Tesis Doctoral

Diseño y evaluación de sistemas de control y procesamiento de señales basadas en modelos neuronales pulsantes

Ángel Francisco Jiménez Fernández

Directores:

Dr. D. Gabriel Jiménez Moreno

Dr. D. Alejandro Linares Barranco

Sevilla, Marzo de 2010

Departamento de Arquitectura y Tecnología de Computadores
Universidad de Sevilla

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
Departamento de Arquitectura y Tecnología de Computadores

Tesis Doctoral

Diseño y evaluación de sistemas de control y procesamiento de señales basadas en modelos neuronales pulsantes

Memoria presentada para aspirar al grado de doctor por:

Ángel Francisco Jiménez Fernández

Ingeniero Informático

Los profesores ponentes y directores:

Dr. D. Gabriel Jiménez Moreno

Profesor titular del Departamento de Arquitectura y Tecnología de Computadores

Dr. D. Alejandro Linares Barranco

Profesor titular del Departamento de Arquitectura y Tecnología de Computadores

Sevilla, Marzo de 2010

Agradecimientos

Este trabajo no hubiese sido posible sin la implicación, colaboración y apoyo de muchas personas, las cuales, en mayor o menor medida han motivado, ayudado y orientado en el desarrollo del trabajo presentado en esta tesis. En especial me gustaría hacer llegar mi más profundo agradecimiento a:

- Mis padres y mi hermana, por todo su apoyo, cariño y comprensión. A ellos está dedicado este trabajo.
- Rocío, por estar a mi lado en todo momento.
- Mis directores Gabriel Jiménez y Alejandro Linares, por sus incontables enseñanzas, sabios consejos y dedicación.
- Antón Civit, Rafael Paz, Lourdes Miró, Francisco Gómez, Manuel Rivas, y en general a todos los miembros del grupo de investigación de Robótica y Tecnología de Computadores de la Universidad de Sevilla, por la ayuda prestada, por compartir conmigo su sabiduría y ser grandes compañeros.
- A los nuevos miembros del Departamento de Arquitectura y Tecnología de Computadores, Manuel Domínguez, Pablo Íñigo y Juan Luis Font, por su contagiosa ilusión y ganas de trabajar.
- A mis amigos más próximos, cuyo apoyo, ánimo y motivación han sido cruciales para la realización de este trabajo.

Índice Principal

Índice de Figuras

Índice de Ecuaciones

Índice de Tablas

1. Introducción y objetivos

1.1. Los sistemas neuro-inspirados, justificación y antecedentes

El comienzo de la vida en la tierra se data hace 4.400 millones de años, desde aquel momento los seres vivos comenzaron a colonizar todo el planeta. A lo largo y ancho de la tierra nos encontramos con una gran diversidad de condiciones medioambientales, desde ambientes favorables hasta otros extremos para la vida, sin embargo los seres vivos han conseguido colonizar exitosamente estos entornos. Una de las claves más importantes para la expansión de la vida ha sido la capacidad de adaptación de los seres vivos, estando dotados por la naturaleza de las cualidades necesarias para poder sobrevivir a un determinado entorno. Además, dada la elevada diversidad de hábitats naturales, los seres vivos no han tenido más remedio que especializarse en la supervivencia en su entorno más próximo, y en consecuencia se ha producido una riquísima diversidad de especies. Las características particulares de cada especie están codificadas en el código genético de cada individuo, las cuales han sido moldeadas desde el origen de la vida gracias a la evolución natural. En 1859 Charles Darwin publicó su teoría de la evolución en el "Origen de las especies", en ella proponía que mediante selección natural los individuos mejor adaptados de una especie en su entorno, tenían mayores probabilidades de sobrevivir y perpetuar así su línea genética, es decir, los individuos más exitosos distribuían su código genético a una mayor cantidad de nuevos individuos. Sin embargo, la transmisión de la información genética de un individuo a otro no está libre de errores, si no que se le incluye un pequeño "ruido" o mutaciones. De manera que el código genético de cada individuo es prácticamente único, presentando pequeñas variaciones que en una u otra medida afectarán a la manera en la que el individuo sobrevivirá en su entorno, siendo ésta la clave de la adaptación y la evolución de las especies. Gracias a la evolución, la naturaleza ha conseguido crear una gran diversidad de especies, logrando una infinidad de soluciones muy eficientes a los problemas de adaptación de los seres vivos en su entorno.

A lo largo de la historia en infinidad de ocasiones los ingenieros se han inspirado en las soluciones alcanzadas por la naturaleza para resolver problemas en los más diversos campos, siendo éste el origen de los sistemas bio-inspirados, encontrándolos a nuestro alrededor cada vez con más frecuencia. Por ejemplo, alguno de los sistemas bio-inspirados, en cierta medida, pueden ser los aviones, submarinos o robots humanoides.

En las últimas décadas la industria ha sufrido una revolución gracias a la aparición de los sistemas computacionales y robóticos. En la industria se le han ido asignando diversas tareas a los robots, en especial tareas peligrosas, aquellas que requieren de una "fuerza sobrehumana", o muy repetitivas y precisas. Sin embargo estos robots están programados para realizar un conjunto muy limitado de tareas, teniendo que estar localizados en un ambiente casi completamente controlado, con una capacidad de adaptación y aprendizaje prácticamente nula, y consumiendo unas cantidades ingentes de energía comparados con los seres vivos. Sin embargo, este hecho contrasta frontalmente con las habilidades de los animales, y sobre todo con la extremada facilidad con la que se desenvuelven en su entorno, los cuales no sólo pueden navegar libre e inteligentemente (dentro de sus posibilidades) por su entorno, si no que son capaces de procurarse su propia energía, aprender, desarrollar actividades sociales,

personalidades propias, comunicarse y organizarse para lograr fines comunes, etc… O en otras palabras, el desarrollo de habilidades sociales, cognitivas y culturales.

En la actualidad la mayoría de los robots están gobernados por comportamientos algorítmicos, procesados por sistemas basados en computadores. En los últimos años los computadores han evolucionado a un ritmo muy alto, alcanzando una capacidad de procesamiento elevadísima, tal y como fue predicho por la Ley de Moore [Moore65]. Sin embargo las habilidades de los robots no han aumentado en la misma medida, si no que van progresando a un ritmo muchísimo más lento. Entonces caben plantearse dos preguntas, ¿se puede modelar con la suficiente fidelidad el comportamiento de un ser vivo para con posterioridad ser codificado algorítmicamente? Y en caso de que así fuera, ¿podría ejecutarse este algoritmo en tiempo real y con suficiente fiabilidad en un computador? Las respuestas a ambas preguntas son en cualquier caso debatibles, y por supuesto dependientes del animal a ser tomado como modelo, pero en nuestra opinión, no parece muy arriesgado afirmar que el ser humano será difícilmente modelable en las próximas décadas. Sin embargo las posibles respuestas a estas preguntas pueden dar lugar a muchas más preguntas, pero de en entre ellas cabe plantearse si los sistemas basados en computador actuales son los sistemas más adecuados para proporcionar a los robots de habilidades cognitivas avanzadas [Penrose89].

La solución a todas estas cuestiones es actualmente desconocida, pero tal vez pueda pasar por tratar de imitar al "controlador" de los propios seres vivos, el sistema nervioso, realizando tareas de "ingeniería inversa", surgiendo así los sistemas neuro-inspirados. Estos son un subconjunto de los sistemas bio-inspirados, los cuales tratan de resolver problemas comunes en la ingeniería mediante el uso de sistemas que están basados en la manera que el sistema nervioso codifica y procesa la información, siendo un campo en continuo desarrollo gracias al trabajo de los ingenieros neuromórficos. El término de ingeniería neuromórfica fue por primera vez usado por Carver Mead en el CalTech al final de los años 80 [Mead89], siendo su objetivo inicial el imitar el comportamiento de las neuronas en el sistema nervioso mediante circuitos analógicos VLSI o aVLSI [Liu02]. Sin embargo el campo de estudio de los ingenieros neuromórficos se expandido en los últimos años, usando tanto circuitos analógicos, como digitales, o de señal mixta para implementar modelos neuronales. Siendo un aspecto muy importante en la ingeniería neuromórfica el entender como mediante el uso modelos neuronales la naturaleza ha creado arquitecturas muy robustas al ruido, increíblemente eficientes, y que además incorporan mecanismos de aprendizaje. La ingeniería neuromórfica agrupa a una serie de investigadores especialistas en los más diversos campos, como son matemáticos, físicos, biólogos, psicólogos e ingenieros de las más diversas ramas.

En 1906 el científico español Santiago Ramón y Caja recibió el premio Nóbel de medicina por sus pioneras investigaciones sobre la estructura microscópica del cerebro, descubriendo que el cerebro estaba formado por un conjunto de células independientes conectadas entre ellas, las neuronas. Sus estudios fueron posibles gracias a los avances en los métodos de tinción y de tecnología de los microscopios, plasmando en láminas ilustradas sus observaciones, pudiéndose encontrar en la Figura 1.1 una reproducción de una de las láminas originales. Las neuronas son un tipo especial de células que transmiten su actividad en forma de impulsos eléctricos o potenciales de acción, y aunque son diversas, la mayoría están

compuestas por tres elementos, el soma, o núcleo de la neurona, las dendritas, compuestas por filamentos de la membrana celular sensible a estímulos externos, y el axón, el cual representa el canal a través del que "viajan" los estímulos generados por una neurona, procesando la información en la conexión entre dendritas y somas, gracias a un fenómeno conocido como la sinapsis [Johnston95].

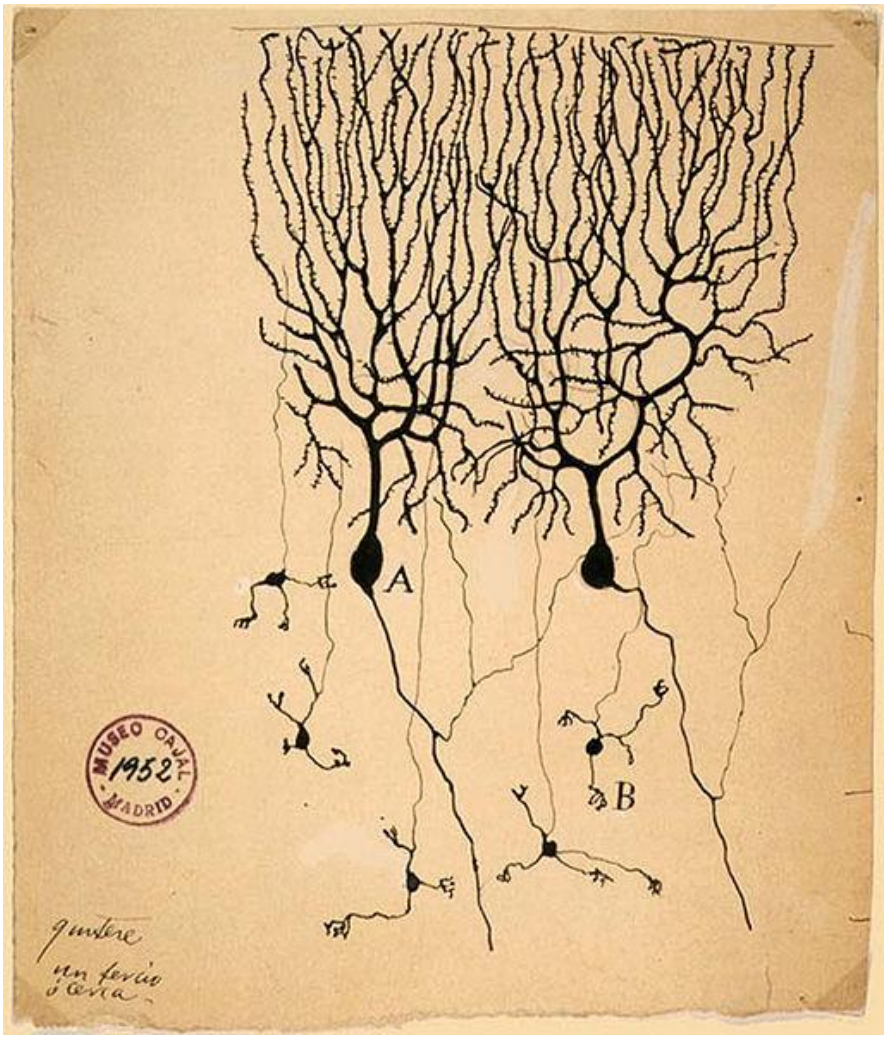


Figura 1.1: Reproducción de una lámina ilustrada de Ramón y Cajal del cerebro de un ave

Las neuronas tienen características fisiológicas muy complejas, en 1952 Hodgkin y Huxley [Hodgkin52] analizaron el comportamiento electrónico de una neurona aislada, estudiando el comportamiento de los canales de sodio y potasio, recibiendo gracias a su estudio el premio Nóbel de medicina en 1963. Demostrando que las neuronas representaban, comunicaban y procesaban la información mediante pequeños pulsos electrónicos en el tiempo, conocidos como potenciales de acción, action potentials o spikes. En su modelo describen el comportamiento de una neurona como una serie de ecuaciones diferenciales no lineales [Lamberti97], un spike consiste en una alteración del voltaje entre membranas de las dendritas de las neuronas, generando las neuronas conectadas nuevos spikes, en base a los spikes recibidos por sus dendritas, estando toda la información codificada en los spikes disparados por las diversas neuronas [Shepherd90][Maass99]. La Figura 1.2 muestra las características

temporales de un spike al ser disparado por una neurona, al principio los canales de la membrana de la neurona se encuentran polarizados con un potencial inferior a un umbral (*Threshold*), no emitiendo, y por tanto, no realizando ninguna actividad mientras este umbral no se supere. En algún momento llegarán a la neurona una serie de estímulos externos, de tal manera que si el potencial de la membrana de la neurona alcanza el umbral, despolarizándose bruscamente, se genera un spike y transmitiéndose un impulso nervioso, conteniendo en él la información procesada. Un breve instante después a la despolarización de la neurona, se polarizará de nuevo hasta "hyper-polarizarse", no siendo capaz de transmitir un nuevo spike hasta transcurrido un período de tiempo conocido como el período refractario.

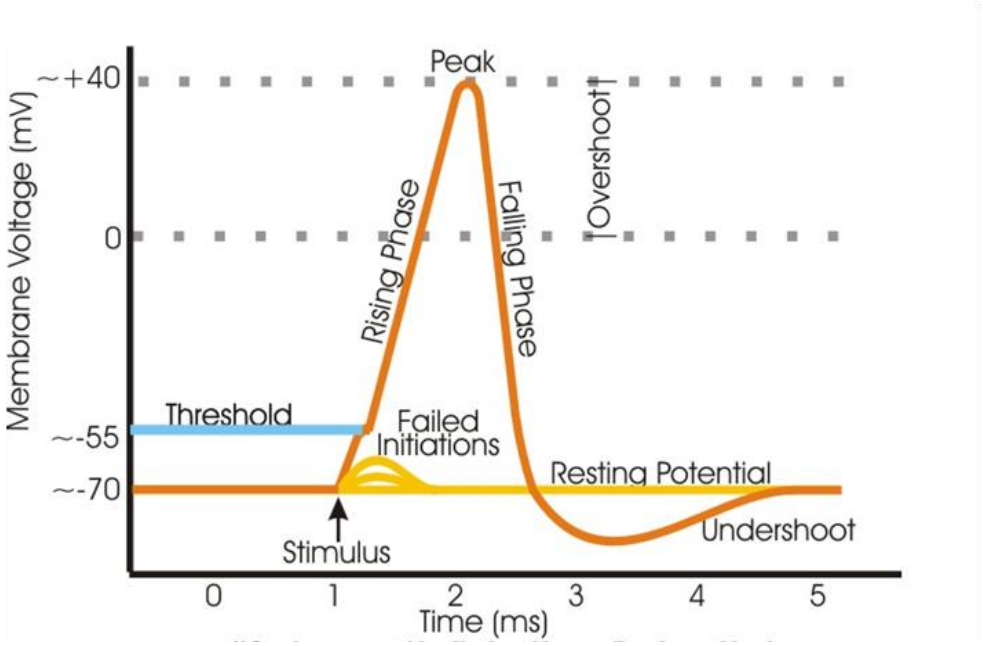


Figura 1.2: Diagrama de un potencial de acción, o spike, generado por una neurona

Uno de los puntos clave es la hipótesis de la representación de la información por parte de las neuronas, mucho se ha debatido sobre como codificar la información en los spikes, Horace Barlow en 1961 propuso varios modelos [Barlow61], siendo un modelo muy aceptado el campo de la ingeniería neuromórfica el que se propone la codificación de la información en la frecuencia de los spikes, siguiendo una modulación en frecuencia de pulsos, spikes, o PFM [Westerman97][Maass99]. De esta manera la información puede ser codificada de forma continua, sin necesidad de realizar una discretización temporal de la información [Fujii96] [Hynna01], es más, un computador digital codificará la información como un número binario, necesitando un elevado número de bits, además de necesitar introducir ciertos "artificios", como representar la información en coma flotante, para poder aumentar la precisión con la que representar la información. Siendo la representación pulsante extremadamente eficiente desde varios puntos de vista, comenzando con su simplicidad, reduciendo la cantidad de canales de comunicación necesarios para transmitir los spikes, y finalmente, aunque no menos importante, proporcionándonos una información continua, en ningún caso discreta. Ambos hechos tienen dos consecuencias muy interesantes, la minimización de canales de comunicación, permitiendo tasa de conectividad entre las neuronas altísimas, y además, al no estar la información muestreada, no se transmite información redundante, sino que sólo se

transmiten los spikes cuando son necesarios, no saturando los canales de comunicación innecesariamente.

El cerebro es el elemento central del sistema nervioso de todos los vertebrados y la mayoría de los invertebrados, localizándose en la cabeza, protegido por el cráneo, y próximo a los órganos sensitivos más importantes, como son: la vista, el oído, el equilibrio, el gusto y el olfato. El cerebro humano es extremadamente complejo y se estima que contiene entre 15 y 33 billones de neuronas, pudiendo estar cada una de ellas conectadas con otras 10 mil neuronas. Estructurándose en capas de neuronas, las cuales están especializadas en procesar una parte de la información, y tienen una funcionalidad más o menos definida [Rakic88] [Shadlen94]. Para realizar el procesado de la información cada neurona está conectada a un campo proyectivo de neuronas a lo largo de diversas capas, fluyendo la información entre capas, y procesándose en este mismo fluir. Por ejemplo, se piensa que el córtex cerebral humano, o neo-cortex, es la parte más "evolucionada" de nuestro cerebro, siendo el responsable de la memoria, la atención, el pensamiento, el lenguaje y la conciencia.

En la Tabla 1-1 realizamos un comparativa cualitativa desde un punto de vista muy general entre la manera en que funcionan un computador y el sistema nervioso de los seres vivos. La primera diferencia que encontramos es que un computador está sincronizado por una señal de reloj global, que le hace reaccionar continuamente cada ciclo de reloj, sin embargo, las neuronas se comportan manera completamente asíncrona, no existiendo ningún mecanismo explícito de sincronización entre ellas. Además el computador es elemento completamente determinista, dictaminando en todo momento tras una sucesión de operaciones aritméticas y lógicas en qué estado debe encontrase, en yuxtaposición, las neuronas responden a un modelo estocástico, dependiendo su reacción de modelos probabilísticos dinámicos. Los sistemas actuales computacionales tienen una alta resolución de la información que manejan, muestreada a un ritmo constante, una vez más, el sistema nervioso es completamente opuesto, la resolución de la información no es tan elevada, pero es capaz de adaptarse a las características de la información para mejorar su representación. En los sistemas computacionales actuales el procesamiento en sí de la información está muy centralizado, en el caso de ordenador personal, o levemente distribuido, como en un clúster, comparado con la manera en que las neuronas procesan la información, ya que cada neurona procesa de manera muy simple una pequeña parte de la información, no dependiendo de otras neuronas, e implementando de esta manera un modelo de procesamiento de la información masivamente paralelo. Para usar los computadores actuales resulta imperativo el proveerles de dispositivos de memoria tanto para almacenar los algoritmos a ejecutar, así como los datos iniciales, intermedios y finales, no obstante, los sistemas nerviosos no necesitan memoria para ninguno de estos fines, ya que por un lado el "algoritmo neuronal" que ejecutan las neuronas está modelado mediante las características fisiológicas de cada neurona y la topología con la que se conecta con otras neuronas, y por otro lado, toda la información relativa al procesamiento simplemente fluye, no se deja almacena en ninguna posición de memoria, sino que parte desde los órganos sensitivos y va siendo procesada a medida que va atravesando capas neuronales. En consecuencia, desde el punto de vista de los sistemas computaciones, la solución alcanzada por la naturaleza se presenta completamente revolucionaria en

muchísimos aspectos, además de los ya comentados, representando el desarrollo de elementos computacionales neuro-inspirados una actividad muy innovadora y prometedora.

Tabla 1-1: Comparativa cualitativa entre un computador y un sistema nervioso

Computador	Sistema nervioso
Reloj centralizado de alta velocidad.	Asíncrono, sin ninguna señal global de reloj.
Dictamina perfectamente y de manera determinista el estado lógico en que debe encontrarse.	Las neuronas se comportan de forma estocástica, respondiendo a modelos probabilísticos.
Alta resolución de la información con una tasa de muestreo constante.	De baja resolución, pero adaptativo. Sin período de muestreo, estando la información contenida en los spikes.
La computación está centralizada o levemente distribuida.	Cada neurona procesa una pequeña parte de la información, implicando una computación completamente distribuida y masivamente paralela.
La memoria está muy "lejos" del computador, necesitando memoria tanto para el algoritmo como para almacenar los datos.	Las características morfológicas y las interconexiones de cada neurona son el algoritmo en sí, la información está contenida dentro de las neuronas.

Las neuronas en el sistema nervioso presentan una alta tasa de conectividad, pudiendo estar cada una conectada a otras 10 mil, como acabamos de exponer. A la hora de implementar sistemas neuro-inspirados artificiales es muy difícil, o prácticamente imposible, alcanzar esta tasa de conectividad entre las neuronas en el interior de un circuito integrado aVLSI, y sobre todo, conectar de manera dedicada las neuronas de distintos chips entre ellas. Sin embargo, aunque la naturaleza nos aventaja en este aspecto, las neuronas presentan tiempos de respuesta del orden de milisegundos, mientras que los tiempos de los circuitos electrónicos actuales están en el orden de los nanosegundos.

La solución al problema de la comunicación entre chips neuro-inspirados fue propuesta por Silvotti en el año 1991 [Silvotti91], la representación dirección-evento, o Address-Event Representation (AER). En la Figura 1.3 se muestra de manera esquemática como se realiza la transmisión de la información mediante la representación AER entre dos chips neuro-inspirados. La representación AER propone el uso de un bus común multiplexado de alta velocidad para la comunicación de los spikes disparados por las neuronas de un chip, el bus AER. La idea es asignar una dirección, address, a cada neurona, de manera que cada vez que una neurona dispare un spike en el chip transmisor, un arbitrador hará que aparezca en el bus AER la dirección de la neurona que ha producido el spike, evento AER, pudiendo ser transmitido de diversas maneras. Una vez recibido un evento AER en el chip receptor, este es decodificado, enviado el spike original a una serie de neuronas receptoras. De esta forma, las neuronas de cada chip se encuentran virtualmente conectadas, fluyendo la información entre ellas, multiplexando el acceso a un canal de comunicación común, sólo accediendo a él las neuronas más activas [Boahen98][Boahen00][Zaghloul04]. Se han propuesto diversas maneras

de implementar el protocolo comunicación de eventos AER. Estos protocolos de comunicación los podemos clasificar en dos grupos, los protocolos paralelos [Boahen98][Boahen00] [Linares03][Häfliger04] y los protocolos serie, serial AER o SAER, [Miro06][Miro07][Berge07] [Fasnacht08][Zamarreño08]. En general los protocolos más extendidos son los protocolos paralelos, en particular en este trabajo vamos a tomar como referencia el usado en el proyecto europeo CAVIAR, especificado en [Häfliger04]. En la actualidad los protocolos serie están en una fase de modelado y desarrollo inicial, no encontrándose en la actualidad implementaciones prácticas aplicadas a los sistemas reales, aunque no nos cabe duda alguna que estos protocolos remplazarán paulatinamente a los protocolos paralelos, tal y como ha ocurrido en general con los sistemas digitales actuales de altas prestaciones.

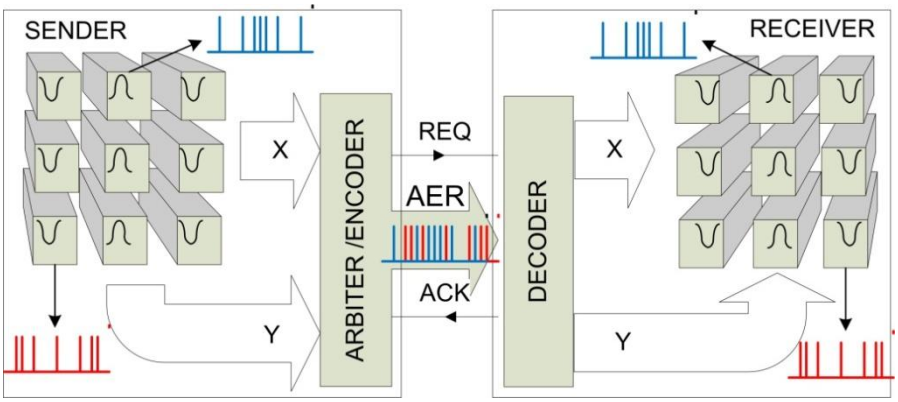


Figura 1.3: Esquema de la transmisión de la información pulsante mediante el uso de la representación AER

Actualmente podemos encontrar una gran variedad de implementaciones hardware de sistemas neuro-inspirados que utilizan la representación AER, encontrándonos una gran variedad de elementos desarrollados en los más diversos centros de investigación. Antes de poder procesar la información pulsante codificada según la representación AER, resulta imperativa su obtención, para ello encontramos una gran variedad de sensores neuro-inspirados. Las retinas son sensores de visión de neuro-inspirados, en las que cada pixel representa a una célula de una retina biológica, emitiendo una serie de spikes cuya frecuencia dependerá de cierta función de la iluminación a la que se ve sometida, existen diversas implementaciones, ya sean en chips aVLSI [Mahowald92]Boahen99][Barbaro02][Culurciello03] [Lichtsteiner05][Lichtsteiner06][Lichtsteiner08][Costas07][Leñero09], retinas sintéticas basadas en FPGA [Paz09b], o retinas stereo [Mahowald94][Philipp04]. Además también podemos encontrar un ejemplo de fusión sensorial en [Jimenez10b], en la que se combinan una retina y un acelerómetro para el diseño de un sistema de estabilización visual neuro-inspirada. Otros sensores bastante extendidos son las cócleas sintéticas, estos sensores son sensibles al sonido, descomponiéndolo en componentes frecuenciales y transmitiéndolas como eventos AER [Lyon88][Lazzaro93][Schaik05][Chan07][Hamilton08], en el Capítulo 8 del presente trabajo propondremos el modelo de una cóclea sintética y las estudiaremos más en profundidad.

Para procesar la información AER existen diversos modelos computacionales y elementos que lo implementan. Un modelo muy usado es el modelo de Integrate & Fire, encontrando diversos chips neuromórficos que contienen capas de neuronas implementando este modelo

[Liu04][Goldberg03]. También encontramos sistemas que realizan convoluciones basadas en pulsos o eventos AER, existiendo una gran variedad de convolucionadores AER, ya sean analógicos [Serrano05][Serrano07][Serrano08], como digitales [Paz08] [Camuñas08]. Además se pueden encontrar nuevos sistemas empotrados de procesamiento AER en [Lujan07a][Lujan07b], aprovechando la potencia del co-diseño hardware/software de sistemas AER, así como propuestas de redes híbridas, en las que se proponen nuevas arquitecturas de redes celulares para integrarlas con sistemas AER [Linares08] Como aplicaciones particulares de los convolucionadores, se han presentado diversos dispositivos para el filtrado de visión AER [Serrano-Gotarredona99][Gomez01]. Otros modelos muy extendidos son, el de la sinapsis probabilística [Westerman97], mediante la que los spikes son procesados mediante el uso de técnicas probabilísticas [Goldberg01], y el modelo Winner-Take-All, en la que sólo la neurona más activa de toda una capa dispara spikes, implementando de esta manera modelos atencionales [Indiveri00][Oster07]. Al igual que los seres vivos tienen la capacidad del aprendizaje, también existen implementaciones basadas en la representación AER pulsante capaces de ser entrenados para aprender patrones, encontrando diversos modelos [Inidvieri99][Hynna01] e implementaciones en circuitos AVLSI [Indiveri06][Yang06] [Häfliger07], así como propuestas para el aprendizaje de robots [Molina08]. Estos sistemas se han utilizado para las más diversas aplicaciones, como por ejemplo son el reconocimiento de voz [Chakrabartty10], la eco-localización [Yu09], categorización de escenas [Thorpe01][Delorme00][Rousselet02][Rousselet03], reconocimientos faciales [Delorme01] y de caracteres [Perez08]. Además en podemos encontrar una gran variedad de ejemplos en los que estos sistemas han sido combinados y fusionados para ser integrados en plataformas robóticas [Lewis01][Lewis03][Vogelstein06][Vogelstein08][Gomez07][Linares07c][Jimenez09c]. El diseño, implementación, e integración de sistemas neuro-inspirados no sería posible sino no existieran herramientas especializadas para la depuración y el desarrollo de sistemas AER. Las llamadas AER-Tools [Paz05][Gomez06], son una serie de elementos que permiten la depuración e interfaz con un ordenador personal de los sistemas AER [Paz05][Paz06] [Berner07], para de esta manera poder visualizar, reproducir, y analizar la información AER que circula por un sistema multi-chip neuro-inspirado basado en la representación AER.

Existen dos grandes citas anuales para los ingenieros neuromórficos, los workshops de Telluride [Telluride] y Capocaccia [Capocaccia], representando un punto de encuentro para investigadores de todo mundo. En estos workshops los investigadores exponen a la comunidad científica sus últimos avances, además de proporcionar la oportunidad de trabajar con personas otros centros de investigación, así como posibilitar la combinación de los más diversos dispositivos hardware neuro-inspirados para el diseño de sistemas de procesamiento de información neuro-inspirada novedosos.

El grupo de investigación de Robótica y Tecnología de Computadores de la Universidad de Sevilla ha participado en los últimos años en varios proyectos de investigación para el desarrollo de sistemas pulsantes AER. Este trabajo ha sido posible gracias al auspicio y financiación del proyecto europeo CAVIAR, y los proyectos nacionales SAMANTA 1/2, y VULCANO, en los que han participado entidades tan importantes como: INI (pionero a nivel mundial en los sistemas neuro-inspirados), el IMSE (líder mundial en el diseño de

convolucionadores AER), o el grupo NeuroCor (expertos en el diseño y control de robots antropomórficos). Los proyectos mencionados pretendían en general la realización de demostradores que permitieran poner a prueba las posibilidades del procesado AER, en especial en lo referente a los campos sensorial y motor.

1.2. Motivación del presente trabajo

Una cuestión pendiente de resolver con respecto al sistema nervioso de los seres vivos es cómo se consigue tal grado de eficiencia usando, en principio y casi exclusivamente, un modelo de la información basado en señales pulsantes, de hecho se desconoce el mecanismo de codificación utilizado. Esta incógnita es difícil de resolver y parece fuera de nuestro alcance en la actualidad, sin embargo en este trabajo sí nos planteamos hacer uso de señales pulsantes bio-inspiradas para resolver problemas comunes en la ingeniería. Es evidente que no vamos a imitar de forma fidedigna al sistema nervioso de los seres vivos, pero el uso de sistemas neuronales pulsantes artificiales sí nos pueden ir mostrando ciertas características que nos irán acercando a dicho conocimiento de la naturaleza. Por otra parte, los nuevos mecanismos y sistemas que podamos obtener con la tecnología pulsante neuro-inspirada pueden representar ciertas ventajas tecnológicas en diversas aplicaciones prácticas.

En la mayoría de los proyectos anteriormente mencionados se han planteado problemas o demostradores para ser resueltos de forma neuro-inspirada, en especial convolucionadores para procesado de imagen, control y aprendizaje. La idea básica es representar la información de forma pulsante y después actuar sobre flujo de pulsos (AER) para, “simplemente quitando y/o poniendo pulsos”, procesar la información. El término “simplemente” debe entenderse como que las operaciones a utilizar son sencillas: quitar y poner pulsos, pero no implica que sea evidente cuales, o cuantos, debemos quitar o poner. De los problemas planteados en dichos proyectos, fundamentalmente dos han servido de motivación al presente trabajo: el diseño de controladores en lazo cerrado mediante sistemas pulsantes (en principio para ser usados con motores eléctricos en robots), y el diseño de filtros simples (también en principio para la síntesis de cócleas artificiales).

Ambas aplicaciones, evidentemente, pueden ser resueltas con la tecnología actual de diversas formas, básicamente podemos utilizar tecnología digital, analógica, o mezcla de ambas. En este trabajo se intentan resolver estos problemas llevando la utilización de los sistemas pulsantes al extremo máximo. Pero lo sorprendente es que los sistemas resultantes son relativamente simples, obteniéndose además una forma de proceder, o metodología, que puede ser de aplicación a otros problemas diferentes a los dos planteados.

La primera decisión a tomar para intentar resolver mediante el procesado de pulsos estos problemas es cómo codificar la información. En diversos trabajos [Maass99][Thorpe96] se recogen numerosas formas de codificar la información para su uso en sistemas neuro-inspirados. En este trabajo nos hemos decantado, en casi todos los casos, por el mecanismo clásico de PFM con pulsos estrechos, por ser el más simple y con mayores posibilidades de procesado. En dicho mecanismo de modulación se representa la información mediante el

tiempo entre pulsos o frecuencia de los mismos. De esta forma una señal analógica se puede representar, por ejemplo, asociando su amplitud en cada instante a una frecuencia de pulsos. Este mecanismo de modulación es muy parecido a la modulación delta-sigma usada en conversores A/D, mostrada en la Figura 1.4, en la que la señal número 5 representa un pulso equiparable a un spike AER/PFM, pero obsérvese que no se cuantiza ni muestrea periódicamente la señal analógica, la información está contenida en el tiempo entre pulsos, y éste es un valor analógico [Morgado04].

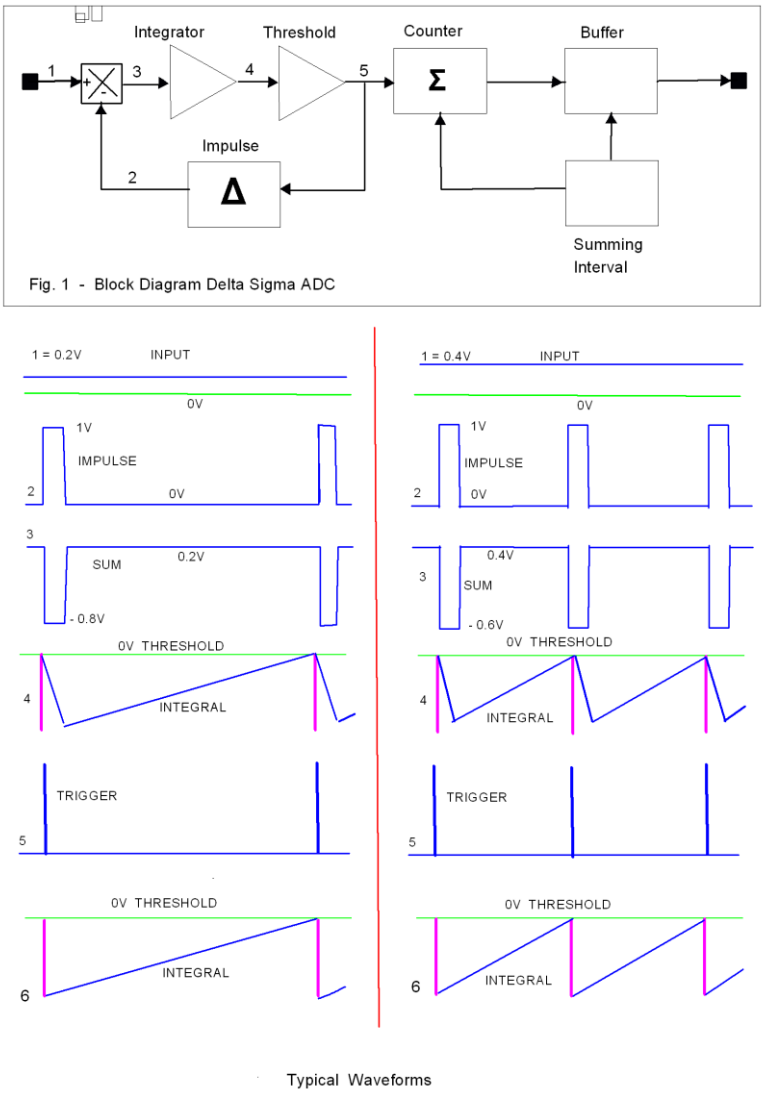


Figura 1.4: Ejemplo de la modulación Sigma-Delta

Se trata por tanto de diseñar dispositivos que sean compatibles con los sistemas AER/PFM y resuelvan, utilizando el "procesado" de esos mismos pulsos, los dos problemas planteados. Pero ¿a qué procesado nos referimos?, es evidente que para procesar el flujo de pulsos vamos a utilizar circuitos digitales y en cierta forma algoritmos, pero la obtención de dichos algoritmos no la vamos a basar en el procesado digital clásico, puesto que las señales pulsantes guardan la información de forma analógica, de manera que resulta natural que se

utilice una "analogía" con este tipo de sistemas para obtener los elementos de procesado. En este sentido podemos remontarnos a trabajos como [Mass99] en los que se planteaba ya, de forma incipiente, una aritmética con pulsos, pero aquí se da un paso más allá, se generaliza una forma de procesado equivalente al analógico, es decir, no sólo se muestra como resolver los dos problemas planteados si no que señalamos además como diseñar sistemas de la misma forma a la utilizada en los computadores analógicos, utilizando la "analogía". Pero con una diferencia fundamental: los sistemas diseñados no precisan de circuitos analógicos, pueden ser implementados en un dispositivo netamente digital como es una FPGA de forma relativamente simple, aunque las señales tratadas sean de naturaleza analógica (tiempo entre pulsos). Resulta evidente que al ser tratadas dichas señales pulsantes mediante dispositivos digitales como las FPGA, mediante la implementación de circuitos digitales síncronos, tenemos finalmente un reloj que cuantiza los valores analógicos del tiempo entre pulsos, en este trabajo vamos a obviar este detalle suponiendo un periodo de reloj muy pequeño frente a los tiempos entre pulsos, por otra parte dicho problema se puede resolver usando circuitos asíncronos. Problemas equivalentes también pueden surgir del valor finito de los registros (saturación) o de la discretización de los mismos (siendo común el hecho de usar registros con números enteros). Por otra parte es lógico pensar que el procesado en el sistema nervioso de un ser vivo no va a ser como el que se muestra en este trabajo, pero sí podemos afirmar que será más próximo que el realizado por un sistema analógico clásico o un computador digital.

En resumen de las funciones de transferencia de los sistemas objeto de estudio y basándose en elementos simples de tratamiento de pulsos (sumador, integrador, derivador…) se diseñan sistemas digitales pulsantes que replican sus correspondientes sistemas analógicos. En cierta forma se muestra, creemos que por primera vez, un procedimiento de aritmética de pulsos con posibilidad de ser utilizada de forma equivalente a un sistema analógico. Buena parte de este trabajo se centra en buscar dichos elementos de procesado simple e identificar sus parámetros a partir de las funciones de transferencia con la que se pretende realizar la analogía. Es evidente que una vez planteado el nuevo sistema habrá que simularlo y caracterizarlo de forma correcta para después probarlo en un demostrador.

Un aspecto importante es como de fidedigna es la representación pulsante PFM con respecto a la señal origen analógica, respuesta a esto la podemos encontrar en [Morgado04]. En general tenemos que las componentes en frecuencia de la señal analógica que se pretenda representar y el índice de modulación son parámetros importantes en este contexto, las componentes de alta frecuencia de la señal analógica, que además sean de amplitud baja, serán difícilmente representables mediante pulsos. En [Morgado04] podemos observar como la representación PFM no puede tener una frecuencia inferior a la de la señal que representa, esto es similar al teorema del muestreo. En este trabajo se obvia estos problemas de representación y suponemos que siempre se tienen señales e índices de modulación correctos.

1.3. Objetivos y estructura de la tesis

Podemos señalar dos tipos de objetivos: unos generales, que pretenden avanzar en el estudio de los sistemas neuronales, y otros específicos, centrados en la resolución de problemas particulares y el diseño de sus correspondientes sistemas reales.

a) Objetivo General: Búsqueda de nuevas arquitecturas computacionales, basadas en sistemas pulsantes análogos a los sistemas neuronales.
    a. En el cerebro se especializan elementos en cada tarea: Cada tarea u operación la vamos a identificar a hardware dedicado.
    b. Adaptar los sistemas computacionales digitales existentes a sistemas basados en la representación pulsante. No sabemos el "programa" del cerebro, pero sí sabemos cómo lo hacen actualmente los computadores, lo cual parece un buen punto de inicio.
b) Objetivos Específicos: Diseñar nuevos elementos que implementen estas operaciones, para lo que se van a realizar aportaciones a dos campos:
    a. Aplicación de los modelos pulsantes al control de actuadores robóticos.
        i. Diseño de interfaces entre motores y sistemas pulsantes.
        ii. Identificación de sistemas equivalentes a los tradicionales para el control de motores.
        iii. Análisis del funcionamiento de los controladores junto con motores reales.
    b. Generalización de los elementos usados en el control para procesar señales codificadas con spikes imitando a los filtros analógicos.
        i. Diseño de filtros de spikes.
        ii. Caracterización e identificación paramétrica de los filtros de spikes
        iii. Simulación y análisis de los filtros diseñados.
    c. Diseño de plataformas de demostración:
        i. Eddie: Diseño y construcción de un robot móvil diferencial neuro-inspirado, como demostrador de los controladores basados en spikes
        ii. La Cóclea Sintética: Propuesta de un sensor de audio, el cual imita al oído interno de los seres vivos, mediante el procesamiento de sonido codificado como spikes.

El trabajo de investigación que presentamos se compone de 9 capítulos, a lo largo de los cuales se pretenden abarcar los objetivos propuestos con anterioridad:

- En el Capítulo 1 se trata la introducción y objetivos, se muestra el estado del arte de los sistemas neuro-inspirados basados en modelos pulsantes.

- Para poder integrar los actuales sistemas neuro-inspirados con plataformas robóticas, es necesario desarrollar nuevos elementos que hagan de interfaz entre los actuadores propios del robot. La primera cuestión a tratar, gira en torno a cómo actuar sobre los motores de un robot con señales codificadas como spikes. Para ello en el Capítulo 2 hemos propuesto dos alternativas para realizar esta tarea, además se presenta el entorno de simulación con el que hemos analizado todos los elementos presentados, así como los resultados de las simulaciones de ambas alternativas, comparándolas en sus fortalezas y debilidades. (Objetivo 2.a.i)

- Los sistemas de control más comunes en la industria son los sistemas de control en lazo cerrado, en el Capítulo 3 vamos a proponer dos nuevos componentes para diseñar sistemas de control equivalentes a controladores proporciones, P, en lazo cerrado, simulándolos con las dos alternativas presentadas en el Capítulo 2, y de nuevo realizando una comparativa entre la idoneidad de usar uno u otro mecanismo de control. (Objetivo 2.a.ii)

- Entre los controladores en lazo cerrado, uno de los más extendidos es el controlador Proporcional-Integral-Derivativo (PID), en el Capítulo 4 vamos a presentar un nuevo elemento que nos permitirá implementar de manera equivalente las componentes integrales y derivativas del controlador, así como la integración de todos los componentes presentados para la obtención de un controlador PID basado en spikes, equivalente a los controladores clásicos analógicos. También se ha simulado y analizado el controlador PID completo, identificando los parámetros del controlador, las fuentes de error y su rendimiento computacional. (Objetivo 2.a.ii)

- Una vez diseñado y caracterizado el controlador PID en el simulador ha sido llevado a la realidad, en el Capítulo 5 exponemos como hemos llevado a cabo esta tarea, comenzando con el diseño de una plataforma hardware para dar soporte al controlador, la plataforma AER-Robot, diseñando nuevos componentes para la gestión externa del controlador y la monitorización de la actividad interna del controlador, analizando finalmente el efecto del controlador PID basado en spikes sobre un motor real. (Objetivo 2.a.iii)

- En el Capítulo 6 proponemos un modelo de control basado en spikes para robots móviles diferenciales como una aplicación práctica de los controles expuestos en el Capítulo 5, construyendo como plataforma de demostración un pequeño robot móvil, Eddie. (Objetivo 2.c.i)

- Tras abarcar todas las cuestiones relativas al control, en el Capítulo 7 vamos a adoptar un nuevo punto de vista sobre los elementos presentados con anterioridad, observándolos de una forma más general desde el punto de vista del procesamiento de señales, diseñando filtros frecuenciales de spikes con modelos equivalentes a los filtros frecuenciales analógicos. En este capítulo partiremos del diseño de un filtro paso de baja de spikes básico, incrementando su complejidad y presentando nuevos componentes, para diseñar filtros más complejos como los filtros paso de alta y paso de banda, además todos los filtros han sido simulados y analizados, identificando a nivel paramétrico sus características. (Objetivos 2.b.i y 2.b.ii)

- En el Capítulo 8, exponemos los modelos que subyacen detrás de los sensores de audio neuro-inspirados pulsantes, las cócleas analógicas, proponiendo un nuevo modelo de cóclea sintética que remplaza los elementos analógicos de las cócleas actuales por los filtros paso de banda

presentados en el Capítulo 7, siendo una aplicación práctica muy prometedora. Sin embargo el modelo propuesto es muy complejo a nivel paramétrico, para resolver la sintonización paramétrica proponemos el uso de un algoritmo genético, siendo un ejemplo de cómo ajustar un sensor neuro-inspirado con técnicas bio-inspiradas de una manera rápida y eficiente. (Objetivo 2.c.ii)

- En el Capítulo 9 se sintetiza el trabajo realizado, generalizando y proponiendo nuevas posibles aplicaciones, para finalmente exponer las conclusiones y aportaciones realizadas.

- Finalmente presentamos las referencias bibliográficas usadas en este trabajo agrupadas en el Capítulo 10, y en el Apéndice A incluimos los esquemáticos de la placa de circuito impreso presentada en el Capítulo 5, la plataforma AER-Robot.

2. Mecanismos de actuación sobre motores de DC mediante neuronas pulsantes

Nuestro primer objetivo consiste en aplicar los modelos de las neuronas pulsantes al control de elementos motrices de las plataformas robóticas e implementarlos en circuitos lógicos programables, como son las FPGA [Maxfield04]. En este capítulo se van a proponer diferentes estrategias para el manejo de motores de DC mediante el uso de neuronas pulsantes, implementaciones concretas de los elementos que conforman dichas estrategias, así como simulaciones de su comportamiento junto con modelos de simulación de motores de DC. Nuestra idea es introducir un elemento entre una secuencia de spikes (o pulsos) y un motor de DC, el cual "adapte" la información contenida en la frecuencia de los spikes, al movimiento en sí del motor. Estos elementos introducidos en una red neuronal pulsante multi-jerárquica, estarían localizados principalmente en las capas de salida de la red.

Las salidas de los circuitos digitales se caracterizan por tener un rango de operación discreto, en su mayoría sólo alternan entre '1' y '0'. Para controlar un motor de DC con señales digitales, cuya velocidad es proporcional al voltaje aplicado en sus bornes, hemos de aprovechar sus características frecuenciales.

Es bien sabido que los motores de DC se comportan como filtros de paso de baja. En la Ecuación [2-1] se muestra la función de transferencia de un motor de DC ideal [Ogata04], pudiendo encontrarse en la Tabla 2-1 el significado físico de cada elemento, así como un ejemplo de valor. En la Figura 2.1 observamos el diagrama de Bode correspondiente a la función de transferencia de los motores de DC que se usarán más adelante en las simulaciones. En el diagrama de Bode del motor de DC se aprecian los dos polos de la función de transferencia del motor de DC, así como los decrementos de la ganancia del motor de DC, además de las pérdidas de fase asociadas a ambos polos, decreciendo asintóticamente hasta 180º.

$$M(s) = \frac{\omega_{rad/s}}{V_{motor}} = \frac{K_p}{LJs^2 + (RJ + LB)s + (RB + K_pK_v)}$$

Ecuación [2-1]

Apoyándonos en la Ecuación [2-1], eliminando los términos multiplicados por 's' (dinámicos), obtenemos la Ecuación [2-2], en la que se muestra la expresión a que responde la ganancia estática de un motor de DC. Finalmente, despejando la velocidad estacionaria del motor, en la Ecuación [2-3], obtenemos la velocidad estacionaria alcanzada por el motor, por cada voltio aplicado a sus bornes. Constatando que la velocidad del motor muestra un comportamiento lineal con respecto al voltaje aplicado.

$$K_{Motor} = \frac{\omega_{static}}{V_{DCmotor}} = \frac{K_p}{RB + K_pK_v}$$

Ecuación [2-2]

$$\omega_{static} = K_{Motor}V_{DCmotor}$$

Ecuación [2-3]

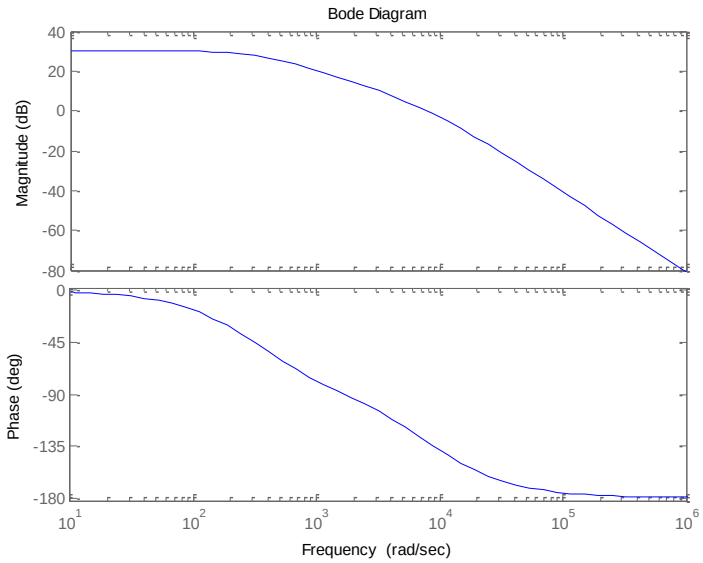


Figura 2.1: Diagrama de Bode del modelo de un motor de DC

Aprovechando las características frecuenciales de los motores de DC, para controlarlos con un sistema digital, se han desarrollado diversas modulaciones digitales, en las que la información no viaja en los niveles lógicos de la señal, sino en su comportamiento temporal o frecuencial. Variando sólo la componente continua del espectro de la señal aplicada al motor, y haciendo que los armónicos introducidos por las señales digitales se encuentren lejos de la banda de paso del motor. Las dos modulaciones digitales clásicas para el control digital de motores de DC son:

- Modulación por Anchura de Pulsos (PWM).
- Modulación por Frecuencia de Pulsos (PFM).

Siendo la más popular la modulación PWM, sobre todo en lo que respecta al control de motores. Aunque en general la modulación PWM se puede considerar como un mecanismo simple de conversión D/A aplicable en sistemas muy diversos, por ejemplo, desde sistemas audio [Soren04] hasta fuentes de alimentación conmutadas.

2.1. Conversión de una señal pulsante a una señal modulada por anchura pulsos:

Una señal modulada siguiendo un esquema PWM tiene un periodo constante (T_{PWM}) y un tiempo en alto variable (T_h). Estando la información contenida en el tiempo que está en alto (T_h), aunque también puede obtenerse de la relación ente T_{PWM} (constante) y Th (variable), el dutty-cycle (Figura 2.3 arriba). El período de las señal PWM, T_{PWM}, ha de tener un valor suficientemente pequeño para que su primer armónico, y los sucesivos, sean filtrados por el motor de DC.

Si aplicamos una señal de PWM directamente a un motor de DC, éste alcanzará en régimen estacionario una velocidad proporcional al dutty-cycle de la señal PWM. Ya que la potencia de la componente en continua del espectro de la señal de PWM, es aproximadamente proporcional a la tensión de alimentación por el dutty-cycle de la señal, tal y

como se muestra en la siguiente Ecuación [2-4]. El resto de componentes han de tener una frecuencia suficientemente elevada para que sean filtradas por el motor [Rivas07].

$$V_{DCmotor} \approx \frac{T_h}{T_{PWM}} V_{PowerSupply}$$

Ecuación [2-4]

Como una primera propuesta para traducir una secuencia de spikes a una señal de PWM, proponemos una arquitectura basada en un modulador PWM y un integrador de spikes (contador digital) [Jimenez08a] La idea radica en que el integrador cuente el número de spikes recibidos a lo largo del período de la señal PWM (T_{PWM}). Cuando T_{PWM} es alcanzado, el contador se "resetea" a cero, y la cantidad de spikes contados se transfiere al modulador de PWM. Esta cantidad de spikes contados responde a una discretización de una señal pulsante con período constante T_{PWM}. El modulador de PWM generará la señal con un dutty-cycle proporcional a la cuenta de spikes del ciclo de PWM anterior.

Para llevar a la práctica esta arquitectura, se ha implementado una entidad VHDL que traduce el tráfico de spikes a una señal de PWM. El esquema de interconexión de las entidades VHDL que componen a este elemento [Jimenez08a] se muestran en la Figura 2.2, a la izquierda el integrador de spikes conectado a las entradas de spikes, y a la derecha el modulador PWM y con la salida de la señal PWM. Ambos elementos están conectados por un bus de varios bits, transfiriéndose la cuenta de los spikes como un número discreto con signo. La cantidad de bits usada en este bus e internamente en el modulador PWM nos dictaminará la resolución de la señal PWM que se generará.

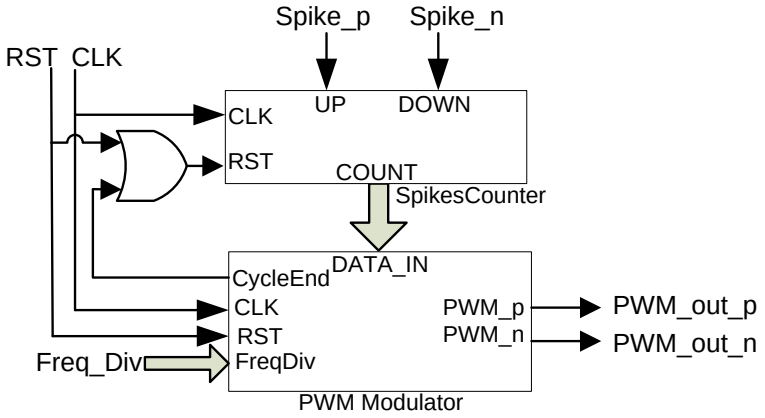


Figura 2.2: Circuito del conversor de spikes a PWM

Obsérvese en la Figura 2.2 como tanto la entrada de los spikes, como la salida PWM tienen un ancho de 2 bits. La entrada de los spikes tiene dos bits para poder recibir spikes con signo (o polaridad), quedando la polaridad del spike determinada por la posición del bit por el que es recibido. Si un spike es recibido por la señal *Spike_p*, el spike se considera positivo; y de forma análoga, por *Spike_n*, será negativo. En caso de recibir un spike positivo, el contador incrementará su valor en '1', sin embargo, en caso de recibir un spike negativo, se decrementará en una unidad. El modulador PWM recibirá la cuenta del integrador con signo a su entrada, seleccionando el canal (o bit) de salida de la señal PWM en base al signo de la

cuenta de entrada. En caso de que la cuenta del integrador sea positiva, la señal de PWM se encontrará en *PWM_p*, y en caso de ser negativa, encontraremos la señal de PWM en *PWM_n*.

Para fijar el valor del período de PWM, T_{PWM}, se dispone de la señal *Freq_Div* de 16-bits (Figura 2.2). El valor introducido en este bus está dirigido a un divisor de frecuencia interno en el modulador, el cual temporizará el modulador PWM. En las siguientes ecuaciones se muestra cómo se ha de calcular el valor del divisor de frecuencia para fijar el período de PWM, en base al número de bits del modulador PWM y el período de reloj de la FPGA.

$$T_{PWM} = (Freq_{Divider} + 1) * 2^{Nº\ Bits} * T_{CLK}$$

Ecuación [2-5]

$$Freq_{Divider} = \frac{T_{PWM}}{2^{Nº\ Bits} * T_{CLK}} - 1$$

Ecuación [2-6]

También podemos calcular el tiempo que la señal PWM estará en alto en base al número de spikes recolectados durante el último período de PWM:

$$T_{HighTime} = Nº\ Spikes \frac{T_{PWM}}{2^{Nº\ Bits}}$$

Ecuación [2-7]

Dado que la frecuencia de los spikes es el número de spikes recolectados en un tiempo fijo, entre este mismo tiempo, podemos calcular el número de spikes recibidos en base su frecuencia:

$$SpikeRate = \frac{Nº\ Spikes}{T_{PWM}} \Rightarrow NºSpikes = SpikeRate * T_{PWM}$$

Ecuación [2-8]

Sustituyendo el número de spikes contados en la Ecuación [2-7], podemos obtener el tiempo en alto de la señal PWM en base a la frecuencia de los spikes recibidos. Obsérvese como el tiempo en alto es lineal respecto a la frecuencia de los spikes. Así como el tiempo en alto crece cuadráticamente con el período de PWM, ya que cuanto mayor sea el período de PWM, más tiempo estará la señal a nivel alto por spikes recibido, y además estará más tiempo recolectando spikes, incrementando la cuenta de spikes para una frecuencia constante.

$$T_{HighTime} = SpikeRate \frac{T_{PWM}^2}{2^{Nº\ Bits}}$$

Ecuación [2-9]

Combinando esta última ecuación con la Ecuación [2-4], en la que mostrábamos el voltaje equivalente en continua en base al dutty-cycle de una señal PWM, podemos obtener el voltaje equivalente aplicado al motor en base a la frecuencia de los spikes de entrada:

$$V_{DCmotor} \approx \frac{SpikeRate \frac{T_{PWM}^2}{2^{Nº\ Bits}}}{T_{PWM}} V_{PowerSupply} = \frac{SpikeRate * T_{PWM}^2}{2^{Nº\ Bits} * T_{PWM}} V_{PS} = \frac{SpikeRate * T_{PWM}}{2^{Nº\ Bits}} V_{PS}$$

Ecuación [2-10]

Según la ecuación anterior el voltaje equivalente aplicado al motor debe ser directamente proporcional a la frecuencia de los spikes de entrada, pudiendo alterar la ganancia del sistema modificando el período de PWM.

Finalmente podemos obtener la velocidad estacionaria teórica de un motor mediante el uso de este componente, en base a los parámetros constantes del sistema, periodo de PWM, resolución del modulador y tensión de alimentación, así como en base a la variable del sistema, que es la frecuencia de los spikes de entrada:

$$\omega_{static} \approx K_{Motor} K_{PWM} SpikeRate = K_{Motor} V_{PS} \frac{T_{PWM}}{2^{Nº\ Bits}} SpikeRate$$

Ecuación [2-11]

En la Figura 2.3 mostramos la salida del modulador PWM ante una entrada de spikes con frecuencia decreciente. Se observa claramente el dutty-cycle de la señal de salida es proporcional a la tasa de spikes medida en el ciclo de PWM anterior. Como cabría esperar y evidencian los resultados de las simulaciones, este elemento introduce una latencia entre la señal de entrada de spikes a su aplicación sobre el motor equivalente a T_{PWM}.

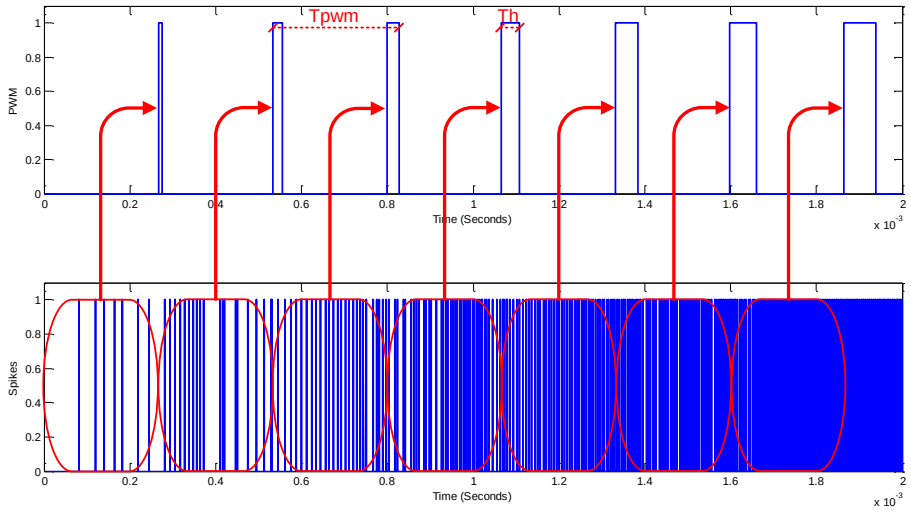


Figura 2.3: Respuesta del conversor PWM ante una frecuencia de spikes de entrada decreciente

A la hora de elegir el período de la señal PWM surge una cuestión acerca del rango dinámico de la señal PWM. Cuanta mayor precisión, bits, tenga el modulador PWM, una mayor tasa de spikes habrá que recibir durante un determinado T_{PWM} para abarcar todo su rango dinámico. Es decir, si tenemos un modulador PWM de 8 bits, se necesitará recibir 128 spikes durante un T_{PWM} para saturarse a '1'; sin embargo, aumentando la resolución 9 bits se necesitarán 256 en ese mismo T_{PWM}. Esto nos obligaría tener que aumentar la frecuencia de los spikes de entrada al doble para tener un rango dinámico completo. Sin embargo, como se aprecia en las ecuaciones anteriores, desde el punto de vista de *Freq_Div*, el número de bits del modulador PWM no influye. En la siguiente ecuación mostramos el número mínimo de spikes necesarios para saturar un modulador PWM en base al valor dado a *Freq_Div*, así como una representación gráfica del mismo en la Figura 2.4.

$$SpikeRate_{Min.\ Sat.} = \frac{Nº\ Spikes}{T_{PWM}} = \frac{2^{Nº\ Bits}}{Freq\_Divider * 2^{Nº\ Bits} * T_{CLK}} = \frac{1}{Freq\_Divider * T_{CLK}}$$

Ecuación [2-12]

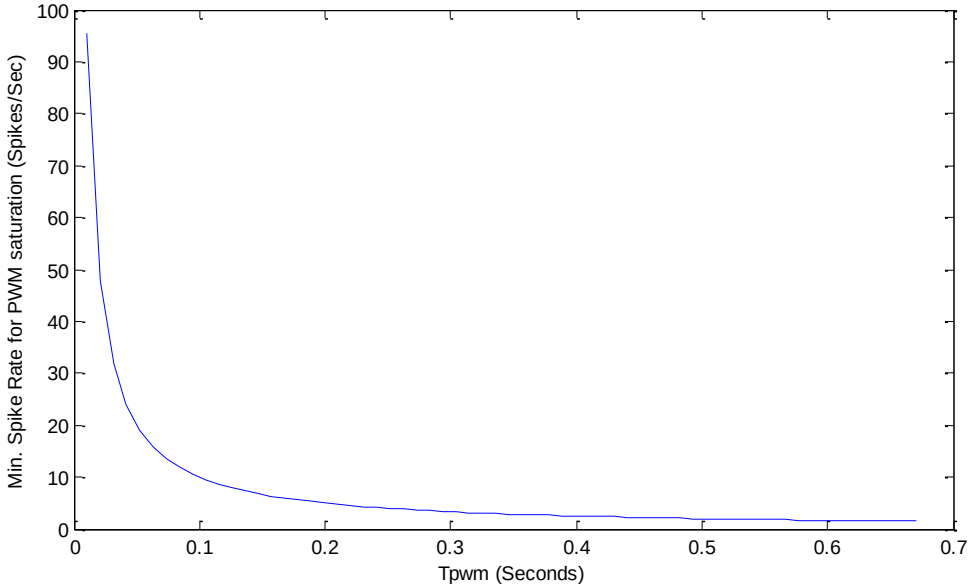


Figura 2.4: Frecuencia mínima de saturación del conversor PWM frente a su divisor de frecuencia

Este método está basado en la medida de la tasa de spikes recibidos durante un tiempo determinado. Podrían desarrollarse otras arquitecturas basadas en la medida del Inter-Spike-Interval (ISI) que paliasen en cierta medida el efecto de la latencia del período de integración. Sin embargo, a priori, esta nueva arquitectura sólo paliaría estos efectos en casos muy determinados (cambios bruscos en la frecuencia de los spikes), no terminaría de evitar la introducción del período de integración.

2.2. Adaptación de una señal pulsante a una modulación por frecuencia de pulsos.

La modulación por frecuencia de pulsos, PFM, resulta especialmente atractiva, ya que coincide con la empleada por los spikes y las neuronas pulsantes en la naturaleza. Una señal modulada mediante PFM sigue un esquema exactamente opuesto a una señal PWM. En la modulación PFM el tiempo en alto es siempre constante[1], pero la frecuencia, o separación temporal entre pulsos, es variable. Estando la información contenida en la frecuencia de la señal. En la Ecuación [2-13] mostramos la tensión en continua equivalente aplicada al motor. Esta ecuación es similar a la usada para el modulador PWM, pero en este caso, en vez de ser el tiempo en alto de la señal fijo y el período (o su inversa la frecuencia) variable, es justo al

[1] Hay otra alternativa para modulación PFM, en la que la señal siempre es cuadrada, esto significaría que tanto su periodo como su tiempo en alto cambiarían a lo largo del tiempo. Este caso no es contemplado en este trabajo, ya que este tipo particular de modulación PFM no es contemplado en los modelos teóricos neuronales.

contrario. Resultando la componente continua de la tensión equivalente en el motor, proporcional a la frecuencia de pulsos aplicada, controlando la ganancia estática en este caso con el tiempo en alto de cada pulso en la señal de PFM, justo al contrario que en la modulación PWM.

$$V_{DCmotor} \approx \frac{T_h}{T_{PFM}} V_{PowerSupply} \approx Th * freq_{PFM} * V_{PS}$$

Ecuación [2-13]

Combinando esta última expresión con la Ecuación [2-3], en la que mostrábamos la velocidad estacionaria del motor frente al voltaje aplicado en sus bornes, podemos obtener la velocidad estacionaria teórica que alcanzará el motor para una frecuencia de pulsos o spikes constante:

$$\omega_{static} \approx K_{Motor} * Th * V_{PS} * SpikeRate$$

Ecuación [2-14]

Siguiendo la filosofía de las neuronas pulsantes, donde intentamos evitar la integración de los spikes en el tiempo, y dado que los spikes siguen una modulación PFM, ¿por qué no aplicarlos directamente al motor de DC, evitando así la introducción de períodos de integración?

La primera dificultad que nos encontramos es la estrechez de los spikes desde el punto de vista temporal (20nS en nuestras FPGA), incapaces de hacer conmutar una etapa de potencia. La solución pasa por incrementar la anchura temporal de los spikes durante un tiempo fijo (*SpikesWidth*) [Rivas07].

Al igual que con los adaptadores PWM, hemos diseñado una entidad VHDL que implementa esta funcionalidad, el S*pikes Expansor*, mostrado en la Figura 2.5. Está compuesto por un contador decreciente, un registro de un 1 bit, y un multiplexor también de 1 bit. De manera que cada vez que un spike es recibido, el contador se carga con el valor del ancho de spike, decrementándose en uno cada ciclo de reloj hasta que llega a cero, parando entonces la cuenta, y estando listo para recibir un nuevo spike. Siendo el bit de cero (/ZERO, activo en baja) del contador que generará la salida, mientras el contador sea distinto de cero, esta señal estará a nivel alto. Para proporcionar la polaridad adecuada a los spikes de salida, con un multiplexor nos encargaremos de redirigir el spike expandido a la señal adecuada. La señal de selección del multiplexor está conectada a un registro de 1 bit, el cual almacena un '0' si el último spike recibido ha sido negativo, y un '1' si el último spike ha sido positivo. Seleccionando así la salida adecuada del spike expandido.

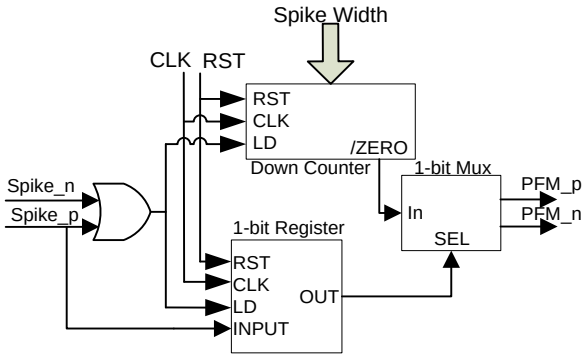


Figura 2.5: Circuito del Spikes Expansor

En la Figura 2.6 nos encontramos un cronograma que muestra el funcionamiento del S*pikes Expansor*. En ella vemos como tras recibir un spike a la entrada (en rojo), inmediatamente se propaga a la salida, y se incrementa durante un tiempo fijo (en púrpura).

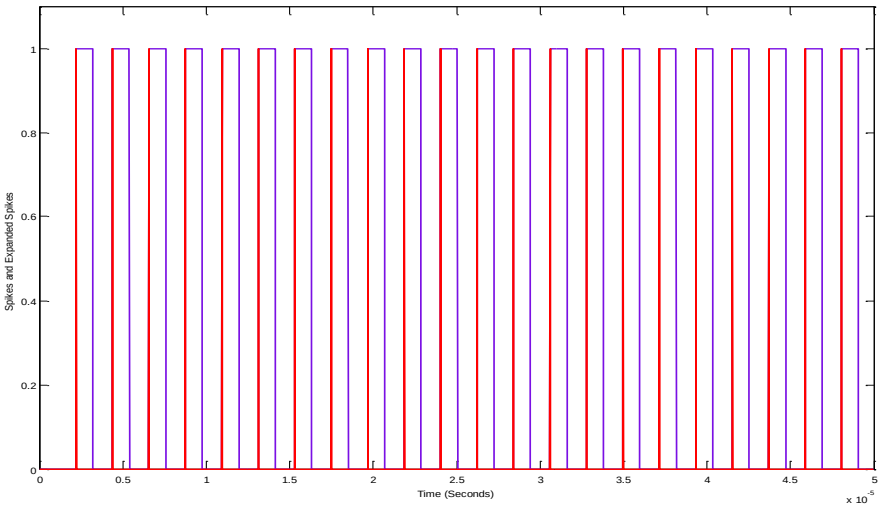


Figura 2.6: Salida del Spikes Expansor ante una entrada de spikes con frecuencia constante

La anchura de los spikes viene determinada por la señal de entrada *spikes_width*, siendo el ancho de los spikes proporcional a este valor, multiplicado por el período de la señal de reloj por la que se rija esta entidad, tal y como se muestra en la Ecuación [2-15]. Un efecto interesante del *Spikes Expansor* es la saturación, ésta ocurre cuando se recibe un spike mientras aún se está incrementando uno anterior. Por ejemplo si tenemos un ancho de spike de 5us, y a los 2us de recibir un spike recibimos otro, el nuevo se incrementará sus 5us, estando la salida del *Spikes Expansor* a '1' durante 7us, en vez de 10us distribuidos homogéneamente en el tiempo. En la Ecuación [2-16] se muestra la expresión a la que responde la frecuencia mínima necesaria para saturar la salida del *Spikes Expansor* a '1'.

$$T_{High} = (Spike\_Width + 1)T_{CLK}$$

Ecuación [2-15]

$$SpikeRate_{Min.\ Sat.} = \frac{1}{(Spike\_Width + 1)T_{CLK}}$$

Ecuación [2-16]

En la Figura 2.7 se muestra la tasa de spikes mínima necesaria para saturar el Spike Expansor. Este parámetro ha de ser elegido en base a las características físicas del motor y de las etapas de potencia, así como de la tasa de spikes máxima que nuestro sistema sea capaz de recibir. Más adelante en este capítulo se mostrará su influencia sobre el sistema.

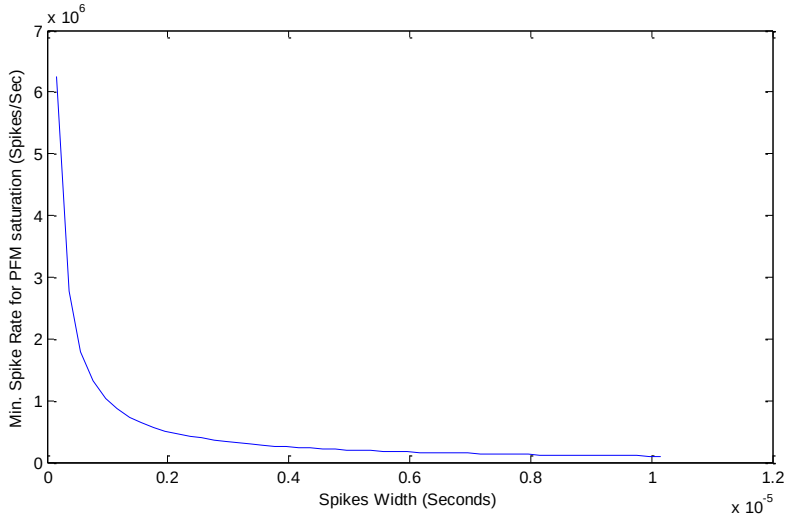


Figura 2.7: Frecuencia mínima de saturación del Spikes Expansor frente al ancho de spike

2.3. Modelos de simulación

Para verificar el comportamiento tanto de los componentes VHDL, como de su interacción con sistemas dinámicos (como son los motores de DC), hemos recurrido al simulador Simulink [SIMULINK], incluido en MATLAB [MATLAB], junto con una extensión de Xilinx para integrar componentes VHDL en Simulink, llamado Xilinx System Generator [SYSGEN]. Para cada uno de los mecanismos de actuación sobre motores de DC basados en spikes, expuestos con anterioridad, se han desarrollado diversas simulaciones en diversos escenarios. En ellas podremos apreciar los diferentes comportamientos que muestran los motores de DC ante los diversos parámetros configurables de estos mecanismos. Así su correlación aproximada con las ecuaciones teóricas.

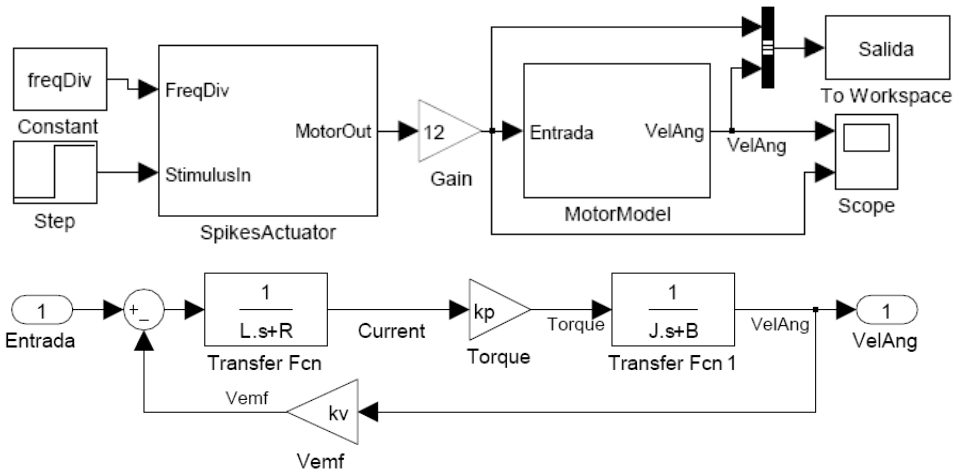


Figura 2.8: Esquemas de simulación en lazo abierto

En la Figura 2.8 mostramos el esquema general de simulación (en particular del PWM), en ella vemos los diversos elementos incluidos en la simulación. A la izquierda el generador de estímulos y una constante, la cual representará los parámetros propios de cada método (*Spike_Width*, *Freq_Div*). En el centro encontramos dos cajas "blancas", la caja de la derecha contendrá el método de actuación propio de cada caso, y la caja de la izquierda, el modelo de un motor de DC de segundo orden, mostrado en detalle en la parte inferior de la Figura 2.8. Finalmente, a la derecha se encuentran las salidas del simulador, por un lado un osciloscopio para comprobar visualmente el resultado de la simulación, y una salida al espacio de trabajo, *workspace*, de MATLAB, donde podremos realizar un procesado a posteriori de los resultados de las simulaciones.

Las simulaciones realizadas exigen una elevada cantidad de memoria y tiempo de cómputo, ya que nuestras FPGA trabajan a 50MHz y han de manejarse una elevada cantidad de muestras. Para hacer viables las simulaciones se ha optado por un motor de DC con una constante temporal muy pequeña. La Tabla 2-1 muestra los valores de los componentes del modelo del motor que se ha usado para las simulaciones. Y cuyo diagrama de Bode fue mostrado en laFigura 2.1. Este modelo es el de un motor real comercializado por Maxon Motors [Maxon], que ha sido elegido para las simulaciones ya que su constante de tiempo es muy pequeña. Sólo necesita 12.8ms para alcanzar el 98% de la velocidad deseada en lazo abierto ante una entrada en escalón. Esta característica ha hecho posible la realización de simulaciones de componentes VHDL reales junto con motores que podríamos encontrar en el mercado.

Tabla 2-1: Parámetros del motor real usado en el simulador

Parámetros	Nombre en la simulación	Valor	Unidades
Voltaje Nominal	-	12	V
Velocidad Máxima	-	402	rad/s
Constante de tiempo	-	3.19	ms
Frecuencia a 0dB	-	1.26	kHz
Resistencia terminal	R	2.06	Ohm
Inductancia terminal	L	0.238e-3	H
Constante de par	K_p	23.5e-3	Nm/A
Constante de velocidad	K_v	24.5e-3	V/(rad/s)
Inercia del rotor	J	10.7e-7	Kgm2
Coeficiente de fricción	B	7.5e-5	Nm/(rad/s)

Según las ecuaciones anteriores y los parámetros proporcionados por el fabricante, la función de transferencia del motor usado en las simulaciones, cuyo diagrama de Bode fue mostrado en la Figura 2.1 es el siguiente:

$$M(s) = \frac{\omega_{rad/sec}}{V_{motor}} = \frac{K_p}{LJs^2 + (RJ + LB)s + \left(RB + K_p K_v\right)}$$

$$= \frac{23.5 * 10^{-3}}{2.553 * 10^{-10}s^2 + 2.222 * 10^{-6}s + 70.72 * 10^{-5}}$$

Ecuación [2-17]

Dado que un sistema de segundo orden responde la siguiente expresión:

$$G(s)=K\frac{\omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2}$$

Ecuación [2-18]

Podemos descomponer nuestro sistema como:

$$M(s)= 33.2282\frac{2.7701 * 10^6}{s^2 + 8.7035 * 10^3 s + 2.7701 * 10^6}$$

Ecuación [2-19]

Analíticamente podemos extraer los siguientes parámetros del motor:

$$K_{Motor}=\frac{K_p}{RB + K_p K_v} = 33.2282\ V\ {}^{rad}/_{sec}$$

$$\omega_n = 1.6644\ * 10^3\ {}^{rad}/_{sec}$$

$$\delta = 2.6147$$

Este motor tiene un coeficiente de amortiguamiento (δ) mayor que 1, mostrando un comportamiento sobre amortiguado. Ofreciendo la siguiente respuesta a un escalón unitario:

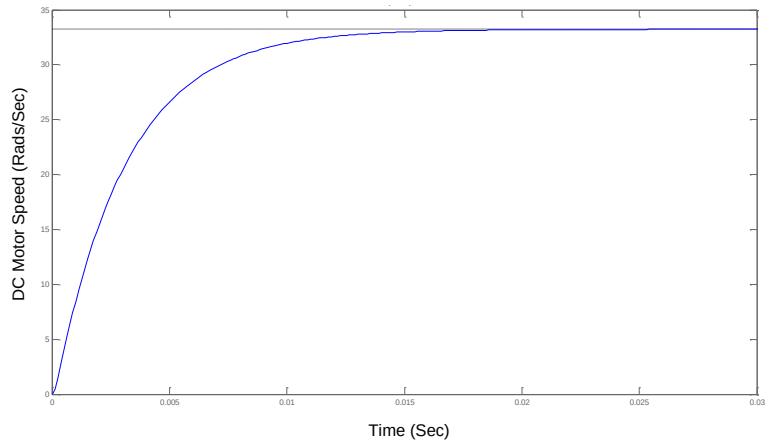


Figura 2.9: Frecuencia mínima de saturación del conversor PWM frente a su divisor de frecuencia

2.3.1. Generación de estímulos del sistema. El generador de spikes exhaustivo bit-wise

En primer lugar, para poder simular los elementos de control que iremos exponiendo a lo largo de este trabajo, necesitamos un elemento capaz de proporcionarnos secuencias de spikes, en base a números discretos aplicados a su entrada, con el fin de excitar a todo el

sistema. Para ello diseñaremos un generador sintético de spikes, de tal manera que la frecuencia de los spikes de salida sea proporcional al valor discreto proporcionado a su entrada; tal y como se muestra en la siguiente ecuación, donde x representa el valor de entrada, y $k_{FreqModulation}$ la constante de modulación en frecuencia:

$$f(x)_{Spikes} = k_{Freq\ Modulation} * x$$

Ecuación [2-20]

Existen diversas propuestas de generadores de spikes sintéticos basados en FPGAs, y orientados generalmente a la generación de imágenes [Linares02] [Linares03] [Gomez05]: 'Scan', 'Uniform', 'Random', 'Random-Square' y 'Exhaustive'. Dado que para aplicaciones de control necesitamos métodos que distribuyan de manera homogénea los spikes en el tiempo, nos vamos a centrar en los métodos exhaustivos [Gomez05][Linares02]. Estos métodos discretizan el tiempo en frames, estando cada frame dividido a su vez por slices, los cuales toman un ciclo de reloj. Disparando spikes en los slices adecuados para obtener una frecuencia de salida adecuada y con una distribución de los spikes lo más homogénea posible.

Uno de estos métodos exhaustivos es el método bit-wise [Paz09a][Paz09b], este método consiste en comparar el valor de entrada con un contador de slices; de tal manera que cuando el valor de entrada sea mayor que el contador de slices se disparará un spike. Sin embargo, si los implementamos de esta manera todos los spikes de un frame serán disparados en los primeros slices. Para solucionar este problema, y lograr una distribución homogénea en el tiempo de los spikes, en vez de comparar el valor de entrada con el contador de slice, vamos a compararlo con el contador de slice con sus bits invertidos. Gracias a la inversión de bits (bit-wise) el valor invertido del contador de slices va alternando entre diferentes valores, evitando una comparación secuencial entre el valor de entrada y contador de slices. En la Tabla 2-2 se observa la secuencia de emisión de spikes para una implementación del método exhaustivo bit-wise de 3 bits, donde en la primera fila se haya el contador de slices, en el resto de las celdas el valor del contador con los bits invertidos (con el que se comparará el valor de entrada), y finalmente en gris se representan los disparados y en blanco los no disparados.

Tabla 2-2: Secuencia de emisión de spikes de un generador exhaustivo bit-wise de 8 bits, para los posibles valores de entrada

Entrada	Contador de slices – Tiempo de frame							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	4	2	6	1	5	3	7
1	0	4	2	6	1	5	3	7
2	0	4	2	6	1	5	3	7
3	0	4	2	6	1	5	3	7
4	0	4	2	6	1	5	3	7
5	0	4	2	6	1	5	3	7
6	0	4	2	6	1	5	3	7
7	0	4	2	6	1	5	3	7

En nuestro caso particular nos interesa poder disparar spikes con signo, es decir, spike positivos así como negativos. Para ello hemos de hacer una adaptación del método bit-wise pudiendo así manejar números con signo, codificados en complemento a 2. En primer lugar, la salida del generador serán dos señales de un bit, disparando en una de ellas los spikes positivos y en la otra los spikes negativos. A continuación, en vez de comparar el dato de entrada con el contador de slices, vamos usar el valor absoluto del dato de entrada para generar los spikes; redirigiéndolos por la señal adecuada en base a su signo. El pseudo-código que describiría este método es el siguiente [Paz09b]:

```
for slice=0:1:max_slice_value
      if (abs(entrada) > bit-wise(slice)) then
            if (entrada > 0) then
                  Spike_p = 1;
                  Spike_n = 0;
            else
                  Spike_p = 0;
                  Spike_n = 1;
            end if;
      else
            Spike_p = 0;
            Spike_n = 0;
      end if;
end;
```

Se ha diseñado un generador que implementa este método en hardware, describiéndolo en VHDL. La idea de este generador es seguir fielmente este método, sus bloques internos pueden verse en la Figura 2.10. Estando situado el contador de slices en la parte superior de la ilustración, junto con el inversor de bits. Abajo a la izquierda vemos la señal de entrada, la cual atraviesa un bloque que calcula su valor absoluto, siendo ésta última la entrada del comparador digital, el cual disparará un spike en caso de que esté activo y que el valor de entrada sea mayor que el contador de slices. Finalmente nos encontramos un demultiplexor, con el que dirigiremos el spike generador por la señal adecuada, siendo la señal de selección del demultiplexor el bit más significativo del valor de entrada (1 en caso de ser un número negativo y 0 en otro caso).

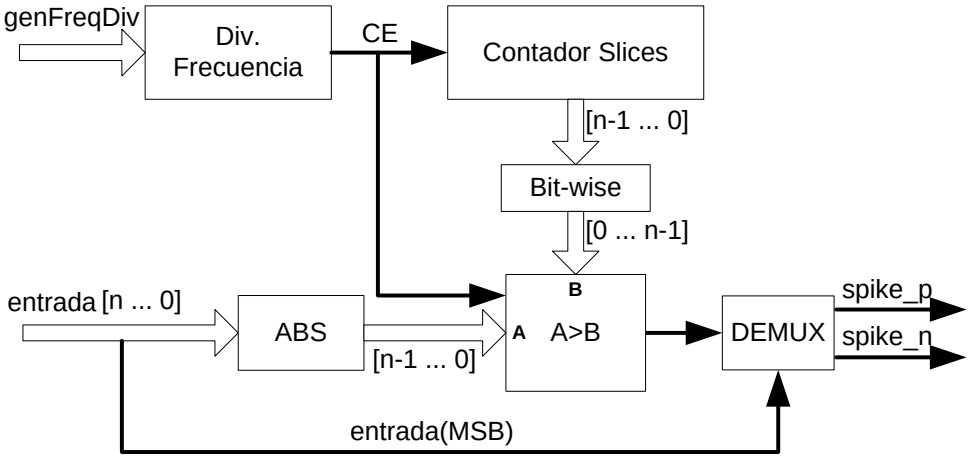


Figura 2.10: Diagrama de bloques del generador exhaustivo bit-wise de spikes

Además hemos añadido un circuito que activa periódicamente una señal de habilitación de circuito (*CE*), esquina superior izquierda, este circuito toma un valor de entrada (*genFreqDiv*), y sólo activa la señal *CE* cuando han pasado un número de ciclos de reloj equivalente a al valor de la señal *genFreDiv*, actuando como un divisor de reloj. Esta señal tiene dos funciones, la primera es que el contador de slices sólo se incremente cuando está activa, así como sólo en este caso se pueden disparar los spikes, ya que si no los spikes ocuparían varios ciclos de reloj.

Analíticamente, la frecuencia de los spikes generados puede ser calculada acorde a la siguiente ecuación:

$$f(entrada)_{spikes} = \frac{F_{CLK}}{2^{n-1}(genFreqDiv + 1)} * entrada$$

Ecuación [2-21]

Donde *Fclk* es la frecuencia de reloj del generador, *n* el número de bits del contador de slices, y *genFreqDiv* el divisor de reloj. Variando estos valores podremos ajustar la recta de modulación del generador a nuestras necesidades en cada caso. Siendo la ganancia del generador:

$$k_{freq\ GenBW} = \frac{F_{CLK}}{2^{n-1}(genFreqDiv + 1)}$$

Ecuación [2-22]

Con el objetivo de poder cuantificar el consumo de hardware del generador exhaustivo bit-wise en una FPGA, más concretamente en una Spartan 3 de Xilinx, hemos sintetizado el generador para diversos número de bits, anotando el consumo en slices de una FPGA, y la frecuencia máxima de reloj a la que puede funcionar el generador. Mostrando los resultados en la siguiente tabla:

Tabla 2-3: Resumen de la síntesis del generador exhaustivo bit-wise para diferentes números de bits

Número de bits	Slices	Máxima Frecuencia
6	31	169.66 MHz
8	34	157.45 MHz
10	40	142.74 MHz
12	48	132.27 MHz
14	49	129.83 MHz
16	50	128.52 MHz

Para comprobar el funcionamiento del generador vamos a comenzar excitándolo con diferentes funciones de entrada, mostrando los resultados en las siguientes figuras. En primer lugar lo hemos excitado con una entrada en rampa, en la Figura 2.11, observándose como según va aumentado la magnitud de la entrada los spikes se encuentran más cercanos en el tiempo, es decir, con mayor frecuencia. A continuación hemos excitado el generador con una

función en seno, en la Figura 2.12, en ella podemos ver spikes tanto positivos como negativos, así como la dependencia de su frecuencia con el valor del seno de entrada.

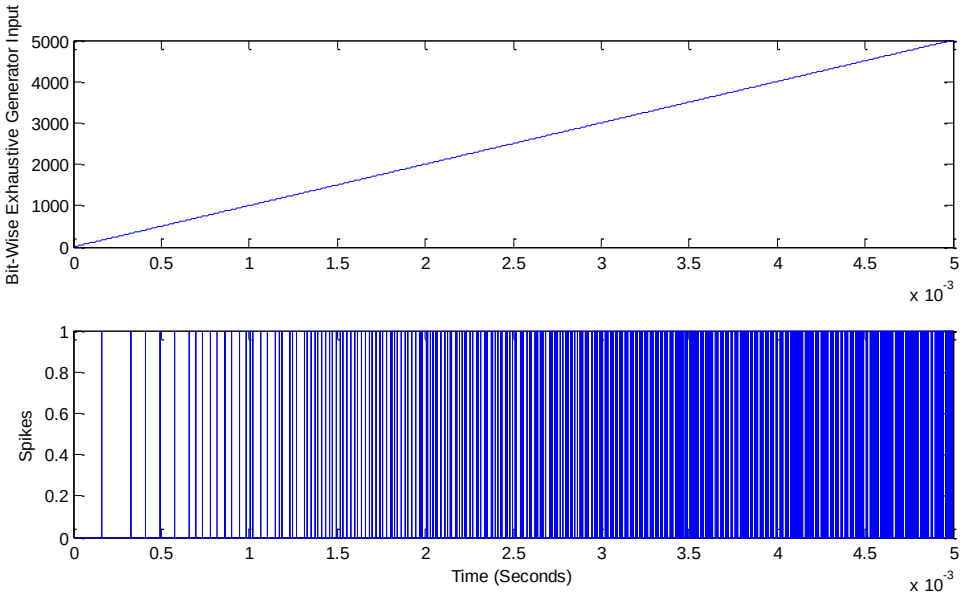


Figura 2.11: Salida del generador exhaustivo bit-wise de spikes ante una entrada en rampa

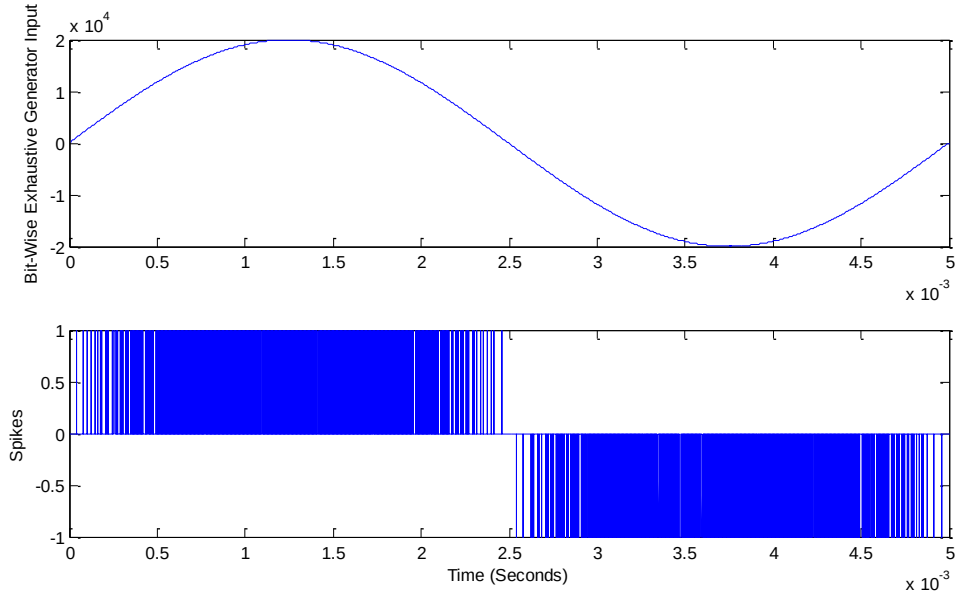


Figura 2.12: Salida del generador exhaustivo bit-wise de spikes ante una entrada en seno

Analizando más en profundidad la respuesta del generador exhaustivo bit-wise implementado, hemos realizado una simulación en la que fijamos el valor del divisor de frecuencia y hacemos un barrido de los posibles valores de la entrada, anotando la frecuencia media de los spikes generados para cada caso. En la Figura 2.13 mostramos las rectas de modulación de generador exhaustivo bit-wise para diferentes divisores de reloj (entre 1 y 10) y 8 bits.

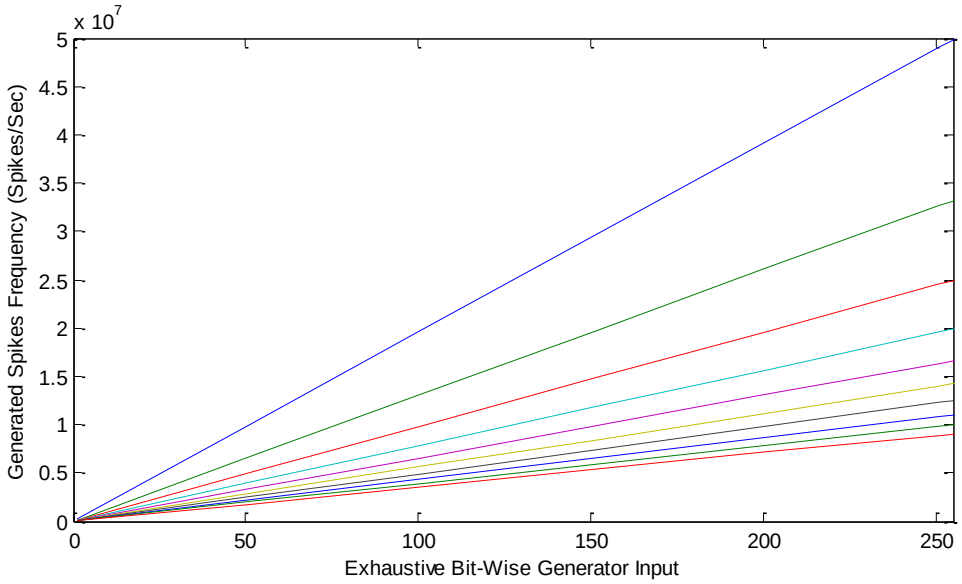


Figura 2.13: Rectas de modulación de un generador exhaustivo bit-wise de spikes de 8 bits y con diversos divisores de frecuencia

Siendo la pendiente de cada una de las rectas de modulación la ganancia de modulación del generador, acorde con la Ecuación [2-22], en la Figura 2.14 mostramos la ganancia de modulación para un modulador de 8 bits.

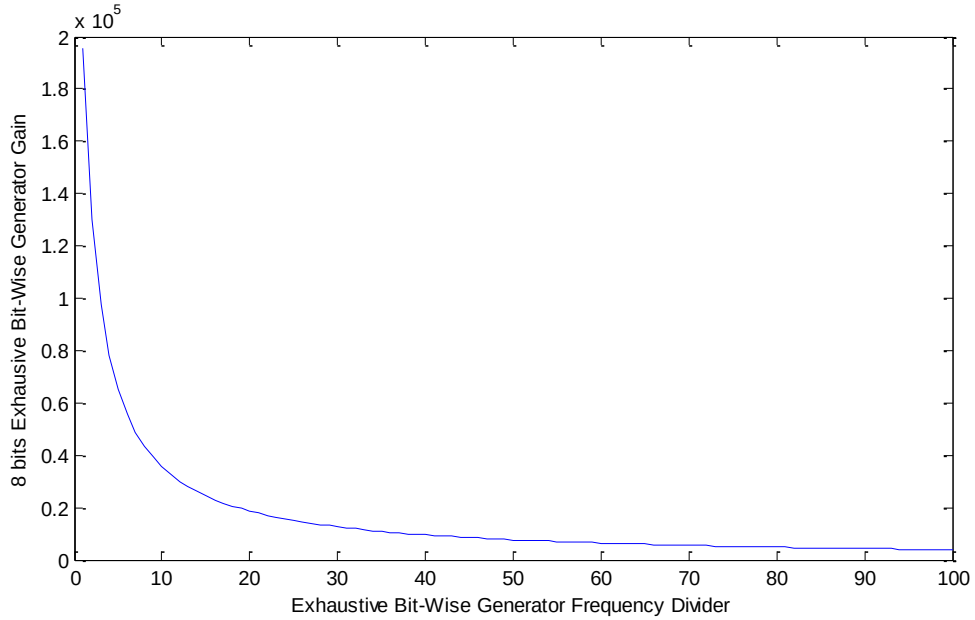


Figura 2.14: Ganancias de modulación de un generador exhaustivo bit-wise de spikes de 8 bits y con diversos divisores de frecuencia

Finalmente, hemos realizado un doble barrido, mostrado en la siguiente ilustración, en el que hemos simulado el generador para distintos números de bits, y distintos divisores de frecuencia, representando la ganancia del modulador como un gráfico en 3 dimensiones.

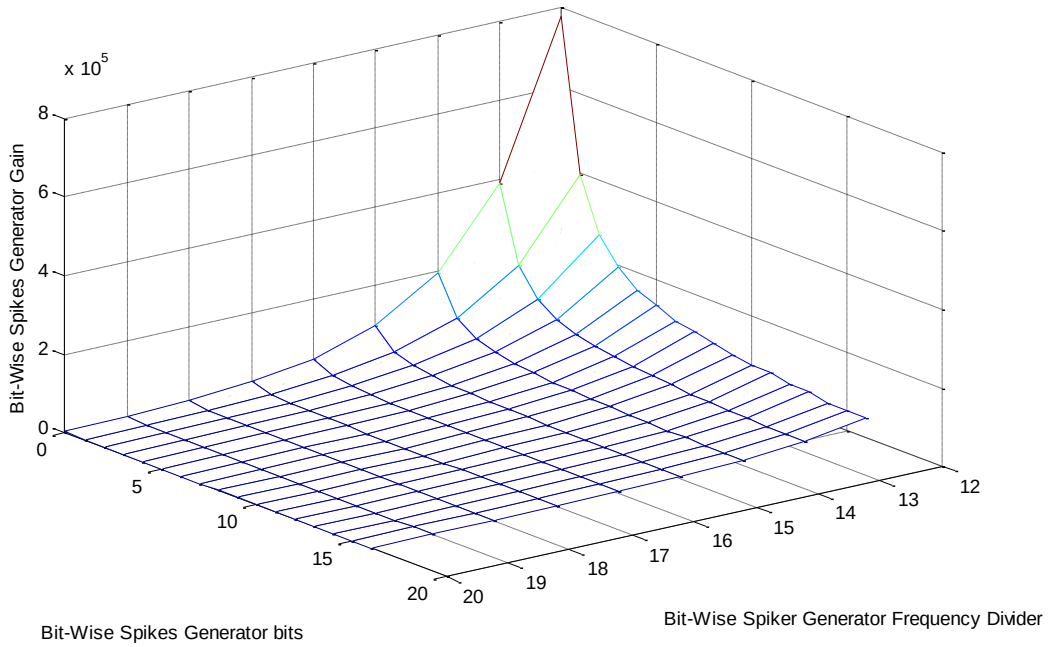


Figura 2.15: Ganancias de modulación del un generador exhaustivo bit-wise de spikes para diferentes números bits y con diversos divisores de frecuencia

A continuación vamos a calcular la frecuencia máxima que puede alcanzar el generador de spikes exhaustivo bit-wise. Dado que la máxima frecuencia de salida nos la encontraremos cuando el valor a la entrada del generador alcance el máximo valor:

$$f_{spikesMax} = \frac{F_{CLK}}{2^{n-1}(genFreqDiv + 1)} * 2^{n-1}$$

Simplificando la expresión anterior obtenemos:

$$f_{spikesMax} = \frac{F_{CLK}}{(genFreqDiv + 1)}$$

Ecuación [2-23]

De esta ecuación podemos deducir que la frecuencia máxima del generador no dependerá del número de bits del generador, sino del valor asignado al divisor de frecuencia.

2.3.2. Simulaciones basadas en PWM

Las primeras simulaciones que vamos a presentar son las basadas en PWM. En la Figura 2.16 se muestra la interconexión de los componentes VHDL que integran el sistema. A la izquierda encontramos los puertos de entrada desde Simulink, están formados por: la señal de reloj y reset, la constante propia de cada método (en este caso Freq_Div / período de PWM), y el estímulo de entrada. Los estímulos de entrada están dirigidos hacia un generador de spikes sintéticos exhaustivos bit-wise. El generador sintético de spikes proporcionará los spikes de entrada del sistema, con una frecuencia proporcional al valor del estímulo, tal y como hemos expuesto en el apartado anterior. Finalmente los spikes (tanto positivos como negativos), se

envían al modulador PWM basado en spikes, generando la señal PWM de salida. Siendo esta salida amplificada y enviada al motor de DC.

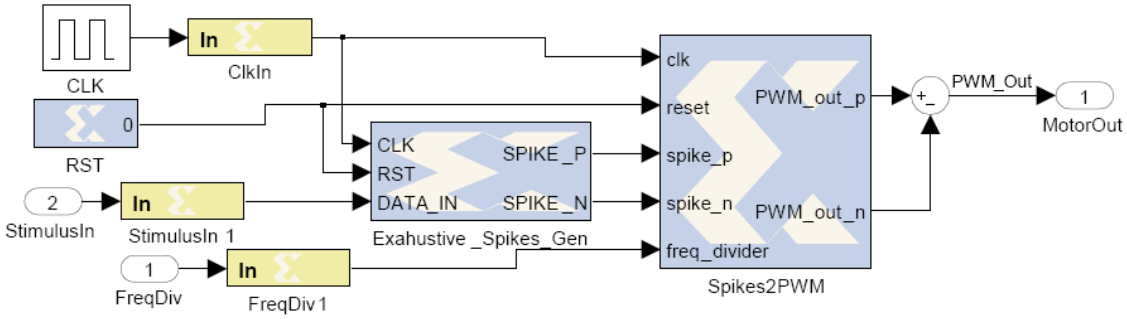


Figura 2.16: Esquemas de simulación del conversor PWM en lazo abierto

La Figura 2.17 es representa las revoluciones alcanzadas por el motor de DC para una frecuencia de spikes fija (305.2 kSpikes/Sec), y diversos períodos de PWM (Tpwm) (desde 41uSec hasta 161uSec). En ella vemos claramente cómo afecta el período de PWM a la respuesta del motor, además de la predicha respuesta sobre amortiguada. En cuanto al comportamiento estático, a mayor período de PWM mayor velocidad alcanzará el motor, ya que más tiempo estará el modulador PWM a '1' por cada spike recibido. Así la ganancia estática asociada a la frecuencia de los spikes puede ser controlada con el período de PWM. Centrándonos en el comportamiento dinámico del motor, se observa claramente como con un período de Tpwm suficientemente pequeño, los armónicos introducidos por la modulación PWM son filtrados por el motor, no produciéndose oscilaciones en la respuesta del motor. Sin embargo, para períodos de PMW muy grandes, los armónicos de la señal PWM se encuentran más cercanas a las bandas frecuenciales de paso del motor, introduciéndose un leve rizado. Finalmente, destacar la latencia introducida por el modulador de PWM, si nos fijamos atentamente al inicio de la actividad del motor, en los casos con un T_{PWM} elevado, se aprecia claramente como el motor tarda más tiempo en arrancar.

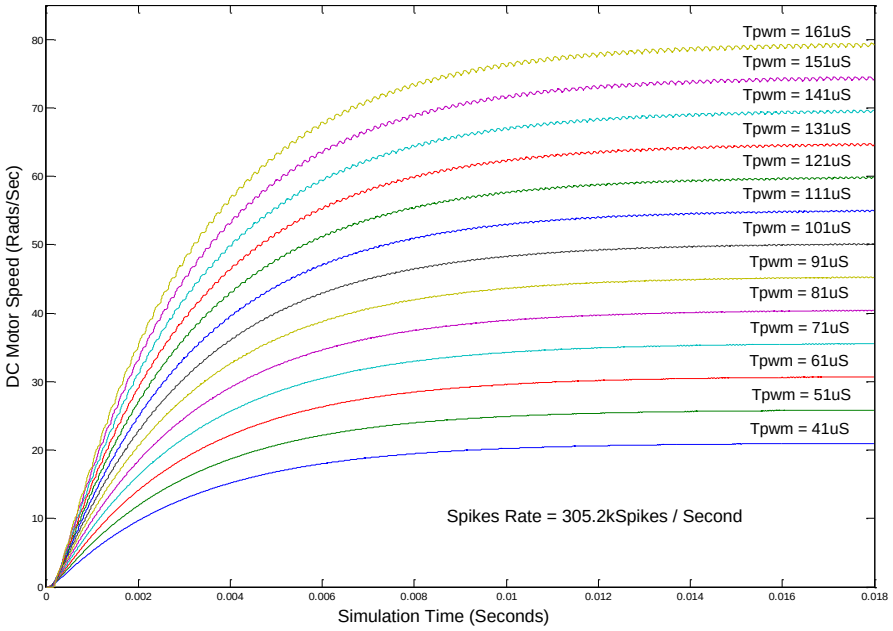


Figura 2.17: Respuestas del motor en lazo abierto, usando PWM, con periodo variable y frecuencia de spikes constante

Haciendo uso de los resultados de las simulaciones anteriores, vamos a compararlos con los resultados teóricos que cabría esperar para la modulación PWM. La Ecuación [2-24] muestra la velocidad estacionaria que debería alcanzar el motor de las simulaciones para la frecuencia fija de las simulaciones, y con diversos períodos de PWM:

$$\omega_{static} \approx \frac{K_{Motor} V_{DC} SpikeRate}{2^{Nº\,Bits}} T_{PWM} = \frac{33.2281 * 12 * 305.18 * 10^3}{2^8} T_{PWM} = 4.7533 * 10^5 T_{PWM}$$

Ecuación [2-24]

La Figura 2.18 la velocidad estacionaria alcanzada por el motor, extraída de las simulaciones anteriores, para los diferentes períodos de PWM (en azul), así como el resultado de sustituir los períodos de PWM de las simulaciones en la ecuación anterior (verde). En ella se aprecia claramente como la ganancia estática del modulador PWM es lineal respecto al período de PWM.

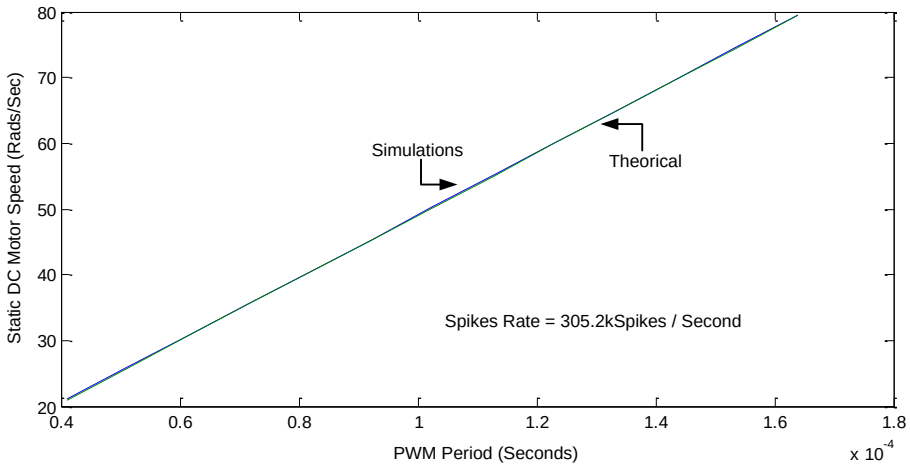


Figura 2.18: Ajuste teórico de la ganancia, usando PWM, con periodo variable y frecuencia de spikes constante

Gracias a los ToolBoxes proporcionados por MATLAB, podemos calcular los coeficientes de recta una regresión que nos aproxime los resultados de las simulaciones, obteniendo la siguiente recta de regresión:

$$y(x) = ax + b = 4.757 * 10^5 x + 1.4575$$

Donde x representa al período de PWM. Comparando la pendiente de esta recta con la pendiente teórica obtenemos:

$$\frac{Pendiente\ Regresión}{Pendiente\ Teórica} 100 = \frac{4.757 * 10^5}{4.7533 * 10^5} 100 = 99.91\%$$

Finalmente, para una tasa de spikes fija, y un período de PWM variable, obtenemos en la simulaciones un ajuste de más del 99.9% respecto a los resultados teóricos.

Para poder medir el rizado introducido por el modulador PWM, hemos calculado la transformada rápida de Fourier (Fast Fourier Transform, FFT) [Mitra93] para cada respuesta asociada a cada período de PWM. En la Figura 2.19 podemos ver los espectros de todas las señales, estando ambos ejes en escala logarítmica, siendo el eje X la frecuencia en hercios y el eje Y la velocidad del motor en radianes por segundo. Obsérvese como la velocidad estacionaria del motor está en la componente continua de la transformada, además observamos armónicos en la frecuencia de PWM usada en cada simulación, así como los sucesivos armónicos de alta frecuencia introducidos por la señal de PWM, separados entre ellos por la frecuencia del PWM. También puede apreciarse claramente el efecto de la ganancia estática asociada al periodo de PWM, estando todas las transformadas sobrepuestas.

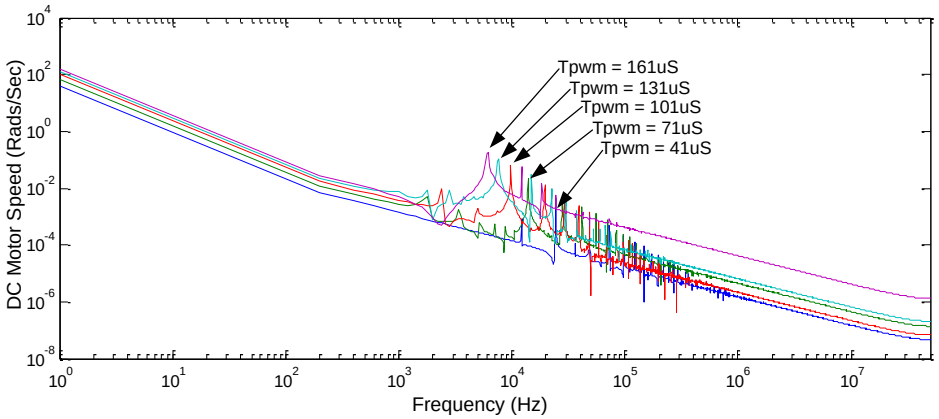


Figura 2.19: Transformada de Fourier de las respuestas del motor, usando PWM, con periodo variable y frecuencia de spikes constante

A continuación, en la Figura 2.20 mostramos la respuesta del motor ante un T_{PWM} fijo (102.4uS) y diversas frecuencias de spikes de entrada (desde 117.6kSpikes hasta 457.7kSpikes). En ellas se observa la ya predicha linealidad de la velocidad del motor ante los diversos dutty-cycles de la señal PWM. Sin embargo cabe destacar el rizado que muestran las respuestas de los motores ante una secuencia de spikes de frecuencia creciente. Este rizado es producido debido a que la señal de PWM está a nivel bajo durante un breve período de tiempo, introduciendo componentes espectrales con relativamente alta potencia, en la banda de paso del motor.

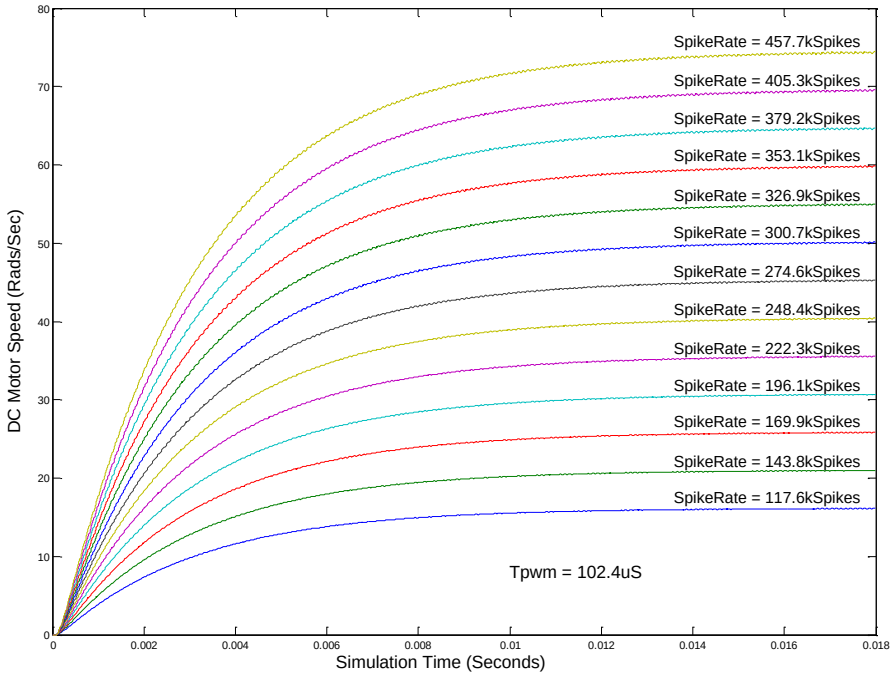


Figura 2.20: Respuestas del motor en lazo, usando PWM, con periodo fijo y frecuencia de spikes variable

Al igual que en el caso anterior, podemos comparar la respuesta de los motores en las simulaciones, con las ecuaciones teóricas planteadas con anterioridad. La Ecuación [2-25] muestra la velocidad estacionaria que debería alcanzar el motor de las simulaciones para un período de PWM fijo, en este caso 102.4uS, y frecuencia variable:

$$\omega_{static} \approx \frac{K_{Motor} V_{DC} T_{PWM}}{2^{Nº\,Bits}} SpikeRate = \frac{33.2281 * 12 * 102.4 * 10^{-6}}{2^8} SpikeRate = 1.595 * 10^{-4} SpikeRate$$

Ecuación [2-25]

En la Figura 2.21 mostramos la superposición de la velocidad estacionaria alcanzada por el motor, tanto de las simulaciones (azul), como de la predicción teórica (verde). Obsérvese como la velocidad alcanzada por el motor es lineal con la frecuencia de spikes aplicada a la entrada del modulador PWM.

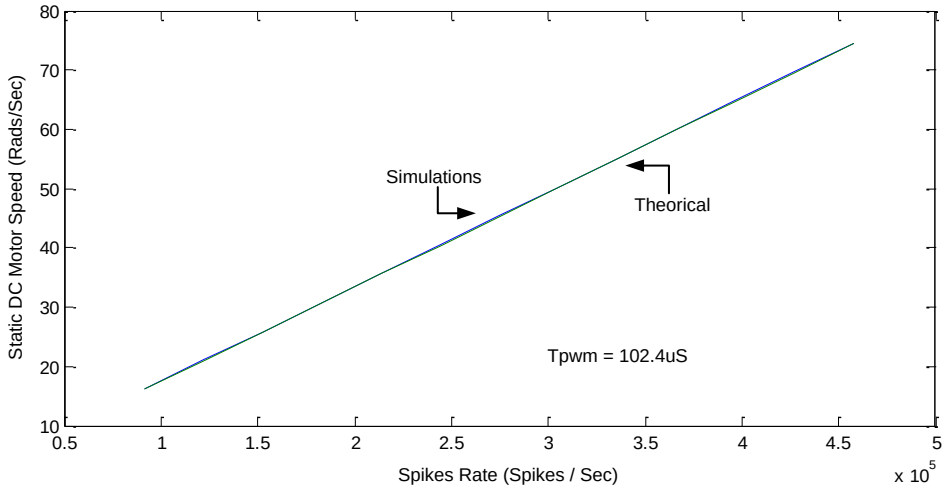


Figura 2.21: Ajuste teórico de la ganancia, usando PWM, con periodo fijo y frecuencia de spikes variable

Finalmente, vamos a calcular una regresión lineal de los resultados de las simulaciones, para así comparar los resultados de las simulaciones con el resultado teórico esperado:

$$y(x) = ax + b = 1.5925 * 10^{-4} x + 1.5579 * 10^{-5}$$

Donde x representa la frecuencia de spikes a la entrada del modulador PWM. Comparando la pendiente de esta recta con la pendiente teórica obtenemos:

$$\frac{Pendiente\ Regresión}{Pendiente\ Teórica} 100 = \frac{1.5925 * 10^{-4}}{1.595 * 10^{-4}} 100 = 99.8433\%$$

Finalmente, para un período de PWM fijo y una tasa de spikes variable, obtenemos en los resultados de las simulaciones un ajuste de más del 99.8% respecto a los resultados teóricos.

También hemos calculado las FFT para cada respuesta del motor, en la Figura 2.22. En ellas, al igual que en el caso anterior, vemos armónicos en la frecuencia del PWM y sus sucesivos armónicos. Pero a diferencia del caso anterior, todos los armónicos están a la misma

frecuencia (ya que el periodo de PWM es el mismo en todos los casos). Solo variando la amplitud de cada armónicos en base a la frecuencia de los spikes aplicados.

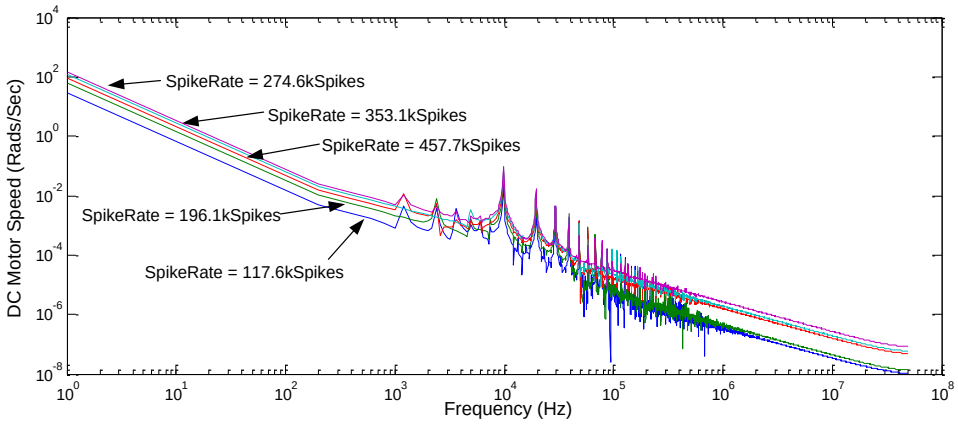


Figura 2.22: Transformada de Fourier de las respuestas del motor, usando PWM, con periodo fijo y frecuencia de spikes variable

2.3.3. Simulaciones basadas en PFM

A continuación vamos a mostrar las simulaciones basadas en PFM. En la Figura 2.23 se muestra la interconexión de los componentes VHDL que integran el sistema. Como puede apreciarse, son exactamente los mismo componentes que en el caso de la modulación PWM, pero variando el método de procesamiento de los spikes. En lugar del modulador PWM nos encontramos el expansor de spikes.

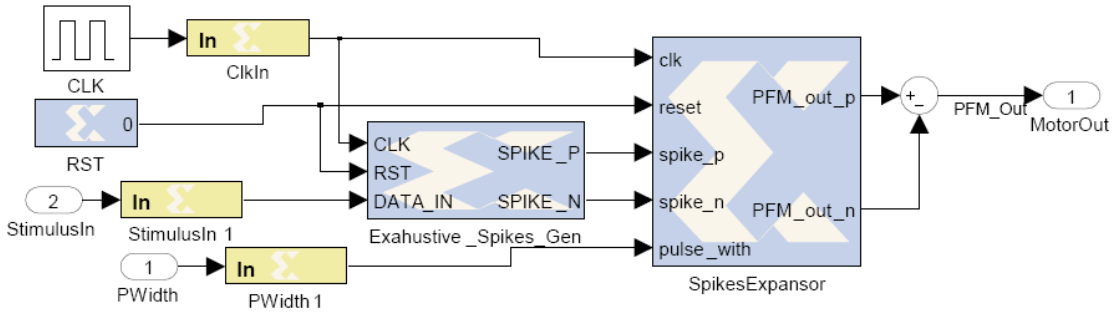


Figura 2.23: Esquemas de simulación del Spikes Expansor en lazo abierto

Al igual que en el apartado anterior vamos a comenzar por un conjunto de simulaciones, mostrada en la Figura 2.24, con una entrada de frecuencia constante (305.18kSpikes), y diversos anchos de spikes (desde 0.14uS hasta 0.6uS). Como se aprecia, a una frecuencia constante, podemos adecuar la ganancia del motor modificando el valor del ancho del pulso, tal y como indicaba la Ecuación [2-26]. A diferencia del PWM el rizado de la respuesta del motor no aumenta con el ancho de spike (o ganancia) usado.

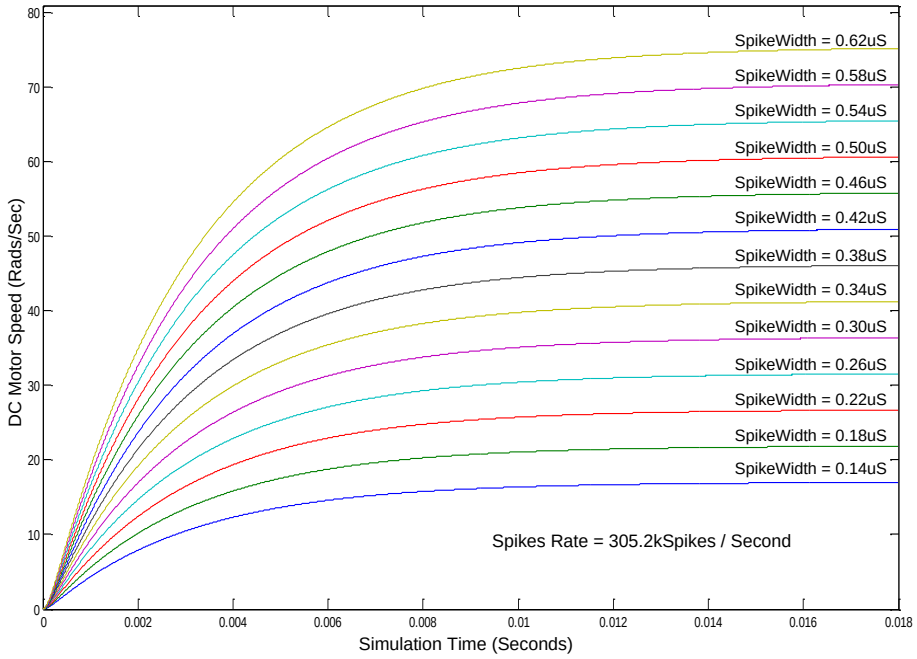


Figura 2.24: Respuestas del motor en lazo, usando PFM, con ancho de spike variable y frecuencia de spikes constante

Análogamente al estudio del modulador PWM, para la modulación PFM podemos comparar la respuesta de los motores en las simulaciones, con las ecuaciones teóricas planteadas con anterioridad. La Ecuación [2-26] muestra la velocidad estacionaria que debería alcanzar el motor de las simulaciones para una frecuencia de spikes fija, 305.18KSpikes/Sec en este caso, y frecuencia variable.

$$\omega_{static} \approx K_{Motor} V_{PS} * SpikeRate * SpikeWidth = 33.2281 * 12 * 305.18 * 10^3 SpikeWidth = 121.69 * 10^8 SpikeWidth$$

Ecuación [2-26]

En la Figura 2.25 mostramos la superposición de la velocidad estacionaria alcanzada por el motor, tanto de las simulaciones (azul), como de la predicción teórica (verde). Obsérvese como la velocidad alcanzada por el motor para una frecuencia fija de spikes es lineal respecto al ancho de spikes utilizado:

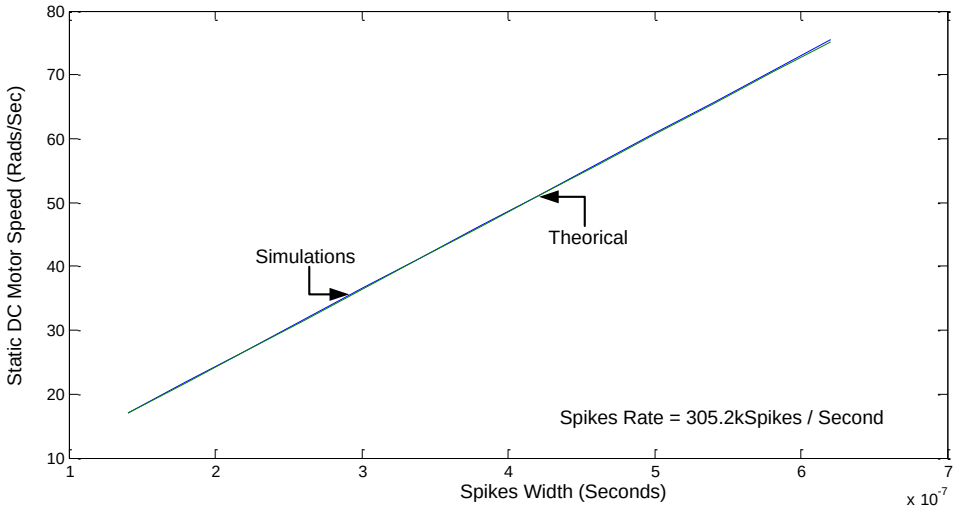


Figura 2.25: Ajuste teórico de la ganancia, usando PFM, con ancho de spike variable y frecuencia de spikes constante

Análogamente al caso anterior, vamos a calcular una regresión lineal de los resultados de las simulaciones. Para así comparar los resultados de las simulaciones con el resultado teórico esperado. Obteniendo la siguiente recta de regresión para una frecuencia de spikes constante de entrada de 305.2kSpikes/Sec:

$$y(x) = ax + b = 1.2136 * 10^8 x + 1.3925 * 10^{-6}$$

Donde x representa la anchura temporal de los spikes de salida. Comparando la pendiente de esta recta con la pendiente teórica obtenemos:

$$\frac{Pendiente\ Regresión}{Pendiente\ Teórica} 100 = \frac{1.2136 * 10^8}{1.2169 * 10^8} 100 = 99.73\%$$

Para una frecuencia de spikes constante a la entrada, y un ancho de spike variable, obtenemos en los resultados de las simulaciones un ajuste de más del 99.7% respecto a los resultados teóricos. Comprobando que podemos variar la ganancia estática asociada a cada spike de manera lineal, simplemente variando la anchura temporal de los spikes.

Para comprobar el efecto de aplicar los spikes directamente al sistema, hemos calculado las FFT de las simulaciones, podemos verlas en la Figura 2.26. En ellas vemos unos armónicos coincidentes con la frecuencia de los spikes aplicados al motor, ya que particularmente en este caso, la frecuencia es constante en todas las simulaciones. A diferencia del caso del PWM, la frecuencia de los armónicos es mucho más alta, siendo estas componentes de alta frecuencia más filtradas por el motor que en el caso del PWM, ya que con el uso del PWM los períodos de PWM eran relativamente grandes comparados con los rangos de frecuencia de los spikes. Estando los armónicos del PWM más cerca de la banda de paso del motor, que los armónicos de los spikes.

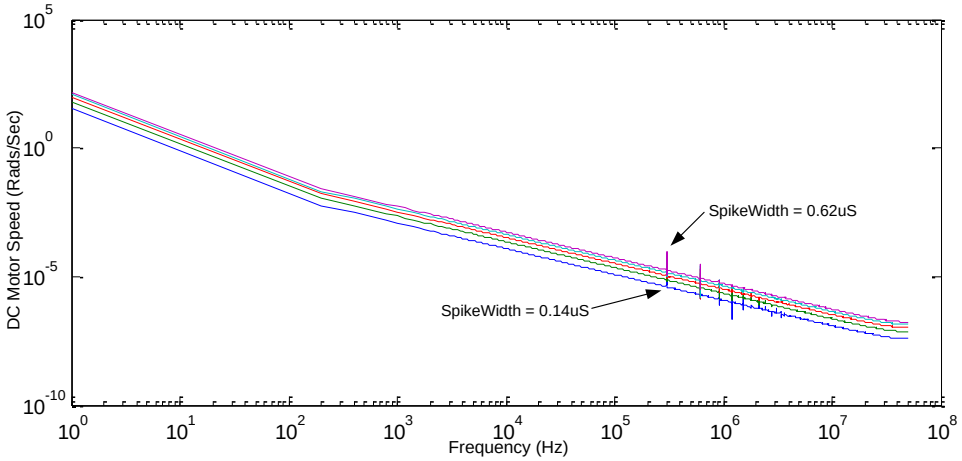


Figura 2.26: Transformada de Fourier de las respuestas del motor, usando PFM, con ancho de spike variable y frecuencia de spikes constante

Finalmente, si aplicáramos este método de actuación a un motor real, nos encontraríamos con tiempo en alto fijo para cada spike, así como una frecuencia de spikes variable a la entrada del expansor de spikes. Variando la frecuencia en base a las decisiones de otras etapas de control en un nivel superior. Para comprobar la respuesta del motor ante un ancho de spike constante (0.32uS), y una tasa de spikes variable (desde 91.5kSpikes, hasta 457kSpikes), realizamos otra secuencia de simulaciones, mostradas en la Figura 2.27.

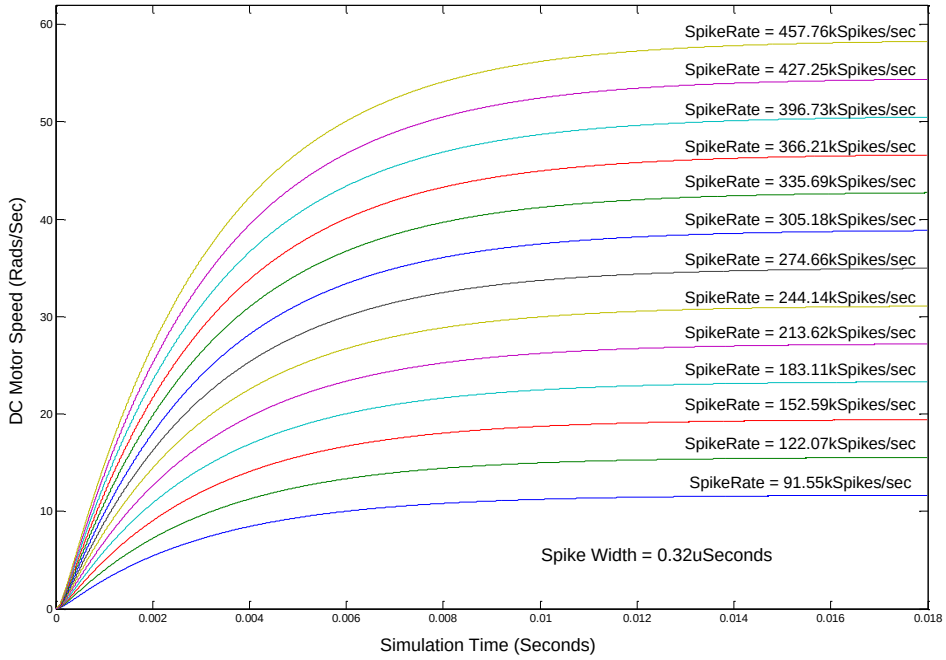


Figura 2.27: Respuestas del motor en lazo, usando PFM, con ancho de spike fijo y frecuencia de spikes variable

La Ecuación [2-27] muestra la velocidad estacionaria que debería alcanzar el motor de las simulaciones para un ancho de spikes fija, 0.32uSec en este caso, y frecuencia de spikes aplicados al motor variable:

$$\omega_{static} \approx K_{Motor} V_{PS} SpikeWidth * SpikeRate = 33.2281 * 12 * 0.32 * 10^{-6} SpikeRate$$
$$= 1.276 * 10^{-4} SpikeRate$$

Ecuación [2-27]

En la Figura 2.28 mostramos la superposición de la velocidad estacionaria alcanzada por el motor, tanto de las simulaciones (azul), como de la predicción teórica (verde). Obsérvese como la velocidad alcanzada por el motor para un ancho fijo de spikes es lineal respecto a la frecuencia de los spikes de entrada.

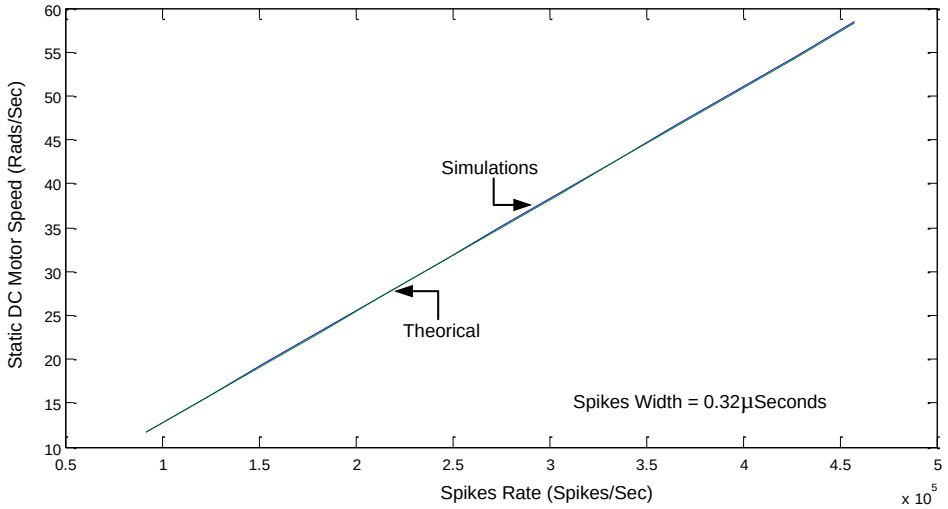


Figura 2.28: Ajuste teórico de la ganancia, usando PFM, con ancho de spike fijo y frecuencia de spikes variable

Finalmente, vamos a calcular una regresión lineal de los resultados de las simulaciones. Para así comparar los resultados de las simulaciones con el resultado teórico esperado. Obteniendo la siguiente recta de regresión para un ancho de spikes constante, 0.32uSeg:

$$y(x) = ax + b = 1.2725 * 10^{-4} x + 3.0665 * 10^{-5}$$

Donde x representa la frecuencia de los spikes de salida. Comparando la pendiente de esta recta con la pendiente teórica obtenemos:

$$\frac{Pendiente\ Regresión}{Pendiente\ Teórica} 100 = \frac{1.2725 * 10^{-4}}{1.276 * 10^{-4}} 100 = 99.7286\%$$

Para una frecuencia de spikes variable a la entrada, y un ancho de spike constante, obtenemos en los resultados de las simulaciones un ajuste de más del 99.7% respecto a los resultados teóricos. Comprobando que podemos variar la velocidad del motor en lazo abierto linealmente con la frecuencia de spikes aplicados, así como variar la ganancia asociada a cada spike linealmente con el ancho de los spikes [Jimenez08a].

En este caso, donde la frecuencia de los spikes es variable, si que resulta más interesante el cálculo de la FFT para cada caso, en la Figura 2.29. Ya que se observan armónicos en las frecuencias de los spikes aplicados al motor, además se observa como a mayor frecuencia más atenuados son por el motor. Introduciendo un rizado en la respuesta del

motor de frecuencia variable (coincidente con la de los spikes de entrada), pero mucho menos severo que en el caso del PWM.

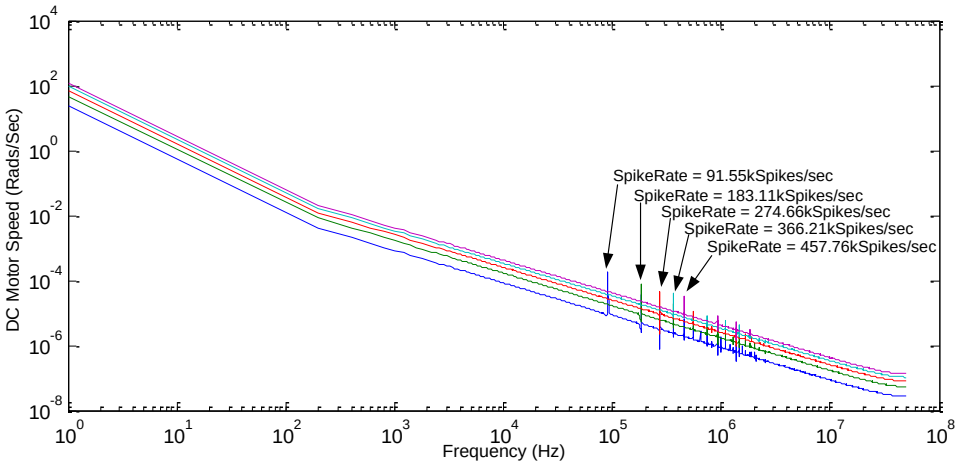


Figura 2.29: Transformada de Fourier de las respuestas del motor, usando PFM, con ancho de spike fijo y frecuencia de spikes variable

A continuación hemos realizado un barrido de ambos parámetros, mostrando en la Figura 2.30 el resultado. Esta figura muestra la característica estática de la velocidad del modelo del motor de las simulaciones. Es decir, la velocidad estacionaria (eje vertical), respecto a la frecuencia de los spikes aplicados (eje horizontal), para diversos anchos de spikes (diferentes líneas). Siendo la ganancia del sistema la pendiente de cada línea. Se aprecia claramente la linealidad de la ganancia del sistema con el ancho de spike. Así como la linealidad de la velocidad del motor con la frecuencia de los spikes, para un ancho de spike, ganancia, fija. En concreto el barrido se ha realizado desde 152.6kSpikes/Sec hasta 1.22MSpikes/Sec, con unos anchos de pulsos desde 0.42 hasta 1.62uSec, con un incremento de 0.1us.

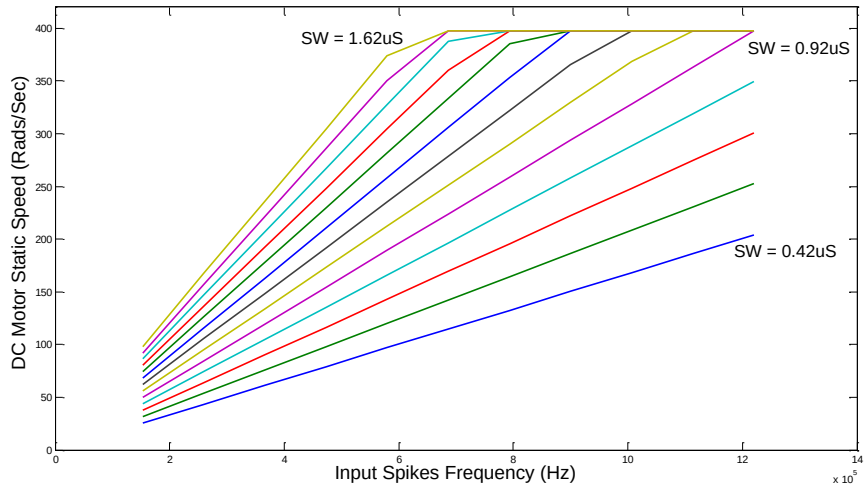


Figura 2.30: Velocidad estacionaria del motor para un barrido del ancho de spikes y la frecuencia de los spikes de entrada.

En la figura anterior observamos como la velocidad del motor alcanza su máximo (se satura) a partir de una determinada frecuencia, para cada ancho de spike. Esto se debe a que la salida del Spikes Expansor se satura a partir de una determinada frecuencia, como expusimos con anterioridad, aplicando el máximo voltaje (12V en este caso) a los bornes del motor.

Finalmente hemos representado la Figura 2.30, como un gráfico en 3 dimensiones, obteniendo la Figura 2.31. Siendo los ejes horizontales la frecuencia de los spikes de entrada y el ancho de spikes, y el eje vertical al velocidad estacionaria del motor. En ella se ve claramente el plano lineal de la velocidad estacionaria del motor, así como la meseta de la saturación.

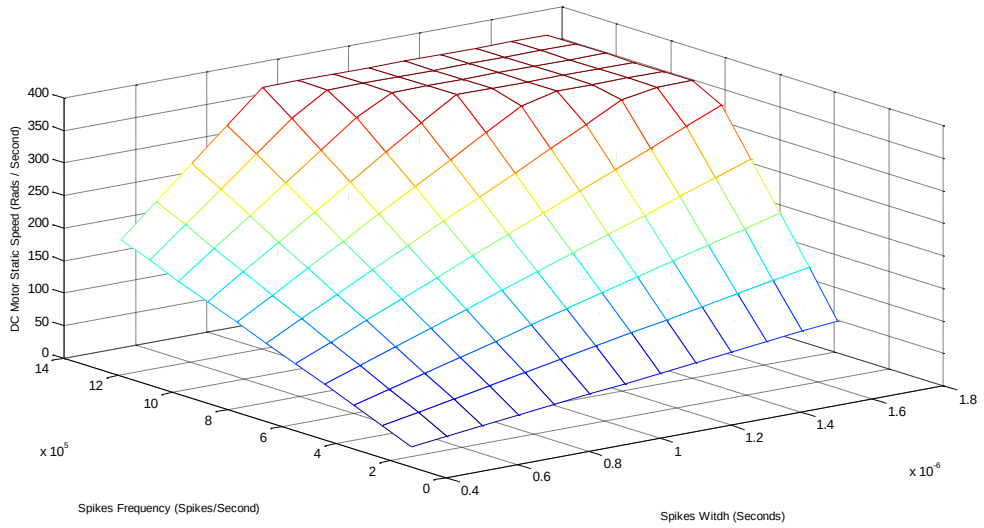


Figura 2.31: Representación en 3D de la velocidad del motor para diversos anchos y frecuencias de spikes

Finalmente podemos definir el comportamiento de nuestro sistema como una ecuación no lineal de dos parámetros: la frecuencia de los spikes (SR), y el ancho de los mismos (SW). Con un nivel de saturación inversamente proporcional al ancho de spike, como mostramos en la Ecuación [2-28]:

$$\omega(SR, SW) = \begin{cases} K_{Motor} V_{PS} * SW * SR, & \boldsymbol{S}R < \dfrac{1}{SW} \\ K_{Motor} V_{PS}, & SR \geq \dfrac{1}{SW} \end{cases}$$

Ecuación [2-28]

2.3.4. Comparativa: PWM vs PFM:

El uso de la modulación PWM introduce un integrador, el cual integra a lo largo del período de la señal PWM. Es decir, se introduce un retraso desde que llega la información en spikes y se transmite al motor la señal PWM, equivalente a Tpwm. Sin embargo, introducir un integrador transforma una secuencia de spikes a un número discreto (valor integrado), y con un período de muestreo conocido (período de la señal PWM). Entre el valor integrado y el modulador PWM sería posible introducir una función de transferencia en el domino Z, e implementar en su interior reguladores en Z, como por ejemplo un PID discreto.

Sin embargo, usando la modulación PFM los spikes son aplicados al motor casi sin retraso alguno, eliminando la latencia introducida por el método PWM. Pero en su contra, al no tener período de muestreo (ni fijo, ni siquiera conocido), la implementación de controles basados en PFM no están extendidos, y pasan por el uso de otro tipo de técnicas, como pueden ser las probabilísticas [Paz08].

La modulación PFM necesita una distribución homogénea de los spikes en el tiempo. Ya que en caso de los spikes se recibiesen con poca homogeneidad temporal (varios spikes en un espacio temporal breve, y un elevada cantidad de tiempo sin recibir ninguno), se introducirían oscilaciones en la respuesta del motor, incluso llegando a saturar por instantes el *Spikes Expansor*. Sin embargo, con la modulación PWM no existe una necesidad tan manifiesta de la distribución temporal de los spikes, ya que los spikes son recolectados durante Tpwm, no importa su distanciamiento temporal, sino la tasa de spikes recibidos a lo largo de Tpwm. Por este motivo, se debería observar inmunidad al usar conjuntamente los moduladores PWM con otros mecanismos de generación de spikes [Linares07b], los cuales no generan secuencias de spikes tan homogéneas como los usados en nuestras simulaciones. El generador de PWM funciona como un filtro con respecto al flujo de spikes.

De las simulaciones concluimos, que la modulación PWM necesita períodos de PWM relativamente pequeños para no introducir rizados en la respuesta del motor. Además, el rizado no sólo depende del período de PWM, sino también del dutty-cycle de la señal. En su contra, la modulación PFM se ha mostrado más flexible en todos estos sentidos, ya que en su descomposición en series de Fourier, presenta sus armónicos a mucha más alta frecuencia que la modulación PWM, siendo éstos filtrados por el motor de DC. Eliminando así parte del rizado mostrado por la modulación PWM. En otras palabras, el modulador PWM disminuye las frecuencias de la señal pulsante, realizando un sub-muestreo o downsampling, adaptándolas al período del PWM. Este hecho tiene una implicación muy clara, y consiste en que necesitamos tasas de spikes dependientes del período de PWM para poder integrar los spikes durante ese tiempo. Sin embargo la aplicación directa de los spikes al motor no requiere tasas tan altas, y además las altas frecuencias de los spikes mas atenuadas por el motor que los armónicos de las señales de PWM. Las simulaciones usando ambas modulaciones se han mostrado muy aproximadas a su modelo teórico, aunque ya conocido para la modulación PWM [Jimenez08a], demuestra que la modulación PFM no tiene que envidiarle en este aspecto.

Con el fin de cuantificar el consumo de recursos físicos hardware, hemos procedido a sintetizar ambos componentes VHDL. La FPGA para la que se ha realizado la síntesis pertenece a la familia Spartan3 de Xilinx, hemos seleccionado precisamente esta FPGA por que será usada de nuevo más adelante en la plataforma AER-Robot. En los reportes de la síntesis se nos informa sobre el consumo de recursos de la FPGA, así como de la velocidad máxima que pueden alcanzar nuestros circuitos. Los circuitos lógicos contenidos en la FPGA se dividen en slices (donde cada slice es un pequeño bloque de lógica), nuestra Spartan 3 contiene un total de 3584 slices. El conversor de spikes a PWM ocupa un total de 53 slices, y puede alcanzar una velocidad de reloj máxima de 140.6MHz. El Spikes Expansor consume 24 slices, y puede operar hasta 157.33MHz. Claramente el Spikes Expansor ocupa muchos menos recursos de la FPGA, gracias a su simpleza, en su contra, el conversor PWM incorpora elementos más complejos (un integrador, y modulador PWM en sí), ocupando casi el doble de hardware. Consecuentemente, el conversor PWM es ligeramente más lento, ya que las señales han de atravesar más circuitos lógicos combinacionales que en caso del Spikes Expansor, el cual, de nuevo muestra mejores gracias a su simpleza.

Tabla 2-4: Comparativa entre los moduladores PWM y PFM

Método de actuación	Consumo Slices	Máxima Frecuencia	Latencia
PWM	53	140.6MHz	Equivalente al periodo de PWM
PFM	24	157.33MHz	Ninguna

Normalmente el periodo para el caso del modulador PWM se fija de forma que la frecuencia fundamental, o más pequeña, esté lo suficiente alta para no producir efectos sobre el sistema, debido a que éste la filtra. Si se fija la anchura de pulso en PFM: Th_{PFM}, y el periodo/frecuencia de PWM para el caso del PWM: $Freq_{PWM}$, entonces consideramos $Freq_{PFM}$ y Th_{PWM} variables, se tiene:

$$T_{hPFM} * Freq_{PFM} = T_{hPWM} * Freq_{PWM} \rightarrow Freq_{PFM} = \frac{T_{hPWM} * Freq_{PWM}}{T_{hPFM}}$$

Ecuación [2-29]

Por ejemplo, si: $Freq_{PWM}$=20kHz y Th_{PFM}=0,001mSec. El factor $Freq_{PWM}/Th_{PFM}$ =20000. Si Th_{PWM} =0,01mSec, la frecuencia es de 200Khz para que la tensión teórica del PFM sea igual que la del PWM. Para Th_{PFM} =0,001mSec sería 20Khz.

En contra posición, si : $Freq_{PWM}$=20kHz , Th_{PFM}=0,1mSec. El factor $Freq_{PWM}/Th_{PFM}$= 200. Si Th_{PWM}=0,01mSeC, la frecuencia es de 2Khz para que la tensión teórica del PFM sea igual que la del PWM. Para Th_{PWM} =0,001mSec sería 200Hz.

En el primer ejemplo la frecuencia que se debe utilizar en PFM es mucho mayor que la de PWM, debido a que el ancho de pulso elegido para PFM es mucho más pequeño que el periodo de PWM. En el segundo ejemplo la frecuencia de PFM a usar es mucho más pequeña que la de PWM, debido a que la anchura de pulso de PFM es más grande que el periodo de

PWM. La anchura de pulso en PFM es importante y tiene el mismo carácter que el periodo para el caso del PWM. Si se elige una anchura de pulso suficientemente pequeña no tiene por que producirse efectos indeseables en el motor, como vibraciones o sonidos, debido a que éste no filtre las componentes de frecuencia indeseadas como se expone en [Ertan04].

Finalmente, desde un punto de vista práctico, el uso de la modulación PFM elimina las pérdidas asociadas a la conmutación innecesaria de la fuente de alimentación, causada por los valores bajos en una señal de PWM. Si el valor de entrada del modulador PWM es muy pequeño, siempre estará conmutando a una frecuencia constante. Mientras que la modulación PFM para valores bajos sólo conmuta la fuente cuando es necesario.

3. Controles en lazo cerrado basados en sistemas pulsantes

A lo largo de este capítulo vamos a exponer el diseño, implementación y simulación de mecanismos para realizar controles en lazo cerrado basados en neuronas pulsantes. En los sistemas de control tradicionales suelen implementarse controles en lazo cerrado, con el objetivo de mejorar la respuesta transitoria del sistema, así como hacerlo más inmune a las perturbaciones externas y efectos del desgaste [Houpis89]. La idea de los sistemas en lazo cerrado consiste en medir la variable a controlar (realimentación) y substraerla con el valor que deseamos alcance (referencia), obteniendo así el error que comete el sistema en cada instante. Finalmente al sistema se le aplicará una función de transferencia del error. Pudiendo dicha función de transferencia variar desde una simple proporción, controles P, hasta funciones dinámicas más complejas, controles PID, predictivos, adaptativos, etc. En la Figura 3.1 mostramos el diagrama de bloques global de un sistema realimentado negativamente, así como en la función de transferencia del sistema equivalente tras realimentarlo unitariamente en el dominio S.

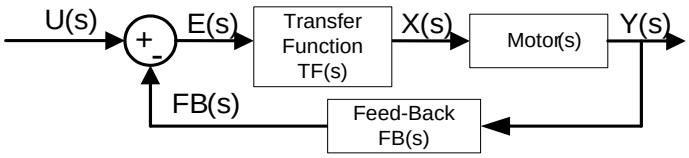


Figura 3.1: Diagrama de bloques de un control en lazo cerrado

$$G(s)_{Lazo\ Cerrado} = \frac{U(s)}{Y(s)} = \frac{TF(s) * M(s)}{1 + FB(s) * TF(s) * M(s)}$$

Ecuación [3-1]

Centrándonos en los sistemas pulsantes, donde la información de la referencia estaría codificada en la tasa de spikes, al igual que la información de realimentación. El error sería una nueva secuencia de spikes, cuya frecuencia vendría proporcionada por la diferencia entre las frecuencias de las señales de entrada (referencia y realimentación). De nuevo, la información del error estará contenida en su tasa de spikes, tal y como mostramos en la siguiente ecuación:

$$E_{SpikeRate} = U_{SpikeRate} - Y_{SpikeRate}$$

Ecuación [3-2]

La idea sería proporcionar al sistema la velocidad de referencia codificada como spikes, y restarla de la velocidad real del motor, también codificada en spikes, obteniendo así el error de la velocidad del motor como spikes. Sólo quedando aplicar los spikes del error al motor, mediante un método de actuación u otro (PWM / PFM). Dado que mediante el uso de ambos métodos de actuación podemos controlar la ganancia estática en lazo abierto del sistema, podríamos implementar controles proporcionales (P). Ajustando la ganancia del sistema mediante de la selección adecuada del período de PWM o tiempo en alto del Spikes Expansor según corresponda en cada caso.

Para realizar esta diferencia, y obtener así el error en tiempo real, proponemos dos mecanismos, aunque sólo el segundo de ellos ha sido implementado:

- Inter-Spike-Interval Difference & Generate, basado en la medida del tiempo entre spikes (Inter-Spike-Interval, ISI), y posterior generación sintética
- Hold & Fire, cancelando spikes consecutivos con signos opuestos.

3.1. Modelo Inter-Spike-Interval Difference & Generate

Este modelo se basa en medir el período de los spikes de entrada, procesándolos para obtener el período de los nuevos spikes de salida como un número discreto, y finalmente aplicando dicho número a un generador de spikes sintético exhaustivo bit-wise (descrito en detalle en capítulos anteriores). Este modelo no ha sido implementado por el alto coste hardware que a priori presenta, ya que como veremos a continuación hacen falta implementar divisores y multiplicaciones de un elevado número de bits, sin embargo, vamos a hacer un análisis preliminar, ya que ha sido de interés por algunos investigadores sobre el procesamiento de información basada en el ISI de los spikes [Liu04].

Como hemos expuesto en la introducción de este capítulo, necesitamos diseñar un elemento cuya frecuencia de salida, la cual representará el error del sistema, sea la diferencia de las frecuencias de entrada:

$$f_{Error} = f_U - f_Y = \frac{1}{T_U} - \frac{1}{T_Y} = \frac{T_Y - T_U}{T_U * T_Y}$$

Ecuación [3-3]

Esta frecuencia, representada por un número entero, ha de ser aplicada al generador sintético de spikes con el fin de generar los spikes del error.

$$f_{Gen} = K_{Gen} * f_{Error}$$

Ecuación [3-4]

Y en el caso particular de que la ganancial del modulador sea 1:

$$f_{Gen} = f_{Error}$$

Ecuación [3-5]

En la Figura 3.4 mostramos una propuesta de la arquitectura interna que podría tener una posible implementación este modelo, a la izquierda las entradas de los spikes (U para la referencia e Y para la realimentación), en el centro un bloque encargado de medir el período de ambas señales así como calcular la frecuencia del error, y a la derecha finalmente el generador sintético de spikes. Para medir el período de ambas señales lo que vamos a hacer es utilizar un temporizador para cada señal, el cual se incrementará en 1 cada ciclo de reloj, de tal manera que cuando un spike se reciba, el valor del temporizador se escribirá en un registro y será puesto a 0. Este registro almacenará el último ISI, así como su signo, siendo la entrada de un circuito aritmético que calculará la frecuencia de salida.

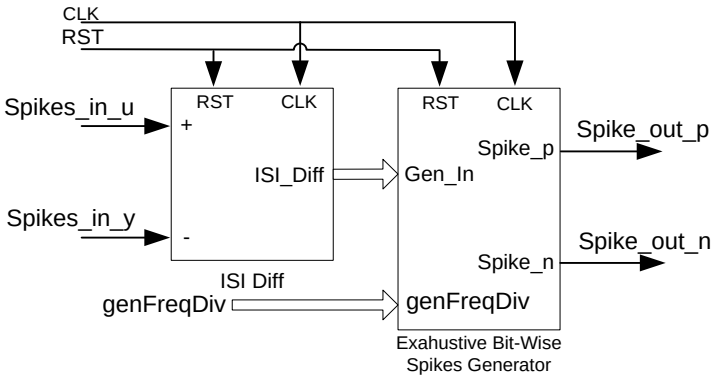


Figura 3.2: Posible arquitectura del ISI-Difference & Generate

Dado que los contadores se incrementan en 1 cada ciclo de reloj, existe una relación entre el número de bits de los contadores y la frecuencia mínima (períodos máximos) de las señales a la entrada. Dado que:

$$ISI = T_{CLK} * NCiclos$$

Ecuación [3-6]

Entonces:

$$ISI_{Max} = T_{CLK} * (2^n - 1)$$

Ecuación [3-7]

Donde Tclk es el período de reloj de la FPGA y n el número de bits del contador. Siendo la mínima frecuencia reconocible:

$$f_{Min} = \frac{1}{T_{CLK} * (2^n - 1)}$$

Ecuación [3-8]

En la siguiente figura mostramos la frecuencia mínima reconocible de las entradas frente al número de bits de los temporizadores, para una frecuencia de reloj de 50MHz. En la figura se ve claramente que para poder reconocer frecuencias menores a 1Hz se necesitan temporizadores de 26 bits.

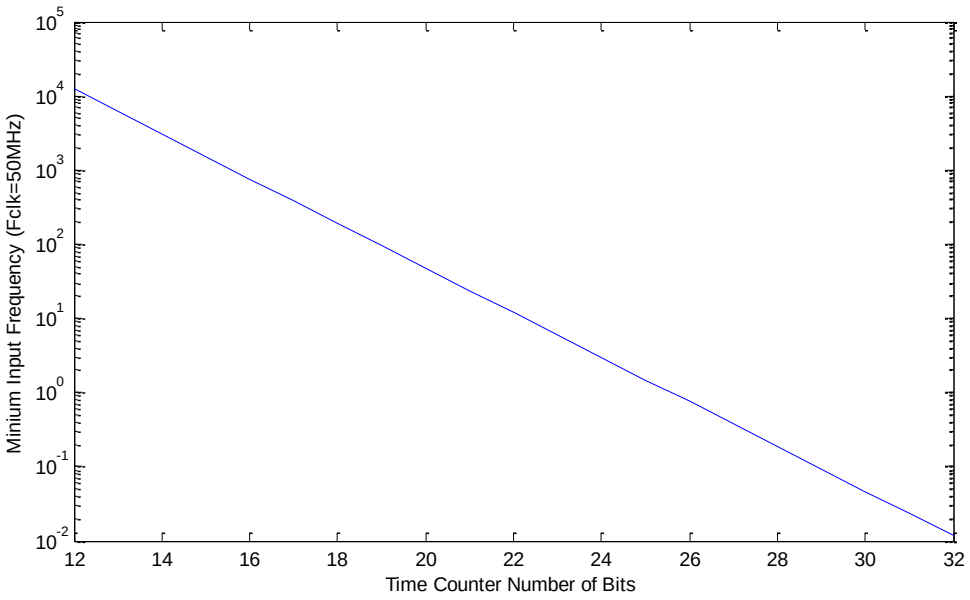


Figura 3.3: Diferencia de frecuencia mínima reconocible en base al número de bits

La necesidad de unos contadores tan grandes nos obliga a realizar operaciones con signo de 26 bits, lo cual supone un coste hardware elevadísimo en una FPGA, por ejemplo la Spartan3 que utilizaremos en capítulos posteriores tiene 6 multiplicadores de 18 bits, necesitando 2 multiplicadores para implementar un multiplicación de 27 bits con signo. Sin tener en cuenta la división, que deberíamos implementar algorítmicamente, consumiendo varios ciclos de reloj y otros dos multiplicadores. Por estos motivos, hemos decidido abandonar este modelo, en busca de otro que sea mucho más ligero desde el punto de vista del consumo de hardware, aunque en principio no proporcione respuestas tan precisas.

3.2. Modelo Hold & Fire.

Manteniendo la filosofía de los modelos neuronales pulsantes, intentando solventar todos los problemas expuestos anteriormente que presentaba el modelo anterior, surge el modelo Hold & Fire [Jimenez09a]. Este modelo se basa en la retención (Hold) de los spikes, esperando a la evolución de las entradas, disparando (Fire) o anulando spikes, en base a la naturaleza de los spikes que se reciban. Es decir, cada vez que llega un nuevo spike este es retenido internamente, esperando a la evolución de las entradas durante un tiempo fijo, tiempo de hold. Si no llega ningún nuevo spike después del tiempo de hold, el spike retenido es disparado. Sin embargo, si llegan nuevos spikes, el spike retenido puede ser disparado, reteniendo un nuevo spike, o cancelado, sin producir ninguna salida y sin retener ningún spike. En la Tabla 3-1 se muestra en detalle el comportamiento del Hold & Fire para las posibles entradas en base a los diferentes spikes retenidos, y en la Figura 3.4 un ejemplo de su comportamiento temporal ante la llegada de un spike positivo por la referencia (U+) para los diversos tipos de spikes de entrada. En definitiva, pensando en una realimentación negativa, la idea es que según vayan llegando los spikes se vayan cancelando mutuamente, produciendo

así la substracción de las tasas de spikes de ambos puertos de entrada, y ofreciendo una salida con la tasa de spikes adecuada.

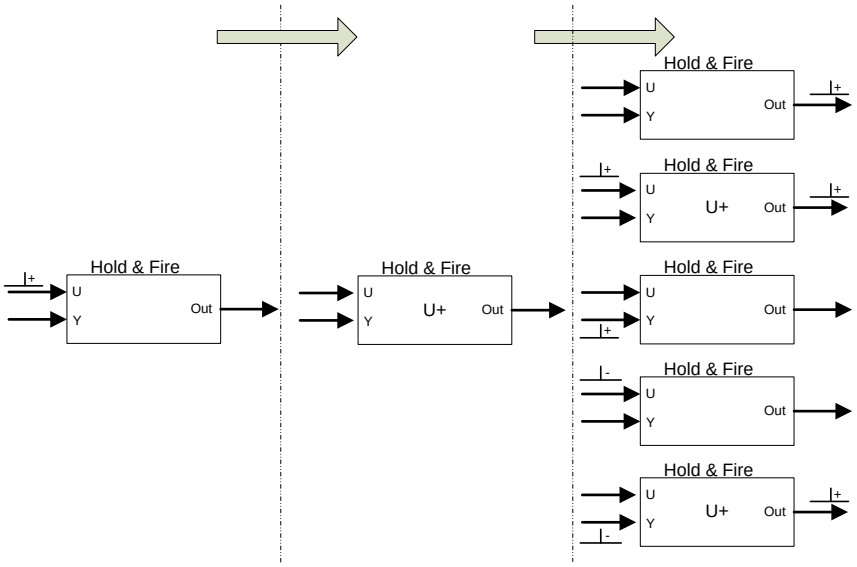


Figura 3.4: Posibles evoluciones del Hold & Fire tras recibir un spike positivo por el puerto U

Tabla 3-1: Estados del Hold & Fire

Retenido	Recibido	Salida	Prox. a ser retenido	Retenido	Recibido	Salida	Prox. a ser retenido
--	U+	---	U+	Y+	---	-	---
	U-	---	U-		U+	---	---
	Y+	---	Y+		U-	-	U-
	Y-	---	Y-		Y+	-	Y+
U+	---	+	----		Y-	---	---
	U+	+	U+	Y-	---	+	---
	U-	---	---		U+	+	U+
	Y+	---	---		U-	---	---
	Y-	+	Y-		Y+	---	---
U-	---	-	---		Y-	+	Y-
	U+	---	---				
	U-	-	U-				
	Y+	-	Y+				
	Y-	---	---				

En cuanto a la implementación concreta de este modelo, hemos diseñado un componente VHDL que responde a esta funcionalidad, cuyo diagrama de bloques se muestra en la Figura 3.5. Como se observa en la figura, está compuesto por cuatro contadores descendentes (uno por cada tipo de spike posible, ±U, ±Y) en los que se retendrán los spikes, y un circuito lógico encargado de gestionar los contadores, así como generar las salidas de la

entidad. Cada contador tiene dos entradas: *set*, carga el contador con el tiempo de hold, y *reset*, pone el contador a cero. Así como dos salidas: *hold*, cuyo valor en alto indica que tiene un spike retenido, y *fire*, señal que nos avisa de que el tiempo de hold a finalizado y ha llegado el momento de disparar un spike retenido. A continuación encontramos un circuito combinacional, el cual gestiona las señales de *set* y *reset* de cada contador, en base a las señales de *hold* y las entradas del sistema, así como disparando los spikes necesarios por su salida, implementando la funcionalidad descrita en la Tabla 3-1. Con el fin de poder observar su comportamiento para diversos tiempos de hold, podemos fijar el tiempo de hold de la referencia y la realimentación independientemente.

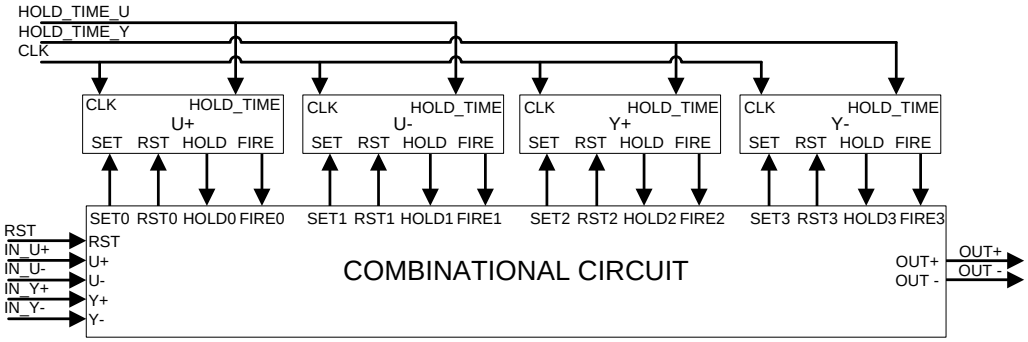


Figura 3.5: Circuito implementado para el Hold & Fire

La Ecuación [3-9] nos permite calcular los tiempos de hold en base a la señal de entrada *hold_time*. Surge inmediatamente una cuestión sobre cuáles deben ser los tiempos de hold. Si los tiempos de hold son muy pequeños, aparece cierta incertidumbre en el sistema, es decir, sólo se cancelaran spikes si da la casualidad de que se disparan muy cerca en el tiempo. Dicha incertidumbre se traduce en que la resta de las tasas de spikes de entrada se realiza compensando spikes positivos con negativos a la salida, y como veremos más adelante, introduciendo perturbaciones en la respuesta del motor. Sin embargo, con tiempos de hold muy grandes, habrá suficiente tiempo entre spikes para que se cancelen, obteniendo una señal de salida cuya tasa de spikes será una resta precisa de las tasas de spikes de entrada. En un principio, para mantener una mínima incertidumbre, es necesario un tiempo de hold al menos mayor que el período de la mínima frecuencia posible de los spikes de entrada, como mostramos en la Ecuación [3-10].

$$T_{Hold} = hold\_time * T_{CLK}$$

Ecuación [3-9]

$$hold\_time_{min} \geq \frac{1}{InputSpikeRate_{min} * T_{CLK}}$$

Ecuación [3-10]

3.3. Modelos de simulación

Al igual que en el capítulo anterior, hemos usado Simulink con Xilinx System Generator para simular estos modelos, junto con modelos dinámicos de motores de DC. Implementado controles proporcionales (P) de velocidad en lazo cerrado basados únicamente en spikes.

Vamos a comenzar con el estudio del comportamiento teórico de nuestro sistema ante una realimentación con un control proporcional. Estos controladores consisten en multiplicar el error del motor, respecto de la referencia proporcionada, por una constante. En el caso de los moduladores PWM y PFM la constante proporcional está representada por el período de PWM y el ancho de Spike respectivamente. Dado que nuestro sistema es de segundo orden, y de forma general, se tiene la siguiente expresión (equivalente a la Ecuación [2-1] para un motor de DC):

$$M(S) = K_{Motor}\frac{\omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2}$$

Ecuación [3-11]

Siendo la ganancia introducida por el método de actuación:

$$K_{spikes} = \begin{Bmatrix} \frac{T_{PWM}}{2^{Nº\,Bits}}V_{PS} & con\ PWM \\ ThV_{PS} & con\ PFM \end{Bmatrix}$$

Ecuación [3-12]

La realimentación que vamos a realizar no es unitaria, ya que hemos de convertir la velocidad del motor a una señal de spikes. Para ello debemos realizar dos operaciones sobre la señal: primero escalarla, desde su velocidad al rango de trabajo del simulador (n-bits), y luego convertirla en spikes, calculando la frecuencia de los spikes en base a la velocidad del motor escalada:

$$K_{feedBack} = K_{scale} * K_{int2Spikes}$$

$$K_{scale} = \frac{2^{Nº\,Bits}}{\omega_{motorMax}}$$

$$K_{int2Spikes} = \frac{1}{2^p * T_{clk}}$$

Ecuación [3-13]

Donde p es la resolución del generador de spikes en bits, y Tclk el período de reloj del sistema. Siendo todos estos parámetros constantes del sistema, como veremos más adelante.

Tal y como se expone en teoría de control tradicional [Ogata04], realimentando el sistema original obtenemos:

$$M_{CL}(S) = \frac{TF(s) * M(s)}{1 + FB(s) * TF(s) * M(s)} = \frac{K_{spikes}M(s)}{1 + K_{feedBack}K_{spikes}M(s)}$$

Ecuación [3-14]

Desarrollándolo:

$$M_{CL}(S) = \frac{K_{Motor}K_{spikes}}{1 + K_{feedBack}K_{Motor}K_{spikes}} * \frac{\omega_{ncl}^2}{s^2 + 2\delta_{cl}\omega_{ncl}s + \omega_{ncl}^2}$$

Ecuación [3-15]

Donde:

$$\omega_{ncl}^2 = (1 + K_{feedBack}K_{spikes}K_{Motor})\,\omega_n^2 \leftrightarrow 2\delta_{cl}\omega_{ncl} = 2\delta\omega_n$$

Ecuación [3-16]

Pudiendo obtener a partir de estas ecuaciones el resto de parámetros característicos teóricos del sistema realimentado en base a la ganancia K de los spikes [Aström09], como por ejemplo:

$$K_{staticCL} = \frac{K_{Motor}K_{spikes}}{1 + K_{feedBack}K_{spikes}K_{Motor}}$$

$$\omega_{ncl} = \sqrt{(1 + K_{feedBack}K_{spikes}K_{Motor})}\;\omega_n$$

$$\delta_{cl} = \frac{\delta}{\sqrt{(1 + K_{feedBack}K_{spikes}K_{Motor})}}$$

Ecuación [3-17]

Sustituyendo las constantes, en el caso particular de nuestro motor:

$$K_{feedBack} = K_{scale} * K_{int2Spikes} = \frac{2^{10}}{398.7386} * \frac{1}{2^{15} * 20 * 10^{-9}}$$

$$K_{staticCL} = \frac{33.2282 * K_{spikes}}{1 + 33.2282 * 3918.6 * K_{spikes}} = \frac{33.2282 * K_{spikes}}{1 + 1.3021 * 10^5 K_{spikes}}$$

Ecuación [3-18]

$$\omega_{ncl} = 1.6644 * 10^3 \sqrt{\left(1 + 1.3021 * 10^5 K_{spikes}\right)}$$

$$2\delta_{cl}\omega_{ncl} = 2\delta\omega_n = 8.7035 * 10^3$$

$$\delta_{cl} = \frac{4.3517 * 10^3}{1.6644 * 10^3 \sqrt{(1 + 1.3021 * 10^5 K_{spikes})}} = \frac{2.6146}{\sqrt{(1 + 1.3021 * 10^5 K_{spikes})}}$$

Ecuación [3-19]

Obteniendo las respuestas de la Figura 3.6 ante un escalón ideal, para diversos valores de la ganancia del sistema. Observándose como el sistema empieza a comportarse como un sistema sub amortiguado a partir de un determinado nivel de ganancia.

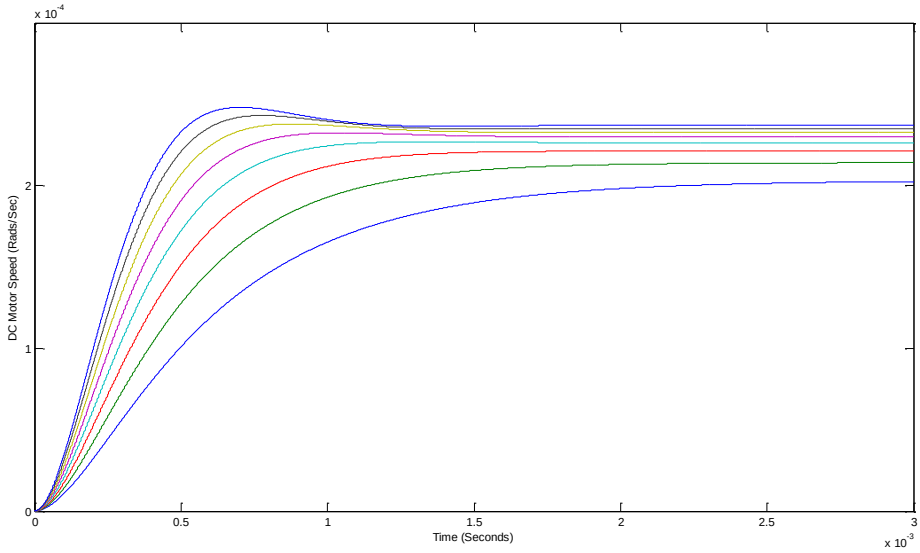


Figura 3.6: Respuestas teóricas ante un escalón del motor con un control P ideal en lazo cerrado

En la Figura 3.7 mostramos los posibles valores que puede tomar el coeficiente de amortiguación en base a la ganancia de los spikes. Resulta de especial interés los valores de la ganancia de los spikes que transforman el sistema entre un sistema sub o sobre amortiguado:

$$\delta_{cl} < 1 \Longrightarrow \frac{2.6146}{\sqrt{(1 + 1.3021 * 10^5 K_{spikes})}} < 1 \Longrightarrow 4.4821 * 10^{-5} < K_{spikes}$$

Es decir, mientras la ganancia de los spikes sea menor que el valor calculado, el sistema será sobre amortiguado, en cualquier otro caso, mostrará una respuesta sub amortiguada.

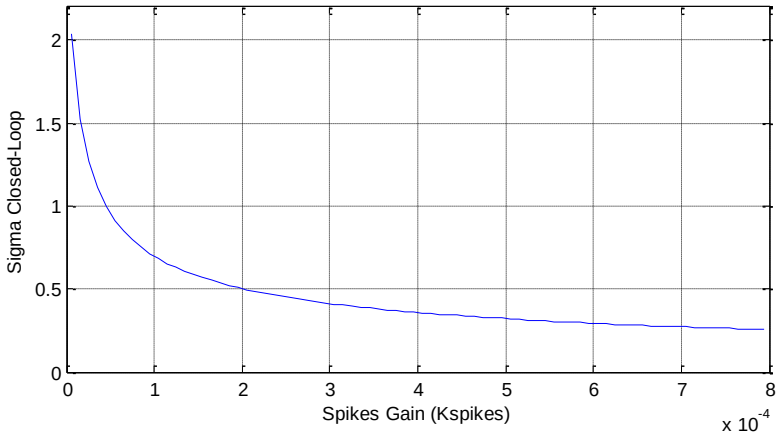


Figura 3.7: Representación del coeficiente de sub-amortiguamiento frente a la ganancia de los spikes

En las respuestas del motor mostradas en la Figura 3.6 también pueden apreciarse como el tiempo de subida es diferente para cada caso, hecho que no ocurría en el caso del lazo abierto. Según la teoría de sistemas, el tiempo de subida depende de la frecuencia natural en lazo cerrado (w_{ncl}) del sistema, modificada en nuestro caso por la ganancia, según la siguiente ecuación [Ogata04]:

$$t_r = \frac{1.8}{2\omega_{ncl}} = \frac{1.8}{2 * 1.6644 * 10^3 \sqrt{\left(1 + 1.3021 * 10^5 K_{spikes}\right)}} = \frac{10.8147 * 10^{-4}}{\sqrt{\left(1 + 1.3021 * 10^5 * K_{spikes}\right)}}$$

La Figura 3.8 muestra la evolución del tiempo de subida:

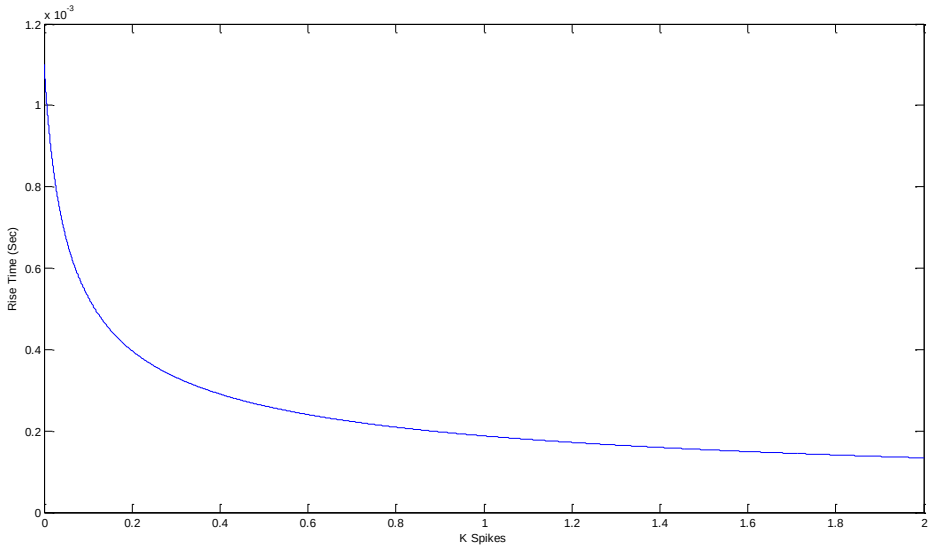


Figura 3.8: Representación del tiempo de subida frente a la ganancia de los spikes

3.3.1. Simulaciones basadas en Hold & Fire

Una vez mostrado el comportamiento teórico del motor, vamos a proceder a verificar el comportamiento de los componentes VHDL expuestos. En la Figura 3.9 mostramos el esquema completo de simulación. Análogamente al capítulo anterior, vamos a ir alternando el bloque de control VHDL por los diversos elementos expuestos. A diferencia del modelo de simulación mostrado en el capítulo anterior, se observa la realimentación de la velocidad del motor al bloque del control VHDL. Dado que Simulink nos proporciona la velocidad del motor como un número discreto, hemos de transformarlo en spikes, para ello usamos otro generador de spikes sintético. Este generador de spikes recibirá la velocidad del motor como un número discreto y los convertirá a una secuencia de spikes, cuya frecuencia representará la velocidad del motor. Como se expondrá más adelante, es usual sensar la velocidad de los motores con encoders ópticos, los cuales nos ofrecen la velocidad del motor codificada mediante pulsos, usando una modulación PFM.

Como referencia, generaremos una secuencia de spikes entre ±1.5MSpikes/Sec. Para ello codificaremos la referencia como un número de 11 bits con signo, lo que implica un rango dinámico de ±1023 posibles valores. Para que la velocidad del motor esté en la misma escala que la referencia, hemos de escalarla, convirtiendo la velocidad, ±398.7Rad/Sec en un número de 11 bits con signo, entre ±1023. En el esquema de simulación de la Figura 3.9 destinamos un módulo de ganancia en Simulink, multiplicando la velocidad del motor por *Kscale*, escalándola al valor adecuado.

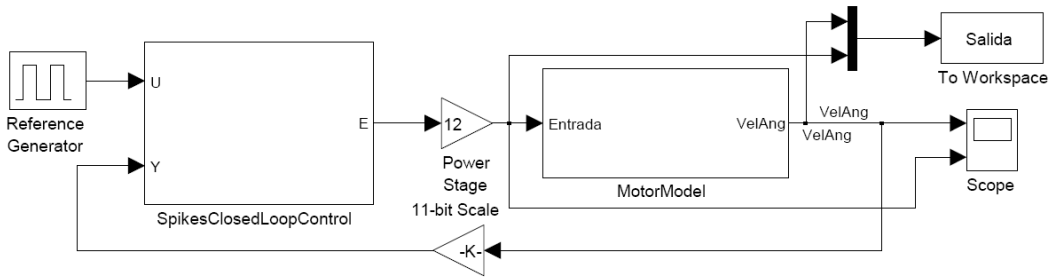


Figura 3.9: Esquema global de simulación en lazo cerrado

El modelo Hold & Fire ha sido simulado tanto con el modulador PWM, como con el Spikes Expansor, obteniendo así las respuestas del motor para los diversos mecanismos de actuación, PWM y PFM. En las Figura 3.10 y la Figura 3.11 mostramos los esquemas de simulación usados, que sólo difieren en el mecanismo de actuación. En la superior tenemos el conversor de spikes a PWM, y en la inferior el Spikes Expansor. En ambas figuras, a la izquierda se encuentran las señales de entrada del sistema, el reloj y la señal de reset, así como las señales propias del control, la referencia, U, y la velocidad simulada del motor, Y ambas señales son convertidas a spikes mediante el uso de los generadores de spikes exhaustivos. Una vez que ambas señales se convierten en spikes, son restadas usando el Hold & Fire. La salida del Hold & Fire es aplicada respectivamente al modulador PWM o el Spikes Expansor en base a cada caso.

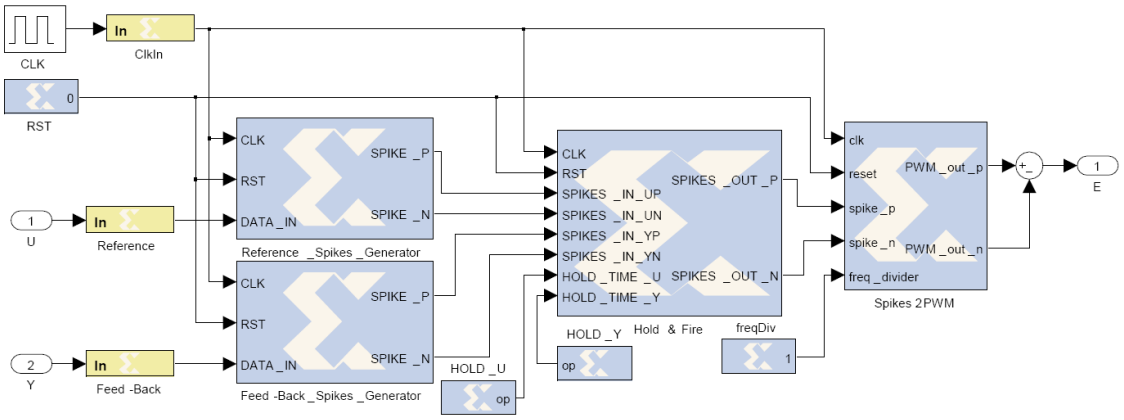


Figura 3.10: Esquema de simulación del Hold & Fire con el modulador PWM

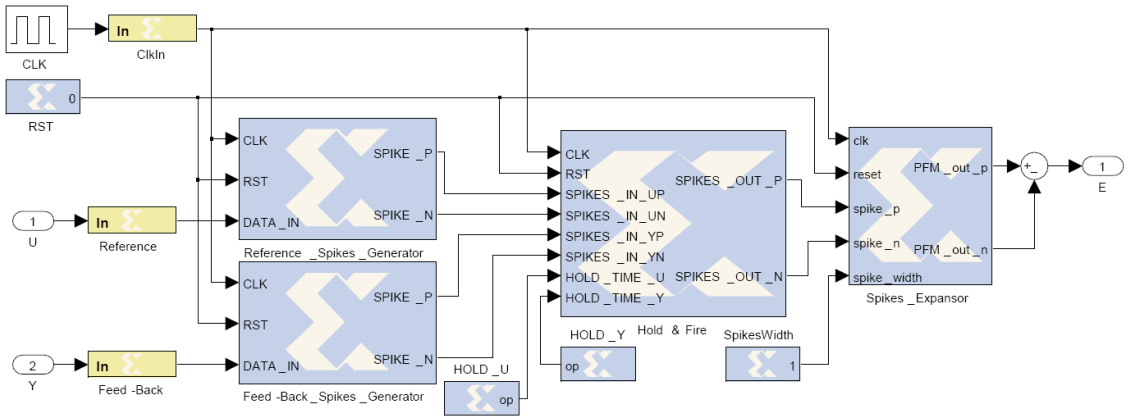


Figura 3.11: Esquemas de simulación del Hold & Fire junto con el Spikes Expansor

En la Figura 3.12 y la Figura 3.13 mostramos unos de los resultados de las simulaciones con un modulador PWM y con un Spikes Expansor respectivamente. En ambas, arriba mostramos la respuesta de la velocidad del motor, así como la referencia aplicada al sistema (nótese que esta referencia está codificada en spikes, pero nosotros la mostramos como un número discreto, en concordancia con la velocidad del motor). En el centro se muestran las señales aplicadas a los bornes del motor, en el caso del PWM se aprecia como la señal es periódica, pero va decreciendo su tiempo en alto según va aumentando la velocidad del motor (y decrementándose el error del sistema). En el caso del Spikes Expansor se observan los spikes extendidos aplicados al motor, así como al principio, al ser la frecuencia del error muy grande, el Spikes Expansor se satura, quedándose su salida fija al nivel alto, sin embargo, según el motor acelera, el error se va reduciendo, dejando de saturar y estabilizándose a una frecuencia fija. Finalmente al pié de las figuras mostramos la velocidad del motor codificada en spikes, donde se observa claramente cómo va alcanzando velocidad, hasta tener una tasa de spikes constantes.

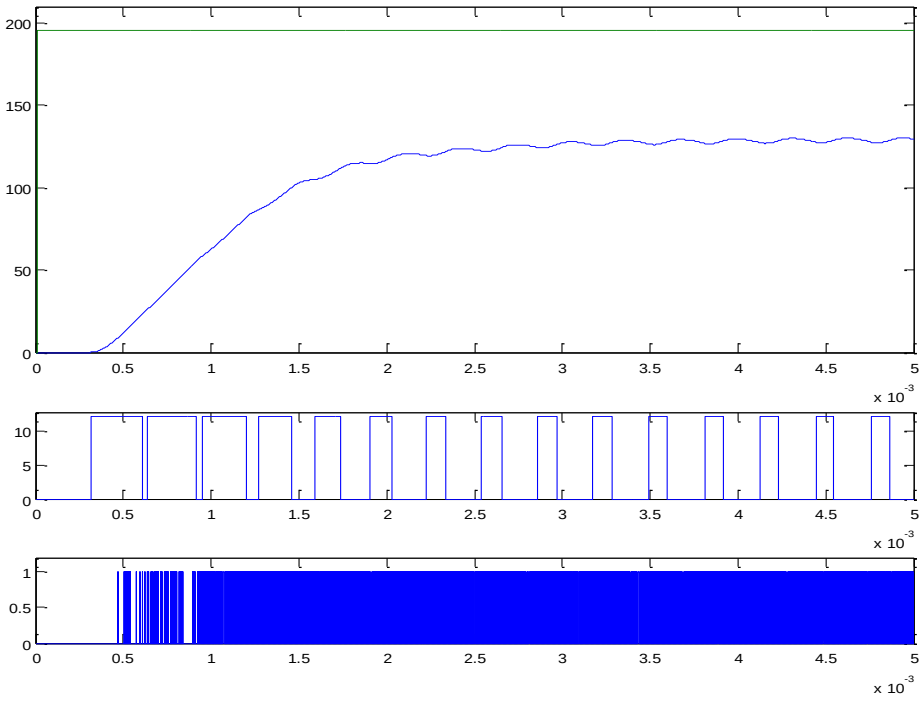


Figura 3.12: Respuesta del motor ante una entrada en escalón, usando Hold & Fire y PWM. Velocidad, arriba; Señal PWM, centro; Spike de entrada del modulador PWM (error)

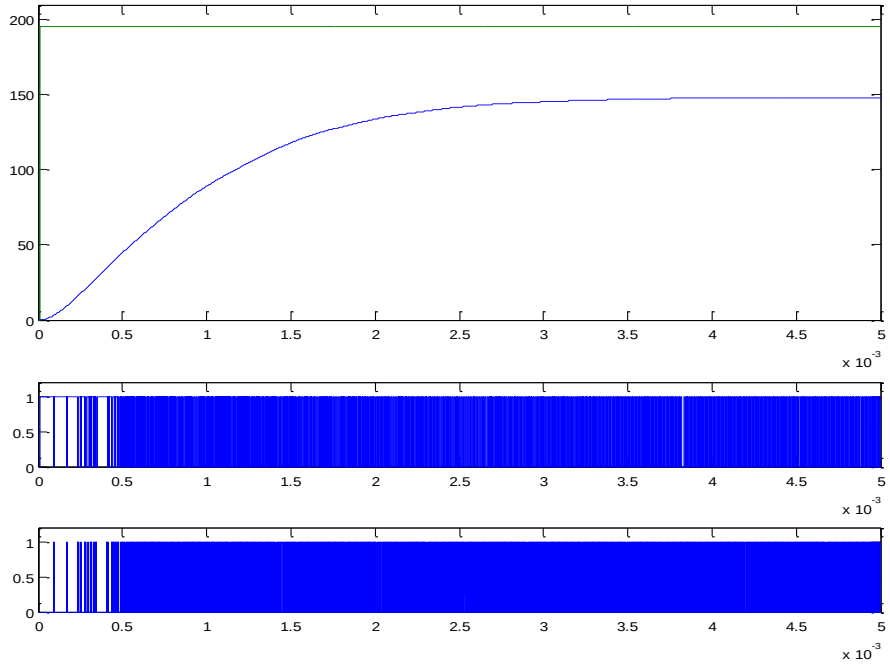


Figura 3.13: Respuesta del motor ante una entrada en escalón, usando Hold & Fire y Spikes Expansor. Velocidad, arriba; Señal PFM, centro; Spike de entrada del Spikes Expansor (error)

3.3.1.1. Simulaciones del Hold & Fire y el modulador PWM

En primer lugar vamos analizar la interacción entre el H&F y el modulador PWM. Para poder entender el comportamiento de la velocidad del motor en la lazo cerrado, hemos comenzar analizando las características del sistema en lazo abierto con el uso del modulador PWM. Como se expuso en el capítulo anterior, variando el período de PWM (Tpwm), podemos variar la ganancia asociada a los spikes de entrada, siendo la ganancia introducida por este componente, como se mostró con la Ecuación [3-12]:

$$K_{PWM} = \frac{T_{PWM}}{2^{Nº\,Bits}} V_{PowerSupply}$$

Ecuación [3-20]

En base a la Ecuación [3-1], el sistema resultante es:

$$M_{CL}(S) = \frac{K_{Motor}K_{PWM}}{1 + K_{feedBack}K_{Motor}K_{PWM}} * \frac{\left(1 + K_{feedBack}K_{Motor}K_{PWM}\right)\omega_n^2}{s^2 + 2\delta\omega_n s + \left(1 + K_{feedBack}K_{Motor}K_{PWM}\right)\omega_n^2}$$

Ecuación [3-21]

Siendo la ganancia estática en lazo cerrado en base al período de PWM:

$$K_{StaticCL} = \frac{\frac{K_{Motor}V_{PS}}{2^{Nº\,Bits}} T_{PWM}}{1 + \frac{K_{feedBack}K_{Motor}V_{PS}}{2^{Nº\,Bits}} T_{PWM}} = \frac{1.5576 * T_{PWM}}{1 + 608.4 * T_{PWM}}$$

Ecuación [3-22]

Para comprobar el efecto en el sistema del período de PWM, tal y como los resultados teóricos predicen, vamos a realizar un conjunto de simulaciones, mostrada en la Figura 3.14, donde la frecuencia de los spikes de entrada en la referencia es constante (915.5 kSpikes-> 233.6 rad/s), y Tpwm variable (desde 0.2mS hasta 0.65mS). Implicando una ganancia entre [0.96-3.07]* 10^{-5}. Se observa claramente como el sistema pasa de ser un sistema sobre amortiguado, para Tpwm menores que 0.3mS, a un sistema sub amortiguado, para períodos mayores. Según las ecuaciones teóricas, el sistema se comporta como un sistema subamortiguado a partir de una ganancia de $4.5*10^{-5}$, sin embargo, con el uso del PWM empieza a serlo a partir de una ganancia de $1.53*10^{-5}$. Esto se sebe al retraso que introduce el modulador de PWM en el lazo de control, efecto que no ocurre con la modulación PFM. Los retrasos puros se pueden modelar como una pérdida constante de fase en el sistema, reduciendo el margen de fase [Ogata04], y haciéndolo así más inestable cuando es realimentado.

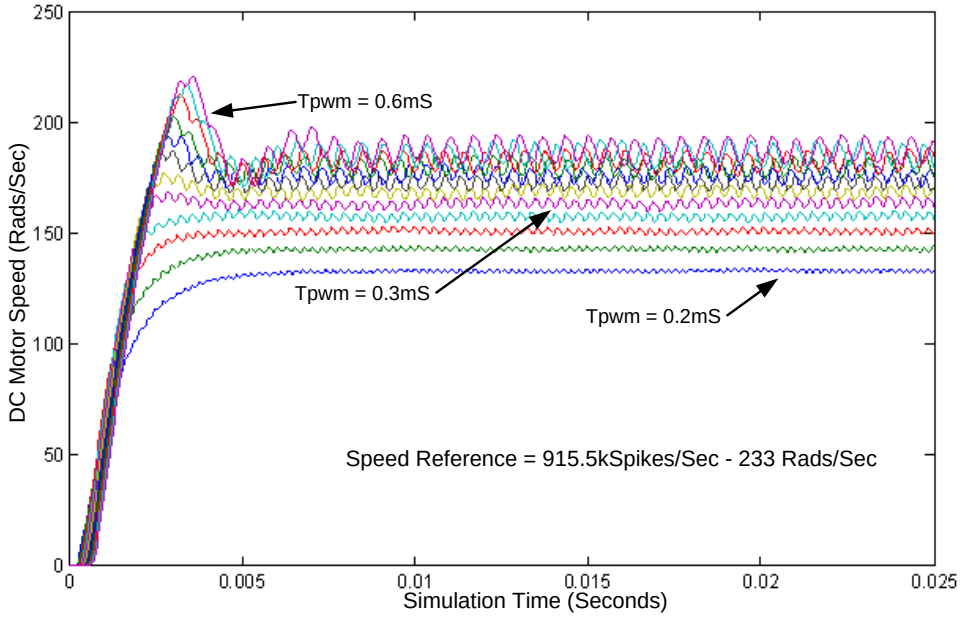


Figura 3.14: Respuesta del motor ante una referencia de spikes constante y diversos períodos de PWM.

Si extraemos la ganancia estática de las simulaciones, y la comparamos con la Ecuación [3-22] (ambas mostradas en la Figura 3.15) podemos comprobar cómo la ganancia del sistema en lazo cerrado efectivamente se comporta como la Ecuación [3-22] predecía, y comprobando que el sistema se comporta como se espera de un sistema de segundo orden realimentado en condiciones similares. Ofreciendo un ajuste entre las simulaciones y las ecuaciones teóricas de más de un 99.23%

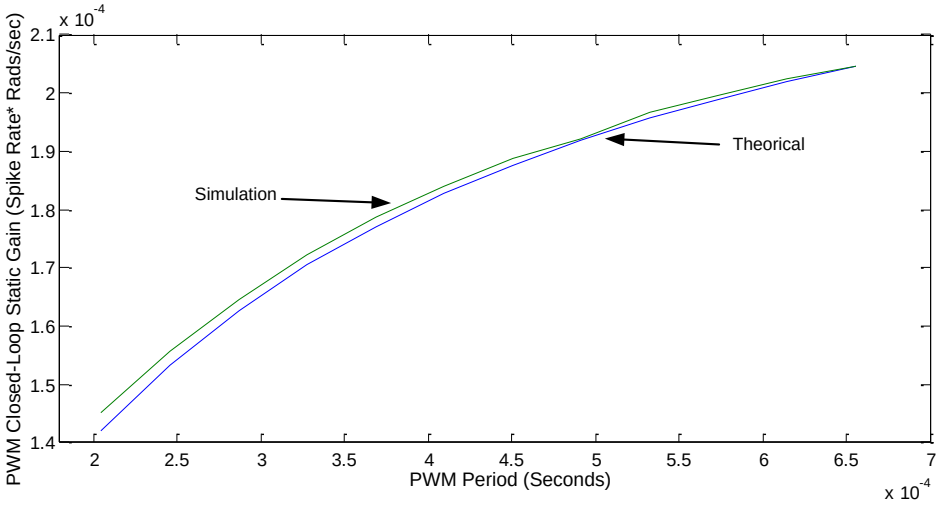


Figura 3.15: Ganancia estática en lazo cerrado mediante el uso de PWM.

Vistos los efectos del período de PWM en el sistema, ahora vamos a pasar a comprobar que efectivamente la respuesta en velocidad del motor es lineal respecto a la frecuencia de los spikes a la entrada. Para ello hemos realizado una secuencia de simulaciones, en las que el período de PWM permanece constante (0.15mSec, implicando una ganancia total en lazo cerrado de $1.215*10^{-4}$), y variamos la frecuencia de los spikes de la referencia de velocidad (desde 152.6 hasta 762.9 kSpikes/Sec). Los resultados de las simulaciones pueden verse en la Figura 3.16, donde se encuentran las referencias de velocidad aplicadas al sistema, así como la respuesta del motor para cada una de ellas.

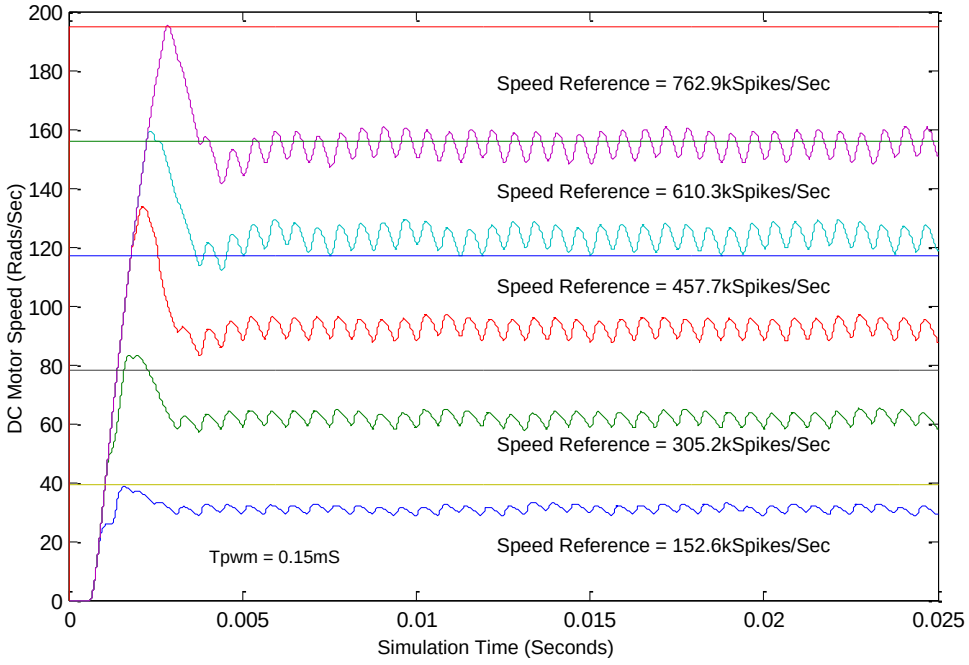


Figura 3.16: Respuesta del motor ante una referencia de spikes variable y período de PWM constante.

Las ecuaciones teóricas predicen que la velocidad del motor debe seguir la siguiente expresión:

$$\omega_{static} \approx K_{StaticCL\_PWM} * SpikeRate = 1.8136 * 10^{-4} * SpikeRate$$

En la Figura 3.17 se muestran la velocidad estacionaria teóricas del sistema (azul), junto con la velocidad estacionaria extraída de las simulaciones (verde). Si hacemos una regresión lineal sobre las velocidades de las simulaciones, obtenemos:

$$y(x) = ax + b = 1.8081 * 10^{-4}x + 0.32$$

Donde x representa la frecuencia de los spikes de la referencia. Comparando la pendiente de esta recta con la pendiente teórica obtenemos:

$$\frac{Pendiente\ Regresión}{Pendiente\ Teórica}100 = \frac{1.8081 * 10^{-4}}{1.8136 * 10^{-4}}100 = 99.7\%$$

Finalmente, para un período de PWM fijo y una referencia de velocidad codificada con spikes variable, obtenemos en los resultados de las simulaciones un ajuste del 99.7% respecto a los resultados teóricos.

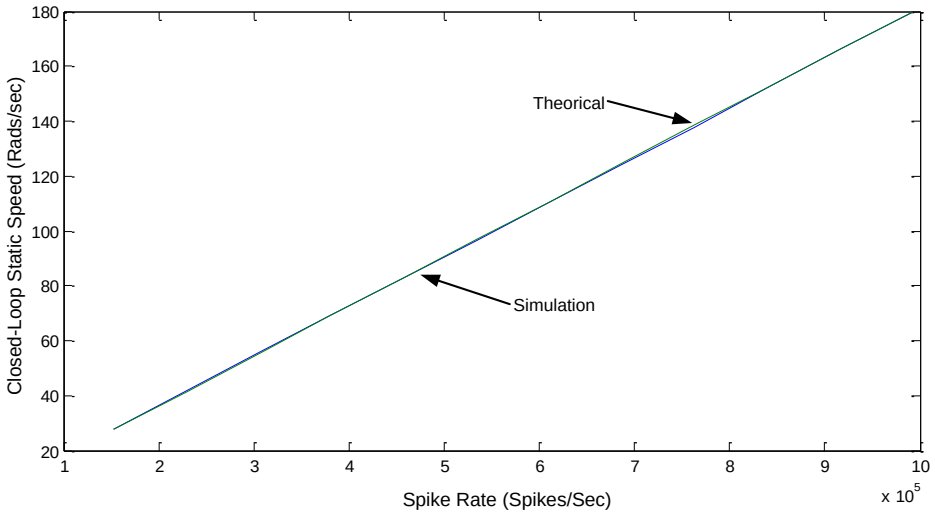


Figura 3.17: Comparativa entre la velocidad estacionaria del motor simulada y teórica, usando el modulador PWM

3.3.1.2. Simulaciones del Hold & Fire con el Spikes Expansor (PFM)

Una vez analizado el comportamiento del sistema usando PWM, vamos a proceder a realizar un análisis similar, pero usando el Spikes Expansor, vamos analizar la interacción entre el H&F y el Spikes Expansor. Como se expuso en el capítulo anterior, la ganancia estática en lazo abierto mediante el uso del Spikes Expansor es:

$$K_{SpikesExpansor} = T_H V_{PS}$$

Ecuación [3-23]

En base a la a Ecuación [3-1], el sistema resultante es:

$$M_{CL}(S) = \frac{K_{Motor}K_{SpikesExpansor}}{1 + K_{feedBack}K_{Motor}K_{SpikesExpansor}} * \frac{(1 + K_{feedBack}K_{Motor}K_{SpikesExpansor})\,\omega_n^2}{s^2 + 2\delta\omega_n s + (1 + K_{feedBack}K_{Motor}K_{SpikesExpansor})\,\omega_n^2}$$

Ecuación [3-24]

Siendo la ganancia estática en lazo cerrado en base al período de PWM:

$$K_{StaticCL} = \frac{K_{Motor}V_{PS}T_H}{1 + K_{feedBack}K_{Motor}V_{PS}T_H} = \frac{398.7386 * T_H}{1 + 1.5576 * 10^6 * T_H}$$

Ecuación [3-25]

Para comprobar el efecto del ancho de spike en el sistema, hemos realizado una primera secuencia de simulaciones, mostrada en la Figura 3.18, en la que aparece la respuesta del motor en trazo continuo y la salida ideal del motor, en trazo discontinuo. En estas simulaciones la frecuencia de los spikes de referencia es constante (915.5kSpikes -> 234.4 rad/Sec) y el ancho de spike variable (desde 0.2uSec hasta 10.2uSec). A primera vista se observa la ausencia del rizado introducido por el uso del PWM. Además a diferencia del uso del modulador PWM, el Spikes Expansor introduce una mínima latencia en el lazo de control (apenas 3 ciclos de reloj), consecuentemente el sistema no pierde tanta fase, pudiendo tener ganancias elevadas sin apenas sobre oscilación. En la figura también se observa claramente como la salida del motor en todos los casos es semejante a la respuesta ideal en estado estacionario, sin embargo, en las respuestas transitorias ambas difieren debido al error introducido por el controlador, el cual será estudiado en detalle en el próximo apartado.

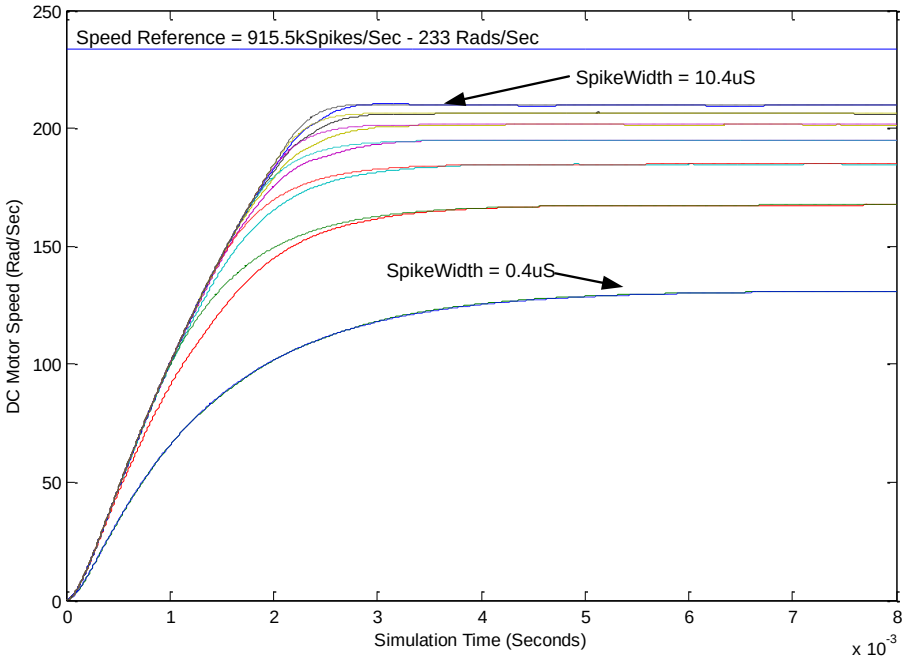


Figura 3.18: Respuesta del motor ante una referencia de spikes constante y diversos anchos de spikes.

La Figura 3.19 muestra la ganancia en lazo cerrado del sistema extraída de las simulaciones (verde), frente a la predicha por las ecuaciones teóricas (azul). Se observa claramente como ambas tienen un comportamiento muy parejo, mostrando un ajuste entorno al 99.93%,

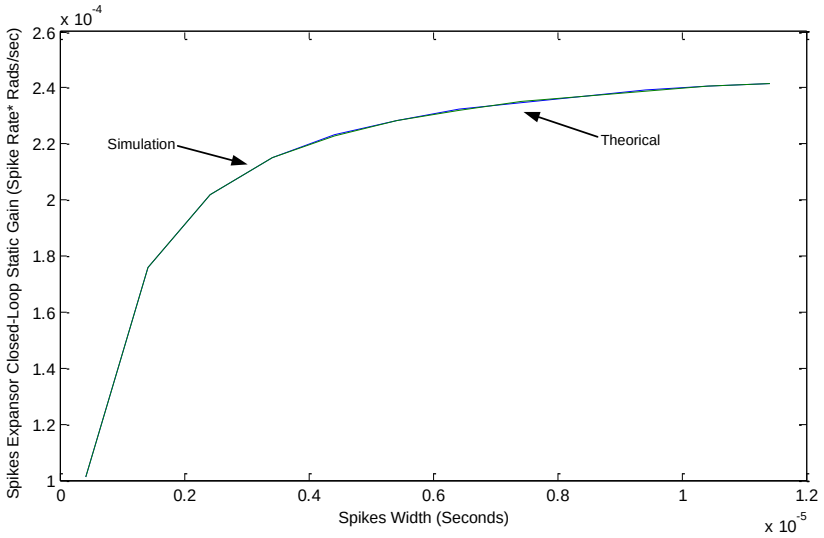


Figura 3.19: Ganancia estática en lazo cerrado mediante el uso del Spikes Expansor.

Vistos los efectos de la ganancia en el sistema, ahora vamos a pasar a comprobar que efectivamente la respuesta en velocidad del motor es lineal respecto a la frecuencia de los spikes a la entrada. Para ello hemos realizado una secuencia de simulaciones, en las que el ancho de spike permanece constante (4.82uSec), y variamos la frecuencia de los spikes de la referencia de velocidad (desde 152.6 hasta 1068.1kSpikes/Sec). Los resultados de las simulaciones se encuentran en la Figura 3.20, en la que mostramos las diferentes referencias de velocidad, y la respuesta del motor para cada una de ellas, estando de nuevo representadas las respuestas de las simulaciones en trazo continuo, y las teóricas en trazo discontinuo. Al igual que en las respuesta del motor en el caso anterior, cuando modificábamos el ancho de spikes, el ajuste obtenido entre los resultados de las simulaciones y la respuesta teórica, es muy bueno en estado estacionario, pero una vez más, en la respuesta transitoria se observa una leve desviación.

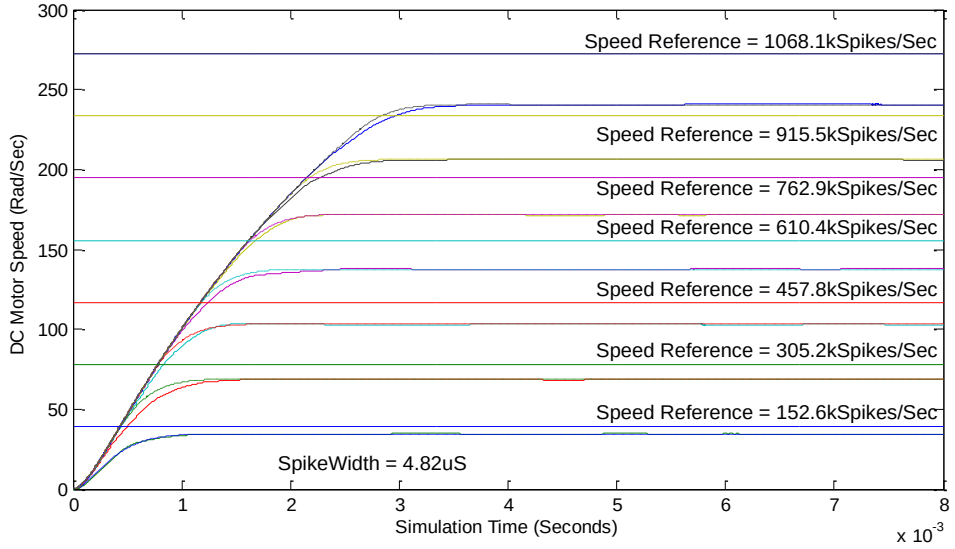


Figura 3.20: Respuesta del motor ante una referencia de spikes variable y ancho de spike constante.

Las ecuaciones teóricas predicen que la velocidad del motor debe seguir la siguiente expresión:

$$\omega_{static} \approx K_{StaticCL_{Spikes}Expansor} * SpikeRate = 2.3125 * 10^{-4} * SpikeRate$$

En la Figura 3.21 mostramos la velocidad estacionaria de las simulaciones (en verde) frente la velocidad estacionaria predicha por la ecuación anterior (en azul). Si realizamos una regresión lineal sobre las velocidades de las simulaciones, obtenemos:

$$y(x) = ax + b = 2.3067 * 10^{-4}x + 0.1643$$

Donde X representa la frecuencia de los spikes de la referencia. Comparando la pendiente de esta recta con la pendiente teórica obtenemos:

$$\frac{Pendiente\ Regresión}{Pendiente\ Teórica}100 = \frac{2.3067 * 10^{-4}}{2.3125 * 10^{-4}}100 = 99.75\%$$

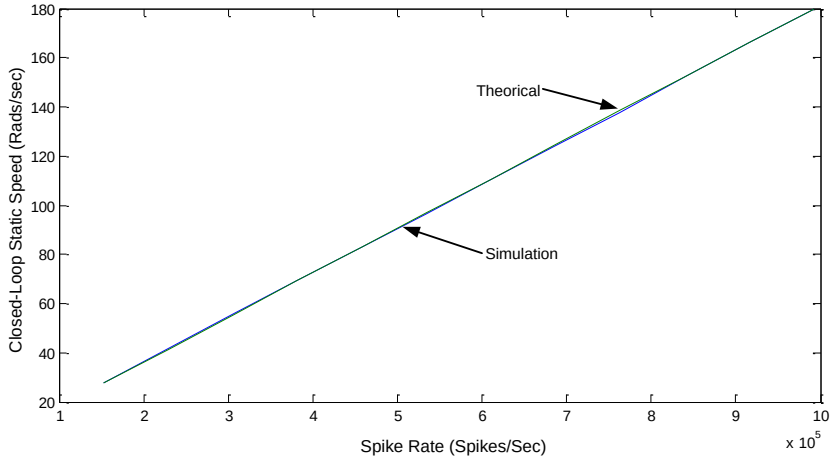


Figura 3.21: Comparativa entre la velocidad estacionaria del motor simulada y teórica, usando el Spikes Expansor

En vista de los resultados obtenidos podemos concluir que el controlador usando Hold & Fire efectivamente se comporta como un sistema en lazo cerrado, controlando la velocidad del motor. Además tanto con el uso del modulador PWM y del Spikes Expansor, las respuestas del motor se ajustan con una elevada precisión a los modelos teóricos.

3.3.2. Fuentes de errores del control proporcional basado en spikes

Tal y como se apreciaba en las figuras 3.16 y 3.18, el controlador en lazo cerrado proporciona en estado estacionario una respuesta por parte del motor muy próxima a la respuesta ideal, sin embargo, también se observan algunas desviaciones en la respuesta transitoria. Dichas desviaciones las achacamos a la desviación de la idealidad de los elementos componentes del control en lazo cerrado, y en consecuencia, a la falta de homogeneidad de los spikes presentes en el controlador.

En primer lugar vamos a exponer las fuentes del error introducido por el Spikes Expansor, y aunque es el último elemento en el controlador, es el más afectado por la falta de homogeneidad temporal de los spikes. La Figura 3.21 ilustra el problema, en el que se deberían extender 3 spikes por un tiempo equivalente a 3 veces el ancho de spikes, en la parte superior de la figura. Sin embargo, en el pie de la figura vemos que los spikes a la entrada del Spikes Expansor no están homogéneamente distribuidos en el tiempo. Todo comienza cuando el segundo spike llega mientras aún se está extendiendo el primero, produciendo una "colisión". Cuando se da esta situación el Spikes Expansor resetea el valor de su contador interno con el valor del ancho de spike, expandiendo el nuevo spike, y "olvidándose" del spike anterior. Como la entrada del Spikes Expansor en este capítulo es un Hold & Fire, como ha disparado un spike positivo de más, lo normal será que dispare uno negativo para compensarlo, de manera que ya no sólo tenemos una colisión, sino una colisión con spikes de distinto signo. En este caso el Spikes Expansor cambiará la polaridad de su salida, y comenzará a expandir el spike negativo. Finalmente llega el spike esperado, siendo extendido con normalidad. En consecuencia el tiempo que ha estado la entrada del motor activa ha sido distinto al deseado, además de presentar un cambio de polaridad. Más adelante mostraremos estos efectos sobre la respuesta del motor en la Figura 3.27.

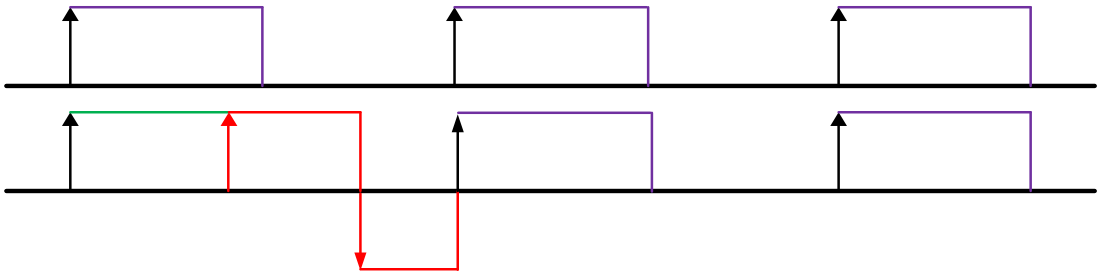


Figura 3.22: Salida del SpikesExpansor ante spikes no distribuidos homogéneamente en el tiempo

A continuación vamos a estudiar el error cometido por el Hold & Fire junto con el generador de spikes Bit-Wise. El error introducido por ambos elementos lo vamos a observar al mismo tiempo, ya que vamos a usar dos generadores Bit-Wise para generar dos secuencias

de spikes para las entradas del Hold & Fire. La Figura 3.23 muestra el escenario de simulación diseñado con dichos fines, tal y como se aprecia a la izquierda en la figura, hemos situado dos generadores sintéticos exhaustivos de spikes a las entradas de un Hold & Fire, estando finalmente la salida de este conectada a un bloque que nos permitirá enviar la información al espacio de trabajo de MATLAB.

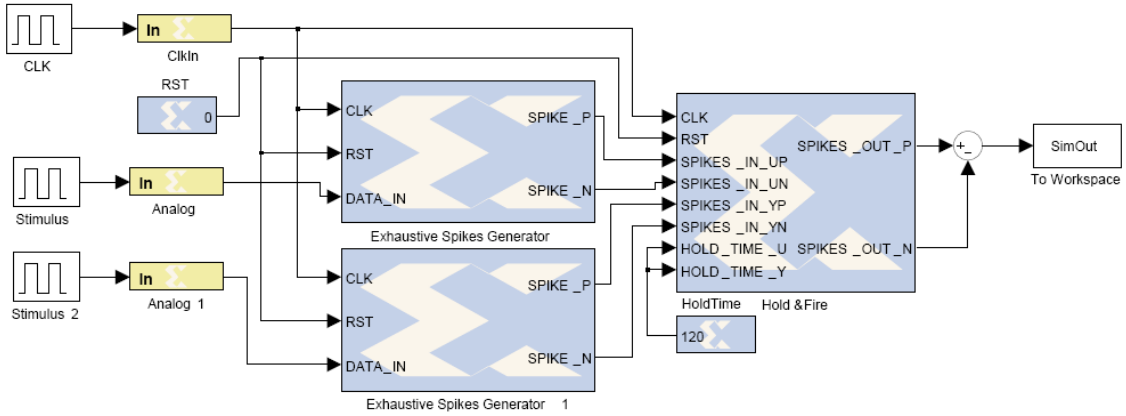


Figura 3.23: Escenario de simulación del Hold & Fire

En primer lugar hemos procedido a analizar la desviación de los spikes de salida del Hold & Fire. El mayor error introducido por este elemento depende de la diferencia en frecuencia en sí de los spikes de entrada, tal y como se muestra en la Figura 3.24. En dicha figura hemos fijado la frecuencia de la entrada positiva, y hemos ido incrementando la frecuencia de los spikes en la entrada negativa. De tal manera que en la gráfica superior la frecuencia de los spikes en la entrada negativa es muy baja, de tal forma que la salida del Hold & Fire son los spikes de la señal aplicada al puerto positivo, sólo cancelando un spike cada vez que se dispara un spike en el puerto negativo, obteniendo así una distribución temporal de los spikes pésima. Sin embargo, cuando vamos incrementando la frecuencia de los spikes del puerto negativo, la homogeneidad de los spikes de salida empieza a crecer, hasta que ambas señales tienen una frecuencia muy parecida, siendo la frecuencia de salida del Hold & Fire más baja, pero mucho mejor distribuida temporalmente.

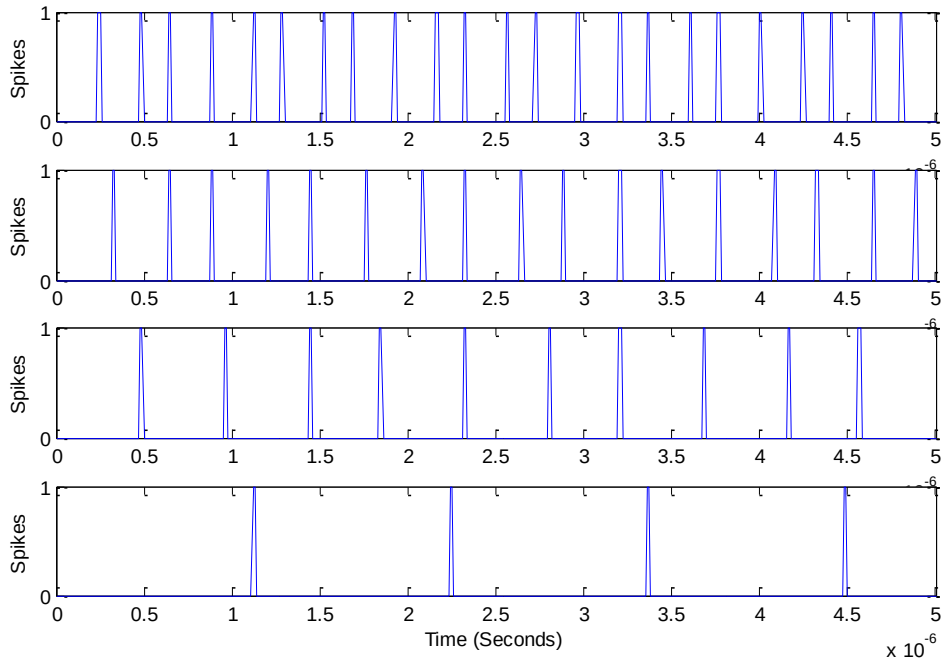


Figura 3.24: Respuesta temporal de Hold & Fire ante señales de spikes con distintas diferencias de frecuencias

Para analizar dicha desviación de distribución en el tiempo hemos medido el intervalo temporal de los spikes, Inter-Spikes-Interval, o ISI, comparándola con el ISI teórico que deberían tener. La Figura 3.25 muestra los resultados tras analizar la desviación en tanto por ciento del ISI de salida del Hold & Fire frente al ISI teórico esperado. La figura muestra una serie de lóbulos, los cuales son debidos a la falta de homogeneidad de los generadores de spikes basados en el método Bit-Wise, siendo esta una de las fuentes de error del sistema, y estudiado en detalle en [Paz09a][Gomez05]. Sin embargo, este error se ve agravado por el error Hold & Fire, superpuesto por una línea discontinua. Pudiendo oscilar entre un 2 y un 35%, dependiendo de la diferencia en frecuencia de los spikes a su entrada.

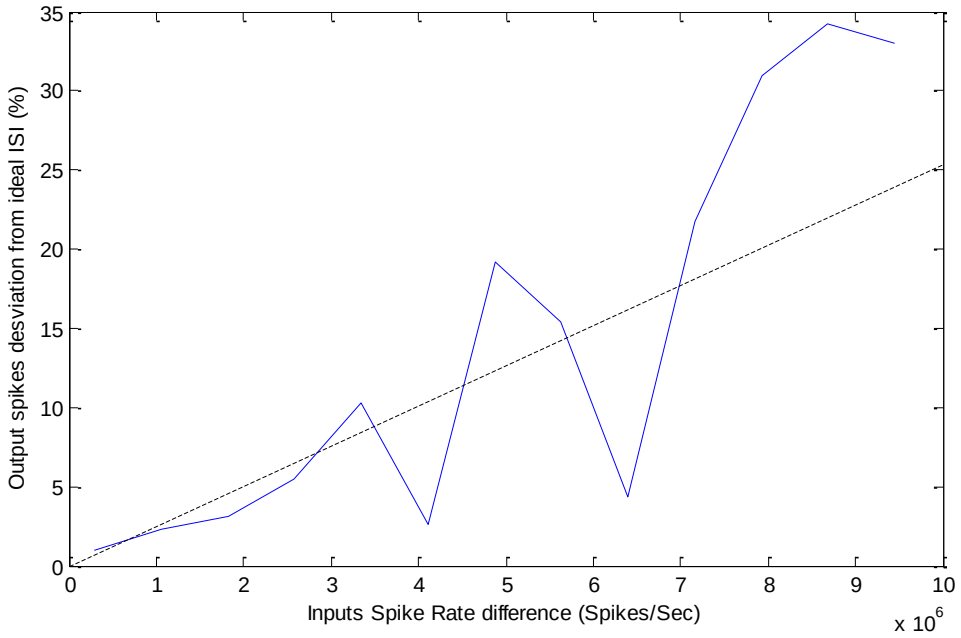


Figura 3.25: Desviación relativa del ISI de los spikes de salida del Hold & Fire frente al ISI ideal

La cuestión que nos queda por abordar es la influencia de los tiempos de hold en el comportamiento del Hold & Fire. Para ello hemos realizado nuevas simulaciones, en las que hemos fijado la frecuencia de los generadores, 230.5KSpikes para la entrada positiva, y 184.3KSpikes para la negativa. A continuación vamos a variar el tiempo de Hold iterativamente, con el fin de comprobar el comportamiento de su salida. Los resultandos de las simulaciones pueden verse en la Figura 3.26. En primer lugar se muestran las señales de entrada generadas, en azul la positiva, y en rojo la negativa. A continuación se muestran las salidas del Hold & Fire para los diversos tiempos de hold, comenzando con el más bajo 20nSec, e incrementándose progresivamente en escalones de 1.5uSec hasta 6uSec. En la figura se ve como ante un tiempo de hold muy pequeño, los spikes no se "esperan" entre ellos, siendo disparados casi inmediatamente, y produciéndose una tasa de cancelaciones entre ellos muy pequeña. De tal manera que emitirá los spikes de la entrada positiva con signo positivo, y los de la entrada negativa con signo negativo, no cancelándose entre ellos, sino más bien compensando spikes positivos con spikes negativos. Sin embargo, según se va incrementando el tiempo de hold, la tasa de cancelación va aumentando, emitiendo menos spikes negativos, y distribuyendo los spikes positivos más homogéneamente en el tiempo. Hasta que finalmente, cuando se le proporciona un tiempo de hold suficientemente alto, por encima del período de ambas señales (nótese que el período más alto de ambas señales es de 5.4uSec, y el tiempo máximo de hold de 6uSec), la tasa de cancelación es la adecuada, no emitiéndose ningún spike positivo, y además consiguiendo una buena distribución temporal de los spikes de salida.

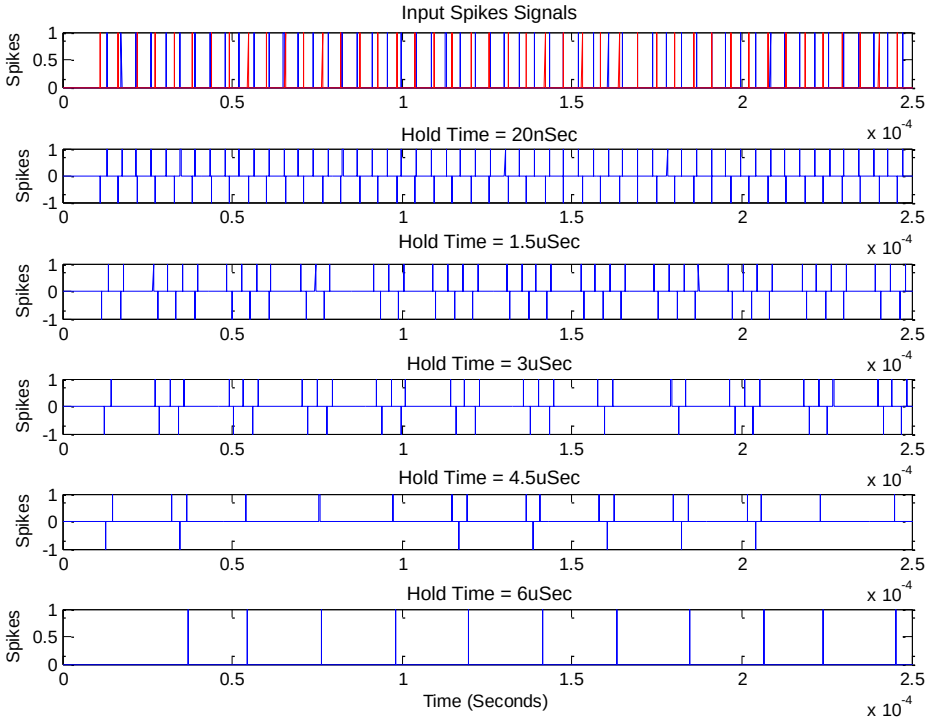


Figura 3.26: Salidas del Hold & Fire ante entradas de distinta frecuencia y diversos tiempos de hold

Para finalizar el análisis del error introducido por el Hold & Fire sólo nos resta ver efecto de los tiempos de hold sobre la respuesta del sistema. Nos vamos a centrar en el uso del Spikes Expansor, ya que el modulador PWM debido a la integración de los spikes a lo largo del período de PWM mitigaría estos efectos. Para ello hemos realizado una serie simulaciones del controlador proporcional en lazo cerrado basado en spikes de la Figura 3.11, cuyos resultados se muestran en la Figura 3.27, en la que la frecuencia de la referencia es fija (305.2kSpikes/Sec), así como el ancho de spike (6uSec), sin embargo hemos ido variando los tiempos de hold desde 0.3uSec hasta 3.3uSec.

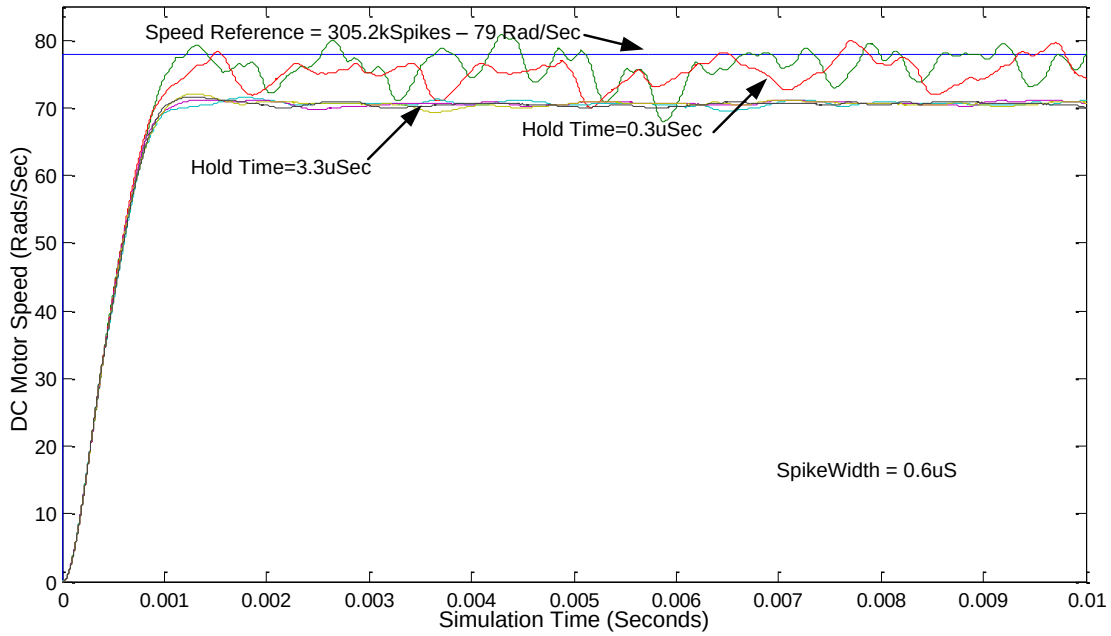


Figura 3.27: Efectos de los tiempos de hold ante una entrada en escalón

En la Figura 3.27 vemos que en el caso de que los tiempos de hold son muy pequeños, los spikes se disparan en un período de tiempo a su vez muy pequeño, y no da lugar a que unos se esperen a otros para cancelarse, así que como ya hemos expuesto, el Hold & Fire compensará el exceso de spikes con un signo, con spikes del signo contrario. En estos casos, el motor responde con una sobre oscilación caótica. Sin embargo, cuando los tiempos de hold son suficientemente grandes, ante las mismas señales de entrada, el Hold & Fire disparará los spikes mejor distribuidos en el tiempo, llegando al motor solamente los spikes necesarios, obteniendo unas respuestas mucho más limpias por su parte.

En conclusión, lo ideal sería tener unos tiempos de hold tan grandes como el período de la frecuencia mínima de los spikes aplicados al Hold & Fire. En muchas ocasiones podemos encontrarnos en ausencia de spikes a las entradas, es decir, frecuencia 0 o período infinito. Siendo una posible simplificación del modelo Hold & Fire el establecer unos tiempos de hold infinitos, ahorrándonos así los contadores de los tiempos de hold, y consumiendo menos hardware.

3.4. Síntesis de las implementaciones del Hold & Fire

En este apartado vamos a analizar el consumo hardware de dos implementaciones del Hold & Fire realizadas, la primera implementación con tiempos de hold finitos y almacenados en registros de 16 bits, y la segunda con tiempos de hold infinitos, ya que como acabamos de exponer, para el control de motores de DC son necesarios tiempos de hold muy altos, tan altos como infinitos. La ventaja de Hold & Fire con tiempos de hold infinitos radica en que necesita menos cantidad de hardware que la versión con tiempos finitos, esto es debido a que ya no son necesarios registros, contadores y comparadores digitales para gestionar los tiempos de hold de cada spike, siendo sólo necesario un registro de 4 bits en el que almacenaremos el

último spike disparado. En la Tabla 3-2 mostramos los resultados de la síntesis de ambas implementaciones. La implementación del Hold & Fire con tiempos de hold finitos necesita 77 slices, mientras que la implementación con tiempos de hold infinitos son solamente 18 slices, gracias a que el hardware se simplifica en gran medida, hasta quedarse reducido a un circuito combinacional con 4 registros de 1 bit. Además gracias a la simpleza del Hold & Fire con tiempos de hold infinitos, la implementación sintetizada puede trabajar a una frecuencia máxima de reloj de más de 244MHz, mientras que el Hold & Fire con tiempos de hold finitos al tener un hardware más complicado, no es capaz de alcanzar los 173MHz.

Tabla 3-2: Consumo hardware y frecuencia de funcionamiento de las implementaciones del Hold & Fire

Implementación del Hold & Fire	Consumo de Slices	Max. Frecuencia de reloj
Tiempos finitos	77	172.46MHz
Tiempos infinitos	18	244.62MHz

4. Controladores proporcionales, integrales y derivativos basados en spikes. El modelo Integrate & Generate

Una vez expuestos los elementos básicos para implementar controles proporcionales (P) en lazo cerrado mediante el uso de spikes, resulta muy interesante el diseño de controladores más complejos como pueden ser los controladores proporcionales-integrales-derivativos (PID), ya que son los más extendidos en la industria, y es un modelo suficientemente simple para controlar la mayoría de los sistemas reales. El controlador PID básico combina las acciones proporcional, derivativa e integral, donde el término proporcional determina la acción ante el error cometido por el sistema en cada momento, el término integral determina la acción ante la suma del error acumulado, y finalmente el término derivativo aporta la variación del error instantáneo. Siendo la suma ponderada de cada uno de estos términos la salida del controlador PID [Astrom09]. Las constantes de esta ponderación serán *kp*, *ki* y *kd*, para los términos proporcional, integral y derivativo respectivamente. Mediante el ajuste de estas tres constantes, el controlador será capaz de proporcionarnos una salida adecuada para satisfacer los requerimientos de comportamiento de la mayoría de los procesos o sistemas. Siendo estos requerimientos los clásicos requerimientos en la ingeniería de control [Ogata04], como puede ser el error estático, el tiempo de subida, el sobre-disparo, el tiempo de asentamiento, etc. En la Figura 4.1 mostramos el diagrama de bloques conceptual de controlador PID, en la que se muestra una posible implementación del controlador PID acorde a la siguiente ecuación:

$$\frac{X(s)}{E(s)} = k_p + \frac{k_i}{s} + k_d * s$$

Ecuación [4-1]

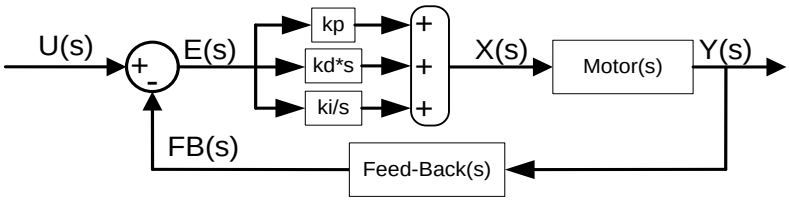


Figura 4.1: Diagrama conceptual de un controlador PID en lazo cerrado

El resumen del efecto sobre la respuesta del motor de cada una de los componentes de un controlador PID [Ogata04] es mostrado en la Tabla 4-1. En ella están presentes las principales características de la respuesta temporal de un sistema frente a los componentes de un controlador PID. En primer lugar encontramos la componente proporcional del controlador, estudiada en profundidad en el capítulo anterior, al incrementar esta componente podemos decrementar el tiempo de subida así como el error en estado estacionario, como ha quedado constatado en el capítulo anterior, sin embargo, en caso de tener un sistema subamortiguado, incrementará la tasa de sobre disparo e introducirá pequeños cambios en el tiempo de establecimiento del sistema. A continuación, si incrementamos la ganancia de la componente integral eliminaremos el error en estado estacionario y decrementaremos el tiempo de subida, sin embargo, estas mejoras en la respuesta del sistema traen consigo mayores tasas de sobredisparo, u overshoot, y tiempos de asentamiento mayores. Finalmente encontramos la

componente derivativa, esta componente nos ayudará a paliar los efectos del integrador, decrementando la tasa de sobreoscilación y el tiempo de asentamiento, sin embargo, por si sólo introduce pequeños cambios en el tiempo de subida y en el error en estado estacionario.

Tabla 4-1: Propiedades cualitativas la respuesta de un sistema con un controlador PID

Componentes del controlador incrementada	Tiempo de subida	Tasa de sobre disparo	Tiempo de asentamiento	Error en estado estacionario
Proporcional	Decrece	Se incrementa	Pequeños cambios	Decrece
Integral	Decrece	Se incrementa	Se incrementa	Se elimina
Derivativa	Pequeños cambios	Decrece	Decrece	Pequeños cambios

Dado que ya hemos expuesto cómo diseñar controles proporcionales y cómo ajustar la ganancia estática en el capítulo anterior, para poder implementar controles PID necesitaríamos dos nuevos elementos. Por un lado un integrador de spikes, es decir un elemento cuya salida sea una nueva secuencia de spikes con una frecuencia proporcional a la integral de la frecuencia de los spikes de entrada. Y por otro lado un derivador de spikes, cuya salida sea otra secuencia de spikes con una frecuencia proporcional a la derivada de la frecuencia de los spikes de entrada. Cuyo comportamiento sea acorde con las siguientes ecuaciones:

$$f_{spikesIntegrator} = k_{integrator} \int_0^{\infty} f_{spikesIn} dt$$

Ecuación [4-2]

$$f_{spikesDerivative} = k_{derivative} \frac{df_{spikesIn}}{dt}$$

Ecuación [4-3]

En este capítulo vamos a proponer un elemento que haga las veces de integrador, el Integrate & Generate, y realimentándolo usando el Hold & Fire vamos a diseñar un derivador de spikes. Para, a continuación introducirlos en nuestro control P, para así poder diseñar controles PID basados en spikes.

4.1. Modelo Integrate & Generate

Este modelo se basa en realizar una integración continua de los spikes recibidos, transfiriendo la cuenta de la integración de los spikes a un generador exhaustivo bit-wise de spikes. La idea es usar un contador de spikes, el cual consiste en un simple contador digital, de manera que cada vez que reciba un spike incrementará su cuenta en 1. Conectando su salida a un generador exhaustivo bit-wise, el cual generará una nueva señal de spikes cuya frecuencia será proporcional al número de spikes contados, *N*, de tal forma que:

$$f_{Int\&Gen} = k_{GenBW} * N$$

Ecuación [4-4]

Dado que la frecuencia media de los spikes de entrada durante un intervalo de tiempo Δt, será la relación entre la variación número de spikes contados, *ΔN* y dicho intervalo de tiempo *Δt*:

$$f_{mediaSpikesEntrada} = \frac{\Delta N}{\Delta t}$$

Ecuación [4-5]

Para obtener la frecuencia instantánea de la frecuencia de los spikes de entrada, hemos de suponer un *Δt* infinitésimamente pequeño, *dt*, transformando la ecuación anterior en:

$$f_{spikesEntrada} = \frac{dN}{dt}$$

Ecuación [4-6]

Despejando la variación del número de spikes contados, *N*:

$$dN = f_{spikesEntrada} dt \rightarrow N = \int_{t1}^{t2} f_{spikesEntrada} dt$$

Ecuación [4-7]

Combinando la última ecuación con la Ecuación [4-4] ,podemos obtener la frecuencia de los spikes de salida del Integrate & Generate en función de la ganancia del generador de spikes.

$$f_{Int\&Gen} = k_{freq\ GenBW} * \int_{t1}^{t2} f_{spikesEntrada} dt$$

Ecuación [4-8]

Finalmente, sustituyendo la ganancia del generador de spikes por la Ecuación [2-22], obtenemos la ecuación completa del comportamiento del Integrate & Generate. Como podemos observar, este bloque viene a comportarse como un integrador con una ganancia equivalente a la pendiente de la recta de modulación del generador exhaustivo bit-wise, mostrada con anterioridad en la Figura 2.15.

$$f_{Int\&Gen} = \frac{F_{CLK}}{2^{n-1}(intFreqDiv + 1)} \int_{0}^{t} f_{Entrada} dt$$

Ecuación [4-9]

Centrándonos en la aplicación de este elemento al control, podemos hacer una analogía a la transformada de Laplace, o transformada en S [Ogata04]. Para que una función, *f(t)*, pueda ser representada mediante la transformada de Laplace, debe cumplir dos condiciones:

a. $f(t) \in \mathbb{R}$ para t>0

b. $f(t)$ sea de orden exponencial, es decir, que esté acotada para que su integral exista y sea finita, en consecuencia deben existir un par de números reales positivos *M* y *a*, de tal manera que:

$$|f(t)| \leq M * e^{a*t}, \forall t \geq 0$$

Dado que las funciones que vamos a utilizar cumplen estas condiciones, podemos hacer una analogía entre la representación pulsante de la información y la transformada de Laplace, para sí poder calcular la función de transferencia del Integrate & Generate como la función de transferencia de un integrador continuo ideal. Resultando la siguiente función de transferencia equivalente:

$$F_{Int\&Gen}(s) = k_{freq\ GenBW} * \frac{F_{Entrada}(s)}{s} \rightarrow \frac{F_{Int\&Gen}(s)}{F_{Entrada}(s)} = \frac{k_{freq\ GenBW}}{s}$$

Ecuación [4-10]

En la Figura 4.2 mostramos la arquitectura interna del componente VHDL que implementa este modelo, a la izquierda las entradas de los spikes, en el centro el contador que integrará los spikes, y a la derecha finalmente el generador sintético de spikes. El integrador es un contador con cuenta ascendente / descendente y saturación. En el que un spike positivo, incrementará su cuenta en uno, y lo se decrementará en una unidad para spikes negativos.

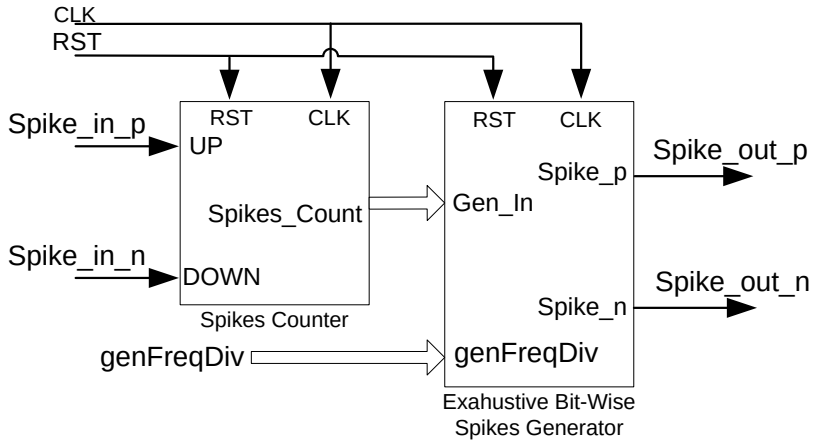


Figura 4.2: Diagrama de bloques del Integrate & Generate

Como el Integrate & Generate tiene un consumo de hardware variable, debido a que el número de bits del generador puede cambiar junto con el ancho en bits del contador de spikes, para poder observar la evolución del consumo hardware, vamos sintetizar el Integrate & Generate para diferentes anchos de bits. Mostrando en la Tabla 4-2 el número de slices que requiere la implementación de este modelo en una FPGA en cada caso. En ella queda constatado cómo el consumo hardware se va incrementando en concordancia con el número de bits, además los circuitos resultantes son más rápidos cuantos menos bits tienen gracias al menor consumo de hardware.

Tabla 4-2: Resumen de la síntesis de la implementación del modelo Integrate & Generate para diversos números de bits

Número de bits	Slices	Máxima Frecuencia
6	32	166.24 MHz
8	36	153.45 MHz
10	42	138.53 MHz
12	51	130.29 MHz
14	53	122.17 MHz
16	54	128.52 MHz

4.2. Simulación del modelo Integrate & Generate

Para verificar el comportamiento de nuestra implementación del modelo Integrate & Generate, vamos a simularlo junto a un generador de spikes. Analizando su salida para diversas señales de excitación. Al igual que hemos actuado a lo largo de los capítulos anteriores, vamos a diseñar un escenario de simulación en el que vamos a situar un generador exhaustivo bit-wise de spikes a la entrada del Integrate & Generate, monitorizando y analizando su respuesta. La Figura 4.3 muestra el escenario de simulación.

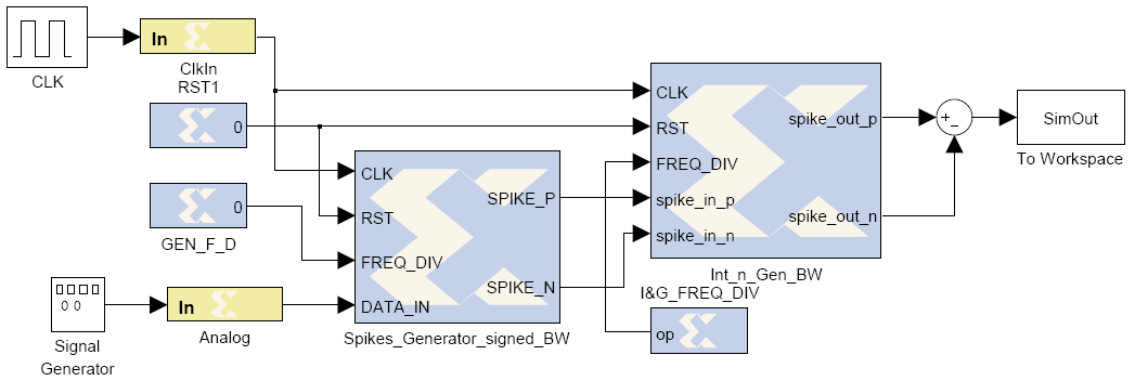


Figura 4.3: Escenario de simulación del Integrate & Generate

En primer lugar vamos a proceder a excitar al Integrate & Generate con una frecuencia de spikes de entrada constante, es decir, una entrada en escalón. Esperando obtener una respuesta en rampa, con una pendiente equivalente a la ganancia del integrador multiplicado por la frecuencia constante de los spikes de entrada.

$$f_{Int\&Gen}(t) = k_{freqGenBW} * f_{in}(cte) * t$$

Ecuación [4-11]

La Figura 4.4 muestra los resultados de la simulación ante una entrada en escalón. Hemos usado un Integrate & Generate de 9 bits, con un divisor de frecuencia de 12 ciclos de reloj, y una señal de reloj de 50MHz, obteniendo así una ganancia del integrador de aproximadamente $1.5024 * 10^4$. Aplicando una señal a la entrada de 505.2kSpikes/Sec. En la parte superior de la figura se muestran la señal de excitación (azul), la integral teórica de dicha señal (verde) y la reconstrucción de la salida del Integrate & Generate (rojo). En la parte inferior de la figura mostramos los spikes del sistema, la entrada (azul), y la salida (verde) del Integrate & Generate.

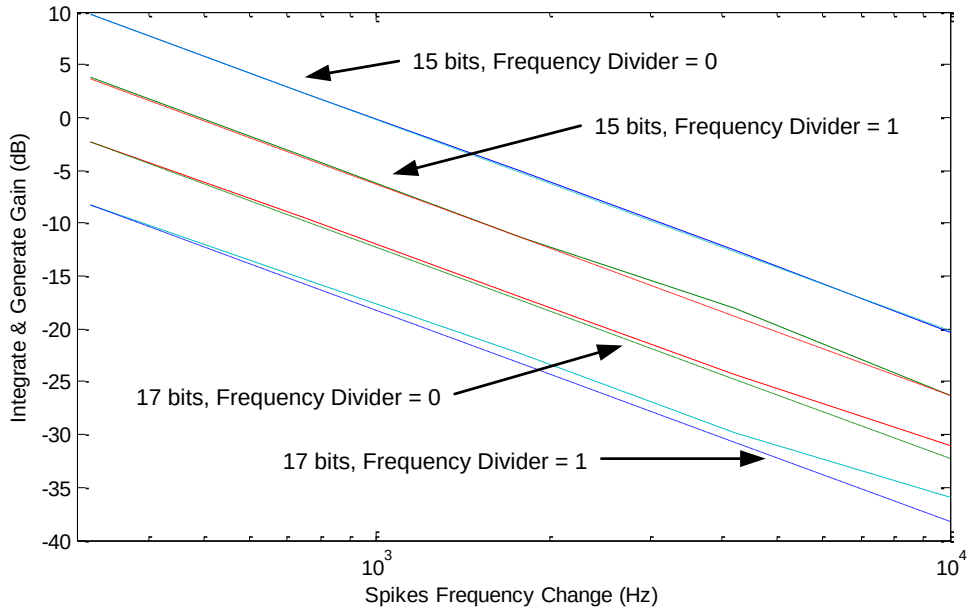


Figura 4.4: Respuesta del Integrate & Generate ante una entrada en escalón

Para comprobar el efecto del divisor de frecuencia en el comportamiento de Integrate & Generate, así como para verificar la veracidad de las ecuaciones teóricas, vamos a repetir la simulación anterior, pero variando el divisor de frecuencia del Integrate & Generate. Es decir, le vamos a proporcionar una frecuencia constante a la entrada, en este caso de 610.3kSpikes/Sec, y vamos observar su comportamiento para diversas ganancias del Integrate & Generate. En la Figura 4.5 se pueden observar los resultados de la simulación, el escalón en frecuencia de entrada se encuentra en la parte inferior de la figura en azul. A continuación, observamos la reconstrucción de la salida del Integrate & Generate para los diversos valores del divisor de frecuencia, en diversos colores y con una línea sólida, así como la integral teórica de la señal de entrada de forma discontinua. La figura muestra cómo el Integrate & Generate se comporta como un integrador, al que le podemos ajustar la ganancia del Integrate & Generate según la Ecuación [4-8] modificando el valor del divisor de frecuencia. De esta forma la pendiente de las salidas de Integrate & Generate son aproximadas a la ganancia de Integrate & Generate, acentuándose la pendiente según decrece el divisor de frecuencia, creciendo en consecuencia la ganancia del Integrate & Generate. En la Tabla 4-3 mostramos los divisores de frecuencia usados, así como la ganancia equivalente para un Integrate & Generate de 9 bits, operando con un reloj a 50MHz.

Tabla 4-3: Ganancia equivalente del Integrate & Generate para diversos divisores de frecuencia

Valor del divisor de frecuencia	Ganancia equivalente del Integrate & Generate
10	$3.5511 * 10^4$
15	$2.4414* 10^4$
20	$1.8601* 10^4$
25	$1.5024 * 10^4$
30	$1.2601* 10^4$

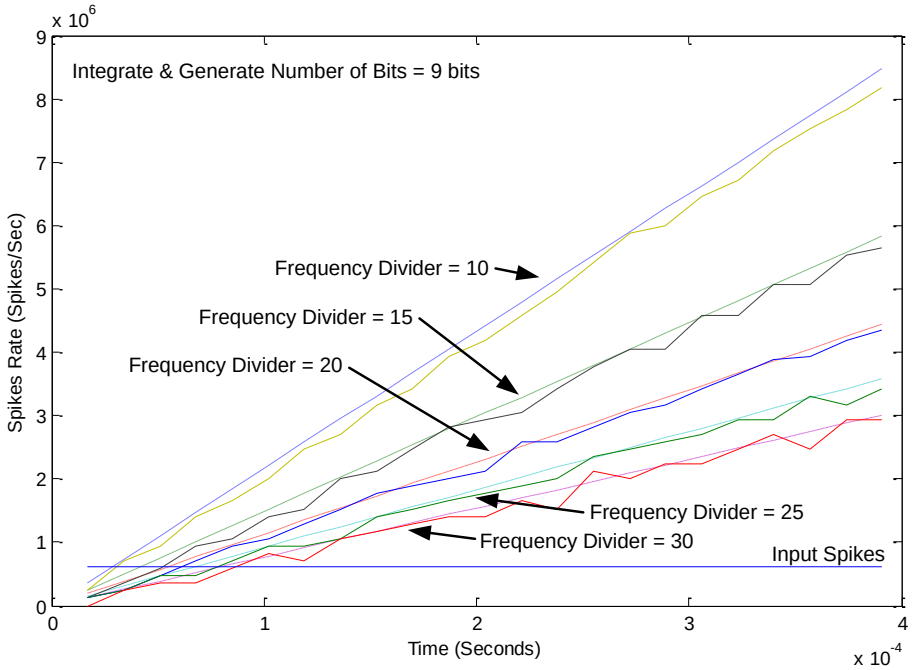


Figura 4.5: Respuesta del Integrate & Generate ante una entrada en escalón para diversos divisores de frecuencia, junto con la respuesta teórica

De forma análoga vamos a observar el comportamiento del Integrate & Generate ante diversos números de bits. Para ello vamos a realizar una nueva secuencia de simulaciones en las que vamos a dejar fijo el valor del divisor de frecuencia, 5 en este caso, y vamos a modificar el número de bits del Integrate & Generate entre 8 y 11 bits; y lo excitamos con una secuencia de spikes con frecuencia fija, 305.2kSpikes/Sec. Los resultandos son mostrados en la Figura 4.6. Al igual que en la anterior figura, se muestra la frecuencia de los spikes de entrada, abajo en azul, así como la reconstrucción de los spikes de salida de Integrate & Generate, en diversos colores y sólidos, y la integral teórica de la entrada, en trazos discontinuos. De nuevo, la figura muestra cómo el Integrate & Generate se comporta como un integrador, y que podemos

ajustar la ganancia del Integrate & Generate según la Ecuación [4-8] modificando el número de bits del generador. En la Tabla 4-4 se muestran las ganancias equivalentes del Integrate & Generate, a 50MHz, con un divisor de frecuencia de 5, y para diversos números de bits.

Tabla 4-4: Ganancia equivalente del Integrate & Generate para diversos números de bits

Número de bits	Ganancia equivalente del Integrate & Generate
8	$1.3021 * 10^5$
9	$0.6510 * 10^5$
10	$0.3255 * 10^5$
11	$0.0814 * 10^5$

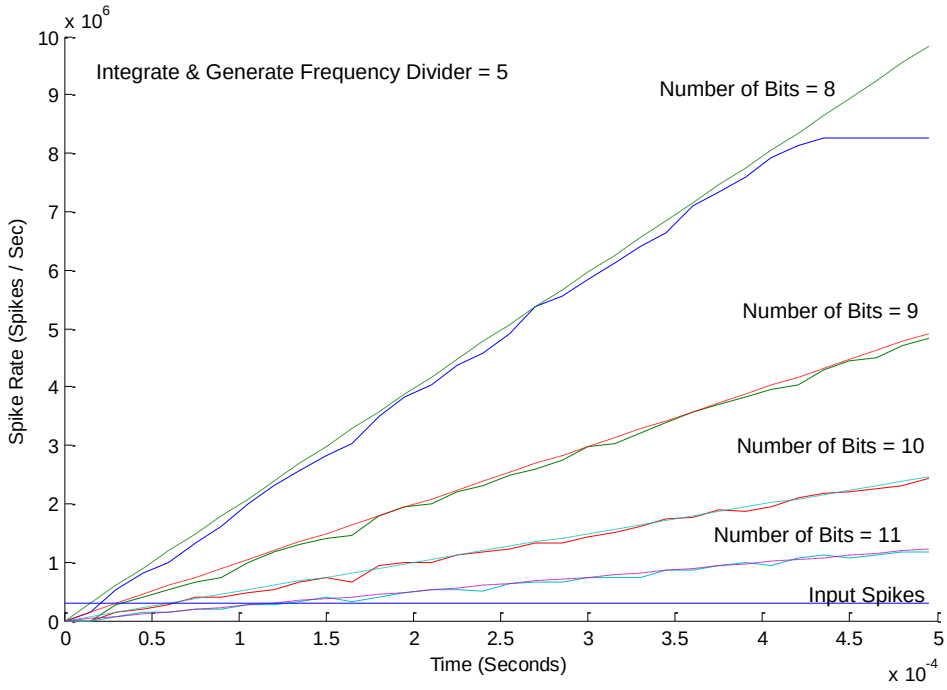


Figura 4.6: Respuesta del Integrate & Generate ante una entrada en escalón para diversos números de bits del integrador, junto con la respuesta teórica

Además en la figura anterior se puede observar cómo la respuesta de mayor ganancia, en azul se ha llegado a saturar. El problema de la saturación es típico de los integradores no ideales. En nuestro caso alcanzamos el estado de saturación cuando el integrador de spikes llega a su máximo nivel, 2^{n-1}, es decir, cuando se han contado 2^{n-1} spikes seguidos de un mismo signo a la entrada del Integrate & Generate. Dado que la frecuencia máxima de salida del Integrate & Generate será:

$$f_{Int\&GenMax} = k_{Int\&Gen} * MaxIntegratedValue$$

Ecuación [4-12]

Siendo el máximo valor de integrador:

$$MaxIntegratedValue = 2^{n-1}$$

Ecuación [4-13]

Y conocida la ganancia del Integrate & Generate expuesta en la Ecuación [2-23], podemos calcular la frecuencia máxima de salida del Integra & Generate según la siguiente ecuación:

$$f_{Int\&GenMax} = \frac{F_{CLK}}{2^{n-1}(intFreqDiv + 1)} 2^{n-1} = \frac{F_{CLK}}{(intFreqDiv + 1)}$$

Ecuación [4-14]

Según sugieren las ecuaciones teóricas, la frecuencia de saturación del Integrate & Generate no depende del número de bits que utilicemos, sino del divisor de frecuencia asignado.

A continuación vamos tratar de verificar la dependencia de la frecuencia máxima de salida del Integrate & Generate con el divisor de frecuencia. Para ello hemos diseñado un escenario de simulación en el que vamos a saturar el Integrate & Generate con diversos divisores de frecuencia, usando frecuencias de entrada suficientemente altas para saturar su salida, y anotando la frecuencia máxima en cada caso. Los resultados son mostrados en la Figura 4.7, en ella podemos encontrar la frecuencia máxima obtenida de las simulaciones en azul, así como la frecuencia máxima teórica del Integrate & Generate según la ecuación anterior, en triángulos rojos. En ella queda constatada la coincidencia de ambas funciones, verificándose así la Ecuación [4-14].

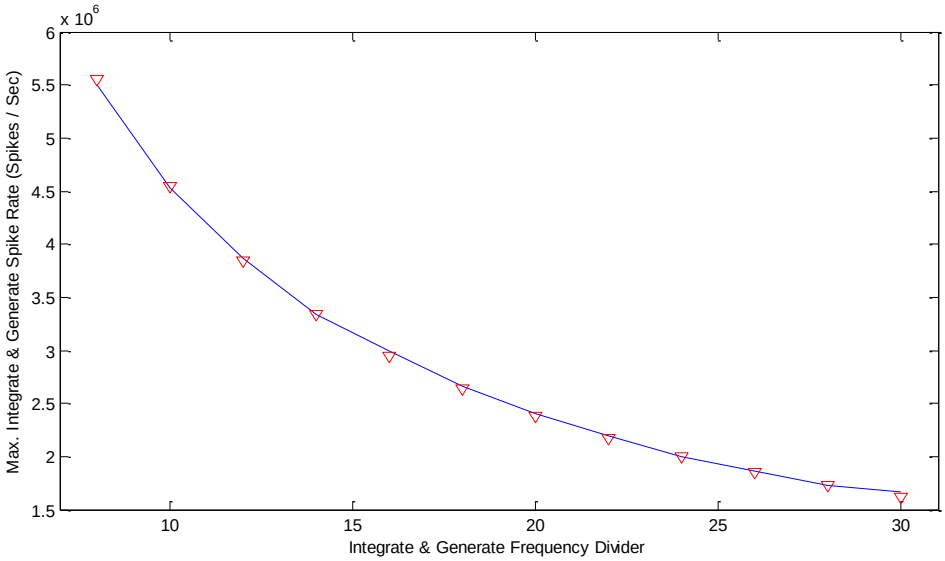


Figura 4.7: Máxima frecuencia de salida del Integrate & Generate para diversos divisores de frecuencia

Finalmente hemos constatado el efecto de "deriva" al que puede llevarnos la saturación del Integrate & Generate, como cualquier otro integrador no ideal con saturación.

Este efecto surge cuando al saturarse el integrador es incapaz de contar más spikes, de tal forma que el contador queda fijo a un valor, dejando de representar la integral de la señal en el tiempo. El problema surge cuando la señal de entrada va variando en el tiempo, ya que la salida del Integrate & Generate dejará de ser fiel a la integral teórica ideal de la señal, introduciéndose una componente continua, u off-set, en la salida del Integrate & Generate. La Figura 4.8 muestra este efecto, en azul encontramos la frecuencia de los spikes de entrada, en verde su integral ideal, y en rojo la salida del Integrate & Generate. Si nos fijamos en la salida de Integrate & Generate, se asemeja perfectamente a la integral ideal al principio, hasta que se satura, quedándose anclada a la frecuencia máxima. Sin embargo, cuando la señal se vuelve negativa, ambas integrales comienzan a disminuir, pero, al no poder alcanzar un valor tan alto el Integrate & Generate, se introduce una diferencia continua entre ambas, o valor de off-set.

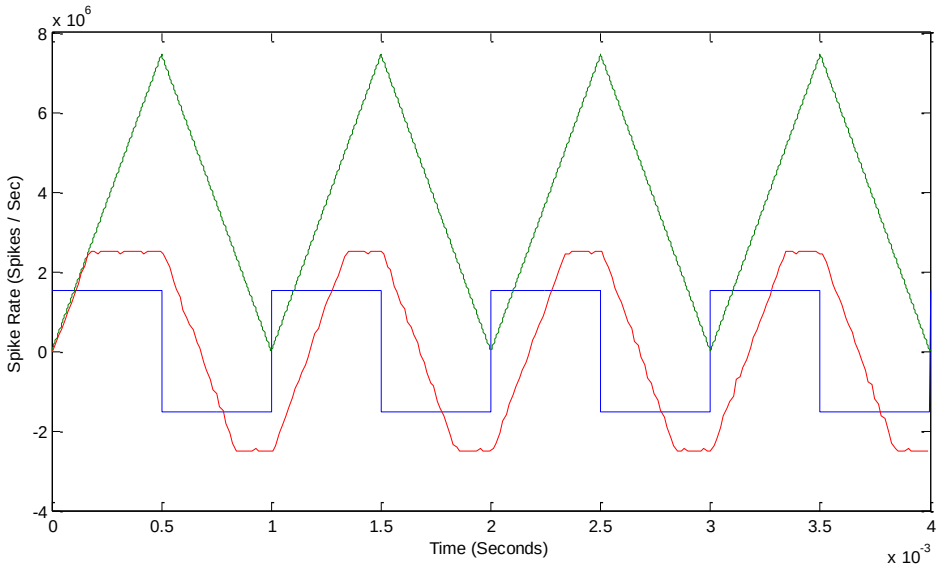


Figura 4.8: Saturación del Integrate & Generate

En resumen, teniendo en cuenta la no linealidad del Integrate & Generate introducida por el efecto de la saturación, podemos formalizar el comportamiento del Integrate & Generate según la siguiente ecuación:

$$F_{IG}\big(In(s)\big) = \begin{cases} \dfrac{k_{Int\&Gen}}{s} * In(s) & si\ F_{IG} \leq k_{Int\&Gen} * 2^{n-1} \\ \dfrac{F_{CLK}}{(intFreqDiv + 1)} & En\ Otro\ Caso \end{cases}$$

Ecuación [4-15]

Con el objetivo de comprobar su comportamiento en el dominio de la frecuencia, hemos excitado el Integrate & Generate con señales sinusoidales de diversas frecuencias, realizando un barrido frecuencial. Anotando para cada frecuencia la potencia de la salida del Integrate & Generate con diversos parámetros, obteniendo así el diagrama de Bode del Integrate & Generate para cada par de parámetros. Los resultados son mostrados en la Figura 4.9, en la que aparecen en trazo continuo la amplitud de la salida del Integrate & Generate, y en trazo discontinuo la salida teórica de un integrador ideal en las mismas condiciones. En el diagrama de Bode se puede observar como la salida del Integrate & Generate es muy parecida

a la salida ideal, presentando una atenuación de 20dB por década. Además varía el valor de su componente continua, u off-set, en virtud de la ganancia equivalente del Integrate & Generate según los parámetros usados.

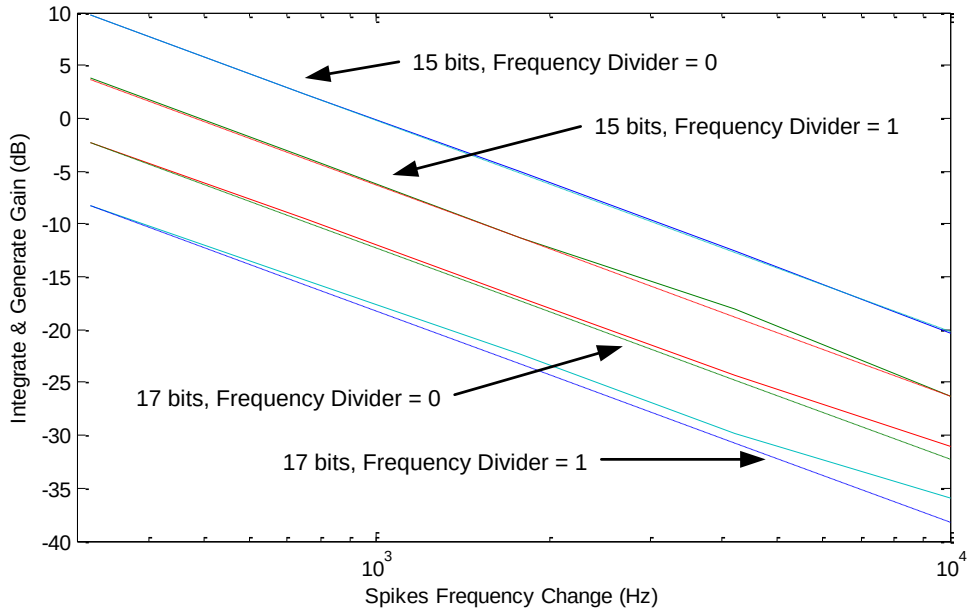


Figura 4.9: Diagrama de Bode del Integrate & Generate

Para concluir el estudio del Integrate & Generate, lo hemos excitado con otras funciones de entrada, distintas a las entradas en escalón usadas anteriormente. Estas funciones han sido una rampa (o diente de sierra), en la Figura 4.10, y una entrada en seno, en la Figura 4.11. En ambas encontramos en la parte superior la frecuencia de los spikes de entrada, azul, la integral ideal de la entrada, verde, y la frecuencia de salida del Integrate & Generate, en rojo. Y en la parte inferior los spikes de entrada, azul, y los de salida del Integrate & Generate, verde. En ambas figuras se puede observar que el Integrate & Generate realiza una integración de los spikes de entrada, generando secuencias de spikes cuya frecuencia es muy próxima a la esperada teóricamente.

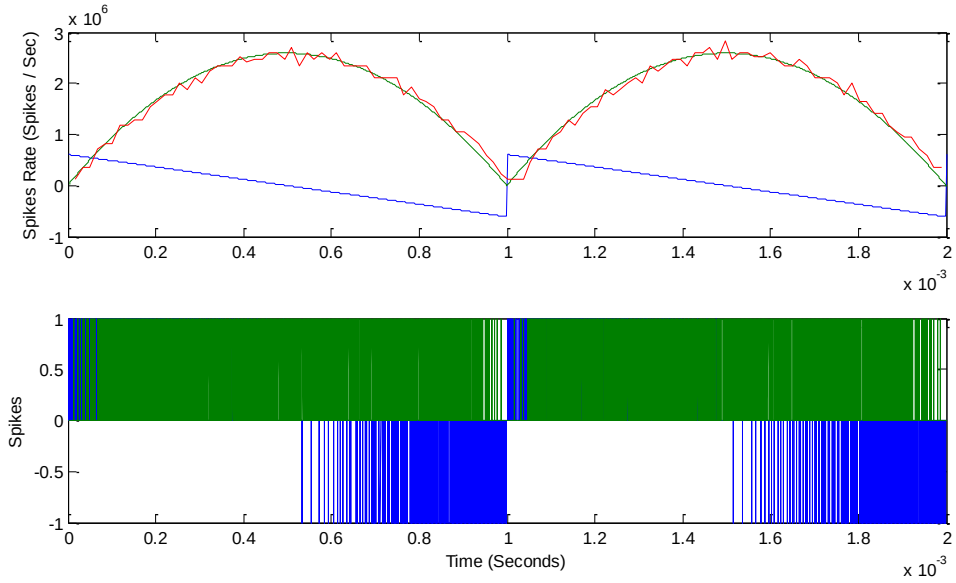


Figura 4.10: Respuesta del Integrate & Generate ante una entrada en sierra

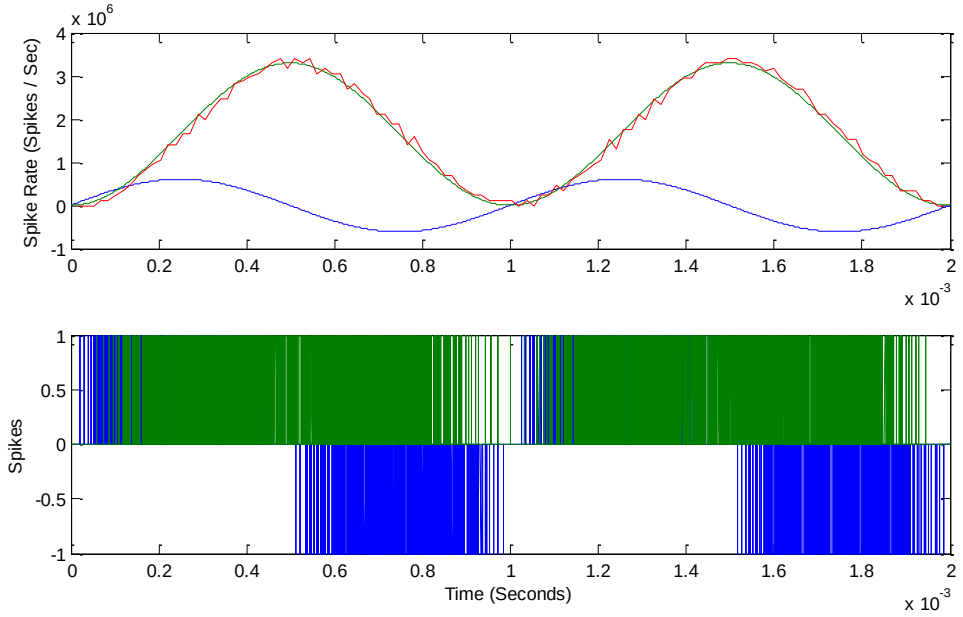


Figura 4.11: Respuesta del Integrate & Generate ante una entrada en seno

4.3. Realimentando el Integrate & Generate: el derivador de spikes

La idea del derivador de spikes, es diseñar un elemento o bloque, cuya entrada sea una señal de spikes, y su salida una secuencia de spikes con una frecuencia proporcional a la derivada de los cambios de la frecuencia de los spikes de entrada. Tal y como expusimos con anterioridad:

$$f_{spikesDerivative} = k_{derivative}\frac{df_{spikesIn}}{dt} \rightarrow \frac{F_{spikesDerivative}(s)}{F_{spikesIn}(s)} = k_{derivative} * s$$

Ecuación [4-16]

Continuando con la analogía de la transformada de Laplace, ¿por qué no realimentar el Integrate & Generate con el Hold & Fire para diseñar nuevos elementos con diferentes funciones de transferencia? Uno de estos nuevos elementos es el derivador de pulsos, y más adelante expondremos otros elementos, como son los filtros paso de baja, alta y banda de señales de spikes.

La Figura 4.12 muestra la topología de la conexión el Hold & Fire (H&F) y el Integrate & Generate (I&G) para diseñar el derivador de spikes. El derivador de spikes es en esencia un filtro paso de alta. Hemos realimentado los spikes de entrada con un Integrate & Generate (I&G), siendo la diferencia entre los spikes de entrada y los integrados la salida del derivador.

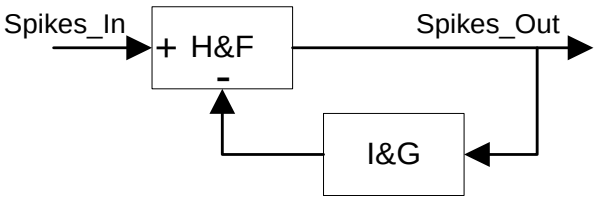


Figura 4.12: Diagrama de bloques del derivador de spikes

Aplicando la teoría básica de sistemas, podemos calcular el sistema equivalente en el domino S compuesto por esta configuración topológica del Hold & Fire y el Integrate & Generate:

$$\frac{F_{spikesOut}(s)}{F_{spikesIn}(s)} = \frac{1}{1 + I\&G(S)} = \frac{1}{1 + \frac{k_{Int\,\&Gen}}{s}} = \frac{s}{s + k_{Int\,\&Gen}}$$

Ecuación [4-17]

Conocida la constante del generador exhaustivo bit-wise integrado en el Integrate & Generate, podemos calcular la función de transferencia como:

$$\frac{F_{spikesDerivative}(s)}{F_{spikesIn}(s)} = \frac{s}{s + \frac{F_{CLK}}{2^{n-1}(intFreqDiv + 1)}}$$

Ecuación [4-18]

Dicha función de transferencia representa a un derivador, con un polo en una frecuencia equivalente a la ganancia del generador (en radianes por segundo). Teniendo una ganancia equivalente:

$$k_{spikesDerivative} = \frac{1}{\frac{F_{CLK}}{2^{n-1}(intFreqDiv + 1)}} = \frac{2^{n-1}(intFreqDiv + 1)}{F_{CLK}}$$

Ecuación [4-19]

A continuación hemos procedido a sintetizar en hardware el derivador de spikes con el fin de realizar una estimación del hardware necesario en el interior de una FPGA para su implementación. Dado que contiene en su interior un Integrate & Generate, la cantidad de hardware necesario, medido en slices, variará en base al número de bits del Integrate & Generate. Los resultados son mostrados en la Tabla 4-5, en ella se aprecia cómo el consumo

hardware aumenta según va aumentando el número de bits, así como se reduce la máxima frecuencia de operación, ya que al aparecer más hardware los "caminos" asíncronos son más largos, consumiendo más tiempo, y alcanzando una frecuencia de reloj menor.

Tabla 4-5: Resumen de la síntesis de la implementación del derivador de spikes para diversos números de bits

Número de bits	Slices	Máxima Frecuencia
6	43	162.41 MHz
8	48	152.31 MHz
10	53	136.87 MHz
12	64	128.45 MHz
14	65	121.61 MHz
16	67	119.52 MHz

4.4. Simulación del derivador de spikes

A continuación, tal y como hemos venido haciendo a lo largo de este trabajo, vamos a construir un escenario de simulación para comprobar el comportamiento del derivador de spikes, mostrado en la Figura 4.13. En este nuevo escenario de simulación vamos a conectar un generador exhaustivo bit-wise de spikes, a la izquierda, con la entrada de un nuevo bloque (*SpikesDerivative*), en el centro. Este último implementa en su interior el derivador de spikes y está compuesto internamente por un Hold & Fire junto con un Integrate & Generate conectados según la Figura 4.12. A continuación vamos a proceder a estimularlo con diferentes señales y parámetros, analizando su respuesta y contrastando los resultados obtenidos con las predicciones teóricas.

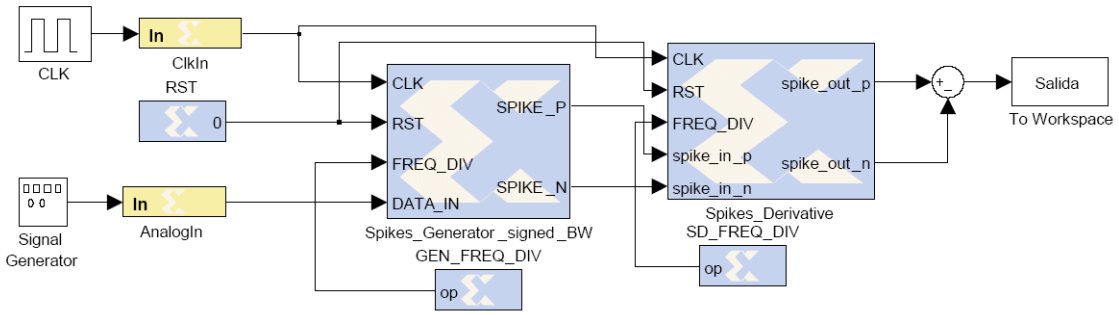


Figura 4.13: Escenario de simulación del derivador de spikes

En primer lugar vamos a excitar al derivador de spikes con una frecuencia de spikes de entrada creciente a ritmo constante, es decir, una entrada en rampa. Esperando obtener una

respuesta en escalón, spikes con frecuencia constate, con una frecuencia equivalente a la ganancia del integrador multiplicado por el incremento de la frecuencia de los spikes de entrada, modificada por el efecto del polo.

$$f_{SpikesDerivate}(t) = k_{spikesDerivative}\frac{d(k_{freqInput} * t)}{dt} = k_{spikesDerivative} * k_{freqInput}$$

Ecuación [4-20]

La Figura 4.14 muestra los resultados de la simulación del derivador de spikes ante una entrada en rampa. En la parte superior de la figura se muestran la señal de excitación (azul), la derivada, con el polo, teórica de dicha señal (verde) y la reconstrucción de la salida del derivador de spikes (rojo). En la parte inferior de la figura mostramos los spikes del sistema, la entrada (azul), y la salida (verde) del derivador de spikes. En la figura se ve como se ajusta la reconstrucción de la salida del derivador de spikes a la salida teórica esperada. Hemos usado un derivador de spikes de 10 bits, con un divisor de frecuencia de 20 ciclos de reloj, y una señal de reloj de 50MHz, obteniendo así una ganancia del integrador de aproximadamente $2.15*10^{-4}$ .Aplicando una señal a la entrada de ±3MSpikes/Sec.

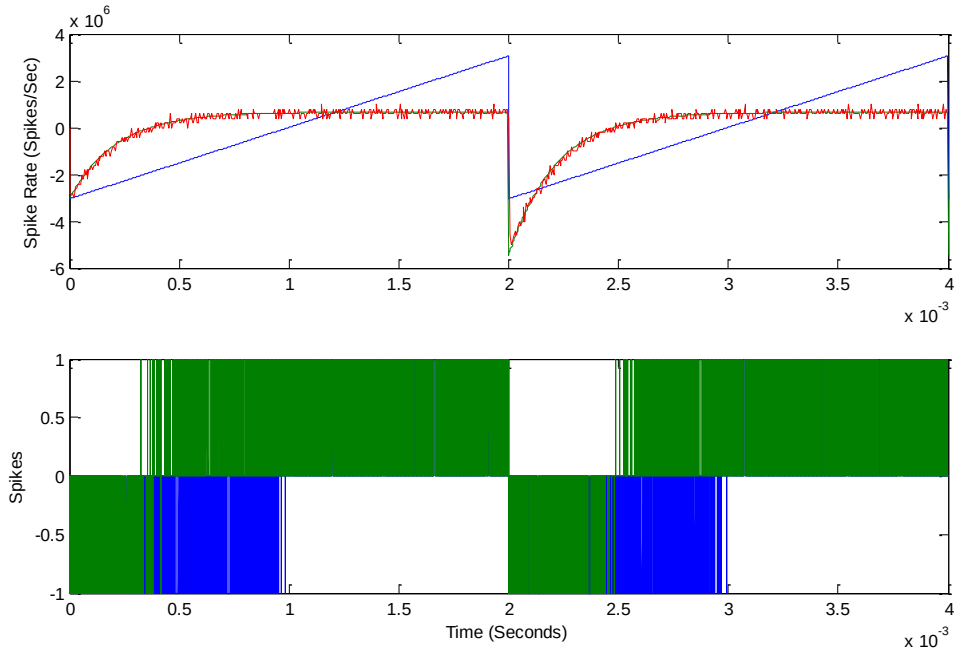


Figura 4.14: Respuesta del derivador de spikes ante una entrada en rampa

Con el fin de poder analizar el efecto del divisor de frecuencia en la respuesta del divisor de spikes, así como para contrastar las ecuaciones teóricas, hemos realizado una secuencia de simulaciones en las que fijaremos a 13 el número de bits del derivador de spikes, y a continuación hemos incrementado secuencialmente el valor del divisor de frecuencia (desde 0 hasta 10). En la Figura 4.15 se pueden observar los resultados de la simulación, la rampa en frecuencia de entrada se encuentra en la parte inferior de la figura en azul, y a continuación se muestra la reconstrucción de la salida del derivador de spikes para los diversos valores del divisor de frecuencia, en diversos colores y con una línea sólida, y la derivada (con

su polo) teórica de la señal de entrada de forma discontinua. La figura muestra que el derivador de spikes se comporta como un derivador con cierta dinámica debido al polo. Así como que podemos ajustar la ganancia del derivador de spikes, junto con la frecuencia del polo, según las ecuaciones [4 - 18] Y [4 - 19] modificando el valor del divisor de frecuencia. En la Tabla 4-6 mostramos los divisores de frecuencia usados, y la ganancia equivalente para un derivador de spikes de 13 bits, operando con un reloj a 50MHz.

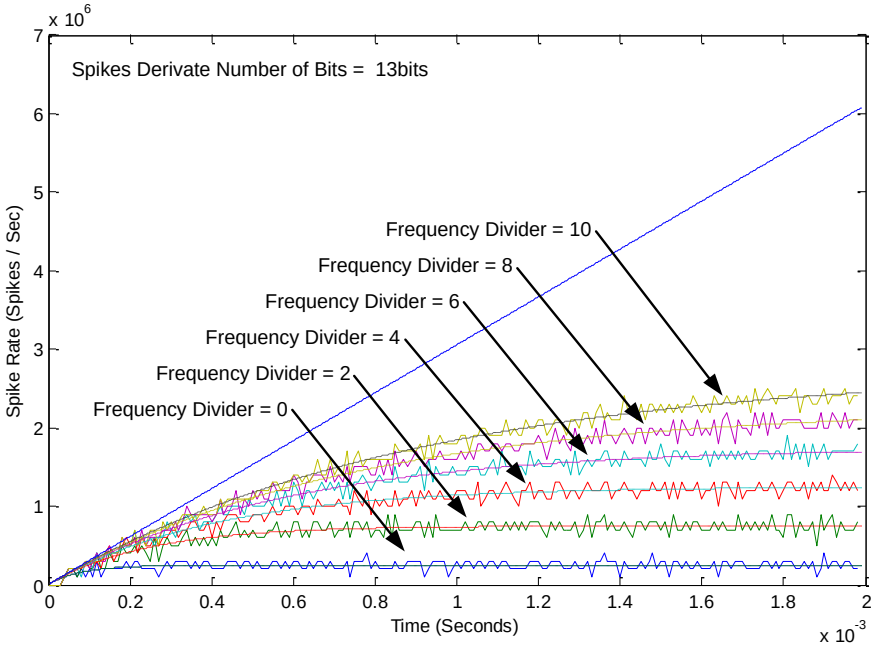


Figura 4.15: Respuesta del derivador de spikes ante una entrada en rampa para diversos divisores de frecuencia, junto con la respuesta teórica

Tabla 4-6: Ganancia equivalente del derivador de spikes para diversos divisores de frecuencia

Valor del divisor de frecuencia	Ganancia equivalente del derivador de spikes	Frecuencia del polo del derivador de spikes (Rad/Sec)
0	$0.0819 * 10^{-3}$	$1.2207 * 10^{4}$
2	$0.2458 * 10^{-3}$	$0.4069* 10^{4}$
4	$0.4096 * 10^{-3}$	$0.2441 * 10^{4}$
6	$0.5734 * 10^{-3}$	$0.1744 * 10^{4}$
8	$0.7373 * 10^{-3}$	$0.1356 * 10^{4}$
10	$0.9011 * 10^{-3}$	$0.1110 * 10^{4}$

El otro parámetro implicado en la respuesta del derivador de spikes es el número de bits del Integrate & Generate que incluye en su interior. Así que vamos a simular su respuesta para la misma señal de estímulo anterior, señal en rampa, fijando el divisor de frecuencia a un valor fijo de 5, y modificando el número de bits del derivador de spikes, dese 8 hasta 14 bits. Los resultados pueden observarse en la Figura 4.16. En ella mostramos la señal de entrada, en azul, la salida del derivador de spikes, en diversos colores con trazo continuo, así como la salida teórica de este componente, de nuevo en diversos colores con trazos discontinuos. Estas nuevas simulaciones constatan que el derivador de spikes se comporta como un derivador con cierta dinámica debido al polo. También podemos ajustar la ganancia del derivador de spikes, junto con la frecuencia del polo, según la Ecuación [4-17] modificando el valor el número de bits del Integrate & Generate que lo compone. En la Tabla 4-7 mostramos los números de bits usados, así como las ganancias equivalentes para un derivador de spikes con el divisor de frecuencia fijado a 5 y operando a 50MHz.

Tabla 4-7: Ganancia equivalente del derivador de spikes para diversos números de bits

Número de bits	Ganancia equivalente del derivador de spikes	Frecuencia del polo del derivador de spikes (Rad/Sec)
8	$0.0154 * 10^{-3}$	$6.5104 * 10^{4}$
10	$0.0614 * 10^{-3}$	$1.6276 * 10^{4}$
12	$0.2458 * 10^{-3}$	$0.4069 * 10^{4}$
14	$0.9830 * 10^{-3}$	$0.1017 * 10^{4}$

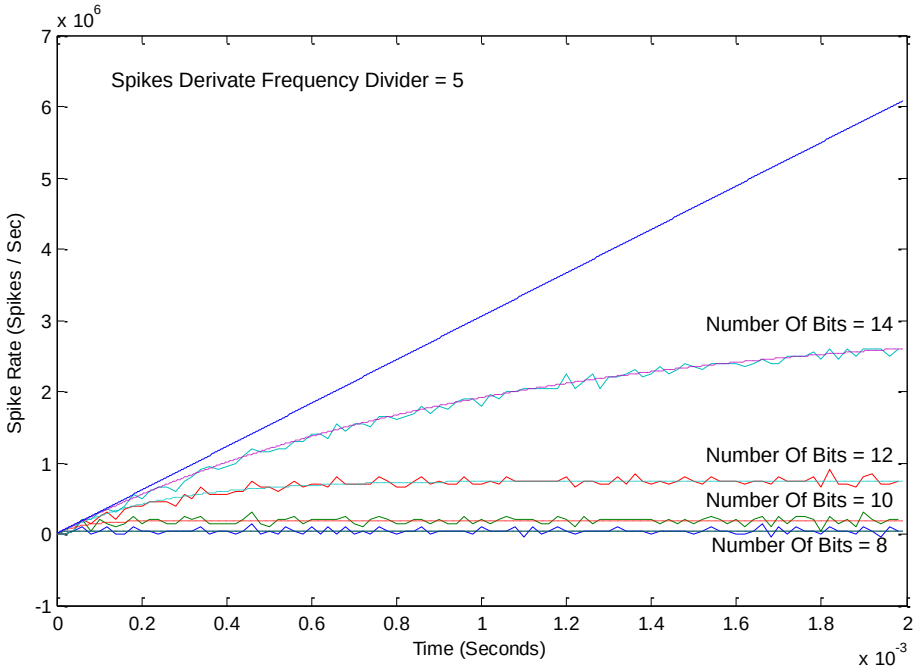


Figura 4.16: Respuesta del derivador de spikes ante una entrada en rampa para diversos números de bits, junto con la respuesta teórica

Dado que el derivador de spikes está basado en un Integrate & Generate, y como expusimos al final del apartado anterior, este componente puede llegar a saturarse quedándose su salida a una frecuencia máxima fija, calculable según la Ecuación [4-13], y sólo dependiente del divisor de frecuencia. Inmediatamente se nos plantea una cuestión acerca del comportamiento del derivador de spikes cuando el Integrate & Generate que integra se satura. Pensando en la topología del derivador de spikes, en la Figura 4.2, la salida del derivador son los spikes de entrada menos los spikes integrados por el Integrate & Generate, en otras palabras, este componente basa su funcionamiento en restarle a su entrada la integral de la diferencia de frecuencia entre ambas señales (entrada e integral). El problema surge cuando la diferencia entre la entrada y la salida teórica del derivador es superior a la frecuencia de saturación del Integrate & Generate. En ese momento el Integrate & Generate se quedará fijado a su máxima frecuencia, siendo incapaz de seguir derivando la señal de entrada, y comportándose como restador continuo (restador de off-set). La siguiente ecuación trata de formalizar este efecto:

$$F_{SD}\big(In(s)\big) = \begin{Bmatrix} \dfrac{In(s) * s}{s + k_{spikesDerivate}} & si\ |F_{SD} - In| \leq F_{MaxIG} \\ In(s) - F_{MaxIG}(s) & En\ Otro\ Caso \end{Bmatrix}$$

Ecuación [4-21]

Para comprobar este efecto hemos realizado una serie de simulaciones en las que hemos forzado la saturación del derivador de spikes. Para ello le hemos excitado con una señal en rampa (frecuencia de los spikes de entrada creciente) con una frecuencia y pendiente suficientemente elevada, así como para diversos divisores de spikes. En la Figura 4.17 se muestran los resultados de las simulaciones, en azul encontramos la señal de entrada, y en diversos colores la salida del derivador de spikes, trazos sólidos, y la salida ideal, en discontinuo. En la figura llama la atención que en todas las respuestas, a partir de cierto punto, siguen de la entrada, pero sustrayéndole la frecuencia de saturación del Integrate & Generate interno del derivador de spikes. Además este punto se va adelantando en el tiempo a la vez que se incrementa el divisor de frecuencia, ya que al decrementarse la frecuencia de saturación del Integrate & Generate, la diferencia entre la entrada y la salida del derivador supera a la de saturación del Integrate & Generate

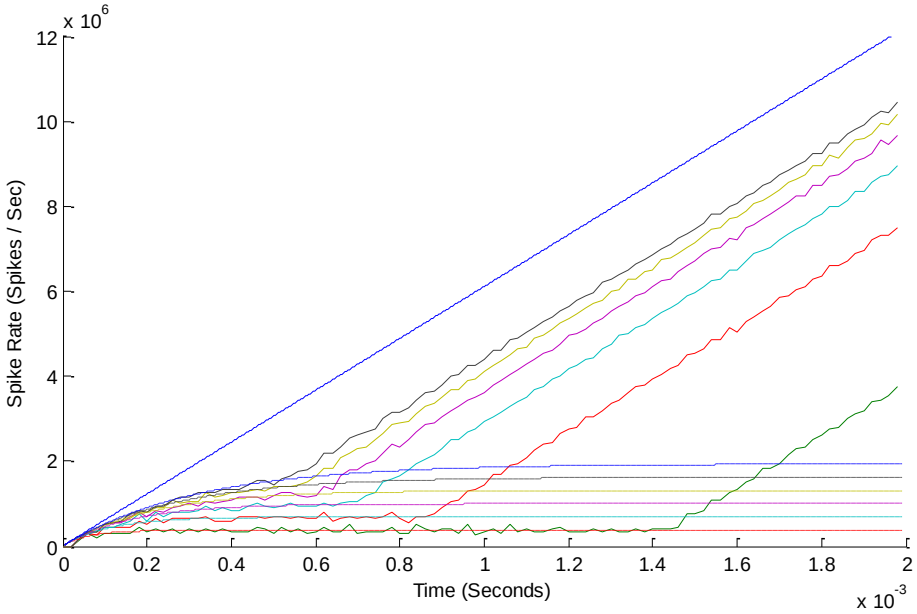


Figura 4.17: Respuesta del derivador de spikes en estado de saturación

Dado que en el caso de saturación del Integrate & Generate, su salida dejará de ser la integral de la entrada, introduciendo los factores de deriva, tal y como se mostró en la Figura 4.8, nos surge inmediatamente una cuestión acerca de si se observarán los mismos efectos en el derivador de spikes. Para responder a esta cuestión hemos simulado el derivador de spikes ante una entrada en forma de "diente de sierra", forzando su saturación un divisor de frecuencia muy elevado, concretamente 40. En la Figura 4.18 podemos ver los resultados de la simulación, en azul la señal de entrada, en rojo la reconstrucción del derivador de spikes, y en verde la salida ideal. En ella se observa cómo al alcanzar cierto punto el derivador de spikes se satura, dejando pasar los spikes de la entrada directamente, tal y como expusimos con anterioridad. Sin embargo, cuando la entrada cambia bruscamente de valor (y en este caso hasta de signo), el derivador vuelve a operar con normalidad, ofreciendo una salida similar a la derivada hasta que vuelve a saturarse. Sin embargo, al igual que en el caso de Integrate & Generate, la salida de derivador de spikes ya no sigue fielmente a la salida ideal, introduciéndose un error de deriva en la salida de este componente.

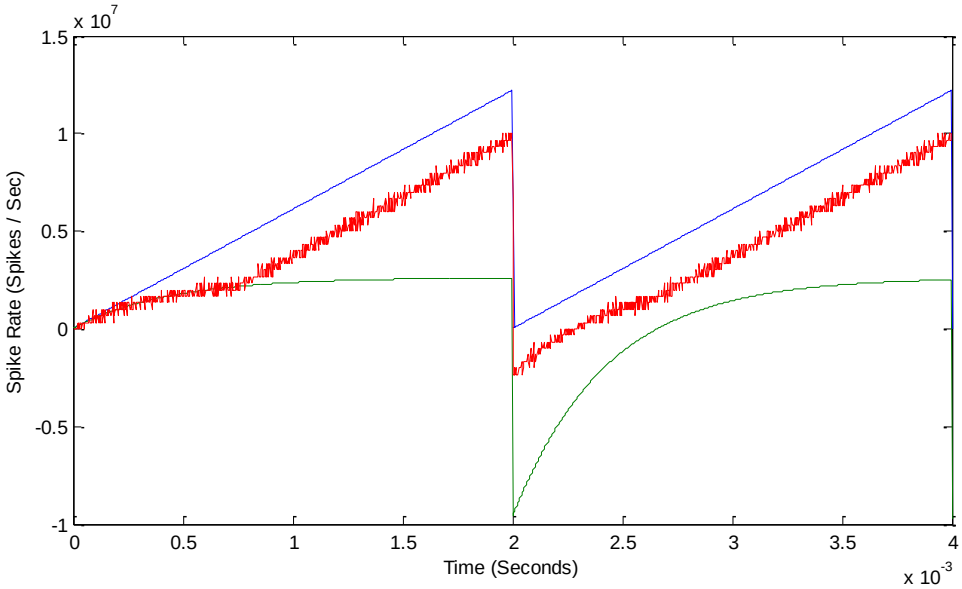


Figura 4.18: Efectos de la saturación el derivador de spikes

De nuevo al igual que hicimos en apartado dedicado al Integrate & Generate, vamos a verificar el comportamiento frecuencial del derivador de spikes. Para ello vamos a excitarlo con funciones senoidales de diversas frecuencias, así como para diversos pares de parámetros. Anotando al potencia de la señal de salida para cada frecuencia de excitación. La Figura 4.19 muestra los resultados de las simulaciones del barrido de frecuencia, en ella representamos el comportamiento frecuencial del derivador de spikes extraído de las simulaciones, en línea continua, junto al comportamiento esperado de un derivador ideal en similares circunstancias. En ella se puede ver que el derivador de spikes se comporta en el plano frecuencial como un filtro paso de alta que hará las veces de derivador. La respuesta del derivador de spikes se caracteriza por tener una amplificación creciente de 20dB por década, hasta que llega a la frecuencia del polo, donde se estabiliza a 0dB. Además la ganancia y frecuencia de paso del derivador de spikes es alterada con los parámetros elegidos para las simulaciones. De nuevo queda constancia de lo próximo que está el comportamiento de nuestro derivador de spikes al comportamiento teórico esperado.

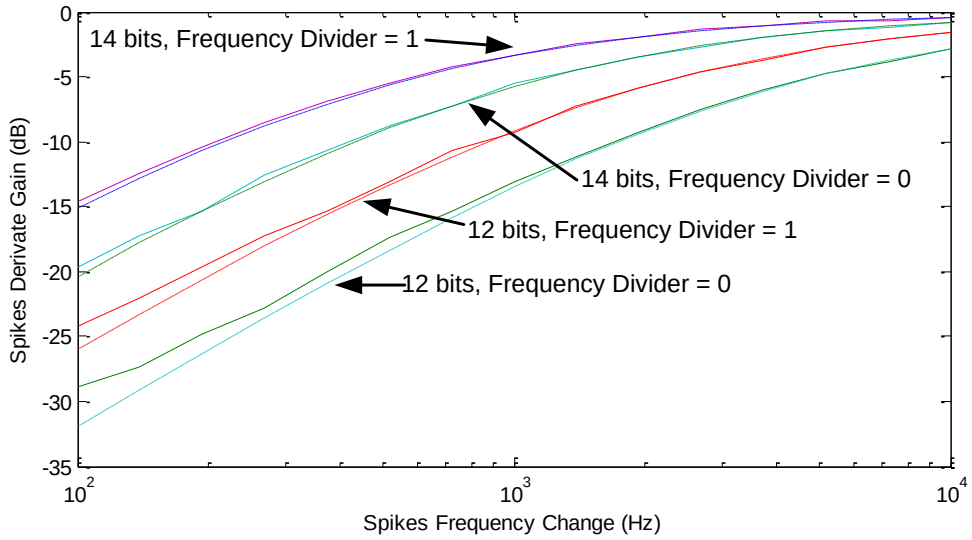


Figura 4.19: Diagrama de Bode del derivador de spikes

Para finalizar este apartado, vamos a excitar al derivador de spikes con otras funciones de entrada. En primer lugar hemos excitado el derivador de spikes con una función de entrada cuadrada, en la Figura 4.20. Es decir, hemos aplicado a la entrada del derivador de spikes una secuencia de spikes a frecuencia constante de 3MSpikes/Sec, pero cambiando con un período de 2mSec, la cual puede verse en la parte superior de la Figura 4.20 en color azul. Además mostramos la reconstrucción de la salida del derivador de spikes en rojo, así como la salida ideal de este componente en verde. Mostrando en la parte inferior de dicha figura los spikes provenientes del generador de spikes, azul, y la salida del derivador de spikes, verde. Esta figura constata la corrección en la operación del derivador de spikes, ya que si comparamos su salida con la salida ideal, vemos como se ajustan bastante bien. Siendo dicha salida la derivada de una señal en escalón, un impulso en cada transición, aplicada a un filtro paso de baja, modelado por la dinámica del derivador de spikes.

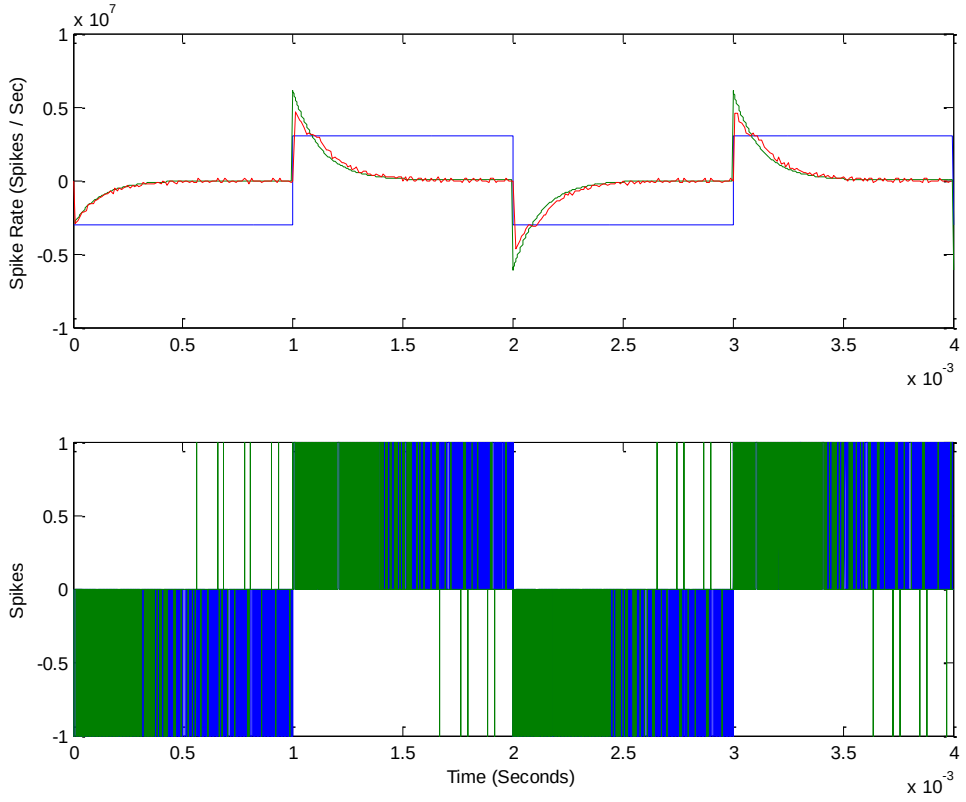


Figura 4.20: Respuesta del derivador de spikes ante una entrada cuadrada

A continuación hemos excitado el derivador de spikes con una función de entrada senoidal, en la Figura 4.21. Esta señal de entrada, que podemos ver en la parte superior de la figura en color azul, se caracteriza por tener una amplitud de 3MSpikes/Sec y una frecuencia de 500Hz. Además mostramos la reconstrucción de la salida del derivador de spikes en rojo, así como la salida ideal de este componente en verde. Mostrando en la parte inferior de dicha figura los spikes provenientes del generador de spikes, azul, y la salida del derivador de spikes, verde. De nuevo vemos el elevado grado de ajuste entre la salida del derivador de spikes y su salida ideal. En este caso la salida es la derivada del seno, es decir el coseno, y como la fase de dicha señal se ve afectada por la dinámica del derivador de spikes. En la figura puede apreciarse que en los primeros instantes de la simulación la salida del derivador de spikes trata de seguir la señal de entrada, sin embargo, a medida que avanza el tiempo la frecuencia de los spikes de salida se ajustan al coseno esperado, este efecto se debe a que al comenzar la simulación los valores iniciales del derivador han sido cero, de la misma manera que ocurre con los filtros analógicos [Oppenheim92]. De tal manera que la máxima frecuencia de los spikes disparados por el derivador de spikes coincidirá por los pasos por 0 del seno de la entrada, y de manera análoga en el caso opuesto, pero no ocurre exactamente así, ya que el polo del derivador desvía levente la fase de la salida.

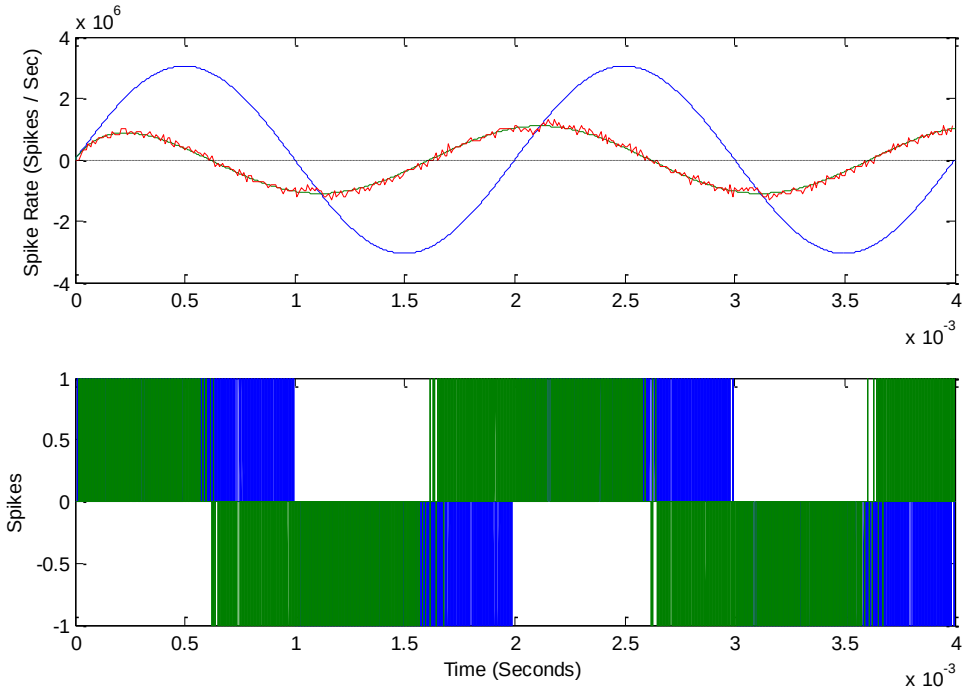


Figura 4.21: Respuesta del derivador de spikes ante una entrada senoidal

4.5. Integración del Integrate & Generate y del derivador de spikes al control, el controlador PID basado en spikes.

Una vez expuestos todos los componentes para poder diseñar un controlador PID, hemos de estudiar la topología con la que los vamos a conectar los componentes presentados, obteniendo de esta manera la arquitectura del controlador, así como las ecuaciones equivalentes resultantes, para que el controlador PID basado en spikes pueda ser ajustado en base a los requerimientos de la respuesta de un determinado sistema. En este apartado no pretendemos obtener ningún ajuste óptimo, ni ninguna respuesta en particular, sino estudiar la analogía de los modelos presentados basados en spikes con los controles tradicionales continuos.

Vamos a diseñar un controlador PID parecido al mostrado anteriormente en la Figura 4.1. Para implementar el controlador PID vamos a necesitar 3 Hold & Fire, con el fin de poder sumar los diversos términos, un Integrate & Generate y un derivador de spikes, así como un Spikes Expansor, para aplicar los spikes a un motor de DC. En la Figura 4.22 mostramos la arquitectura del controlador PID basado en spikes. La señal de error E(s) entrará por la izquierda de la figura, donde a continuación para implementar los términos integral y derivativo, usaremos el Integrate & Generate y el derivador de spikes. Sin embargo, la única manera de controlar el término proporcional es mediante el uso del Spikes Expansor. Así que lo que haremos será propagar el error sin más, para luego aplicarle la ganancia asociada al tiempo en alto de los spikes. Nótese que para sumar dos señales de spikes con un Hold & Fire, en vez de restarlas, lo que tenemos que hacer es simplemente invertir los spikes de una de sus entradas, es decir, conectar a la componente positiva de una de sus entradas, una señal de spikes negativos, y viceversa.

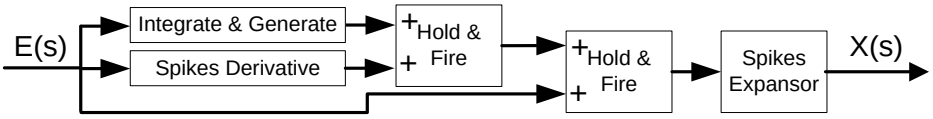


Figura 4.22: Arquitectura del controlador PID basado en spikes

Sin embargo, esta ganancia afectará a todos los miembros integrantes del controlador PID. La siguiente ecuación nos muestra la función de transferencia equivalente de nuestro sistema:

$$PID(s) = \frac{X(s)}{E(s)} = \left(1 + \frac{k_i}{s} + \frac{s}{s + k_d}\right) * k_p$$

Ecuación [4-22]

Esto significa que la ganancia del Spikes Expansor afectará a todo el sistema. Siendo las ganancias equivalentes del término integral y derivativo proporcionales a la ganancia del Spikes Expansor. Sustituyendo las ecuaciones de las ganancias de cada componente del controlador y añadiendo el polo de nuestro derivador, podemos obtener la ecuación del controlador PID basado en spikes más detallada:

$$PID(s) = \left(1 + \frac{F_{CLK}}{2^{nI-1}(IG_{FD}+1)} * \frac{1}{s} + \frac{s}{s + \frac{F_{CLK}}{2^{nD-1}(SD_{FD}+1)}}\right) * (SpikeWidh + 1) * T_{CLK} * V_{PS}$$

Ecuación [4-23]

Donde IG_{FD} y SD_{FD} son los divisores de frecuencia del Integrate & Generate y del derivador de spikes respectivamente. Así como *nI* y *nD* son los números de bits para cada componente, *SpikeWidth* es el ancho de pulso en ciclos de reloj, F_{clk} la frecuencia de reloj del sistema, y V_{PS} es el voltaje proveniente de la fuente de alimentación, y que será aplicado a los bornes del motor.

4.6. Simulación del controlador PID basado en spikes

A continuación vamos a proceder a simular el controlador PID basado en spikes, para poder comprobar su correcto funcionamiento y proximidad a un controlador PID ideal. Para ello hemos construido un nuevo escenario de simulación en Simulink junto con Xilinx System Generator, mostrado en la Figura 4.23. En ella mostramos sólo el bloque de control PID basado en spikes, en la que las entradas del controlador, referencia y realimentación, se encuentran a la izquierda, y conectadas a su vez a dos generadores sintéticos de spikes Bit-Wise, los cuales convertirán estas señales en spikes. A continuación encontramos un Hold & Fire, el cual restará ambas señales, y proporcionará a su salida el error del controlador. Dicho error será la entrada del controlador PID basado en spikes, el cual alberga en su interior un controlador acorde con el expuesto en la Figura 4.22. Para que finalmente los spikes de salida del controlador PID lleguen a un Spikes Expansor, el cual extenderá los spikes a su entrada y los aplicará al motor.

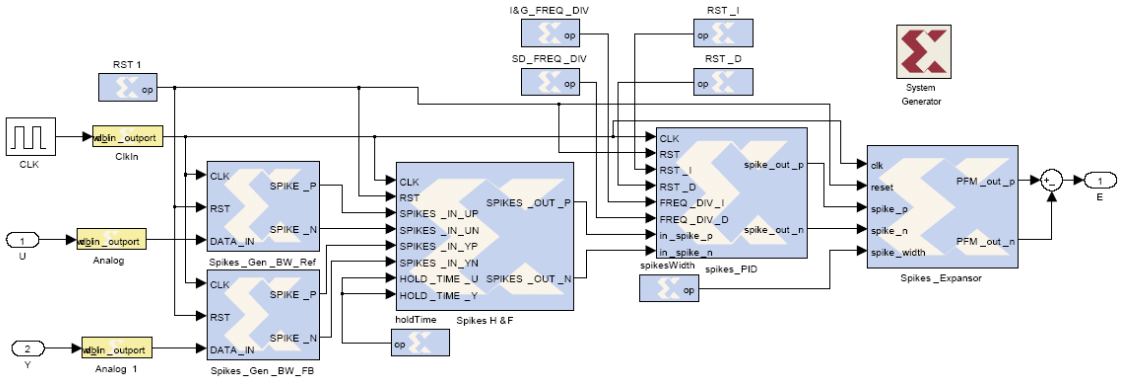


Figura 4.23: Escenario de simulación del controlador PID basado en spikes

Para comprobar el comportamiento del controlador PID después de integrar el Integrate & Generate y el derivador de spikes, vamos a simularlo por partes, es decir, vamos a comenzar inhabilitando el derivador de spikes para poder probar sólo un controlador proporcional e integral, o PI, para después inhabilitar el Integrate & Generate obteniendo un controlador proporcional y derivativo, PD, y finalmente habilitando todo los elementos para simular el controlador proporcional integral y derivativo, PID, basado en spikes.

a) Control PI

En primer lugar vamos a mantener en estado de reset el derivador de spikes, obteniendo así un controlador PI basado en spikes. El propósito de un controlador PI es eliminar el error en estado estacionario provocado por la componente proporcional. El control integral actuará cuando hay una desviación entre el valor de realimentación y el de referencia, integrando esta desviación en el tiempo, multiplicándola por la constante de integración y sumándola al término proporcional. Dicho controlador consta sólo de dos parámetros, la ganancia integral y la proporcional, así que vamos a simular la respuesta del motor ante ambos parámetros, para poder observar de esta manera su comportamiento. Para comenzar el controlador ha sido simulado fijando el número de bits del integrador a 16, es decir, fijando el término integral, y modificando el ancho de spike iterativamente, entre 1.6uSec y 4uSec, variando así el término proporcional. Sin embargo, el modificar el ancho no sólo afecta al término proporcional, sino también al término integral, tal y como expone la Ecuación [4-23]. Los resultados obtenidos tras simular el controlador bajo estas condiciones se encuentra en la Figura 4.24. En dicha figura mostramos la referencia del controlador junto con las respuestas del motor para cada caso, estando la respuesta del controlador en trazo sólido, y la respuesta del motor con un controlador similares características e ideal en trazo discontinuo. En la figura se aprecia como gracias a la adición de un término integral al controlador, el motor consigue alcanzar la referencia de velocidad ante una entrada en escalón, eliminando el error en régimen permanente que introducía el controlador P basado en spikes. Además el motor comienza a mostrar respuestas propias de un sistema subamortiguado, siendo este comportamiento debido a la pérdida de fase del sistema equivalente por el uso del integrador [Houpis89]. Las respuestas muestran un sobredisparo de similar magnitud, sin embargo el tiempo de subida y asentamiento de la señal se decrementa a medida que aumentamos el ancho de los spikes. Contrastando la respuesta de nuestro controlador basado en spikes frente

a un controlador ideal, observamos cómo hay una leve desviación entre ellas, siendo debida al error introducido por los elementos integrantes del controlador, como expusimos con anterioridad. Pero añadiendo más fuentes de error, ya que nos encontramos ahora 3 Hold & Fire, además del error introducido por los generadores Bit-Wise alojados en el interior de los Integrate & Generate.

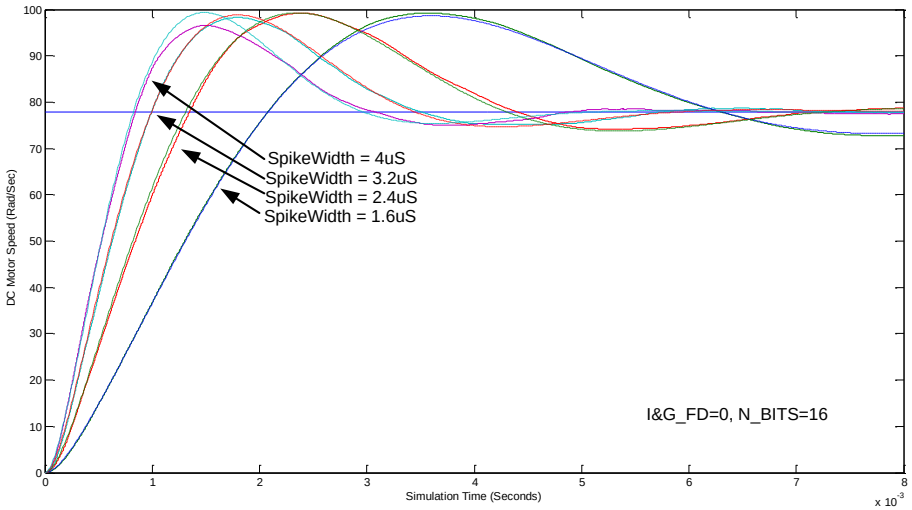


Figura 4.24: Simulación de la respuesta del motor con un controlador PI basado en spikes para diferentes anchos de spikes

Como siguiente experimento hemos fijado el valor del ancho de spike, o término proporcional, a 2uSec, y vamos a modificar el número de bits del Integrate & Generate, entre 15 y 19, obteniendo así distintas ganancias del término integral. La Figura 4.25 muestra los resultados de la simulaciones, en los que observamos cómo el término integral va disminuyendo a medida que usamos un número de bits de mayor, traduciéndose en valor de sobre disparo cada vez menor, pero con un tiempo de subida cada vez mayor [Aström09]. Llegando incluso en los dos últimos casos a tener una ganancia integral muy pequeña, mostrando la respuesta un comportamiento sobreamortiguado, y en el último caso necesitando una elevada cantidad de tiempo para alcanzar la referencia. En el caso de las respuestas subamortiguadas, a medida que decrementamos el número de bits, y en consecuencia aumentamos la ganancia del Integrate & Generate, la sobreoscilación se vuelve cada vez más severa, sin embargo conseguimos reducir el tiempo de subida de la señal en gran medida.

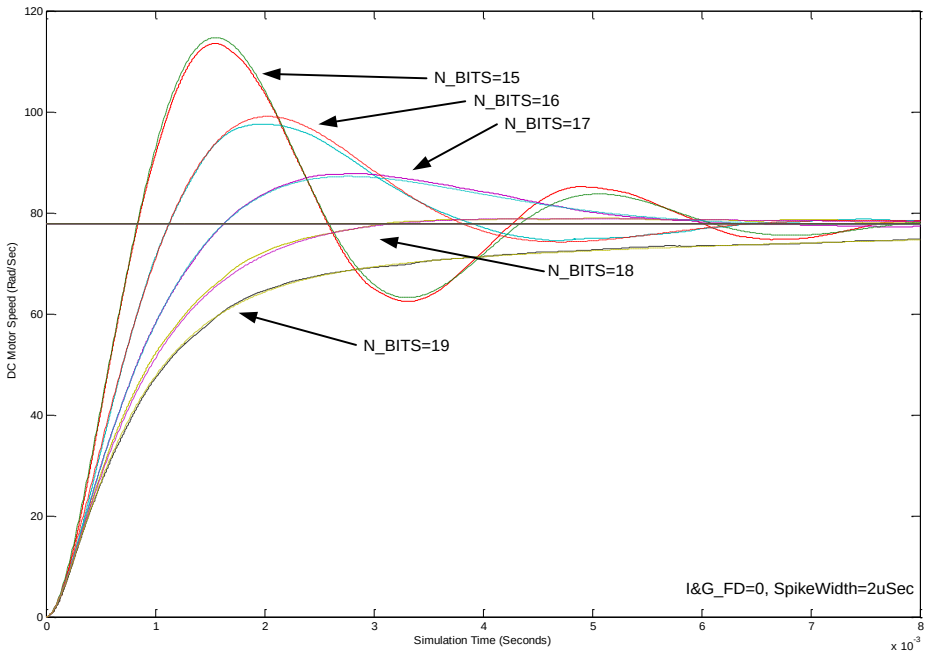


Figura 4.25: Simulación de la respuesta del motor con un controlador PI basado en spikes para diferentes valores del número de bits del Integrate & Generate

a) Control PD

A continuación vamos estudiar el comportamiento del derivador de spikes junto con el motor, para ello hemos dejado el Integrate & Generate en estado de reset, obteniendo así un control PD basado en spikes. La componente derivativa se manifiesta cuando hay un cambio en el valor del error, y trata que la respuesta del sistema se anticipe la variación del error, corrigiéndolo proporcionalmente con la misma velocidad que se produce. Este controlador suele mejorar la estabilidad del sistema, reduciendo las oscilaciones o haciéndolo más amortiguado, sin embargo, no logra anular el error en régimen permanente [Ogata04]. Este controlador tiene, al igual que al controlador PI, dos parámetros, la ganancia proporcional y la ganancia del término derivativo. En primer lugar hemos simulado la respuesta del controlador PD fijando el número de bits del derivador de spikes a 14, y modificando el valor del ancho de los spikes, alterando de este modo ambos términos, el proporcional, y el derivativo. La Figura 4.26 muestra los resultados de las simulaciones. En ella se aprecia cómo el controlador basado en spikes viene a comportarse como un controlador PD, siendo incapaz de alcanzar la referencia, y mostrando un comportamiento sobreamortiguado gracias a la fase aportada por el término derivativo. Además se ve cómo la ganancia del controlador se incrementa con el ancho de spike, asintóticamente a 1, sin alcanzar nunca la referencia. Sin embargo, este controlador es el que muestra una mayor desviación de la respuesta ideal. Esta desviación es achacable a que ante una entrada en escalón, el motor intentará alcanzar la referencia de la entrada, haciendo que el error entre la velocidad del motor y la referencia disminuya. Al disminuir el error del sistema la salida del derivador de spikes serán spikes negativos, que habrán de restarse en un Hold & Fire con el término proporcional, de tal forma que en

ocasiones llegarán spikes negativos al Spikes Expansor, invirtiendo el voltaje en los bornes del motor innecesariamente, e introduciendo la desviación que se aprecia en la figura.

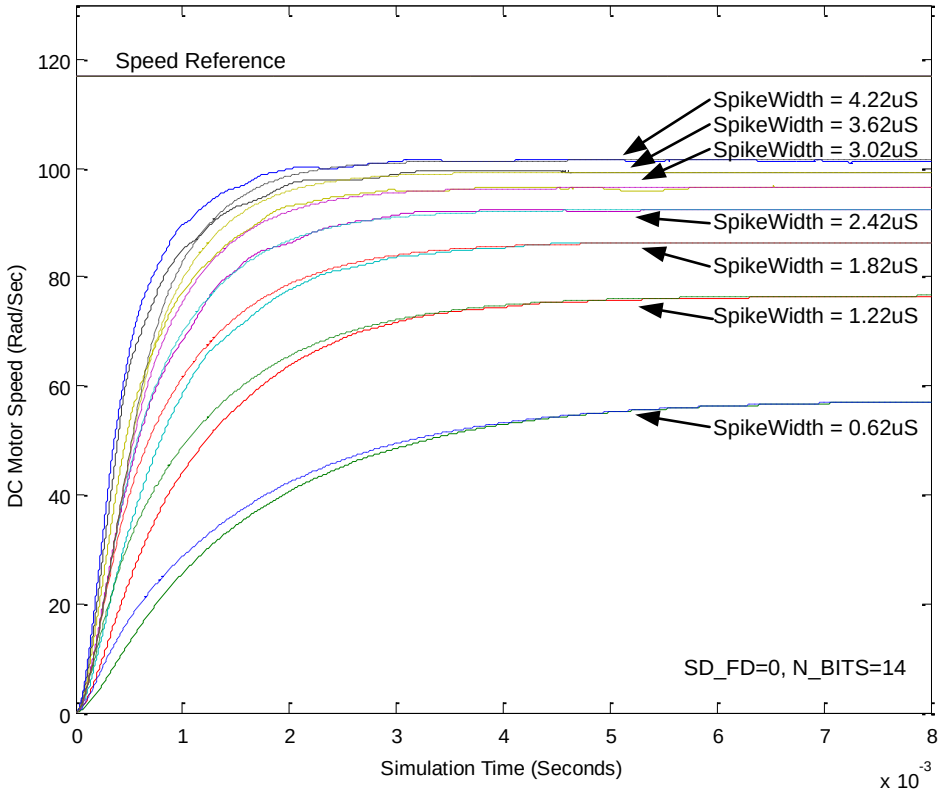


Figura 4.26: Simulación de la respuesta del motor con un controlador PD basado en spikes para diferentes anchos de spikes

Para sólo modificar el término derivativo del controlador PD basado en spikes, hemos fijado el valor del ancho de spikes a 2uSec, y modificado el número de bits del derivador de spikes. Los resultados de las simulaciones se muestran en la Figura 4.27, como indicaba la Tabla 4-1, al aumentar la componente derivativa del controlador sólo conseguimos pequeños cambios en decremento del tiempo de subida y del error en estado estacionario de la velocidad del motor.

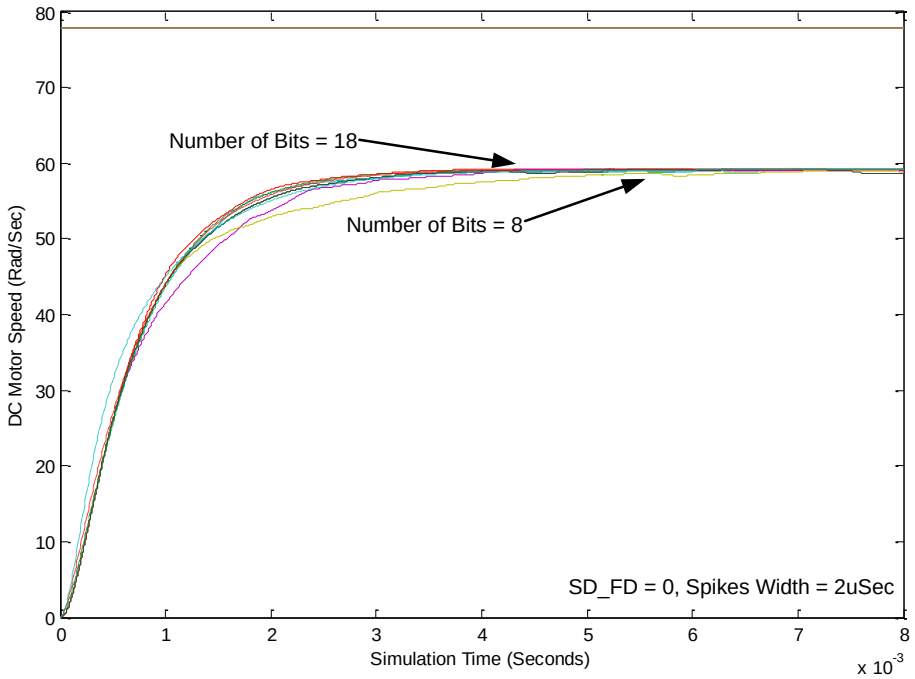


Figura 4.27: Simulación de la respuesta del motor con un controlador PD basado en spikes para diferentes números de bits del derivador de spikes

a) Control PID

Finalmente hemos simulado el controlador PID completo, con todos los términos, proporcional, integral y derivativo habilitados. Este controlador consta de tres parámetros, la ganancia proporcional, la integral y la derivativa. Para poder observar el comportamiento del controlador PID basado en spikes, vamos a ir modificando sus parámetros uno a uno, fijando el valor de dos de los tres parámetros, y modificando el tercero. En primer lugar hemos fijado el número de bits del Integrate & Generate y del derivador de spikes a 17 bits y 10 bits respectivamente, así como sus divisores de frecuencia a 0. A continuación hemos ido incrementando el ancho de spike desde 0.8uSec hasta 4uSec, realizando una serie de simulaciones cuyos resultados pueden encontrarse en la Figura 4.28. En ellas observamos respuestas parecidas a las del controlador PI, pero con la tasa de sobredisparo reducido gracias a la componente derivativa, y al igual que en el controlador anterior, el tiempo de subida de la señal se reduce a medida que incrementamos el ancho de los spikes, pero con un leve incremento de la sobre oscilación. En este controlador es en el que se producen las mayores desviaciones teóricas, ya que incorpora un mayor número de elementos introductores de error en el controlador.

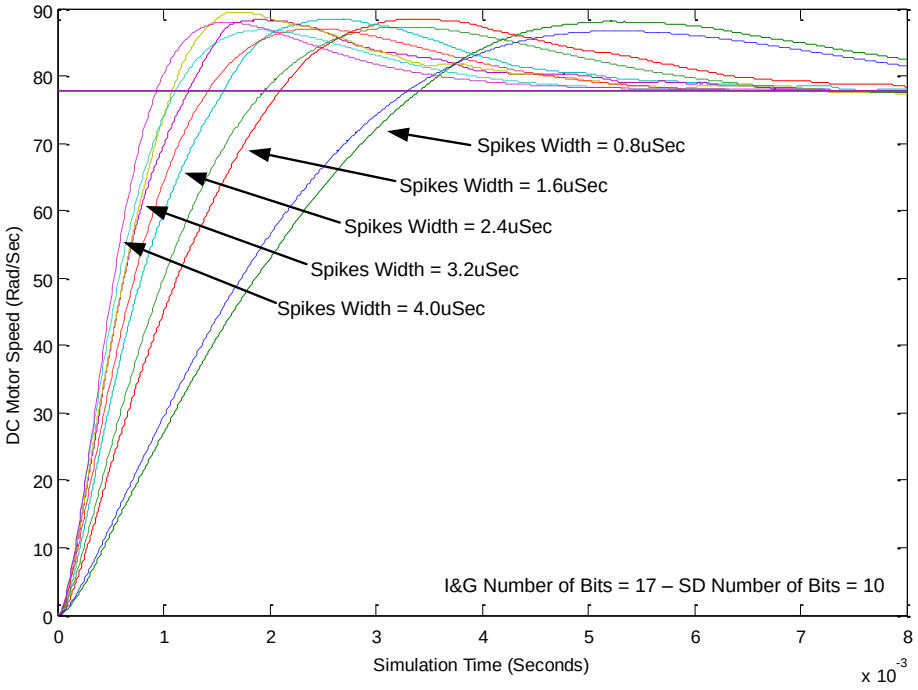


Figura 4.28: Simulación de la respuesta del motor con un controlador PID basado en spikes para diferentes anchos de spikes

A continuación hemos procedido a comprobar el efecto de la ganancia del Integrate & Generate sobre la respuesta del motor. Con este fin hemos fijado el ancho de los spikes a 2.4uSec, así como el número de bits del derivador de spikes a 10. A continuación hemos realizado una serie de simulaciones en las que hemos ido incrementando el número de bits del Integrate & Generate desde 15 a 19 bits, cuyos resultados se ofrecen en la Figura 4.29. En las respuestas del motor mostradas en la figura se aprecia cómo al establecer altas ganancias del término integral, la salida del motor presenta un comportamiento subamortiguado, con una tasa de sobre oscilación y tiempo de subida decreciente a medida que aumenta el número de bits del Integrate & Generate. Hasta el punto que la ganancia de la componente integral es tan pequeña que el sistema comienza a comportarse de manera sobreamortiguada, presentando un tiempo de subida muy elevado. Además las respuestas muestran una desviación frente a la respuesta ideal acorde con la ganancia del Integrate & Generate, ya que los efectos de las no idealidades del controlador se acentúan ante frecuencias tan altas.

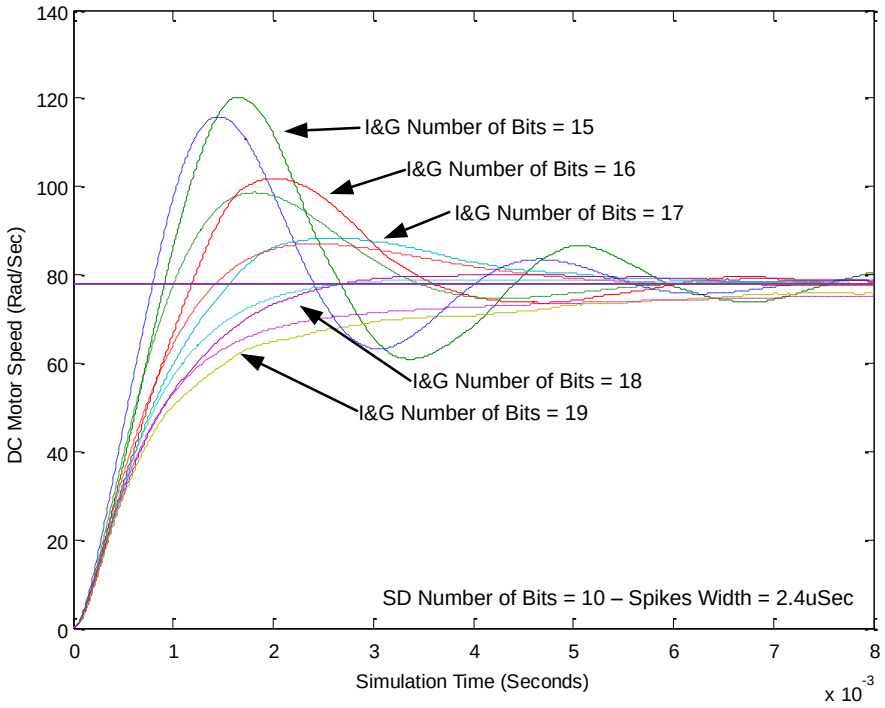


Figura 4.29: Simulación de la respuesta del motor con un controlador PID basado en spikes para diferentes números de bits del Integrate & Generate

Finalmente hemos fijado el ancho de los spikes a 2.4uSec, el número de bits del Integrate & Generate a 17, y su divisor de frecuencia a 0, modificando el número de bits del término derivativo, de 8 a 12 bits, para comprobar el efecto del término derivativo. La Figura 4.30 muestra los resultados de las simulaciones, en los que vuelve a quedar constancia que el término derivativo se decrementa, aunque no en gran medida, la tasa de sobre disparo y el tiempo de subida.

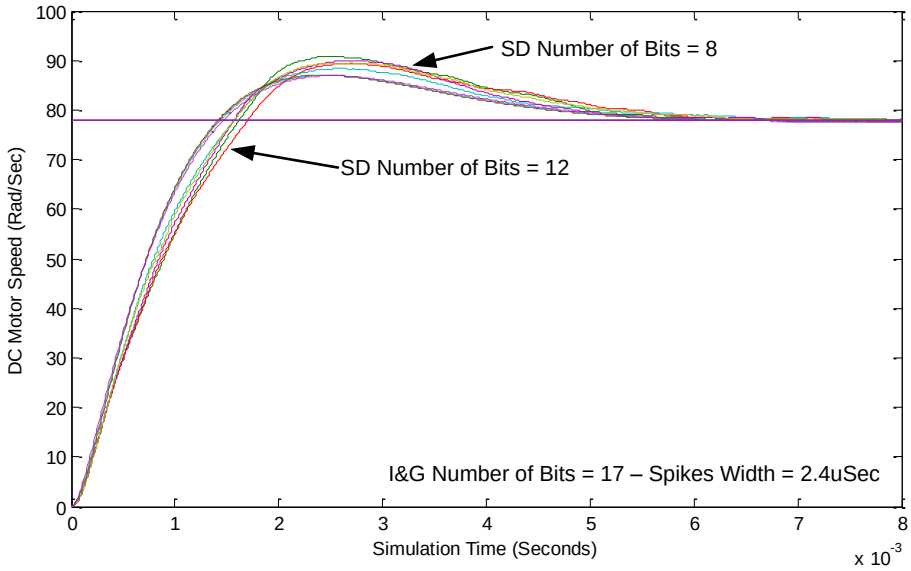


Figura 4.30: Simulación de la respuesta del motor con un controlador PID basado en spikes para diferentes números de bits del Derivador de spikes

4.7. Consumo hardware y operaciones equivalentes

En este apartado vamos a estudiar el consumo hardware, así como la eficiencia del controlador PID basado en spikes que hemos ido desarrollando a lo largo de este trabajo. La Tabla 4-8 muestra los resultados del consumo hardware así como del número de operaciones equivalentes que puede alcanzar nuestro controlador PID basado en spikes. En primer lugar vamos a analizar el consumo de hardware, para ello hemos sintetizado el controlador PID basado en spikes con diversos números de bits del Integrate & Generate y del derivador de spike, ya que este parámetro debe ser definido en tiempo de síntesis y modifica las necesidades hardware, al sintetizar contadores e integradores con distinta cantidad de bits. El Hold & Fire elegido para la síntesis ha sido el Hold & Fire con tiempo de hold infinito, ya que en el estudio de errores del controlador P quedó constatado que para esta aplicación es interesante tener unos tiempo de hold infinito. En la Tabla 4-8 aparecen en primer lugar el número de bits del Integrate & Generate y del derivador de spikes usados en la síntesis, a continuación, mostramos el número de slices consumidos de la FPGA, la frecuencia de operación máxima del controlador PID, el porcentaje de ocupación del controlador en una FPGA de familia Spartan3, así como el número máximo de controladores PID basados en spikes del mismo tamaño que podría albergar en su interior. Obsérvese en la tabla que el consumo de hardware crece acorde con el número de bits elegido para cada elemento componente del controlador PID, además que el derivador de spikes tiene en su interior un Integrate & Generate, el incremento de consumo de slices aumenta de igual manera que en un Integrate & Generate. De tal forma que en el segundo y tercer caso de la tabla, cuando intercambian el número de bits ente 14 y 18, el consumo hardware es exactamente el mismo. También resulta interesante el número de controladores albergables en una FPGA, ya que podríamos controlar entre 25 y 30 motores con un control PID en lazo cerrado, de una forma masivamente paralela.

Finalmente hemos calculado el número de operaciones equivalentes de nuestro controlador, comparado con el sistema digital ideal equivalente. Para ello calculado el número de operaciones que son necesarias para llevar a cabo un control PID digital [Ogata96], en tiempo discreto. Descomponiendo el controlador PID digital nos encontramos que el integrador ha de realizar las siguientes operaciones:

$$I(i) = I(i-1) + K_I * e(i) * T_S$$

Ecuación [4-24]

Las cuales suponen una suma y una multiplicación. Análogamente, la derivada puede calcularse como:

$$D(i) = \frac{K_D * (e(i) - e(i-1))}{T_S}$$

Ecuación [4-25]

Contando el número de operaciones necesarias para ejecutar estos componentes del control, añadiendo una multiplicación del control P, y dos sumas para sumar los tres componentes, y una resta para la realimentación, obtenemos un total de 4 sumas, 4 multiplicaciones y una división. Dado que la complejidad de estas operaciones es distinta para

un computador, es decir, el computador no invierte el mismo número de ciclos de reloj para realizar una simple suma, que una compleja división, es habitual asociarle un "peso ponderado" a cada operación. De manera que una suma tiene un peso de 1, sin embargo una multiplicación tiene un peso de 4, equivalente a 4 sumas, y finalmente a las divisiones se les aplica un peso de 9, equivalentes a 9 sumas [Patterson09]. Calculando el número de operaciones necesarias para implementar un controlador PID discreto, obtenemos:

4sumas+4productos+1división= 4+16+9=29 operaciones

Nuestro controlador PID basado en spikes realiza todas estas operaciones de forma equivalente en paralelo, sin tener en cuenta ningún tipo de overhead entre las instrucciones, y sin ningún tipo de bloqueo de control, estructural o de salto [Patterson09]. Así que para calcular el número de operaciones discretas equivalentes alcanzables por el controlador PID basado en spikes vamos a multiplicar la frecuencia máxima de operación del controlador por el número de operaciones equivalentes que es capaz de computar, mostradas en la última columna de la Tabla 4-8. En ella no sólo mostramos el número de operaciones, sino el número de operaciones máximo que podría alcanzar una FPGA de bajo coste (en torno a los 30€), repleta de este tipo de controladores.

Estos controladores tienen dos ventajas fundamentales frente a los controladores discretos. Por un lado, gracias a la representación de la información como información pulsante, la información se transmite sin necesidad de complejos buses con un gran ancho, sólo necesitamos 2 bits para comunicar información con suficiente precisión, y además el procesamiento basado en spikes resulta muy simple (la operación más compleja ejecutada es sumar 1 a un contador), obteniendo circuitos con un consumo hardware muy bajo, permitiendo así la ubicación de una gran cantidad de controladores en una FPGA. Sin embargo, en un controlador discreto nos encontramos una elevada cantidad de multiplicaciones y divisiones, que son complejas de ejecutar, encontrando sólo unas pocas unidades funcionales en los DSP, como son los multiplicadores y divisores en coma flotante, que permitan la ejecución de estas operaciones, necesitando secuenciar el controlador para ir reutilizando estas unidades. Por otro lado, nuestros controladores son masivamente paralelos, a diferencia de los sistemas digitales actuales, desde los simples microcontroladores, hasta complejos procesadores superescalados, supersegmentados, y multinúcleo, que apenan alcanzan un paralelismo a nivel de instrucciones. Teniendo que ejecutar el algoritmo de control PID en un tiempo limitado, y no permitiendo el control en paralelo más que de un pequeño número de motores.

Tabla 4-8: Resumen de la síntesis y rendimiento del controlador PID basado en spikes

Número de bits del integrador	Número de bits del derivador	Slices	Máxima Frecuencia (MHz)	Porcentaje de uso XC3S400, en %	Cantidad máxima de controladores	Operaciones equivalentes por segundo. 1 controlador / máximo FPGA (GOPs)
14	14	120	134.35	3.3482	29	3.89 / 112.99
14	16	127	125.79	3.5435	28	3.64 / 102.14
16	14	127	125.82	3.5435	28	3.64 / 102.17
16	16	133	119.41	3.7109	26	3.4 / 90.04
18	18	148	116.75	4.1295	24	3.3 / 81.26

5. Diseño, implementación y análisis de controles en lazo cerrado basados en spikes sobre sistemas reales

En este capítulo vamos a exponer cómo se han llevado a la realidad los controles expuestos en los capítulos anteriores. Nuestro objetivo es analizar la respuesta de motores reales junto con los controles basados en spikes. Para ello hemos diseñado una plataforma completamente configurable, la plataforma AER-Robot [Linares06a]. Dada la gran versatilidad de esta plataforma podemos implementar en su interior una elevada variedad de elementos, en nuestro caso particular vamos a almacenar en su interior un controlador PID basado en spikes, monitorizando los spikes usando el bus AER, y analizando sus respuestas en un PC. Nos vamos a central en la implementación de controles PID usando el Spikes Expansor, ya que en el simulador es el elemento que mejor comportamiento ha mostrado, así como un modelo más "próximo" a la biología.

El sistema construido se muestra en la Figura 5.1 (el cual será presentado en los próximos capítulos como Eddie), donde podemos ver una fotografía a la derecha, y a la izquierda un diagrama conceptual del sistema completo. La idea es conectar un motor de DC real (en nuestro caso fabricado por Maxon Motors) a la AER-Robot para poder experimentar con él. Para poder modificar los parámetros de nuestro controlador vamos a conectar la AER-Robot, que incluye un puerto USB, a un PC. Además con el objeto de monitorizar la respuesta de los motores vamos a conectarla a través del bus AER a la USBAERmini2, la cual actúa como un puente entre el bus AER y el bus USB. Recibiendo así en el PC toda la actividad del interior de la AER-Robot para poder analizarla a posteriori [Jimenez09b].

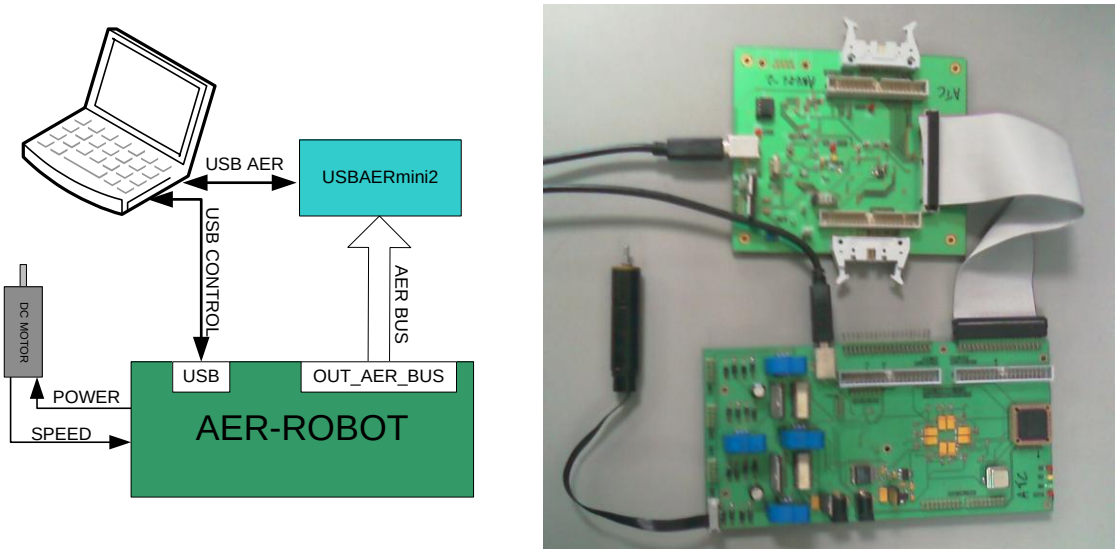


Figura 5.1: Diagrama de bloques y fotografía del sistema de test de los controles implementados

Para poder llevar el controlador PID basado en spikes a la realidad nos han surgido varias cuestiones, las cuales serán abordadas en detalle en los próximos apartados. En primer lugar daremos una visión superficial sobre la tarjeta AER-Robot, ya que será el soporte físico del controlador. A continuación, surgen varias cuestiones acerca del controlador PID basado en spikes en sí mismo, estas cuestiones son: cómo medir la velocidad del motor mediante spikes,

el mecanismo a usar para poder gestionar los parámetros del controlador de manera externa a la FPGA (desde un PC por ejemplo), cómo modificar el número de bits tanto de la componente integral como derivativa, ya que este parámetro es definido en tiempo de síntesis e inalterable en tiempo de ejecución, y finalmente cómo poder monitorizar la actividad interna de los spikes del controlador mediante la representación AER. Tras abordar toda esta problemática, localizada en el interior de la FPGA, expondremos cómo hemos programado el microcontrolador para dar soporte a la gestión del controlador, y proporcionaremos una descripción somera de la USBAERmini2 usada para monitorizar el tráfico AER que generará el controlador PID basado en spikes. Finalizando este capítulo con el análisis experimental del controlador PID basado en spikes junto con el motor utilizado.

5.1. La plataforma AER-Robot

La plataforma AER-Robot ha sido diseñada conceptualmente como un puente entre motores de DC, y el bus AER [Linares06a][Linares07a]. La idea de esta plataforma es recibir información AER y actuar en consecuencia sobre motores de DC, además de sensarlos y enviar el valor de los sensores como información AER [Jimenez07]. Desde el punto de vista de una red neuronal, la plataforma AER-Robot representaría una capa de entrada/salida de una red AER, donde la entrada son los sensores del robot y la propia información AER, y las salidas, la actuación sobre cada motor, así como la información sensorial codificada como eventos AER. No obstante, como expondremos más adelante, la AER-Robot tiene suficiente potencia para comportarse como mucho más que un simple puente. La funcionalidad requerida para la plataforma de AER-Robot es la siguiente:

- Ofrecer una plataforma completamente programable, con los puertos de E/S AER necesarios, en nuestro caso 4.
- Actuar sobre 4 motores de DC.
- Poder sensar estos motores mediante sensores analógicos y encoders ópticos de cuadratura.
- Ofrecer una vía de comunicación con un PC para establecer los parámetros de funcionamiento de la plataforma de desarrollo.

La plataforma AER-Robot es completamente reprogramable y escalable, estando basada en una arquitectura de FPGA más microcontrolador. La Figura 5.3 nos muestra el diagrama de bloques la plataforma AER-Robot. En la figura podemos ver como la FPGA está conectada al microcontrolador, a una etapa de potencia para motores de DC, y a diversos puertos AER. Para poder escalar la plataforma se le han añadido 4 puertos AER, pudiendo así conectar tantas AER-Robots en cascada como se necesite. Además en cada AER-Robot hemos incluido sensores de efecto Hall para medir la intensidad de los motores en todo momento, y reservado varios pines para poder conectar los encoders ópticos de los motores directamente a la FPGA. La FPGA pertenece a la familia Spartan 3 de Xilinx [SPARTAN3], y en concreto es la XC3S400; el microcontrolador pertenece a la familia de los 8051, es fabricado por Sillicon Laboratories, y es el C8051F320 / 342 [C8051F320].

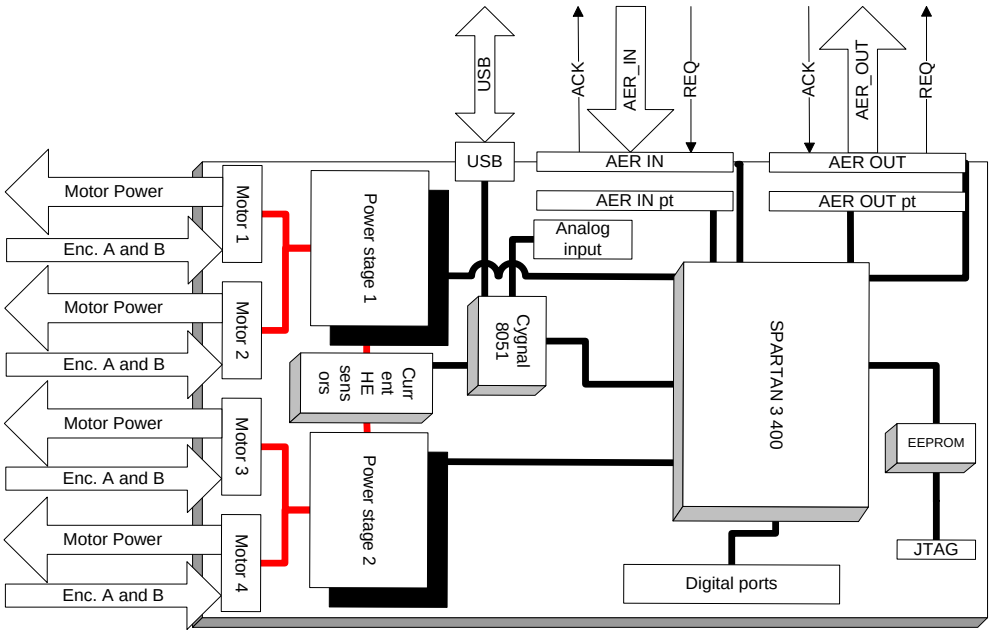


Figura 5.2: Diagrama de bloques de la plataforma AER-Robot

La idea subyacente de la AER-Robot es usar la FPGA para los elementos de procesado que necesiten una velocidad muy alta, como son los eventos AER, o los controladores basados en spikes. Estando el microcontrolador reservado para las tareas lentas, o las dificultosas de realizar en la FPGA, como es la comunicación USB con un PC, la conversión de analógico a digital de sensores, o como veremos más a adelante, la decodificación de un receptor de radio control.

En resumen, las principales características de la plataforma AER-Robot son:

- Spartan 3: XC3S400
  - 400k puertas
  - 56k bits de memoria RAM distribuida, y 288k bits de memoria RAM en bloque
  - 4 Digital Clock Managers (DCM)
  - 16 Multiplicadores dedicados
- Microcontrolador: C8051F320 / 342
  - 24 / 48MHZ
  - Memorias con 2304k bytes de RAM interna y 16k bytes de memoria flash
  - USB 2.0 Full-Speed
  - ADC 200ksamples
  - Variedad puertos Serie

- 4 puertos AER, permitiendo la conexión de múltiples AER-Robots en cascada.
- Aislamiento electrónico entre la etapa lógica y la de potencia, permitiendo alimentaciones separadas con el fin evitar ruido indeseado en la etapa lógica debido al uso de motores de alta potencia.

- 4 etapas de potencia independientes para motores de DC de hasta 24V.
- 2 canales de encoder óptico para cada motor, permitiendo el sensado independiente de cada motor.

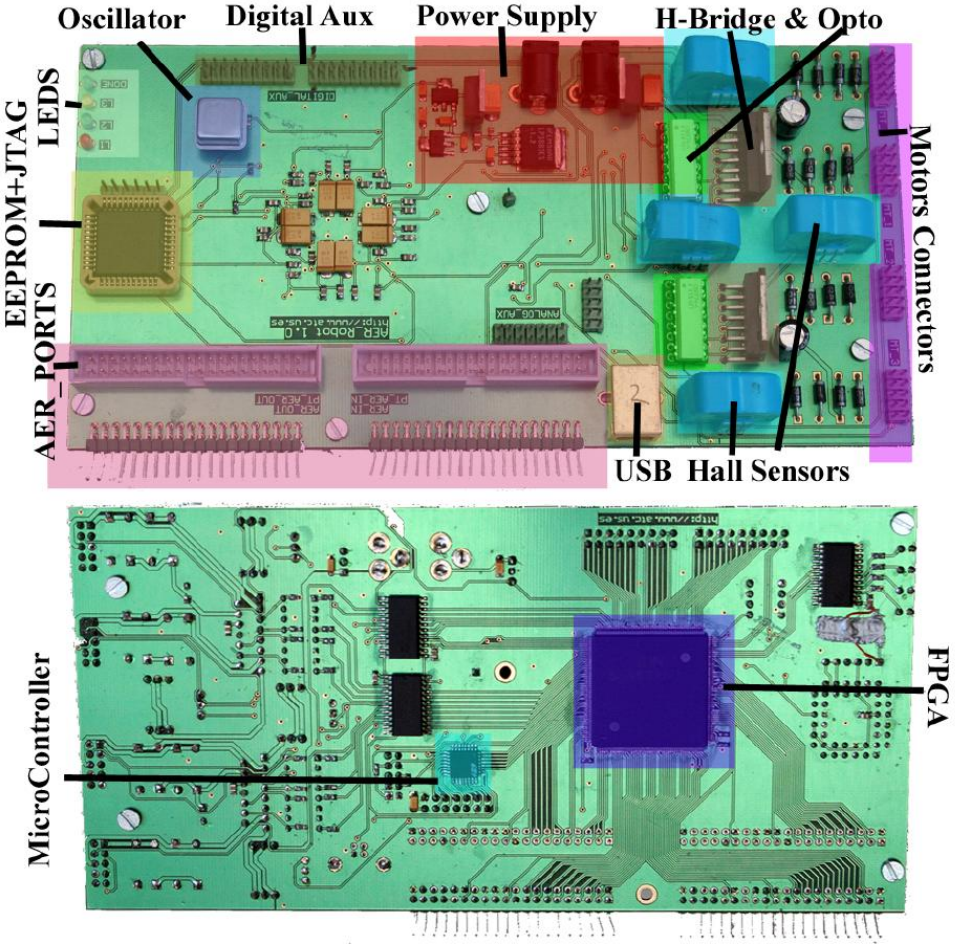


Figura 5.3: Fotografía de ambas caras de la tarjeta AER-Robot

5.2. Implementación real del controlador PID basado en spikes

En este apartado vamos a exponer como hemos implementado un controlador PID basado en spikes en el interior de la FPGA incluida en la AER-Robot. Nuestra idea es realizar un análisis análogo al realizado en los capítulos anteriores, pero sin el uso del simulador. Para ello vamos a implementar un controlador PID basado en spikes parametrizable en el interior de la FPGA, excitarlo con diversos parámetros y entradas, y monitorizar las respuestas del control a través del bus AER.

La implementación completa del sistema se ha realizado en varios niveles. Por un lado, en el nivel más bajo, está el firmware almacenado en la FPGA, el cual implementa un controlador PID basados en spikes, un monitor AER, así como mecanismos para gestionar todos los parámetros del controlador. Por otro lado está el microcontrolador, el cual actúa en este caso como un dispositivo USB, de tal manera que desde un PC le podemos enviar los parámetros de los controles, así como las funciones de excitación del sistema, a la FPGA.

Finalmente, conectado al bus AER nos encontramos con la tarjeta USB2AERmini, la cual envía al PC por el puerto USB la información AER generada por la AER-Robot.

5.2.1. Arquitectura del controlador

Como acabamos de exponer, en el interior de la FPGA están los elementos encargados del control. Hemos implementado un control basado en varios Hold & Fire e Integrate & Generate, y en un Spikes Expansor, completamente configurable, que no sólo permite cambiar sus parámetros, sino también su topología, para convertirlo en un control en lazo cerrado o abierto, en base a nuestras necesidades. Además gracias a un registro de control vamos a poder habilitar o deshabilitar los componentes integrales o derivativos del controlador, teniendo así controles P, PI, PD o PID basados en spikes. Podemos ver el diagrama de bloques del control implementado en la Figura 5.4. Los spikes de excitación del controlador son generados por el generador sintético de spikes bit-wise (a la izquierda) según le ordene el microcontrolador. Siendo estos spikes una de las entradas de un Hold & Fire, y restados con los spikes provenientes de la realimentación, obteniendo así el error del sistema. Los spikes del error del sistema son aplicados a un módulo que contendrá el controlador PID (en el centro). Finalmente extendemos los spikes de la salida del controlador PID en el tiempo mediante el uso de un Spikes Expansor (a la derecha), y son aplicados directamente al motor de DC. Sin embargo, para poder cambiar la topología del control, de un control en lazo cerrado a uno en lazo abierto, hemos añadido un multiplexor (con dos canales de dos bits) entre el controlador PID y el Spikes Expansor, de tal manera que podemos seleccionar la entrada del Spikes Expansor entre los spikes del generador directamente (lazo abierto) o la salida del controlador PID basado en spikes (lazo cerrado). El bit que selecciona el canal del multiplexor, y en consecuencia la topología del control, estará almacenado en un registro de estado, como expondremos más adelante.

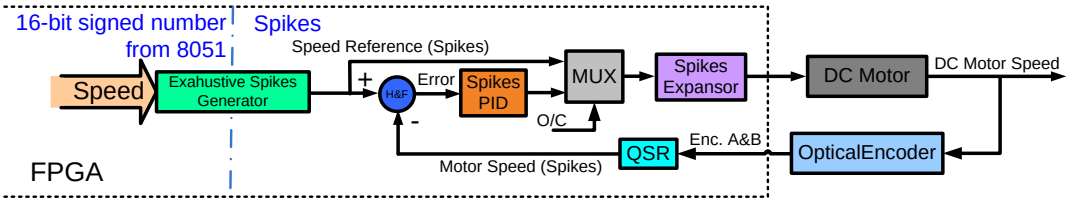


Figura 5.4: Diagrama de bloques del control en lazo cerrad basado en spikes

5.2.2. Realimentando el sistema, midiendo la velocidad del motor

En la figura anterior se muestra un nuevo componente aún no presentado (en celeste), usado para convertir la velocidad angular del motor (la realimentación del sistema) en spikes, el Quad Spikes Rate, o QSR. Uno de los sensores muy comúnmente usado para medir la velocidad de rotación de un motor son los encoders ópticos incrementales [HEDS550]. Estos sensores codifican la velocidad del motor mediante una señal modulada en PFM, donde recibimos una secuencia de pulsos, con dutty-cycle constante, pero frecuencia variable. En nuestro caso particular, nuestros encoders tienen dos canales de pulsos, las señales A y B,

desfasadas 90º entre ellas. Pudiendo calcular la velocidad del motor a partir de la frecuencia de estas señales, y el sentido de giro, o signo, en base a la fase relativa entre ellas. Para traducir estas señales a spikes, hemos implementado este nuevo componente VHDL, el ya mencionado QSR. Este componente está compuesto por una máquina de estados finita (Finite State Machine, FSM), la cual determina el sentido de giro, y dispara un spike por cada flanco de la señales A y B. El diagrama de estados de este componente es mostrado en la Figura 5.5. Los spikes de salida estarán homogéneamente distribuidos en el tiempo, ya que las señales A y B tienen un desfase preciso de 90º entre ellas, pudiendo tener así el cuádruple de precisión usando estos sensores.

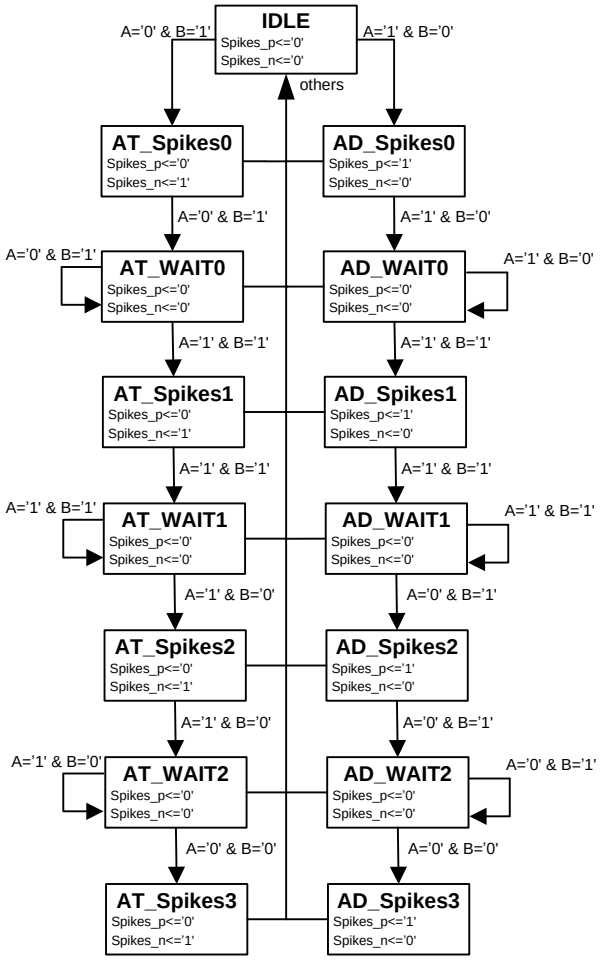


Figura 5.5: Diagrama de estados del conversor de las señales del encoder a spikes

Haciendo uso de ChipScope, proporcionado por Xilinx [CHIPSCOPE], hemos capturado las entradas del QSR, las señales A y B, y su salida, los spikes generados, directamente del interior de la FPGA mostradas en la Figura 5.6. En la figura se observa el giro positivo del motor, ya que la señal A se dispara antes que la señal B, y los spikes positivos generados en cada flanco de las señales A y B. También puede apreciarse la perfecta distribución de los spikes de salida en el tiempo, ya que las señales A y B tienen una fase precisa (cuando la velocidad está en régimen estacionario) de 90º.

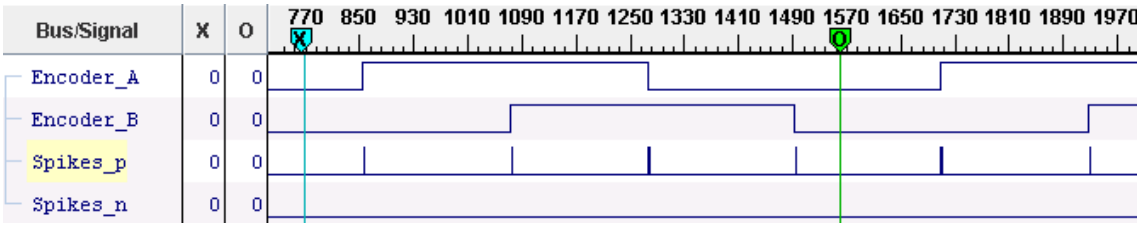


Figura 5.6: Spikes disparados a partir de las señales A y B del encoder óptico

El encoder incorporado en el motor que será usado en los próximos apartados para el análisis experimental del controlador PID basado en spikes, tiene una elevadísima resolución, de 500 pulsos por vuelta. Tras conectarlo al QSR, obtenemos 2000 spikes por vuelta, y dado que dicho motor tiene una velocidad máxima de unas 115 revoluciones por segundo (rps), la frecuencia máxima de los spikes proveniente de los encoders será de unos 230kSpikes/Sec. Esta tasa de spikes es suficientemente elevada para que los armónicos de los spikes sean filtrados por el propio motor, además de proporcionando un rango de control muy alto. Sin embargo, se puede dar el caso de que el encoder usado no tenga una resolución tan alta, proporcionando una tasa de spikes relativamente pequeña. En este caso habría que usar otras técnicas para convertir las señales A y B del encoder óptico en spikes. Como por ejemplo, medir el período de las señales A y B, y en base a dicho período generar una nueva secuencia de spikes usando un generador sintético de spikes, como podría ser el general exhaustivo Bit-Wise.

5.2.3. Gestión externa de los parámetros del controlador

Para gestionar cómodamente los parámetros de los elementos del controlador, hemos implementado un componente que hace de puente entre los elementos internos de la FPGA y un puerto serie síncrono para sistemas empotrados muy extendido, el Serial Peripheral Interface o SPI [Catsoulis05]. La idea es que el microcontrolador gestione los parámetros del control a través del puerto SPI. Estos parámetros son: modo de funcionamiento, el valor del generador de spikes, tiempos de hold, selección del banco de Integrate & Generate y del derivador de spikes, así como los divisores de frecuencia de cada uno de ellos, y el ancho de los spikes aplicados al motor.

El bus SPI (Serial Peripheral Interface) fue desarrollado por Motorola para realizar transmisiones serie síncronas en modo full-duplex entre un dispositivo maestro y periféricos esclavos, aunque también existen las arquitecturas multi-maestro. Los dispositivos SPI pueden trabajar en modo maestro (master), o esclavo (slave). El dispositivo que actúa como master debe proporcionar las señales de reloj así como activar las señales de selección del dispositivo esclavo con el cual quiere mantener una comunicación. Sólo pudiendo haber un dispositivo ejerciendo de master en el sistema, mientras que podemos tener tantos dispositivos esclavos como podamos necesitar [Catsoulis05].

Para poder seleccionar el parámetro adecuado en el que escribir, así como asignarle un valor, hemos introducido un bus de datos y direcciones en el interior de la FPGA, gestionado por un controlador SPI esclavo, de manera que mediante el bus de direcciones apuntaremos a

un registro concreto, y en el bus de datos le especificaremos el valor a escribir en el registro. En la Figura 5.7 mostramos el controlador SPI esclavo, junto el bus de datos / direcciones y los elementos de control. Siendo en este caso el bus de datos de 16bits y el de direcciones de 8bits. Además se ha incluido una línea de habilitación de escritura, la cual avisa a todos los elementos conectados al bus interno de la FPGA que hay una pareja de datos/direcciones válidas.

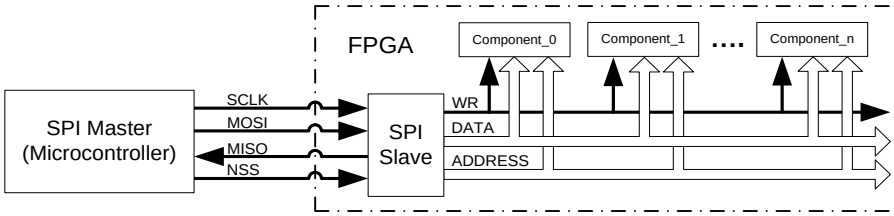


Figura 5.7: Arquitectura de datos y direcciones basadas en SPI

El controlador SPI esclavo está compuesto por un registro de desplazamiento y una FSM, cuyo diagrama de estados es mostrado en la Figura 5.8. La comunicación SPI comienza con la selección de esclavo (señal NSS a 0) por parte del microcontrolador. Una vez seleccionado, el controlador comienza a esperar los flancos adecuados de la señal SCLK (es este caso en el flanco de bajada) para capturar secuencialmente los bits presentes en la entrada de datos (MOSI, Master Output Slave Input), y desplazar el registro interno por cada bit. El registro interno está directamente mapeado al bus de datos y direcciones, así que una vez se han leído el número de bits adecuados (en este caso 24), sólo resta proporcionar un pulso en la señal WR para que el registro interno apuntado por el bus de direcciones almacene los 16 bits presentes en el bus de datos.

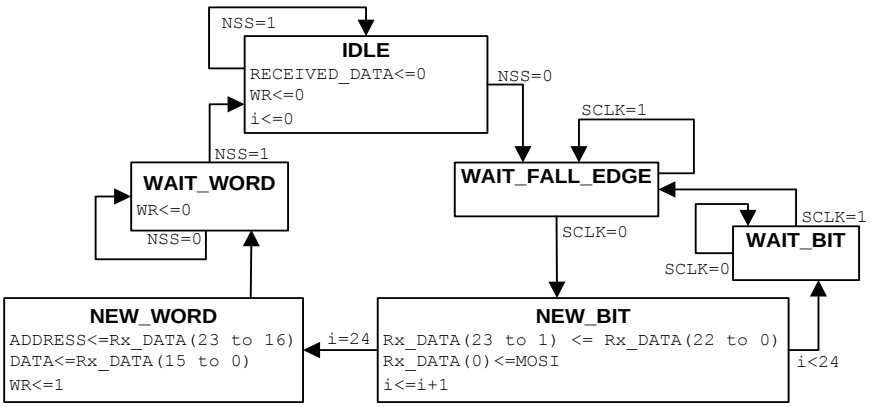


Figura 5.8: Máquina de estados finita del controlador SPI

Las tramas SPI que recibe el controlador tienen 3 bytes, 24 bits, en las que lo primero que se transmite es la dirección a escribir (un byte), y a continuación se envían los 16 bits de datos. De tal manera que cada vez que se reciban los 24 bits, estaremos apuntando a un registro del controlador en el bus de direcciones, y el valor a escribir en el registro apuntado se encontrará presente en el bus de direcciones.

23							16	15							8	7							0
ADDRESS (7 – 0)								MSB_DATA (15 – 8)								LSB_DATA (7 – 0)							

En la Figura 5.9 mostramos un cronograma del componente SPI esclavo capturado en el interior de la FPGA, de nuevo haciendo uso de ChipScope. Este cronograma muestra la transmisión de una trama de test, la cual pretende escribir en la dirección 0xAA el valor 0x00FF. En el cronograma, la comunicación comienza bajando la línea de selección de esclavo, NSS. Tras bajar NSS, comienza a oscilar la señal del reloj SPI (SCLK), y empiezan a escribirse bits en el bus por parte del master SPI (MOSI). Los bits recibidos por la línea MOSI son capturados en el flanco de bajada de SCLK, desplazando el registro de desplazamiento interno (Rx_SPI) un bit hacia la izquierda. Finalmente, cuando se han recibido los 3 bytes, en el bus de direcciones y datos aparecen los valores correctos, sólo resta proporcionar un pulso en la señal WR para que todos los componentes conectados a estos buses sepan que hay unos datos de dirección / datos válidos en el bus.

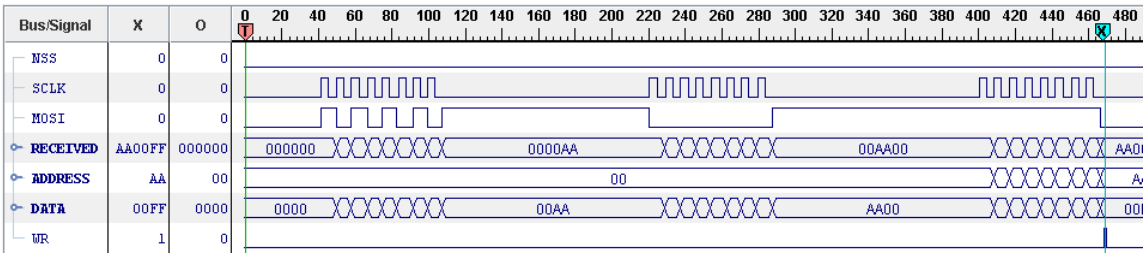


Figura 5.9: Cronograma del controlador SPI

Cada componente del control ha sido "envuelto" por un envoltorio, o wrapper, el cual contendrá los registros que almacenen los valores de los parámetros, así como la lógica combinacional necesaria para decodificar el bus de direcciones, tal y como muestra la Figura 5.10. Para organizar todos los componentes en el interior del control, hemos proporcionado a cada componente una dirección base, por ejemplo, si tenemos un Hold & Fire, con dos parámetros, y dirección base 0x80, podremos direccionarlos como 0x80+0 y 0x80+1 respectivamente. De tal manera que todos los componentes estarán siempre pendientes del bus de direcciones, y cuando en el bus haya una dirección perteneciente a uno de sus registros y un pulso en WR, el componente direccionado escribirá el contenido del bus de datos en el registro del parámetro adecuado.

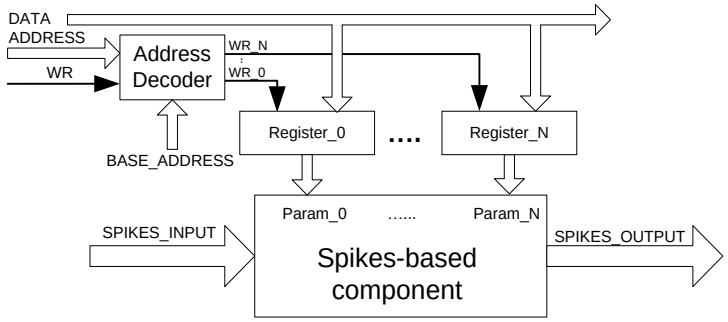


Figura 5.10: Envoltorio usado para parametrizar cada componente

Particularmente en esta implementación se han organizado los componentes de acuerdo a la Tabla 5-1. En primer lugar nos encontramos un registro de estado, en el que

seleccionamos el modo de funcionamiento del controlador (lazo abierto / cerrado), así como los elementos activos en lazo cerrado, el integrador y el derivador de spikes, y con el que podemos encender o apagar los LEDS de la AER-Robot. Como vimos anteriormente, el modo del controlador es seleccionado mediante el uso de un multiplexor, estando el 6º bit del registro de estado directamente conectado con la entrada de selección de canal del multiplexor, seleccionando así el modo deseado. Además los bits 4º y 5º están conectados a las señales de reset del Integrate & Generate y del derivador de spikes, de forma que manteniendo uno u otro componente podremos configurar el controlador en modo P, PI, PD o PID. En la siguiente dirección usada, la dirección 0x02, nos encontramos con el generador sintético de spikes bit-wise, en la que el dato proporcionado representa la entrada del generador, produciendo una secuencia de spikes con una frecuencia que puede ser calculada según la Ecuación [2-21]. A continuación nos encontramos al Spikes Expansor, el cual sólo tiene un parámetro, el ancho de spike con el que finalmente serán aplicados al motor, siendo el ancho de spikes calculable mediante la Ecuación [2-15]. En la posición 0x04 nos encontramos con el selector de banco del Integrate & Generate, seleccionando el banco deseado con el valor escrito en esta posición, que debe estar comprendido entre 0 y 3, al contener 4 bancos de Integrate & Generate. Si queremos modificar el valor de cada uno de ellos, tendremos que escribir en la dirección del selector de banco incrementándole el número de banco más uno, es decir, si queremos escribir un nuevo valor en el registro del divisor de frecuencia de un Integrate & Generate, tendremos que escribir en la dirección 0x05, para el del segundo banco en 0x06 y así sucesivamente. Inmediatamente después, en la dirección 0x09 están situado el banco de derivadores de spikes, el cual se comporta exactamente igual que el banco de Integrate & Generate. Tras los bancos situamos los tiempos de hold de los 3 Hold & Fire usados en el controlador, entre las posiciones 0x0E y 0x13, pudiendo modificar los tiempos a nuestro antojo acorde con la Ecuación [3-3]. Y finalmente el monitor AER, llamado monitor AER masivo, ha sido situado al final del mapa de direcciones, ya que es el componente que más direcciones ocupa. Aunque será descrito en profundidad en los siguientes apartados, en este apartado adelantamos que este componente necesita una dirección de memoria por cada spike a monitorizar a través del bus AER. En nuestro caso particular, tendremos 8 señales a monitorizar, compuesta cada una de ellas por dos spikes para representar el signo, así que necesitará 16 direcciones de memoria, de la 0x20 a la 0x2F. Cada una de estas direcciones representa la dirección (Address) de 16 bits asignada a cada spike monitorizado por el bus AER, es decir, si escribimos en la dirección 0x21 0x5555, cada vez que dispare un spike asociado a la primera dirección, en bus AER se disparará la dirección 0x5555, como veremos en los próximos apartados.

Tabla 5-1: Espacio de direcciones interno del controlador PID en lazo cerrado

Dirección SPI	Entidad	Comentarios
0x00	Registro de estado	[5]- Closed-loop [4]-D [3]-I [2-0] leds
0x01	Reservada	No usado
0x02	Generador de spikes Bit-Wise	Número de 16bits con signo
0x03	Spikes Expansor	Ancho de spike
0x04-0x08	Banco de Integrate & Generate	+0 Selección de banco, entre 0-3 +1 Banco 0: Divisor de frecuencia +2 Banco 0: Divisor de frecuencia +3 Banco 0: Divisor de frecuencia +4 Banco 0: Divisor de frecuencia
0x09-0x0D	Banco de derivadores de spikes	+0 Selección de banco, entre 0-3 +1 Banco 0: Divisor de frecuencia +2 Banco 0: Divisor de frecuencia +3 Banco 0: Divisor de frecuencia +4 Banco 0: Divisor de frecuencia
0x0E-0x0F	Hold & Fire (error)	Tiempos de hold
0x10-0x11	Hold & Fire (ID)	Tiempos de hold
0x12-0x13	Hold & Fire (PID)	Tiempos de hold
0x20-0x2F	Monitor AER masivo	Dirección AER de cada spike

5.2.4. Adaptación del Integrate & Generate y del derivador de spikes

De todos los componentes integrantes del controlador, hay dos componentes que comparten una peculiar característica, el Integrate & Generate y el derivador spikes. Esta característica consiste en que uno de sus parámetros, en concreto el ancho de bits, debe ser definido en tiempo de síntesis. Debido a que en base a este parámetro se han de sintetizar contadores de diversos anchos de bits, siendo imposible modificarlo una vez que el circuito ha sido sintetizado. Esta característica de estos componentes nos obliga a sintetizar todo el controlador cada vez que deseemos modificar este parámetro, consumiendo una gran cantidad de recursos y tiempo. La solución por la que hemos optado para solventar esta problemática ha sido sintetizar a la vez varios Integrate & Generate y derivadores de spikes con distintos números de bits, y agruparlos en bancos. Concretamente para este controlador hemos diseñado un banco con 4 Integrate & Generate de diversos bits. De tal manera que para ajustar la ganancia del Integrate & Generate o del derivador de spikes, seleccionaremos el componente con el número de bits adecuados, y realizaremos el ajuste final de la ganancia con el divisor de frecuencia de cada componente. La Figura 5.11 nos muestra el diagrama de bloques del banco de Integrate & Generate que hemos diseñado, en el que hemos añadido cuatro Integrate & Generate, de 12 a 18 bits, y un registro de estado. La idea es seleccionar, mediante el bus interno de la FPGA, en el registro de estado del banco el Integrate & Generate que queremos utilizar, de manera que el registro de estado mantendrá activas la señales de reset de todos los Integrate & Generate excepto del seleccionado en el registro de estado del

banco, sólo funcionando un Integrate & Generate al mismo tiempo. Como se aprecia en la figura, los spikes de entrada son aplicados a todos los Integrate & Generate, y sus salidas están conectadas a una puerta OR, de forma que los spikes de salida de cualquiera de los Integrate & Generate la podrán atravesar, sin embargo, al sólo estar activo un Integrate & Generate a la vez, sólo nos encontramos con los spikes del Integrate & Generate seleccionado. Además los Integrate & Generate han sido conectados al bus interno de la FPGA, de manera que podamos ajustar el divisor de frecuencia de forma independiente para cada uno de ellos. Un banco de semejantes características ha sido diseñado para el derivador de spikes, de manera que sólo hemos necesitado sustituir los Integrate & Generate por derivadores de spikes de diversos número de bits, de 16 a 22 bits en nuestro caso.

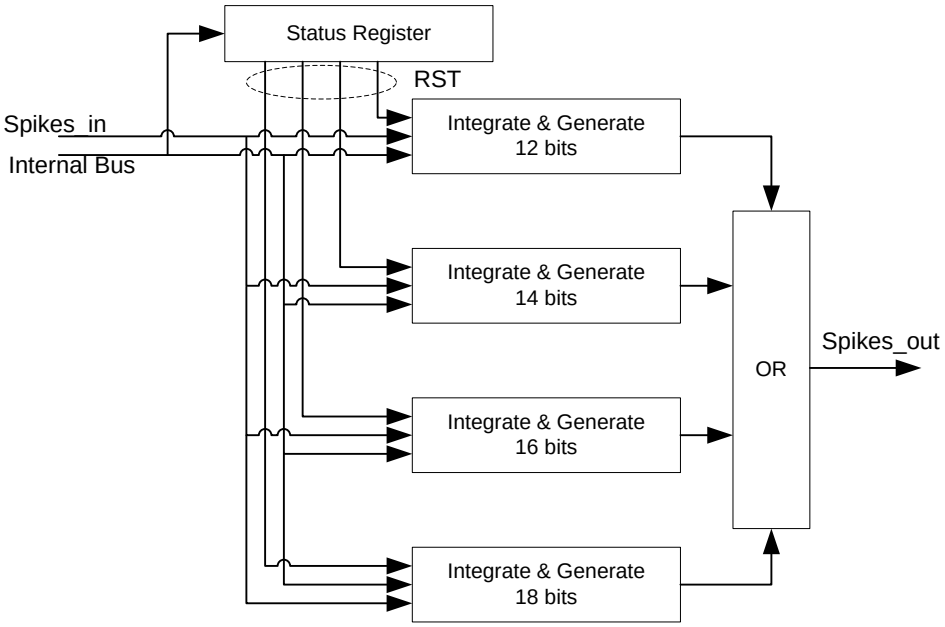


Figura 5.11: Diagrama de bloques del banco de Integrate & Generate

5.2.5. Monitorización del control mediante el uso del bus AER

Para monitorizar la actividad interna de los spikes que se disparan entre los diversos elementos del control, vamos a codificar cada spike según un espacio de direcciones AER [Boahen98], para ser transmitido a través del puerto AER, donde otra herramienta AER (la USBAERmini2 en este caso y expuesta en detalle más adelante), será la encargada de recogerlos y transmitirlos a un PC para su posterior procesamiento. Con este fin hemos diseñado un componente VHDL que implementa esta funcionalidad, el monitor AER masivo [Jimenez09b]. Este componente, cuyos elementos se muestran en la Figura 5.12, ha sido diseñado para hacer "fotografías" del valor de los buses de los spikes, codificar los spikes mediante la representación AER, y enviar los eventos AER a través del puerto AER paralelo. En nuestro diseño todas las señales de spikes están conectadas a la entrada de este componente como se aprecia en la Figura 5.12. Este componente usa dos memorias estructuradas como memorias FIFO (First Input - First Output), la idea de la primera FIFO, la FIFO de spikes, es almacenar las "fotografías" de los spikes, de manera que los spike son alineados como palabras de un tamaño en bits de 2 por el número de spikes a monitorizar (hemos de tener en

cuenta tanto los spikes positivos como los negativos), además vamos a hacer una operación OR entre ellos, cuya salida está conectada a la señal de escritura (WR) de la FIFO de spikes. De tal manera, que en caso de que se dispare un solo spike, tomamos una fotografía de todos los spikes, ya que se disparan en paralelo, pudiendo tener más de un spike al mismo tiempo, y el bus AER los comunica secuencialmente. Así que en esta FIFO se almacenarán "fotografías" de los spikes, y estando la FSM (situada debajo de la FIFO de los spikes en la Figura 5.12, y cuyo diagrama de estados se muestra a la izquierda) encargada de ir tomando cada "fotografía" para recorrerla bit a bit, o spike a spike. De forma que cada vez que se encuentre un 1 lógico en una posición determinada, significará que se ha disparado un spike, debiendo obtener la dirección AER asociada a la posición de ese spike de la memoria ROM de mapeo, y almacenarla en la FIFO de eventos AER. La FIFO de eventos AER es en la que se van almacenando los eventos AER a ser transmitidos a través del bus AER. Finalmente, hemos conectado una FSM (a la derecha de la figura) entre la FIFO de eventos AER y el puerto AER paralelo de salida, esta FSM se encarga de ir tomando los eventos de la FIFO AER, e ir transmitiéndolos usando el protocolo AER paralelo asíncrono. En resumen, los spikes que se disparan en paralelo son almacenados en la FIFO de los spikes, después son procesados secuencialmente por la primera FSM, almacenando de manera secuencial su dirección AER en la FIFO AER, y finalmente comunicando los eventos AER uno a uno a través del puerto AER paralelo, usando el protocolo de *hand-shake* asíncrono que fue establecido para el proyecto CAVIAR [Häfliger04] [Serrano09].

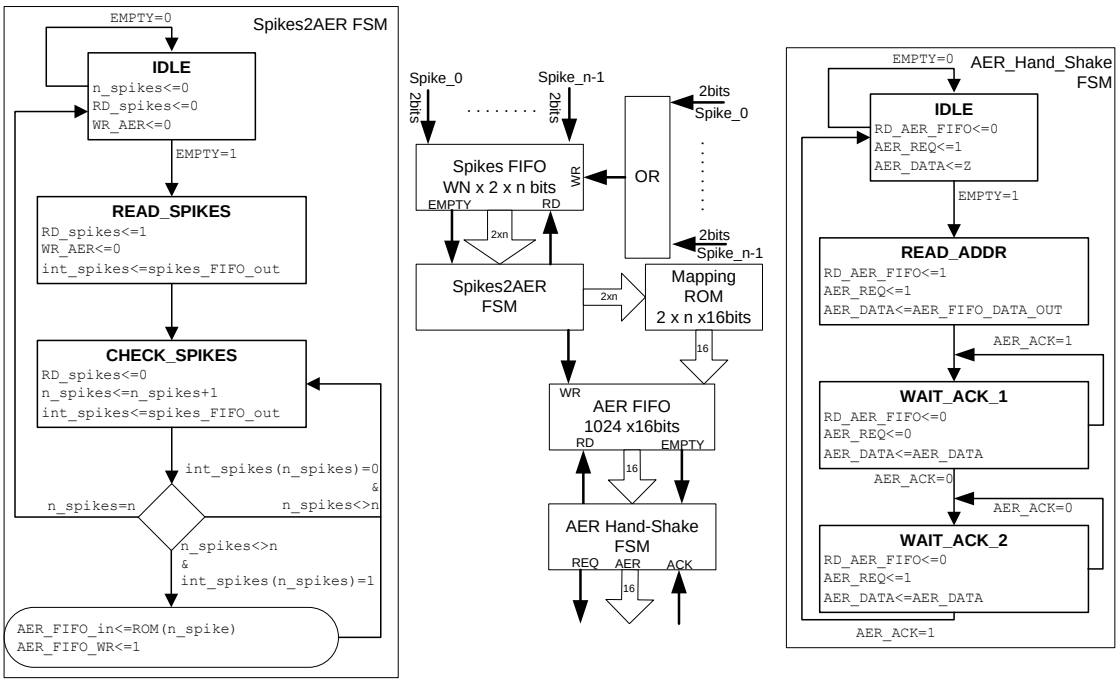


Figura 5.12: Estructura interna del monitor AER masivo

En la Figura 5.13 mostramos un cronograma de este componente capturado desde el interior de la FPGA. Todo comienza cuando los spikes se disparan y son capturados por la FIFO de spikes (1). En caso de que la FIFO de los spikes deja de estar vacía, la máquina de estados *Spikes2AER* comienza a funcionar, en primer lugar lee los spikes capturados de la FIFO y los almacena en un registro interno (2). A continuación, ciclo a ciclo recorremos los spikes

capturados (3), y por cada spike disparado, buscaremos en la memoria de mapeado su dirección AER, introduciéndola en la FIFO AER (4). Finalmente, la máquina de estados AER Hand-Shake se encarga de ir cogiendo los eventos AER de la FIFO AER y comunicarlos a través del puerto AER mediante el protocolo de *hand-shake* (5) asíncrono de 4 fases antes mencionado [Häfliger04]. Como se aprecia en el cronograma, desde que se dispara un spike hasta que el monitor baja la línea de Request del bus AER, pasan 7 ciclos de reloj en el mejor de los casos, y 9 en el peor, ya que este retardo depende de la posición del spike a la entrada del monitor.

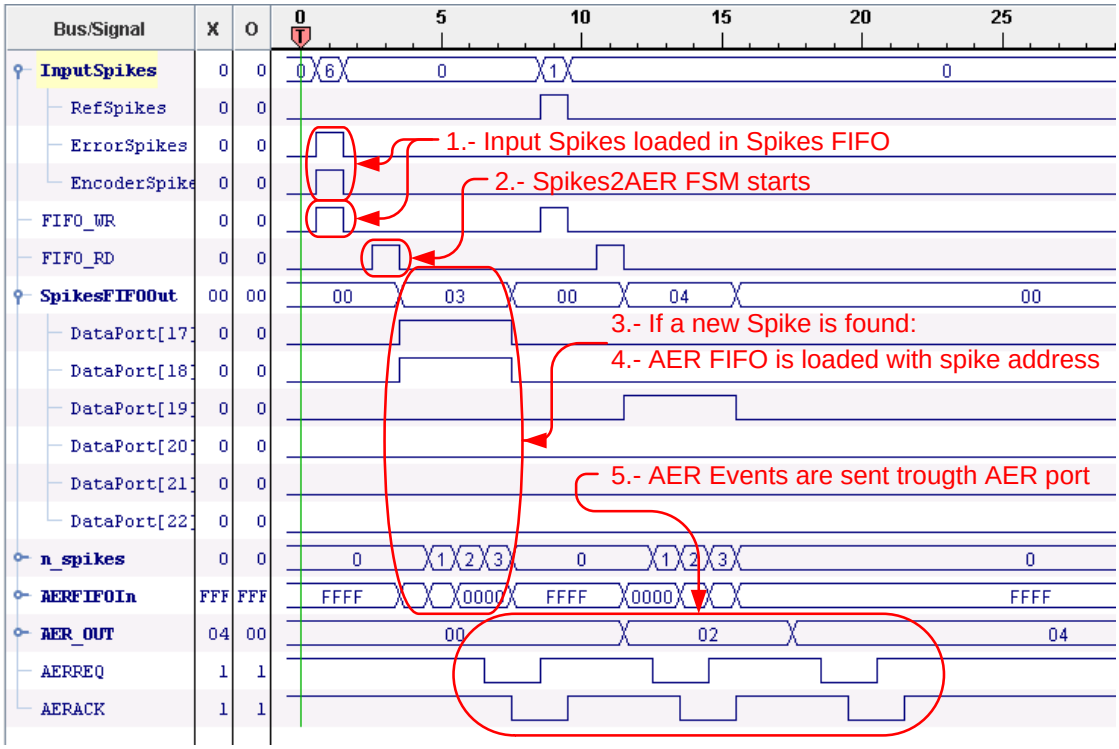


Figura 5.13: Cronograma del monitor AER masivo

Como expusimos en el aparatado anterior, este componente es completamente programable, pudiendo asignar una dirección AER a una señal de spikes a nuestro antojo. Para ello este componente ha sido conectado al bus de datos / direcciones interno de la FPGA, estando la organización de los registros en detalle la Tabla 5-2. En ella podemos ver cómo existe una sub-dirección para cada señal de spikes, tanto para los spikes positivos como negativos, a partir de la dirección base, BASE_ADDRESS, proporcionada al monitor AER masivo. Cada dirección toma por defecto el valor de su desplazamiento, es decir, la dirección alocada en la posición BASE_ADDRESS+0x00 valdrá 0, y la alojada en BASE_ADDRESS+0x09, tomará el valor 9. Aunque el controlador PID presentado a continuación sólo necesita monitorizar 14 spikes, 7 señales, más adelante presentaremos un controlador PID de posición, no de velocidad como hemos venido haciendo hasta ahora, y dicho controlador necesitará dos direcciones más la posición del eje del motor.

Tabla 5-2: Espacio de direcciones detallado del monitor AER del controlador PID basado en spikes

Direccion SPI	Señal de control	Valor de reset
BASE_ADDRESS +0x00	Speed Ref pos.	0
+0x01	Speed Ref neg.	1
+0x02	Speed Error pos.	2
+0x03	Speed Error neg.	3
+0x04	Motor Speed pos.	4
+0x05	Motor Speed neg.	5
+0x06	Integrate & Genrate pos.	6
+0x07	Integrate & Genrate neg.	7
+0x08	Spikes Derivative pos.	8
+0x09	Spikes Derivative neg.	9
+0x0A	Int. Gen. + Spikes Der. pos.	10
+0x0B	Int. Gen. + Spikes Der. neg.	11
+0x0C	Int. Gen. + Spikes Der. + error pos.	12
+0x0D	Int. Gen. + Spikes Der. + error neg.	13
+0x0E	Motor Position pos.	14
+0x0F	Motor Position neg.	15

5.2.6. Resultados de la síntesis del controlador PID basado en spikes

Una vez diseñados y ensamblados todos los componentes del controlador PID basado en spikes, hemos procedido a sintetizarlo. Un sumario de la síntesis del sistema es mostrado en la Tabla 5-3. En él podemos observar el uso de los recursos de la FPGA. El controlador diseñador consume 1737 slices de los 3584 que contiene nuestra Spartan 3, usando el 48% de los bloques lógicos de la FPGA, de los que el 13% son usados para las memorias FIFO del monitor de spikes, pudiendo operar como máximo a 98.34MHz. Este consumo es, a priori, tan elevado por el uso de los elementos periféricos del controlador, como es el controlador SPI esclavo, o el monitor AER, además hemos necesitado introducir bancos de integradores y derivadores, sintetizando cuatro veces el mismo componente aunque sólo se use una vez. Resultando evidente que la programabilidad y posibilidad de testeo introduce un elevado coste en recursos hardware

Tabla 5-3: Resultados de la síntesis del controlador PID en lazo cerrado

Number of BUFGMUXs	1 out of 8	12%
Number of External IOBs	37 out of 141	26%
Number of LOCed IOBs	37 out of 37	100%
Number of Slices	1737 out of 3584	48%
Number of SLICEMs	236 out of 1792	13%

5.2.7. Uso del microcontrolador como gestor del control

Toda la responsabilidad del control ha sido delegada a la FPGA, sin embargo, como hemos ido exponiendo continuamente a lo largo de este trabajo, los controles presentados tienen multitud de parámetros, cuya variación con motivos experimentales es muy interesante. Para gestionar los controles en particular, y todos los componentes de la FPGA en general, hemos hecho uso del microcontrolador incluido en la AER-Robot. El microcontrolador está conectado al puerto SPI de la FPGA, de hecho él es el elemento SPI Master del bus. A través del puerto SPI escribiremos en los registros internos de la FPGA, como expusimos en el apartado anterior. Dado que el microcontrolador incluido en la AER-Robot implementa un puerto USB 2.0 Full-Speed, vamos a asignarle la responsabilidad de hacer de puente SPI-USB, programándolo para tales efectos. De tal manera que desde un PC le enviemos paquetes USB con los valores deseados de los parámetros, y el microcontrolador los emitirá a través del puerto SPI.

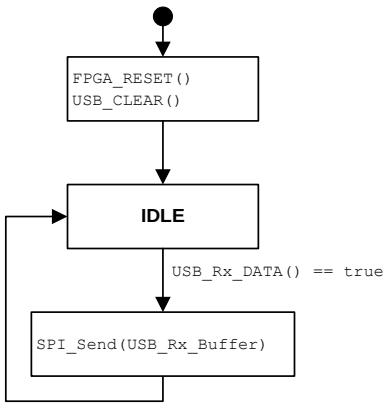


Figura 5.14: Máquina de estados del microcontrolador

La Figura 5.14 muestra el diagrama de estados del firmware del microcontrolador. El microcontrolador, al inicializarse, espera a que la FPGA descargue el firmware desde su EEPROM mediante una bandera. Una vez que la FPGA tiene descargado su firmware, le proporciona un reset global a los elementos de control, para que se queden inicializados. Después de resetear el firmware de la FPGA, espera a recibir un paquete USB. Tras recibir alguno lo envía a través del puerto SPI. Como expusimos en el apartado referente al

controlador SPI esclavo de la FPGA, el microcontrolador envía los bytes de 3 en 3, primero la dirección del registro destino, y luego 2 bytes de datos. Cómo los buffers del puerto USB son de 64 bytes, y normalmente no los usamos enteros, hemos añadido una dirección especial, 0xFB, que indica el final del buffer "útil". Si el microcontrolador la encuentra, éste descarta el paquete USB recibido, finalizando las transmisiones a la FPGA, y volviendo al estado de espera o IDLE.

5.3. Monitorización de la información AER: La tarjea USBAERmini2

Para monitorizar los spikes que se disparan en el interior de la AER-Robot, su visualización y posterior análisis, se ha usado una tarjeta USBAERmini2. Esta tarjeta es un puente completo entre el bus AER y el bus USB de un PC [Berner08]. Este dispositivo permite secuenciar (reproducir) y monitorizar (capturar) tráfico AER con una resolución temporal tanto de 1uSec como 0.2uSec. Puede ser tanto insertada como monitor entre dos dispositivos AER (modo pass-through), o puede ser usada en modo terminal (como haremos en el siguiente apartado).

La USBAERmini2 tiene una arquitectura basada en microcontrolador más CPLD. El microcontrolador, fabricado por Cypress y de la familia FX2LP [CypressFX2], incluye un puerto USB2.0 high-speed (velocidad máxima de 480Mbits). La CPLD ha sido fabricada por Xilinx, en particular pertenece a la familia Coolrunner 2 [CoolRunner2], y tiene 256 macroceldas. En la CPLD hay implementadas cuatro FSM, para manejar los puertos AER (tanto de envío como de recepción), generar marcas de tiempo (timestamps), y para leer / escribir las FIFOs USB del FX2LP.

En la Figura 5.15 se muestra el diagrama de bloques de la USBAERmini2. Para generar los timestamps se usa un contador de 14bits. Cada vez que se desborda, manda un mensaje al PC, el cual incrementa otro contador en el PC de 18bits, teniendo así un timestamp con una resolución de 32bits. La CPLD tiene conectado un reloj de 30MHz, obteniendo una tasa de monitorización de 6 mega-eventos AER por segundo de pico, pudiendo mantenerse a una frecuencia constante de 4.5 mega-eventos, aunque estas tasas están limitadas por la capacidad del PC. La máxima frecuencia del secuenciador es de 3.75 mega-eventos. Este dispositivo fue diseñado para ser simple y barato de fabricar, así como simple de utilizar, es puramente plug-and-play. Podemos conectar a una cadena AER varias USBAERmini2, pudiendo sincronizarlas entre ellas para obtener timestamps sincronizados.

En el PC nos encontramos el jAER, este software es un interface entre la USBAERmini2 y MATLAB. jAER es un proyecto de código libre, u *opensource*, escrito en java, el cual permite el interfaz en dispositivos AER, permitiendo la visualización y procesamiento de la información AER [jAER]. Las clases del jAER son accesibles desde MATLAB, pudiendo así estimular dispositivos AER y capturar información AER fácilmente.

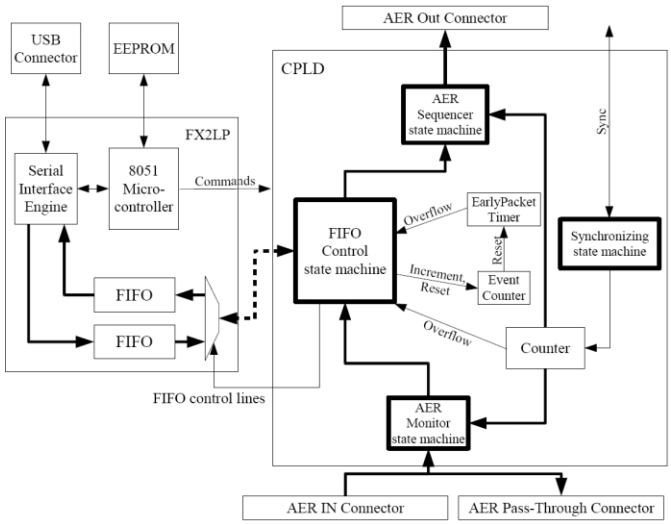


Figura 5.15: Diagrama de bloques de la USBAERMini2

5.4. Excitación, monitorización y análisis del control

En este apartado vamos a analizar la respuesta de un motor real usando nuestros controles basados en spikes. Para ello vamos a analizarlo teóricamente, para más adelante estudiar su comportamiento en lazo abierto y lazo cerrado con diversos controladores, y ante diversas funciones de entrada.

5.4.1. Análisis teórico del modelo del motor

Vamos a comenzar con el análisis teórico del modelo de motor usado en nuestros experimentos. Para ello primero lo analizaremos en lazo abierto. El motor usado ha sido fabricado por Maxon Motors, en concreto un modelo perteneciente a la familia A-max26 [Amax26] junto con una reductora planetaria del mismo fabricante [GP26], y sus parámetros más importantes se muestran en la siguiente tabla:

Tabla 5-4: Parámetros del motor real usado para los experimentos

Parámetro	Nombre	Valor	Unidades
Voltaje Nominal	-	12	V
Máxima Velocidad	-	726.3	rad/s
Constante de Tiempo	-	176.1	Ms
Frecuencia 0dB	-	3.77	kHz
Resistencia Terminal	R	2.5	Ohm
Inductancia Terminal	L	0.2276e-3	H
Constante de Par	Kp	13.9e-3	Nm/A
Constante de Velocidad	Kv	13.9e-3	V/(rad/s)
Inercia del Rotor	J	14e-7	Kgm2
Coeficiente de fricción	B	1.48e-5	Nm/(rad/s)
Relación de reducción	-	1:13	-
Resolución del encoder	-	500	Pulsos por vuelta
Frecuencia máxima el encoder	-	500	kHz

Según las ecuaciones Ecuación [2–1] y los parámetros proporcionados por el fabricante, la función de transferencia del motor usado en las simulaciones es la siguiente:

$$M(s) = \frac{\omega_{rad/sec}}{V_{motor}} = \frac{K_p}{LJs^2 + (RJ + LB)s + (RB + K_p K_v)}$$
$$= \frac{13.9 * 10^{-3}}{3.186 * 10^{-10}s^2 + 3.506 * 10^{-6}s + 2.297 * 10^{-4}}$$

Ecuación [5-1]

Mostrando el diagrama de Bode de este motor en la siguiente figura:

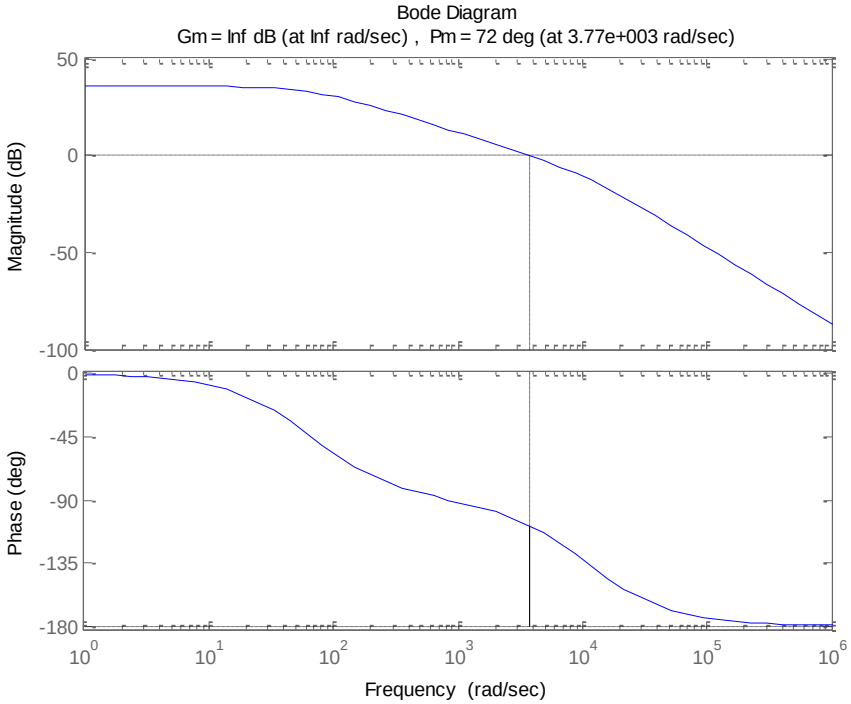


Figura 5.16: Diagrama de Bode del motor analizado

Como se expuso en el capítulo anterior durante el análisis del modelo teórico del motor de las simulaciones, podemos descomponer la función de transferencia del motor como:

$$M(s) = 60.5244 \frac{7.2097 * 10^5}{s^2 + 1.1 * 10^4 s + 7.2097 * 10^5}$$

Ecuación [5-2]

Analíticamente podemos extraer los siguientes parámetros del sistema:

$$K_{Motor} = \frac{K_p}{RB + K_p K_v} = 60.5244\ V\ {}^{rad}/_{sec}$$

$$\omega_n = 849.1\ {}^{rad}/_{sec}$$

$$\delta = 6.4775$$

Este motor tiene un coeficiente de amortiguamiento (δ) mayor que 1, mostrando un comportamiento sobre amortiguado en lazo abierto, ofreciendo la siguiente respuesta ante un escalón unitario:

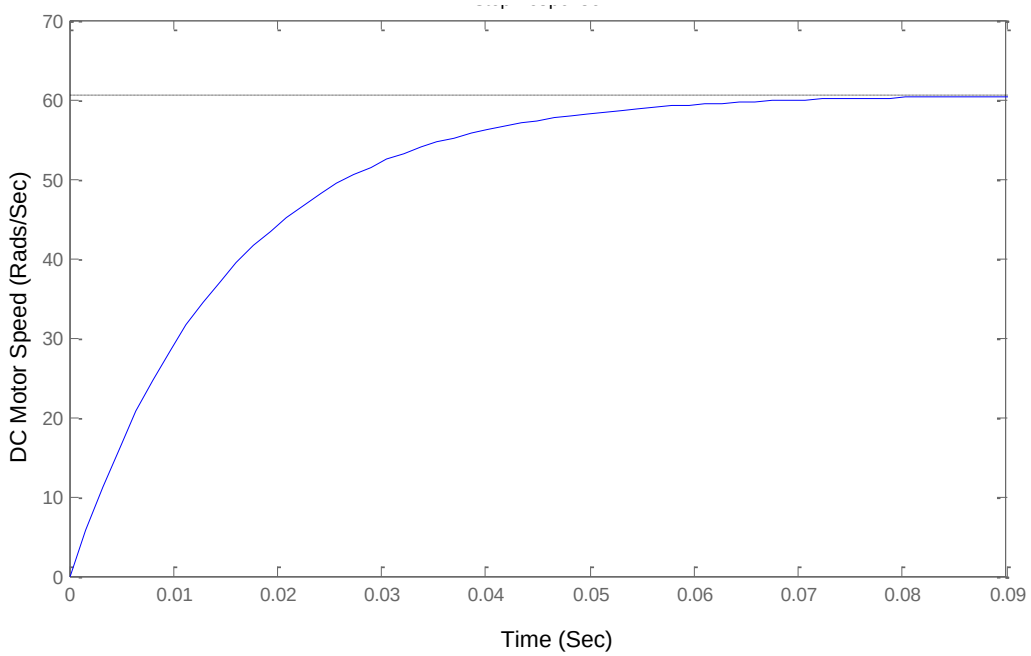


Figura 5.17: Respuesta teórica del motor ante un escalón ideal

5.4.2. Análisis experimental en lazo abierto

Para la excitación y monitorización del control hemos usado MATLAB, de tal forma que por un puerto USB nos comunicamos con el microcontrolador de la AER-Robot, mediante el cual generamos los spikes de referencia del sistema y ajustamos los parámetros del control. Simultáneamente por otro puerto USB conectamos al PCB la USBMiniAER2, usada como monitor del tráfico AER del controlador 0.

En primer lugar vamos a analizar el comportamiento del motor en lazo abierto ante una entrada en escalón, para una entrada con frecuencia constante y distintos anchos de spikes. Para generar el escalón de frecuencia sólo tendremos que escribir en el generador de spikes (dirección 0x00) el valor de la frecuencia del escalón deseado. A continuación mostramos el script de MATLAB utilizado para esta tarea. El script comienza inicializando la tarjeta AER-Robot y las variables. Luego se introduce en un bucle temporal, en el que esperará el tiempo de monitorización que deseemos. La primera vez que se ejecute el bucle se escribirá en el registro del generador de spikes, generando así el escalón de entrada. Por otro lado a lo largo del tiempo de monitorización hemos de ir capturando el tráfico proveniente de la USBAERmini2, en el que se encuentra toda la información presente en el controlador. Finalmente, capturada la información AER, sólo queda convertirla en números discretos y presentarla por pantalla.

```
AER_ROBOT_STARTUP  %Inicialización AER-Robot
AER_ROBOT_UPLOAD_CONFIG(dev,[SPI_SPIKES_EXP;1;0]) %Fijamos el ancho de
spike
monitortime=2.5; %Tiempo de monitorizacioin
if (isempty(usb0)) %Inicialización USBAERmini2
 usb0=usb.CypressFX2MonSeqFactory.instance.getFirstAvailableInterface;
end
if ~usb0.isOpen()
       usb0.open
end
usb0.setOperationMode(1);
%Inicialización variables
inaddr=[]; %%Address
ints=[];   %%TimeStamps
ftime=0;
%Activamos la captura de eventos AER
usb0.setEventAcquisitionEnabled(true);
tic %Bucle Temporal
while toc<monitortime
    %Capturamos los paquetes de la USBAERmini2
    inpacket=usb0.acquireAvailableEventsFromDriver.getPrunedCopy();
    %Y añadimos la nueva información AER a la ya capturada
    ints=[ints; inpacket.getTimestamps()];
    inaddr=[inaddr; inpacket.getAddresses()];
    %Si es la primera vez que se ejecuta el  bucle,
    if(ftime==0)    %activamos el generador de spikes
        AER_ROBOT_UPLOAD_CONFIG(dev,[SPI_SPIKES_GEN;0;150]);
        ftime=1;
    end
 end

%Paramos el generador de spikes
AER_ROBOT_UPLOAD_CONFIG(dev,[SPI_SPIKES_GEN;0;0]);
%Paramos la USBAERmini2, y leemos los últimos paquetes
inpacket=usb0.stopMonitoringSequencing();
inaddr=[inaddr; inpacket.getAddresses()];
ints=[ints; inpacket.getTimestamps()];
 %Convertimos el tráfico AER a números discretos
[realTime,TotalDiscreteValues]=spikes2discrete;
 %Presentamos la frecuencia de los spikes en pantalla
plot(realTime,totalDiscreteValues(:,:))
```

En la Figura 5.18 mostramos el resultado de ejecutar este script iterativamente, pero modificando el ancho de spike. Hemos fijado la frecuencia de los spikes de entrada a 152.59kSpikes/Sec, y variado el ancho de spike desde 4uSec hasta 94.uSec. Como predecían las ecuaciones teóricas, y mostraban los resultados de las simulaciones, la ganancia de los motores reales puede ser modificada variando el ancho de spike.

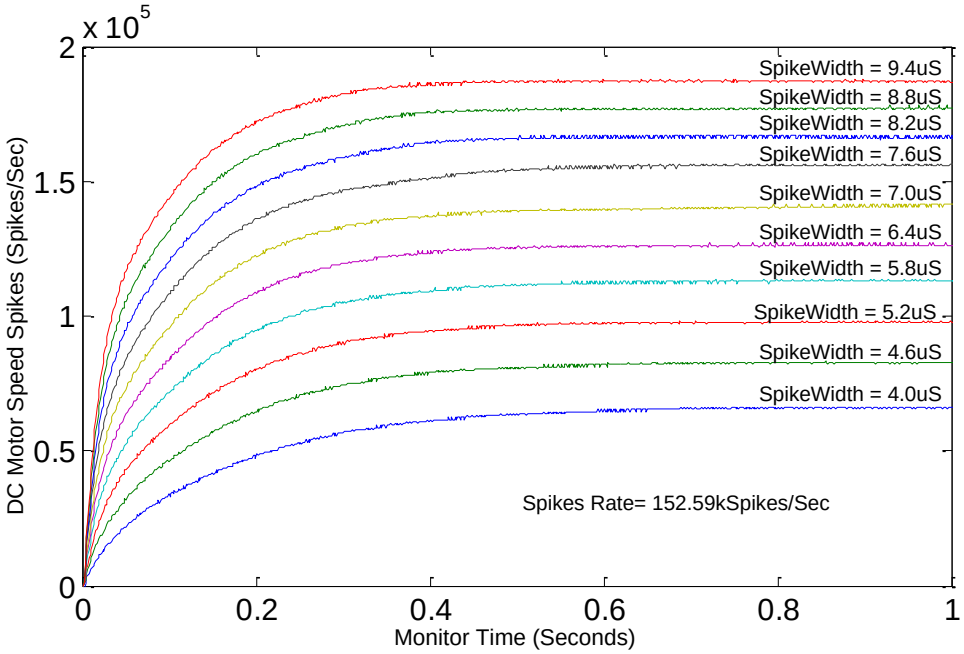


Figura 5.18: Respuesta del motor en lazo abierto para una tasa de spikes constante y ancho de spike variable

Tal y como ya hicimos en capítulos anteriores cuando analizábamos las simulaciones en lazo abierto, vamos a contrastar la ganancia teórica del sistema con la experimental. En la Figura 5.19 mostramos la velocidad en régimen estacionario para una frecuencia de spikes fija, respecto del ancho de spike usado. En azul mostramos la velocidad del motor experimental, la cual no es todo lo parecida que debiera con la velocidad teórica, en verde. Sin embargo, al realizar una regresión lineal de la velocidad experimental, en rojo, ésta no resulta ser tan distinta de la velocidad teórica. Constatando así que el modelo del motor usado no es más que precisamente un modelo, con no idealidades, pero sin embargo, en el caso medio, viene a ser similar con la teórica.

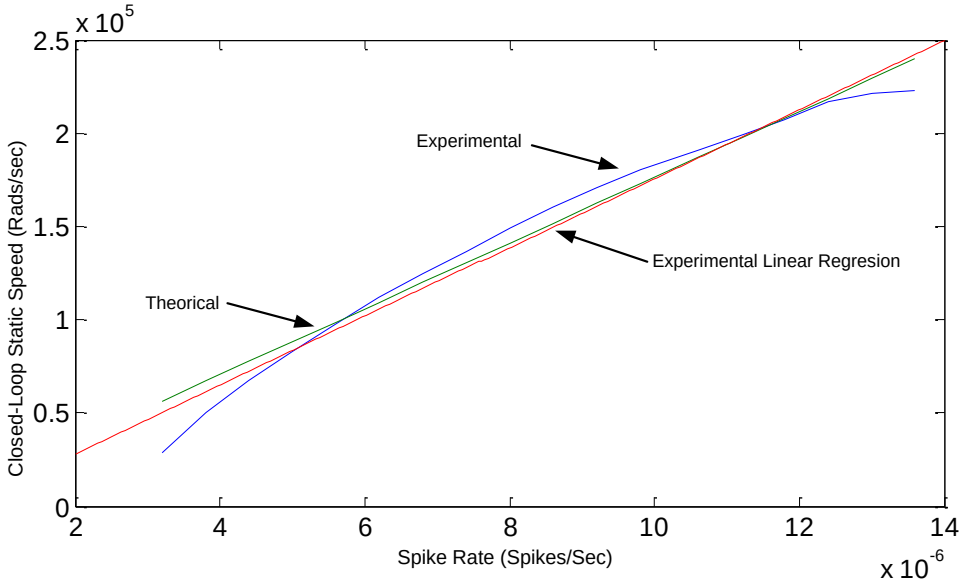


Figura 5.19: Comparativa entre los resultados teóricos y reales de la velocidad estacionaria del motor

Para comprobar la linealidad de la velocidad del motor respecto de la frecuencia de spikes de entrada hemos excitado el motor con diversas frecuencias de spikes (desde 22.8kSpikes/Sec hasta 53.4kSpikes/Sec) y un ancho de spike constante (5.12uSec), mostrando los resultados en la Figura 5.20.

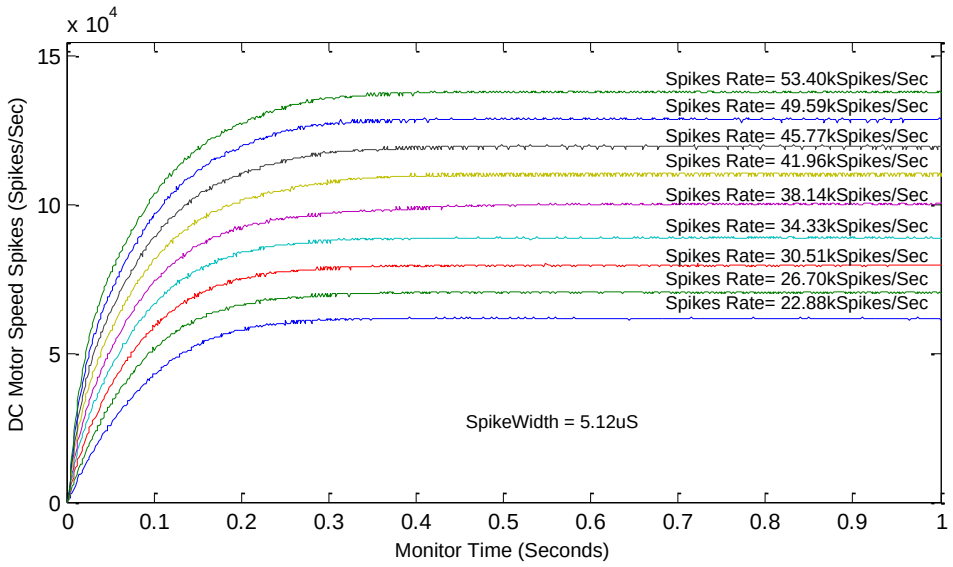


Figura 5.20: Respuesta del motor en lazo abierto par a ancho de spike constante y un spike rate variable

La Figura 5.21 compara la velocidad en régimen estacionario del motor experimental (azul) con la teórica (verde). Como puede observarse ambas tienen una pendiente parecida pero están desplazadas verticalmente, como se mostraba en la Figura 5.19, la ganancia no es del todo lineal con el ancho de spike, así que tomando esa ganancia experimental, hemos calculado la velocidad pseudo-teórica, en rojo, la cual parece que concuerda bastante bien con la experimental. Debiéndose este efecto a que la circuitería de aislamiento y potencia acortan siempre en la misma medida los pulsos dependiendo de su anchura, introduciendo esta no idealidad, siempre de la misma manera para cada determinado ancho de spike.

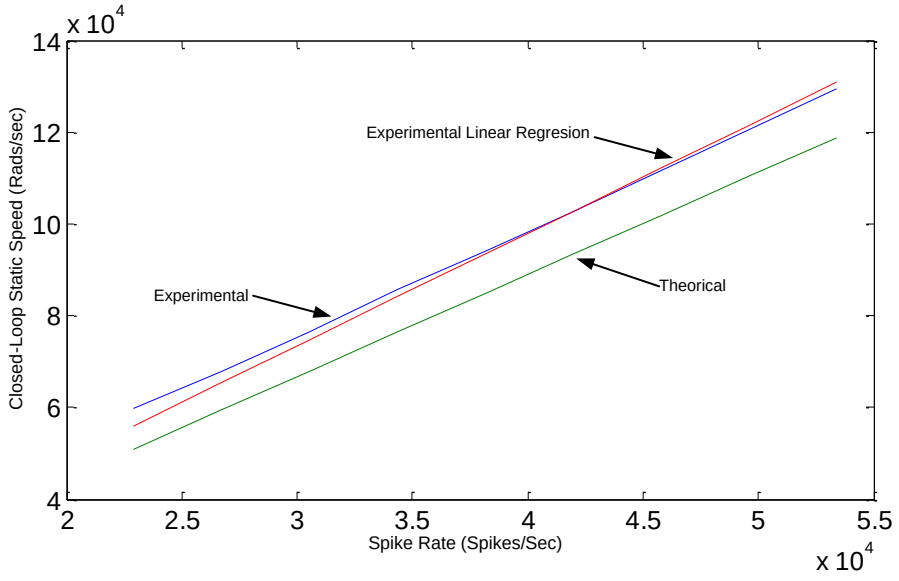


Figura 5.21: Comparativa entre los resultados teóricos y reales de la velocidad estacionaria del motor

Finalmente, hemos extraído la característica estática del motor mediante el uso del Spikes Expansor, mostrada en la Figura 5.22. Para ello hemos realizado un barrido tanto en frecuencia como en ancho de spikes. En los ejes horizontales encontramos el ancho de spikes (izquierda) y la frecuencia de los spikes (derecha). El eje vertical, Z, representa la velocidad del motor codificada en spikes. Observamos la región de operación lineal, así como una meseta que representa la saturación del Spikes Expansor con anchos de spike elevados.

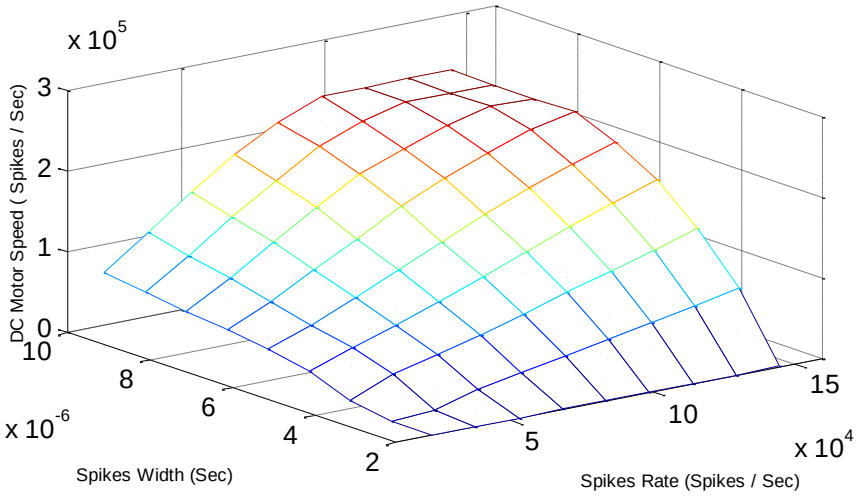


Figura 5.22: Velocidad estacionaria del motor para diversas frecuencias y anchos de spike

5.4.3. Análisis experimental del controlador P basado en spikes

En este apartado vamos a analizar las respuestas del motor en lazo cerrado usando un controlador P. Para ello vamos a modificador el registro de estado de la FPGA, fijando el Integrate & Generate y el derivador de spikes en estado de reset. A continuación vamos a excitar el motor con diversas funciones de excitación (escalón, rampa y seno), además de analizar la actividad interna de los elementos de control para verificar su funcionamiento.

En la Figura 5.23 mostramos la respuesta del motor en lazo cerrado ante una entrada en escalón. Hemos aplicado un escalón, o referencia de velocidad, de 185kSpikes/Sec, para dos anchos de spike (10uSec y 75uSec). En la figura podemos ver la referencia de velocidad (azul), las velocidades reales del motor (rojo y amarillo), y el error cometido en cada instante (verde y morado). Tal y como predecía el simulador, el sistema parece comportarse como un control proporcional en lazo cerrado, en él se ha reducido el tiempo de subida, y siempre aparece un error en régimen permanente.

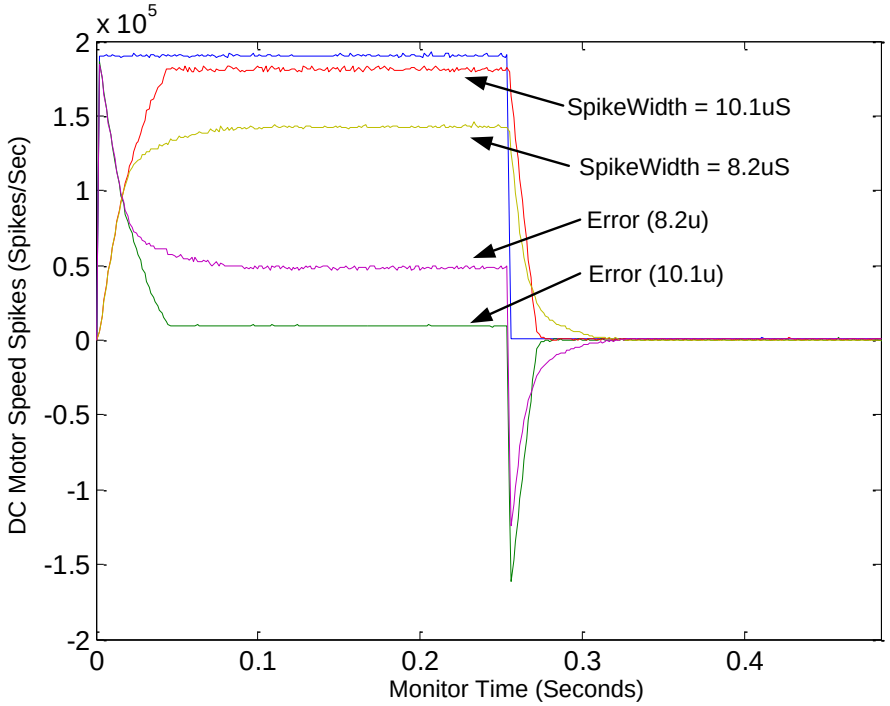


Figura 5.23: Respuesta del motor en lazo cerrado para dos anchos de spike distintos, incluido el error

Para analizar el efecto del ancho de spike en el sistema hemos proporcionado una referencia de velocidad constante (91.55kSpikes/Sec) y diversos anchos de spike (desde 3.2uSec hasta 13.4uSec), mostrando los resultados en la Figura 5.24. En la figura se aprecia cómo el ancho de spike afecta a la ganancia estática del sistema, tal y como el simulador predecía.

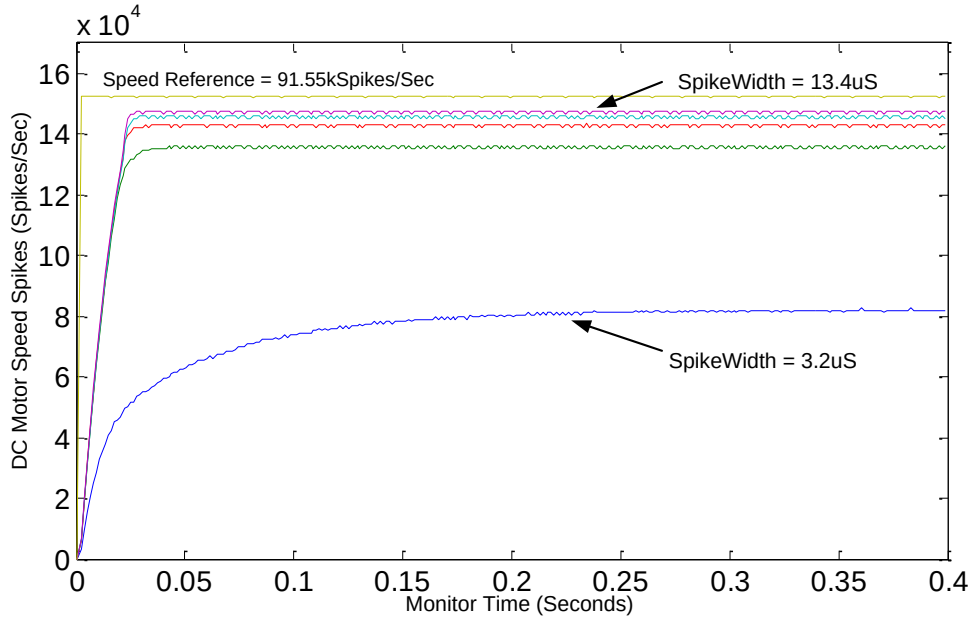


Figura 5.24: Respuesta del motor en lazo cerrado para un spike rate constante y un ancho de spike variable

La Figura 5.25 muestra la ganancia estática del sistema, eje Y, en lazo cerrado para diversos anchos de spike, eje X. En azul mostramos la ganancia experimental del sistema, en verde la ganancia teórica, y en rojo la ganancia teórica del sistema basada en los resultados experimentales del apartado anterior. Esta figura parece constatar que el sistema se comporta como control en lazo cerrado y que efectivamente la ganancia del control puede ser fijada con el ancho de spike, pero sin embargo, hay que tener en cuenta la no linealidad detectada.

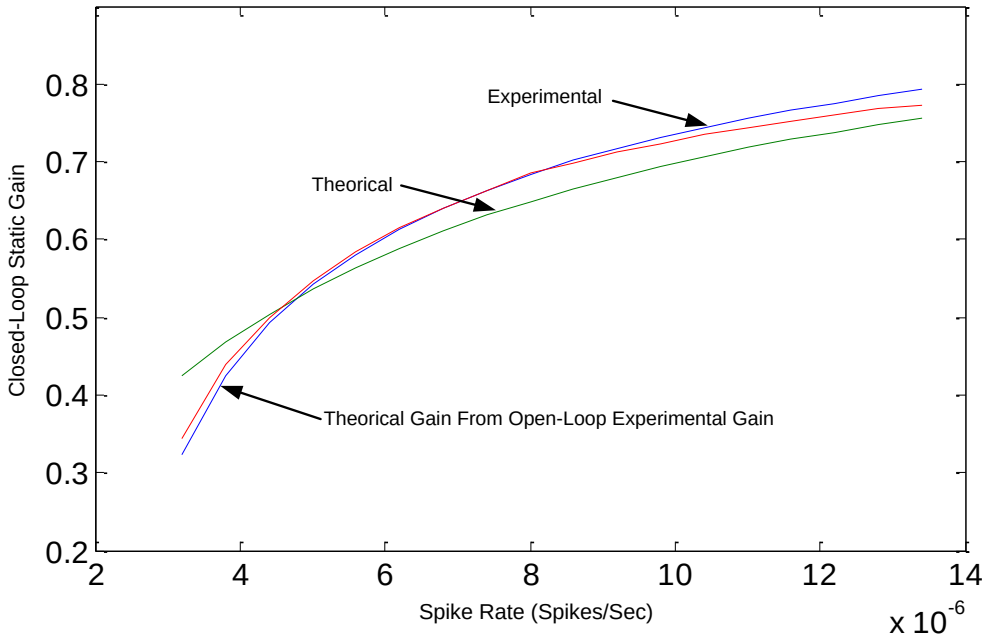


Figura 5.25: Comparativa entre los resultados teóricos y reales de la ganancia estática del motor en lazo cerrado

Para comprobar la linealidad de la ganancia del sistema frente a la frecuencia de referencia, hemos realizado otro conjunto de capturas, en las que la frecuencia de referencia es variable (desde 7.63kSpikes hasta 68.6 kSpikes) y el ancho de spike es constante (5.12uSec), en la Figura 5.26. En ella vemos como la velocidad del motor intenta alcanzar la velocidad de referencia, pero nunca llega a ella porque la ganancia equivalente en lazo cerrado es menor que 1.

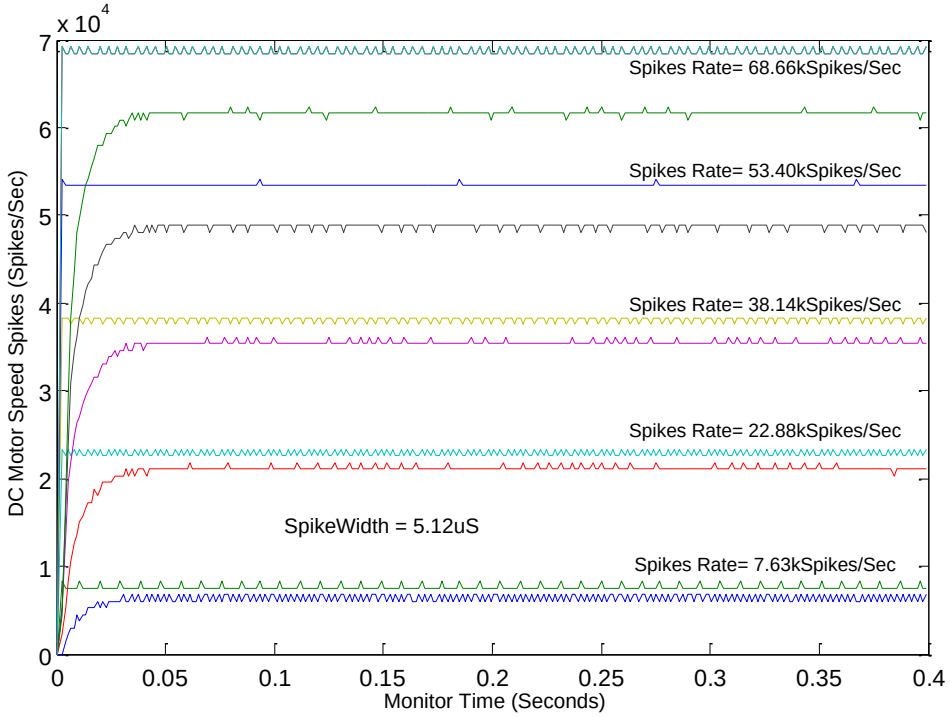


Figura 5.26: Respuesta del motor en lazo cerrado par a ancho de spike constante y un spike rate variable

La Figura 5.27 muestra la velocidad estacionaria alcanzada por el motor para cada frecuencia de referencia. En ella vemos la velocidad experimental (azul), la velocidad teórica (rojo), y la velocidad teórica obtenida a partir de la ganancia en lazo abierto experimental (verde). Aunque el comportamiento de la velocidad experimental debería ser una línea recta, no lo es del todo, comportándose ligeramente como una parábola. Al igual que en casos anteriores, la velocidad teórica no corresponde completamente con la experimental, sin embargo, si calculamos la velocidad teórica desde la ganancia en lazo abierto experimental obtenemos un mejor ajuste.

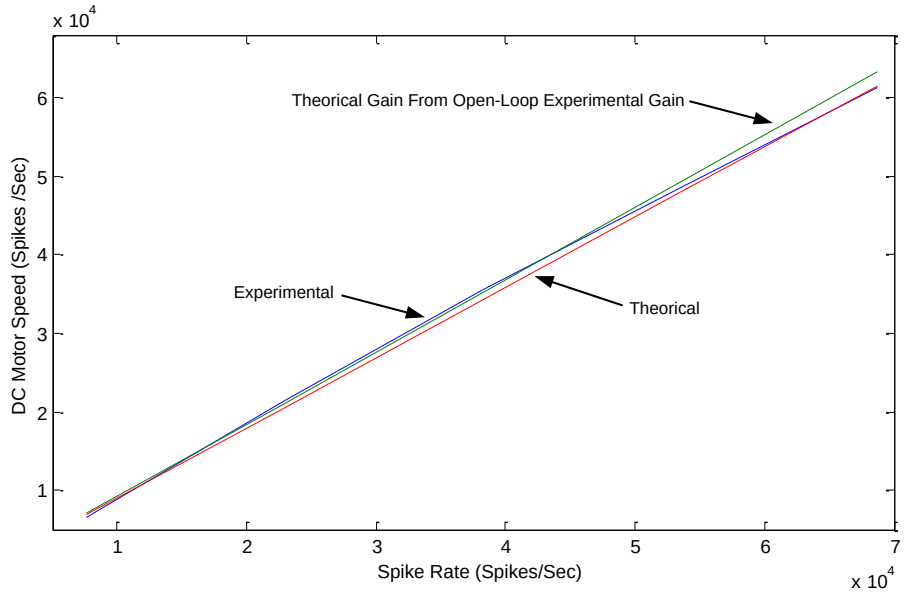


Figura 5.27: Velocidad estacionaria del motor en lazo cerrado para diversas frecuencias de referencia

A continuación vamos a excitar al sistema con otros dos tipos de funciones de excitación, rampa y seno. Para ello vamos a ir cambiando paulatinamente el valor de entrada del generador sintético de spikes, codificando en dichos valores la función de excitación. Por ejemplo en el caso de la rampa iremos incrementando el valor del generador a intervalos fijos de tiempo. Los resultados de aplicar una rampa de entrada al sistema para diversos anchos de spikes (desde 0.32uSec hasta 14uSec) se muestran en la Figura 5.28. En dicha figura se ve la señal de referencia (en azul) compuesta por pequeños escalones, siendo el tiempo entre escalones equivalente al tiempo que tarda el microcontrolador en recibir los paquetes desde el puerto USB y enviarlos por el puerto SPI. Como es conocido en la teoría del control [Houpis89], la velocidad del motor nunca alcanzará la referencia, ya que para ello es necesario introducir un integrador en el lazo de control. Además se observa que la pendiente de las respuestas del motor, no son las mismas que la pendiente de la referencia, ya que la pendiente de las respuestas serán la ganancia en lazo cerrado multiplicado por la pendiente de la referencia, siendo siempre la primera menor que uno, y obteniendo en consecuencia una pendiente menor a la de la referencia.

Finalmente hemos excitado el sistema con un seno a una frecuencia de 2Hz, modificando de nuevo el ancho de spike para analizar la respuesta del sistema, mostrado en la Figura 5.29. De nuevo se muestra la referencia del sistema, en azul, así como las respuestas de los motores para diversos anchos de spikes, desde 5uSec hasta 40uSec No sólo observamos cómo el ancho de spike afecta a la ganancia del sistema, sino cómo modifica la función de transferencia del sistema equivalente, variando la fase entre las diversas respuestas, tal y como predijo la Ecuación [2-4]. Quedando nuevamente de manifiesto cómo el sistema se comporta como un control en lazo cerrado.

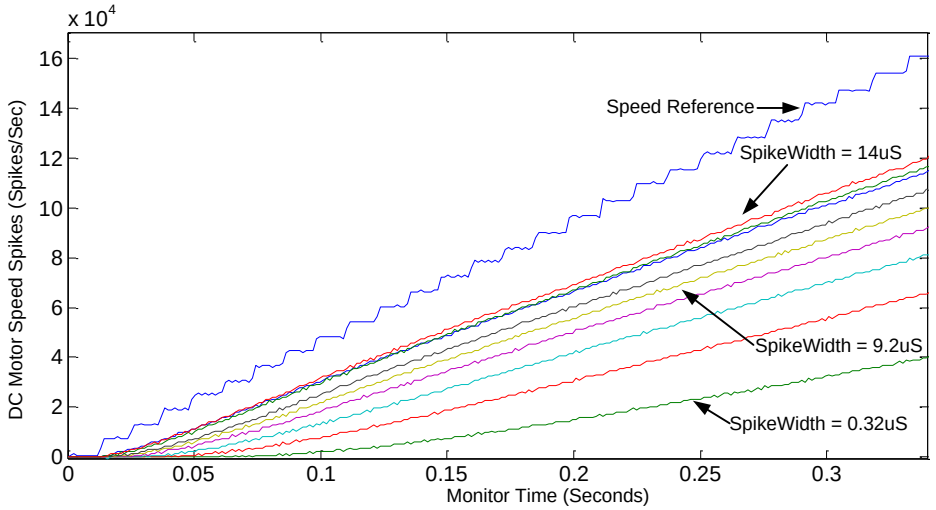


Figura 5.28: Respuestas del motor en lazo cerrado ante una entrada en rampa para diversos anchos de spike

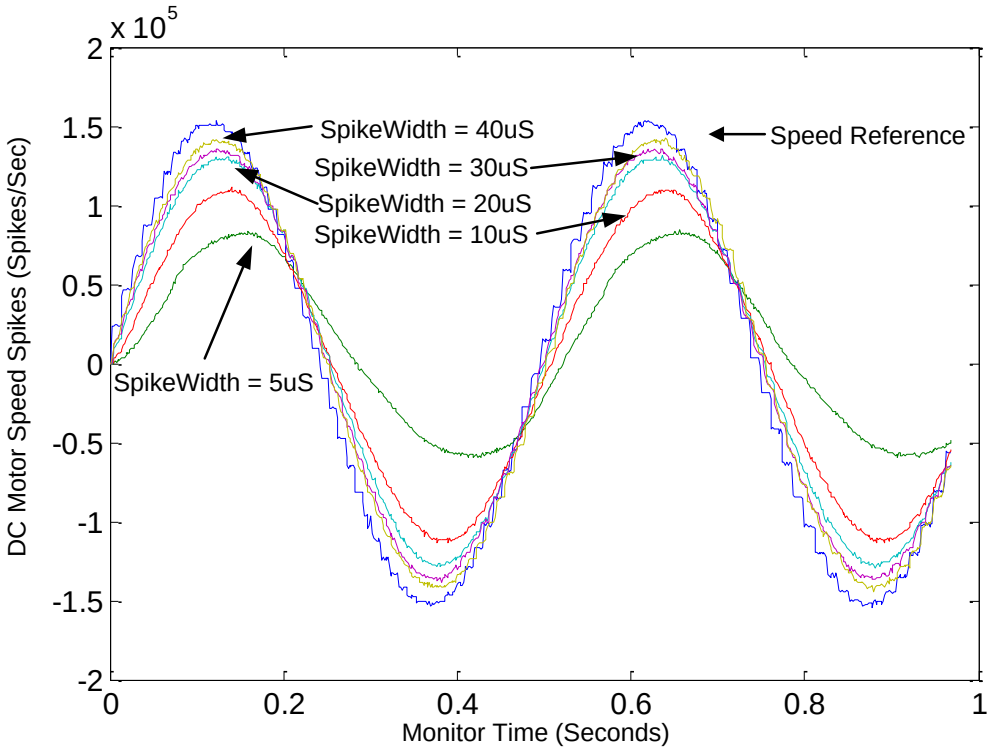


Figura 5.29: Respuestas del motor en lazo cerrado ante una entrada en seno para diversos anchos de spike

5.4.4. Análisis experimental del controlador PI basado en spikes

Tras analizar el comportamiento del controlador proporcional basado en spikes, vamos a configurar el controlador en modo PI. Para ello vamos a modificar el registro de estado de la FPGA, fijando el derivador de spikes en estado de reset. En la Figura 5.30 mostramos la actividad de los spikes de cada componente del controlador PI, monitorizados a través del bus AER. La figura contiene la respuesta del controlador ante una entrada en escalón, en ella pueden observarse las señales implicadas en el control, en primer lugar encontramos la referencia de velocidad, azul, a continuación están el error del sistema y la velocidad del motor, en rojo y verde respectivamente, y finalmente las señales del controlador, en amarillo la salida del Integrate & Generate y en negro la señal aplicada al motor, la cual contiene la suma de los spikes de error y del Integrate & Generate. En la figura puede verse como gracias al uso del controlador PI hemos anulado el error en régimen estacionario de la velocidad del motor presente en el controlador P, y que además el sistema se comporta cómo un sistema subamortiguado, debido a la pérdida de fase asociada a la componente integral del controlador [Ogata04].

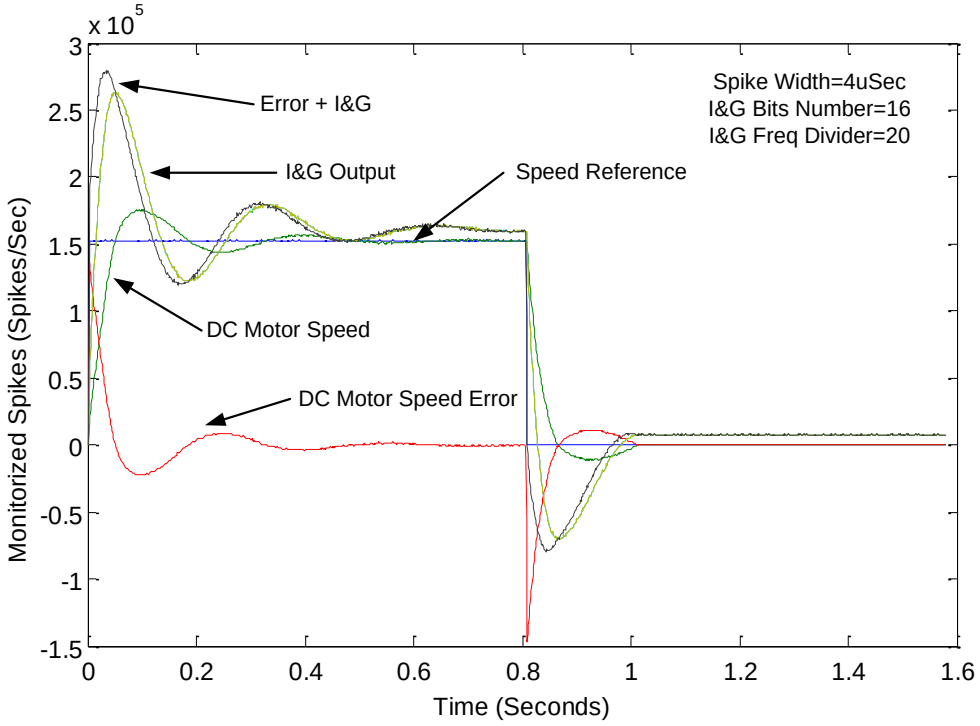


Figura 5.30: Actividad interna del controlador PI basado en spikes ante una entrada en escalón

En la Ecuación [4-9], donde se mostraban los parámetros del controlador PI basado en spikes, vemos que los parámetros implicados en este controlador son tres, en acho de spike, que afecta a la ganancia estática y ganancia del Integrate & Generate, así como el número de bits y el divisor de frecuencia del Integrate & Generate, que sólo afectan a la ganancia del integrador. Así que vamos a ir modificando dichos parámetros y monitorizando la respuesta del sistema para poder así comprobar su correcto funcionamiento. Vamos a comenzar el análisis experimental observando el efecto del ancho de los spikes en la respuesta del motor, para ello vamos a fijar los parámetros del Integrate & Generate, seleccionando el banco de Integrate & Generate número 3, de 16 bits, y fijando el divisor de frecuencia a 20, para a continuación ir modificando iterativamente el ancho de los spikes. Los resultados son mostrados en la Figura 5.31, en la que mostramos la velocidad de referencia, así como las respuestas del motor. En ellas se aprecia cómo el ancho de spike afecta a la respuesta del motor, ya que para valores pequeños, 2.52uSec, el tiempo de subida de la señal es muy elevado, así como el tiempo de asentamiento, sin embargo, a medida que el ancho de spikes aumenta, el tiempo de subida de la velocidad disminuye, así como el tiempo de pico o sobre disparo. Sin embargo la proporción de sobre disparo aumenta acorde con el ancho de pulso.

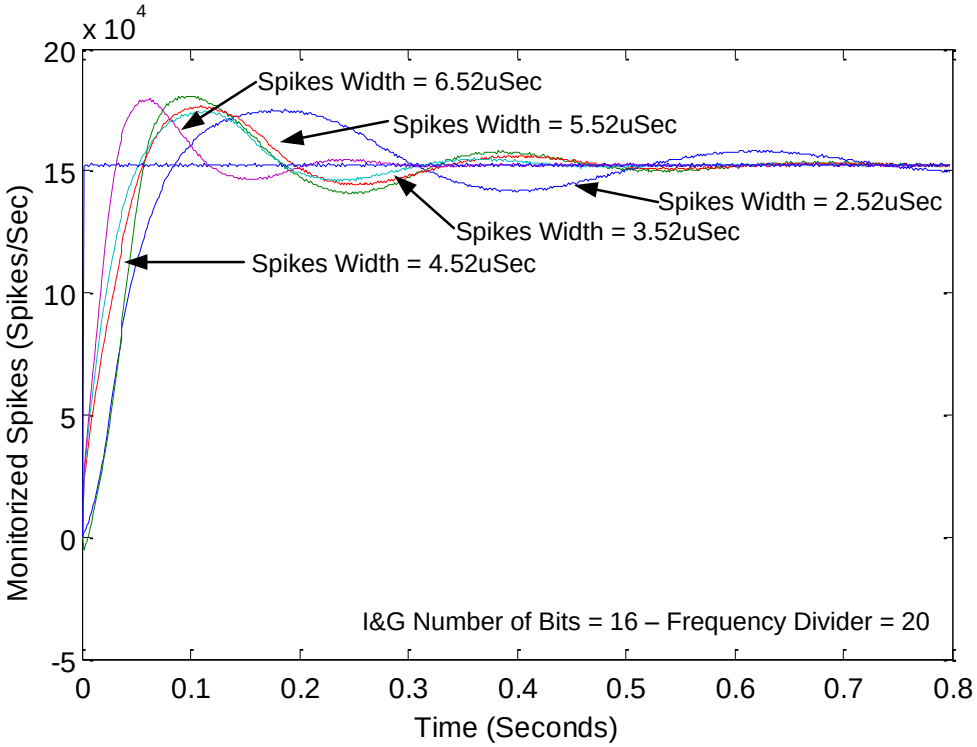


Figura 5.31: Respuestas del motor con un controlador PI basado en spikes para diversos anchos de spikes

En la Figura 5.32 mostramos los tiempos de subida, pico y asentamiento de las respuestas del motor para los distintitos anchos de spikes, además superpuesta a estas últimas, encontramos el porcentaje de sobre disparo en cada caso. Tal y como acabamos de comentar, en ella queda constatado como al ir incrementando el ancho de spikes el tiempo de subida de la velocidad de motor se va reduciendo, así como el tiempo de pico y asentamiento. Sin embargo, la tasa de sobre disparo va aumentando, hasta situarse cerca del 25%.

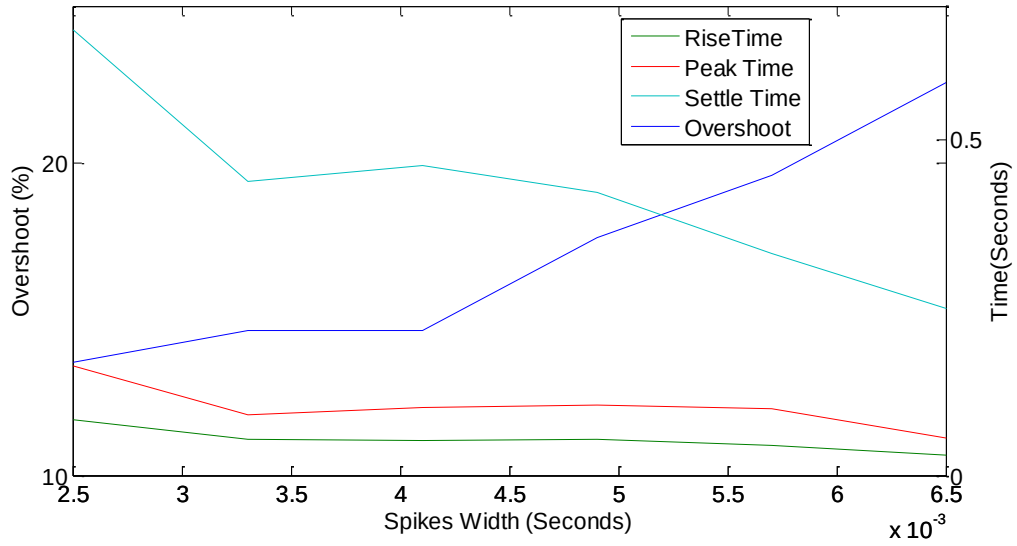


Figura 5.32: Sobreoscilación y tiempos de la respuesta del motor con un controlador PI basado en spikes para diversos anchos de spikes

A continuación vamos a realizar el experimento inverso, hemos fijado el ancho de los spikes a 5.12uSec, y vamos a ir cambiando de banco de Integrate & Generate, así como los divisores de frecuencia. De esta manera vamos a comprobar el correcto funcionamiento de los bancos de Integrate & Generate, así como el efecto de modificar estos parámetros. Tal y como mostraba la Ecuación [4-9], y dado que cambiar de banco de Integrate & Generate sólo supone cambiar el número de bits de este componente, la modificación de estos parámetros sólo afectará a la ganancia del término integral del sistema. En la Tabla 5-5 mostramos los parámetros utilizados, y en la Figura 5.33 las respuestas del motor para cada par de parámetros. En la figura vemos cómo la respuesta del motor para una ganancia del Integrate & Generate muy baja tarda un tiempo enorme en alcanzar la velocidad de referencia, ya que al tener tan poca ganancia hay que esperar a que el Integrate & Generate integre una gran cantidad de spikes provenientes del error. Sin embargo, a medida que la ganancia del Integrate & Generate va aumentando, el tiempo de subida del motor va disminuyendo, hasta el punto que comienza a mostrar respuestas subamortiguadas, con tiempos de subida, de pico y asentamiento cada vez menores, pero con tasas de sobre disparos cada vez más elevados.

Tabla 5-5: Parámetros del controlador PI basado en spikes

Caso de prueba	Numero de bits	Banco seleccionado	Divisor de frecuencia
1	12	1	10
2	12	1	20
3	14	2	10
4	14	2	20
5	16	3	10
6	16	3	20
7	18	4	10
8	18	4	20

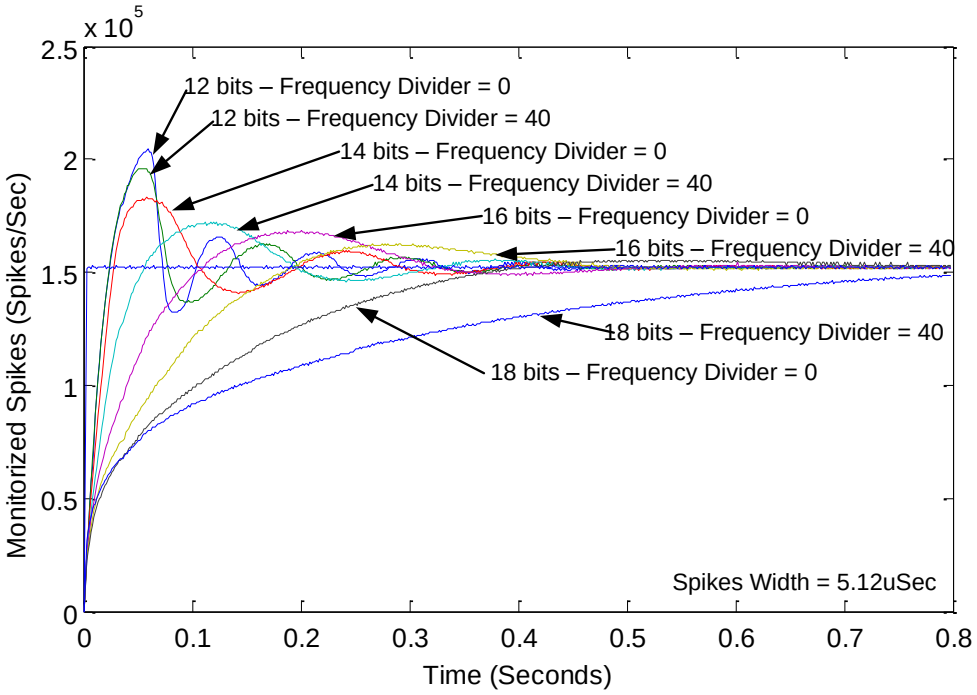


Figura 5.33: Respuestas del motor con un controlador PI basado en spikes para diversos parámetros del Integrate & Generate

En la Figura 5.34 mostramos los tiempos de subida, pico y asentamiento de las respuesta del motor para los parámetros del Integrate & Generate, así como el porcentaje de sobre disparo en cada caso. En ella puede verse claramente cómo al tener un término integral con una elevada ganancia, la respuesta del motor presenta un tiempo de subida y pico muy reducidos. Sin embargo, la tasa de sobredisparo es enorme, del 35%. A medida que disminuimos este término, la velocidad del motor presenta un tiempo de subida y pico muy elevados, pero una tasa de sobredisparo cada vez menor, situándose por debajo del 5%.

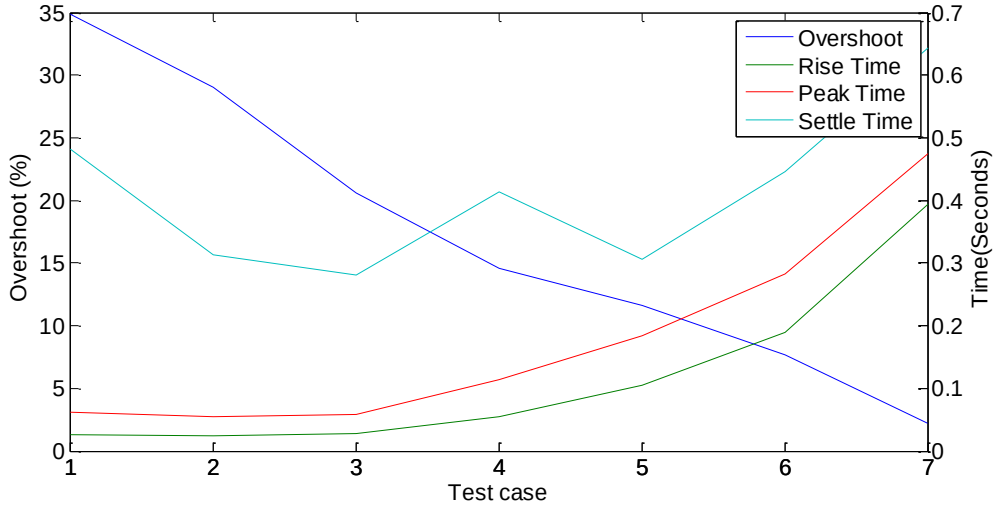


Figura 5.34: Sobreoscilación y tiempos de la respuesta del motor con un controlador PI basado en spikes para diversos parámetros del Integrate & Generate

5.4.5. Análisis experimental del controlador PD basado en spikes

A continuación, vamos a configurar el controlador en modo PD. Para ello vamos a modificador el registro de estado de la FPGA, fijando el Integrate & Generate en estado de reset, obteniendo un controlador PD basado en spikes. Al igual que en el caso del controlador PI vamos a comenzar mostrando la actividad de los spikes internos del controlador PD ante una entrada en escalón en la Figura 5.35. En ella mostramos en primer lugar la referencia de velocidad en escalón, azul, así como la velocidad del motor y el error del sistema, en verde y rojo respectivamente, a continuación puede verse la salida del derivador de spikes, en amarillo, así como la suma de los spikes del derivador y del error, en negro. Como se ve en la figura el motor vuelve a comportarse como un sistema sobre amortiguado, y el controlador introduce de nuevo un error en régimen permanente. Si nos fijamos en la salida del derivador de spikes, consiste en un "impulso" cuando la referencia cambia desde cero, y dicho "impulso" va decayendo a medida que el error decae, hasta valer cero cuando el error se estabiliza.

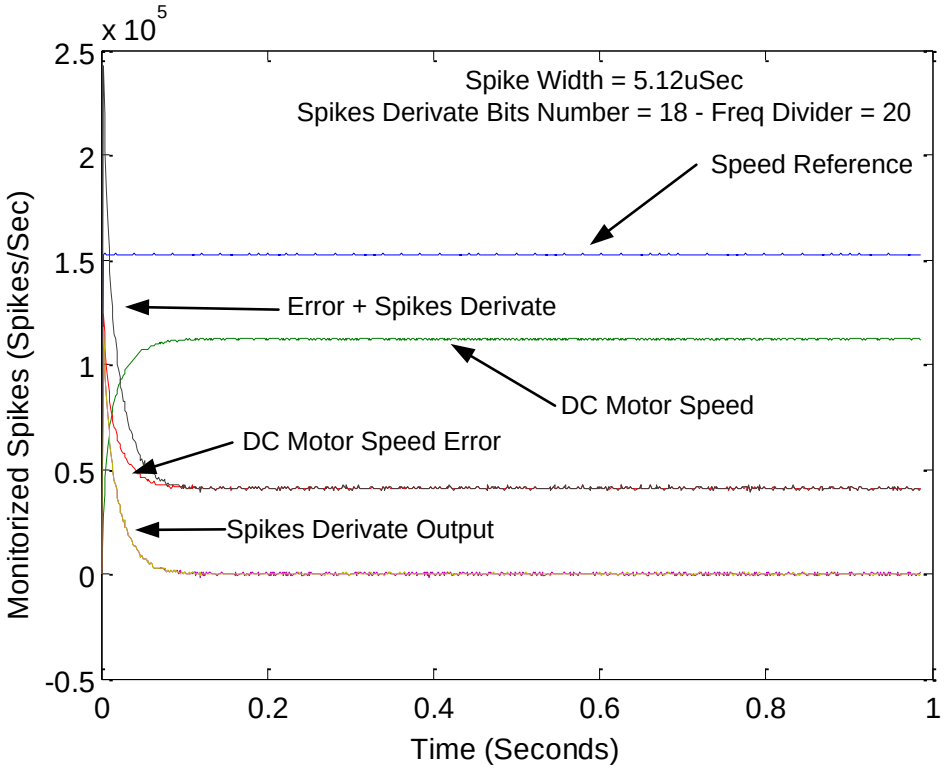


Figura 5.35: Actividad interna del controlador PD basado en spikes ante una entrada en escalón

En la Ecuación [4-19] se mostraban los parámetros del controlador PD basado en spikes, en ella se constata que los parámetros implicados en este controlador son tres, el acho de los spikes, que afecta a la ganancia estática y ganancia del derivador de spikes, así como el número de bits y el divisor de frecuencia del derivador de spikes, que sólo afectan a la ganancia del derivador. A continuación vamos a ir modificando dichos parámetros y monitorizando la respuesta del sistema para poder así comprobar su correcto funcionamiento. En primer lugar hemos fijado el banco del derivador de spikes al banco 1, 16bits, y su divisor de frecuencia a 50, y a continuación hemos ido modificando el ancho de spikes desde 4uSec hasta

10uSec. En la Figura 5.36 se pueden encontrar las respuestas del motor para cada caso. Como acabamos de comentar, el motor se vuelve a comportar como un sistema sobreamortiguado, y además se vuelve a introducir el error en régimen permanente al no haber ningún integrador. Sin embargo las respuestas del controlador son casi análogas al controlador P basado en spikes, aumentando la ganancia estática a medida que aumenta el ancho de spikes.

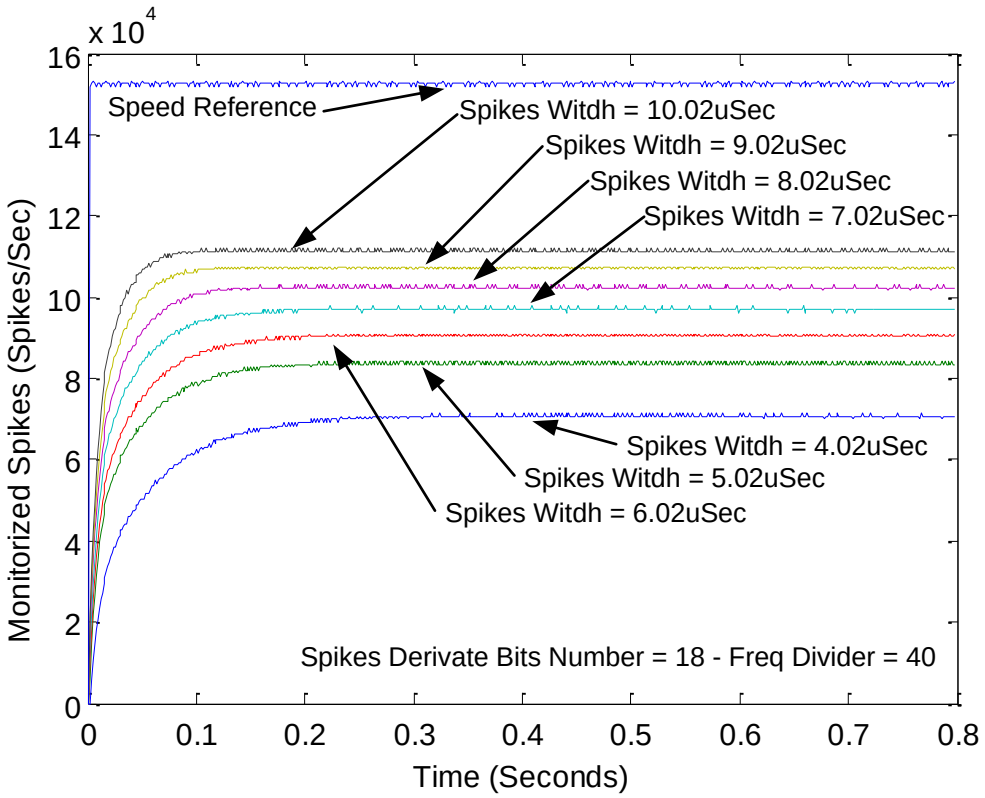


Figura 5.36: Respuestas del motor con un controlador PD basado en spikes para diversos anchos de spikes

A continuación hemos ido cambiando los parámetros del derivador de spikes, alternando entre todos los bancos de derivadores, y modificando sus divisores de frecuencia acorde con la Tabla 5-6, y con un ancho de spike fijo, 3.6uSegundos. Los resultados pueden encontrarse en la Figura 5.37, en ellos se observa que no hay mucha diferencia aparente entre usar un derivador u otro, esto se debe a que la salida del derivador tiene tanta frecuencia que satura al Spikes Expansor, y para que esos spikes tuvieran efecto, habría que aumentar el voltaje de la fuente.

Tabla 5-6: Parámetros del controlador PD basado en spikes

Caso de prueba	Numero de bits	Banco seleccionado	Divisor de frecuencia
1	16	1	20
2	16	1	30
3	18	2	20
4	18	2	30
5	20	3	20
6	20	3	30
7	22	4	20
8	22	4	30

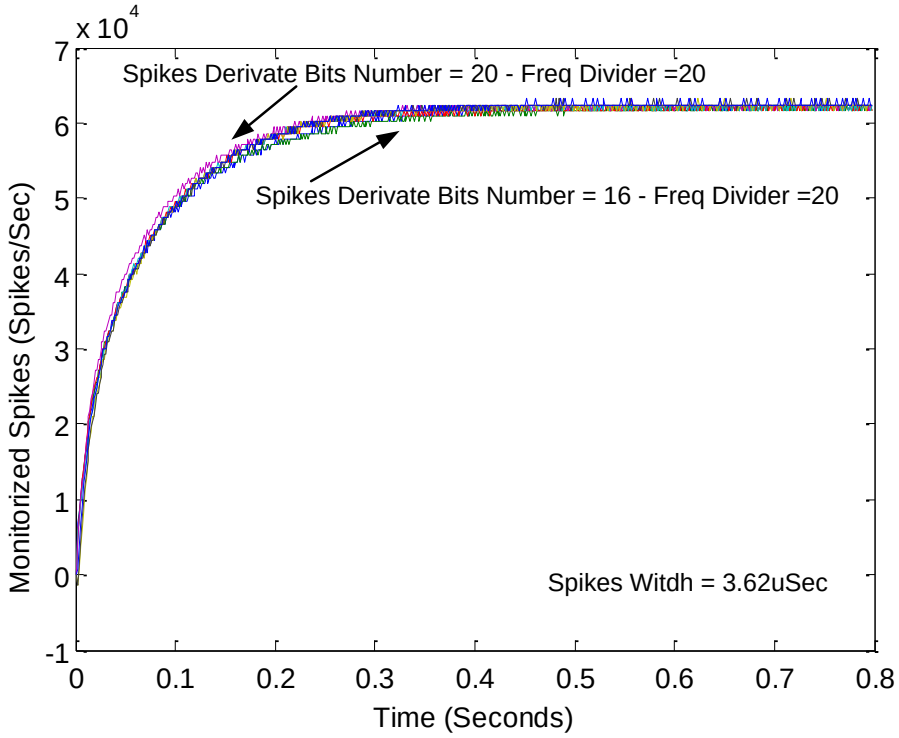


Figura 5.37: Respuestas del motor con un controlador PD basado en spikes para diversos parámetros del derivador de spikes

5.4.6. Análisis experimental del controlador PID basado en spikes

Finalmente vamos a permitir el funcionamiento de todos los elementos de nuestro controlador, no reseteando ninguno, controlando la velocidad del motor con un controlador PID basado en spikes. En este apartado no pretendemos proporcionar ningún método de ajuste del controlador, de los mucho existentes [Aström09], ni ajustar el controlador para cumplir ciertas restricciones, sólo pretendemos observar el comportamiento del motor para diversos parámetros del controlador, verificando así comportamiento. En la Figura 5.38 mostramos la reconstrucción de la actividad de los spikes de cada componente del controlador PID. La figura contiene la respuesta del controlador ante una entrada en escalón, en ella pueden observarse las señales implicadas en el control, en primer lugar encontramos la referencia de velocidad, azul, a continuación están el error del sistema y la velocidad del

motor, en rojo y verde respectivamente, y finalmente las señales del controlador, en celeste la salida del Integrate & Generate y en violeta la salida del derivador de spikes. La suma de ambas en amarillo, y finalmente en negro la señal aplicada al motor, la cual contiene la suma de los spikes de error, del Integrate & Generate y del derivador de spikes.

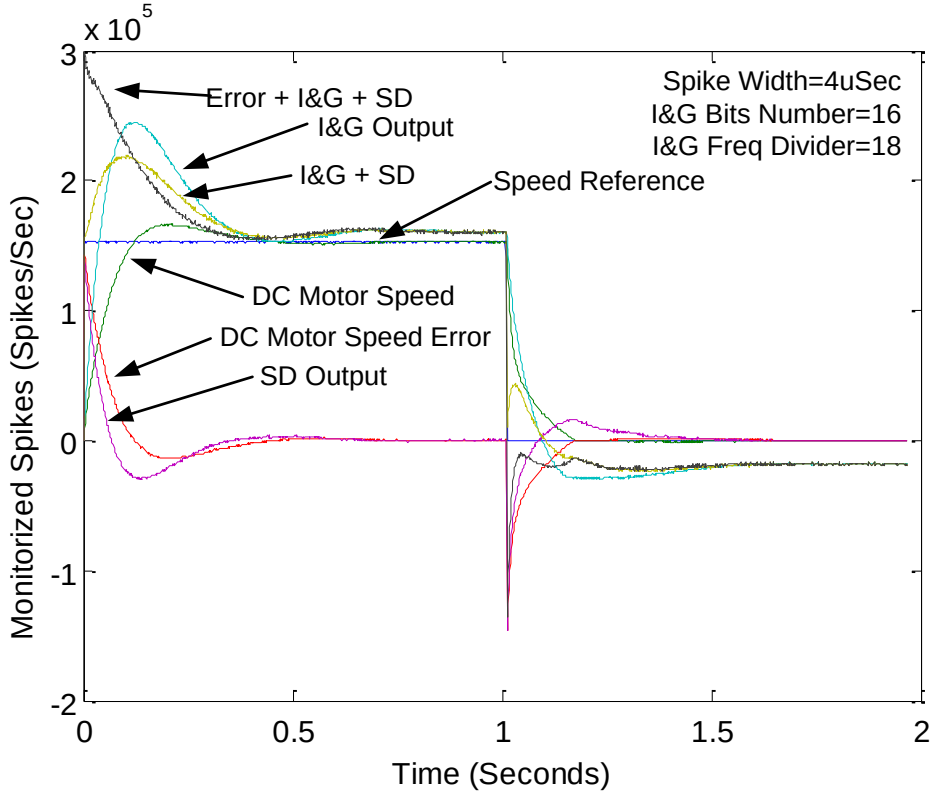


Figura 5.38: Actividad interna del controlador PID basado en spikes ante una entrada en escalón

En la Figura 5.39 mostramos los eventos AER monitorizados antes de reconstruirlos como números discretos. Los eventos mostrados son los correspondientes a la Figura 5.38, en los que se ve la actividad de todos los eventos del sistema, y codificados acorde a la Tabla 5-2. El eje horizontal representa los ticks temporales de la USBAERmini2, de 0.2uSegundos, y el eje vertical las direcciones de los eventos AER, siendo cada punto azul de la figura un spike disparado dentro del controlador PID. El espacio de direcciones está comprendido entre las direcciones 0 y 13, donde las direcciones pares representan spikes positivos, y las impares spikes negativos, codificados acorde con la Tabla 5-2.

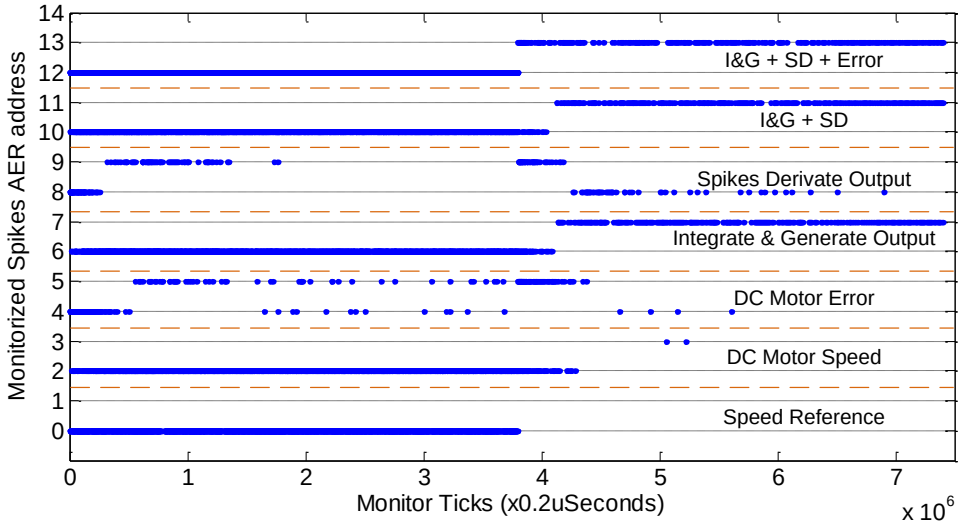


Figura 5.39: Eventos AER del controlador PID basado en spikes monitorizados

Dado la gran cantidad de parámetros que muestra el controlador, hemos seleccionado varios casos de prueba, mostrados en la Tabla 5-7. Lo que hemos hecho es ir configurando el controlador con cada caso de prueba, para a continuación aplicar una velocidad de referencia en escalón, mostrando los resultados en la Figura 5.40. En la figura se muestran las respuestas del motor para cada caso de prueba. Gracias al uso del controlador PID podemos obtener diferentes tipos de respuestas, consiguiendo paliar la sobreoscilación introducida por el uso de un integrador gracias al derivador de spikes, pudiendo obtener respuestas con tiempos de subida muy rápidos y poca sobreoscilación, como ocurre en los casos 1 y 2. Pero también podemos sacrificar el tiempo de subida para obtener una menor sobreoscilación, casos 4 y 6, hacer justo lo contrario, buscar los menores tiempos de subida, casos 3 y 5, a costar de sufrir una elevada sobreoscilación.

Tabla 5-7: Casos de prueba del controlador PID basado en spikes

Caso de prueba	Ancho de spikes (uSec)	Banco seleccionado (I)	Divisor de frecuencia (I)	Banco seleccionado (D)	Divisor de frecuencia (D)
1 – azul	4.02	2	20	1	40
2 – verde	5.14	2	30	1	3
3 - rojo	5.14	2	20	1	2
4 – celeste	4.02	2	40	1	6
5 – violeta	2.82	2	20	1	6
6 - amarillo	5.14	3	25	2	6

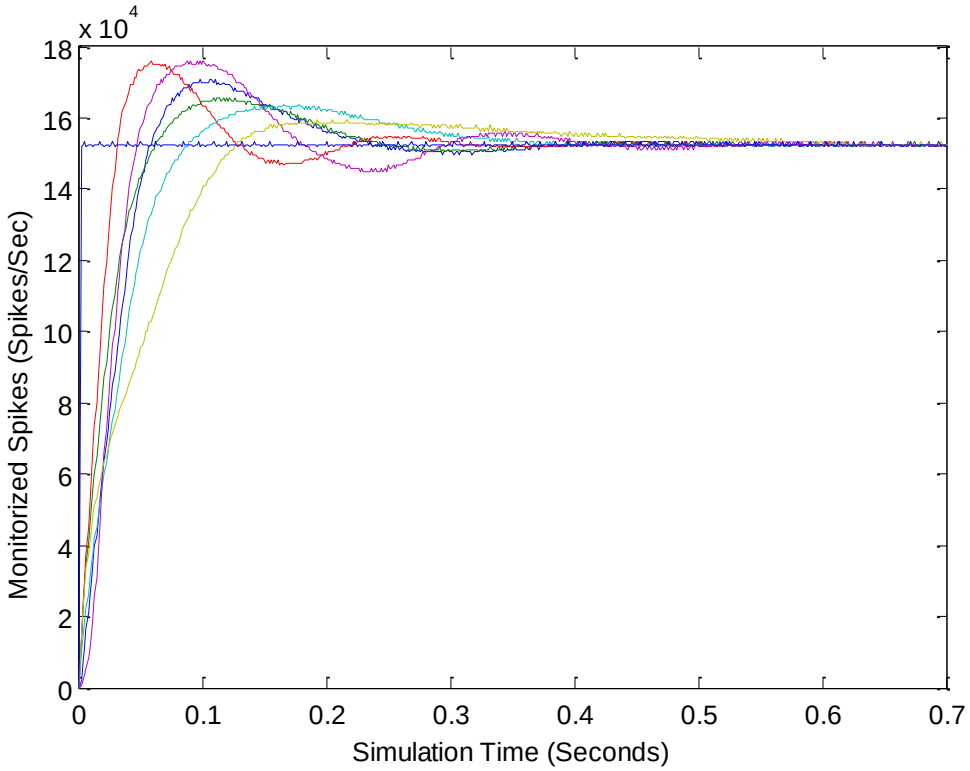


Figura 5.40: Respuestas del motor con un controlador PID basado en spikes para diversos parámetros

En la Figura 5.41 mostramos los tiempos de subida, de pico y de asentamiento, así como la tasa de sobre disparo para caso de prueba anteriormente expuesto. La respuesta que menor tiempo de subida presenta es la del caso 3. Sin embargo, también es la segunda que posee el mayor porcentaje de sobredisparo, en torno al 15%. Si la comparamos con la respuesta del caso 5, vemos cómo ésta tiene un tiempo de subida mayor, pero sin embargo, posee una mayor tasa de sobreoscilación, siendo a nuestro parecer una respuesta peor que la presentada por el caso 3. Por otro lado, si nos fijamos en la respuesta del caso 2, esta presenta una buena relación entre el tiempo de subida y la tasa de sobredisparo, y puede ser una configuración desde la que partir para afinar el controlador para este motor para muchas aplicaciones, siendo una de ellas el robot móvil diferencial que será presentado en el próximo capítulo.

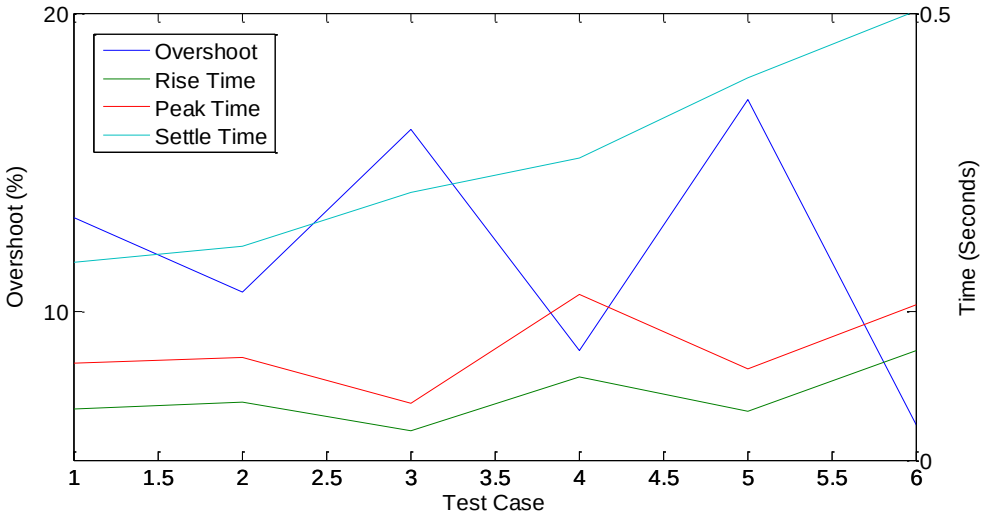


Figura 5.41: Sobreoscilación y tiempos de la respuesta del motor con un controlador PID basado en spikes para diversos parámetros

Para finalizar los experimentos con el controlador PID basado en spikes, vamos comprobar su comportamiento para distintas referencias de velocidad, usando para ello la configuración del caso de prueba número 2. En la Figura 5.42 mostramos los resultados, en los que pueden observarse las respuestas del motor para cada referencia de velocidad, siendo esta alcanzada en todos los casos, mostrando un error en régimen permanente nulo, así como una sobre oscilación cercana al 11%.

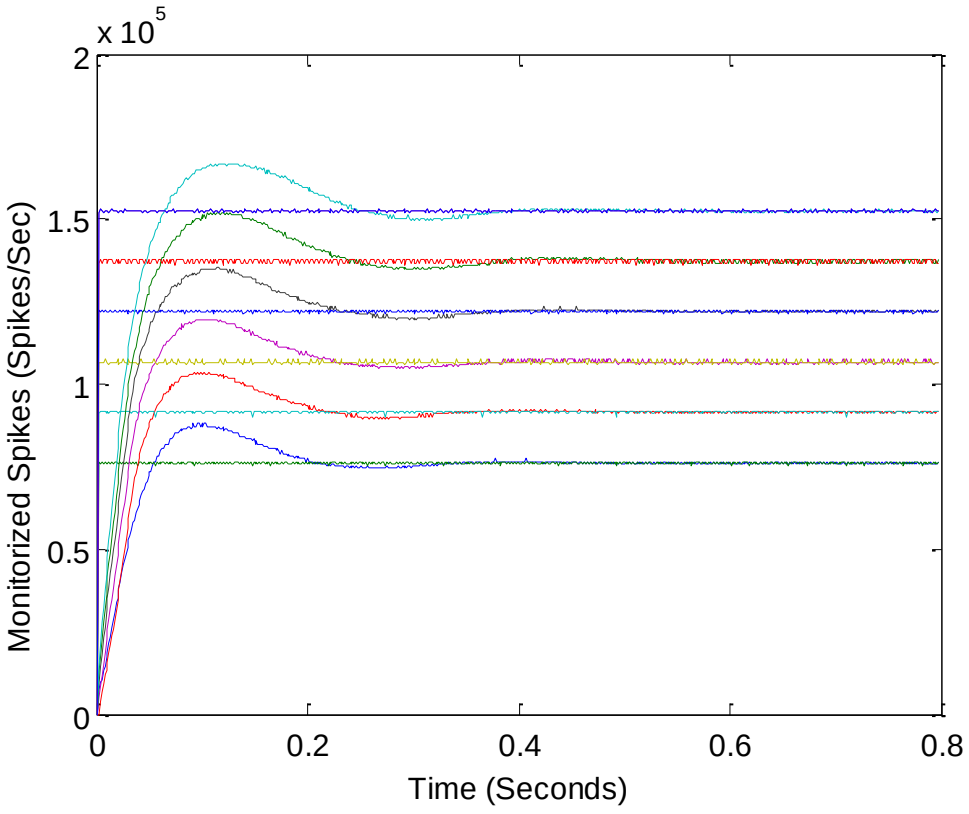


Figura 5.42: Respuestas del motor con un controlador PID basado en spikes para diversas referencias de velocidad

5.5. Control PID de posición basado en spikes

Hasta ahora únicamente hemos expuesto controladores de velocidad basados en spikes, sin embargo, otro controlador muy interesante a desarrollar es el controlador de posición. Ya que es comúnmente usado en robots manipuladores, como son los brazos robóticos o manos antropomórficas [Varona07]. De manera que vamos a apoyarnos en el diseño del controlador de velocidad presentado, para diseñar un controlador PID de posición basado en spikes, este controlador parte de la hipótesis de que será usado en un robot con restricciones en su movimiento, como es el caso de un brazo robótico. De manera, que para moverse, por ejemplo, desde 340º hasta 1º, no avanzará hasta desbordar los 360º y llegar a 1º, si no que retrocederá hacia atrás hasta alcanzar 1º.

Como hemos expuesto con anterioridad el sensor del que disponemos para medir el movimiento del robot es un encoder óptico incremental, el cual nos proporciona la velocidad del motor como un tren de spikes, para poder obtener la posición a partir de su velocidad, debemos integrar los spikes de la velocidad, de tal manera que la frecuencia de los spikes de la posición serán proporcionales a la integral temporal de la frecuencia de los spikes de la velocidad del motor:

$$f_\theta(t) = k_i \int f_\omega(t)dt$$

Ecuación [5-3]

Para integrar los spikes de la velocidad el motor hemos optado por usar un Integrate & Generate, de tal forma que los spikes provenientes del encoder serán la entrada de Integrate & Generate, obteniendo a su salida la posición del motor codificada mediante spikes. El controlador PID de posición basado en spikes finalmente diseñado se muestra en la Figura 5.43. Este controlador es exactamente igual a la Figura 5.4, pero con la adición del Integrate & Generate añadido en la realimentación, en verde oscuro al pie de la figura. Siendo su salida la señal que se restará con la referencia, obteniendo así el error de posición del motor en cada momento. Siendo los spikes de este error finalmente procesados por el controlador PID basado en spikes, y aplicados al motor a través del Spikes Expansor. Además hemos eliminado el multiplexor que nos permitía alternar entre un control en lazo abierto o cerrado, ya que el control de posición de un motor de DC en lazo abierto no tiene mucho sentido en nuestro caso.

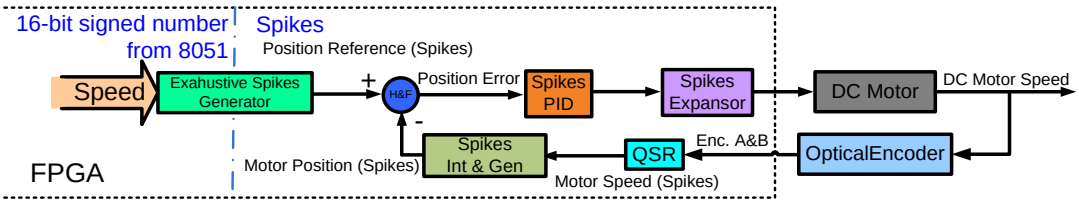


Figura 5.43: Diagrama de bloques del controlador PID basado en spikes de posición

Para ajustar la ganancia del Integrate & Generate hemos usado la Ecuación [4-9], en la que se mostraba la ganancia del Integrate & Generate en función del número de bits y del divisor de frecuencia. El número de bits ha sido determinado en base a la resolución del

encoder y la relación de reducción del motor, de tal manera que el número de spikes que se disparan a lo largo de una vuelta serán:

$$Número\ de\ spikes\ por\ vuelta\ con\ reductora = \frac{Número\ de\ spikes\ por\ vuelta}{Relación\ de\ reducción} = 2000 * 13 = 26000$$

Ecuación [5-4]

Siendo este el número mínimo de spikes que el Integrate & Generate tiene que ser capaz de integrar a lo largo de una vuelta de 360º. Además obtenemos una resolución de:

$$\frac{360º}{Número\ de\ spikes\ por\ vuelta\ con\ reductora} = 0.011º\ por\ spike$$

Para poder integrar el número de spikes que se producen en una vuelta necesitamos un número de bits del Integrate & Generate suficientemente grande para poder albergar esta cantidad. El elemento encargado de integrar los spikes será el contador que está a la entrada del Integrate & Generate, teniendo un ancho equivalente al número de bits, debiendo ser éste adecuado para poder contar todos los spikes de una vuelta. Hemos calculado el número de bits como la mínima potencia de 2 que puede representar este valor teniendo en cuenta el signo:

$$N = \lceil \log_2 (26000) \rceil + 1 = 15 + 1 = 16 bits$$

Ecuación [5-5]

Ya que con un número en complemento a 2 de 16 bits podemos representar los valores comprendidos entre – 32768 y 32767. Además dejamos un margen de 6768 spikes, para que en caso de sobreoscilación de la posición no se produzca la saturación del integrador.

A continuación hemos pasado a fijar el valor del divisor de frecuencia del Integrate & Generate. Con este valor vamos a ajustar la frecuencia máxima de los spikes de salida del Integrate & Generate, así como su ganancia final. Dado que la USBAERmini2 no puede transferir más de 5Meventos AER por segundo, para poder monitorizar el sistema no podemos usar una frecuencia muy alta, así que hemos usado una frecuencia máxima entorno a 500kSpikes/Segundo. En consecuencia hemos asignando un valor de 99 al divisor de frecuencia, usando la Ecuación [4-13] podemos obtener una frecuencia máxima de salida del Integrate & Generate de:

$$Max(f_{I\&G}) = \frac{F_{CLK}}{1 + IG\_FD} = \frac{50 * 10^6}{1 + 99} = 500 kSpikes/seg$$

Ecuación [5-6]

En consecuencia la ganancia del Integrate & Generate puede ser calculada haciendo uso de la Ecuación [2-22]:

$$k_{I\&G} = \frac{F_{CLK}}{2^{n-1}(1 + IG_{FD})} = \frac{50 * 10^6}{2^{16-1}(1 + 99)} = 15.2588$$

Ecuación [5-7]

A lo largo de los 360º de una vuelta serán integrados 26000 spikes, tanto positivos como negativos, así que el rango de control de la posición estará comprendida entre:

$$Max(f_\theta) = k_{I\&G} * 26000 = 15.2588 * 26000 = 396.73kSpikes/seg$$

Ecuación [5-8]

Nótese que para implementar un controlador de posición la resolución y frecuencia del encoder óptico no es tan crítica como en el caso del controlador de velocidad. Ya que al integrar los spikes del encoder usando un Integrate & Generate, podemos ajustar la ganancia de éste último, generando una nueva secuencia de spikes acorde con nuestras necesidades

Tras añadir los spikes de la posición del motor al monitor AER hemos realizado diversos experimentos con el controlador PID de posición basado en spikes. En primer lugar mostramos en la Figura 5.44 en detalle las señales del controlador ante una entrada de referencia de la posición en escalón. En ella podemos ver como el motor alcanza la posición deseada con un error en régimen permanente nulo, y muestra una respuesta subamortiguada. Además pueden encontrase las señales internas del controlador, como la salida del Integrate & Generate, y el derivador de spikes. Finalmente también podemos observar la velocidad del motor, antes de integrarla, en forma de campana, de manera que el motor acelerará hasta alcanzar la posición que comienza a aproximarse a la referencia, momento en el que el controlador comienza a frenar al motor, modificando levemente la velocidad del motor hasta estabilizar la posición en la referencia proporcionada.

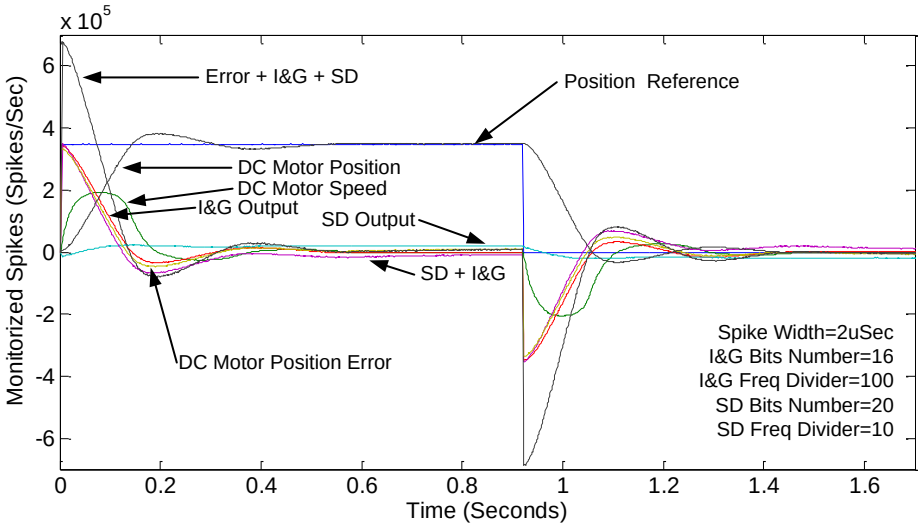


Figura 5.44: Respuesta del controlador PID de posición basado en spikes ante una entrada en escalón

Tras mostrar la respuesta del controlador de posición basado en spikes ante una respuesta en escalón, hemos formulado varios casos de prueba para los diversos valores del controlador PID localizado en el lazo de control. Los parámetros usados para cada caso de prueba pueden encontrarse en la Tabla 5-8, así como los resultados obtenidos ante una entrada en escalón para cada caso de prueba en la Figura 5.45. Todas las respuestas de los casos de prueba tienen un comportamiento subamortiguado, con tiempos de subida más o menos similares, pero con diversas tasas de sobreoscilación, destacando este aspecto en el caso de prueba número 5, siendo el que menor tasa de sobreoscilación presenta.

Tabla 5-8: Casos de prueba del controlador PID de posición basado en spikes

Caso de prueba	Ancho de spikes (uSec)	Banco seleccionado (I)	Divisor de frecuencia (I)	Banco seleccionado (D)	Divisor de frecuencia (D)
1 – azul	4.0	4	200	2	4
2 – rojo	5.12	4	200	2	3
3 – violeta	5.12	4	250	2	2
4 – negro	4.0	4	300	2	6
5 – verde	4.8	4	250	2	6
6 – celeste	5.12	4	300	2	6

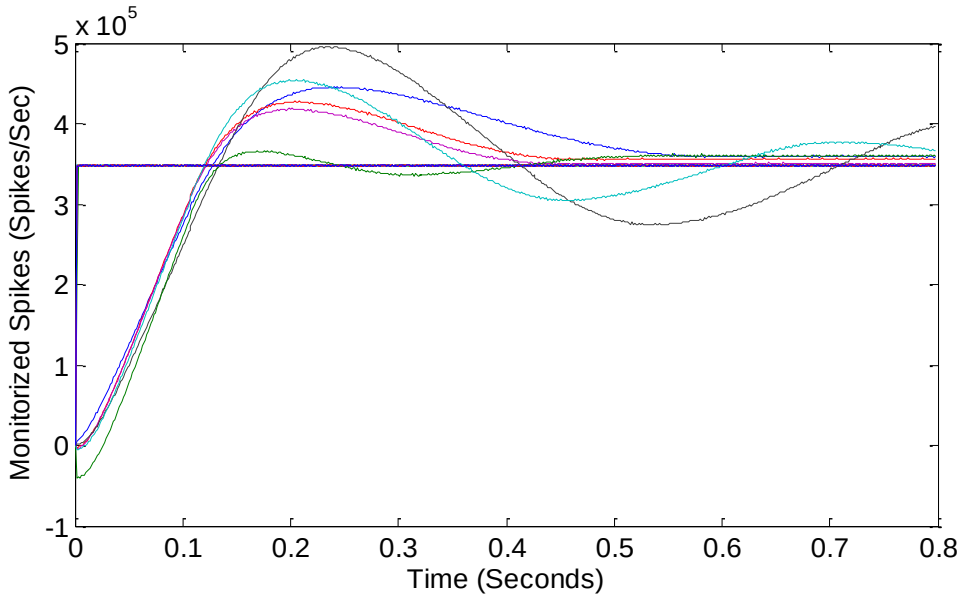


Figura 5.45: Respuesta del controlador PID de posición basado en spikes ante una entrada en escalón y diversos parámetros del controlador

A continuación hemos fijado los parámetros del controlador acorde con el caso de prueba número 3, y hemos procedido a excitarlo con diversas referencias de posición. La Figura 5.46 muestra la respuesta del motor para cada entrada en escalón de la posición. La figura constata como la posición del motor cambia hasta alcanzar la referencia proporcionada, aunque ésta varíe. Sin embargo en algunos casos se puede apreciar cómo se introduce un pequeño error en régimen estacionario, este error lo achacamos a la saturación del Integrate & Generate del controlador PID basado en spikes, ya que hemos usado divisores de frecuencia muy elevados para ajustar el término integral del controlador.

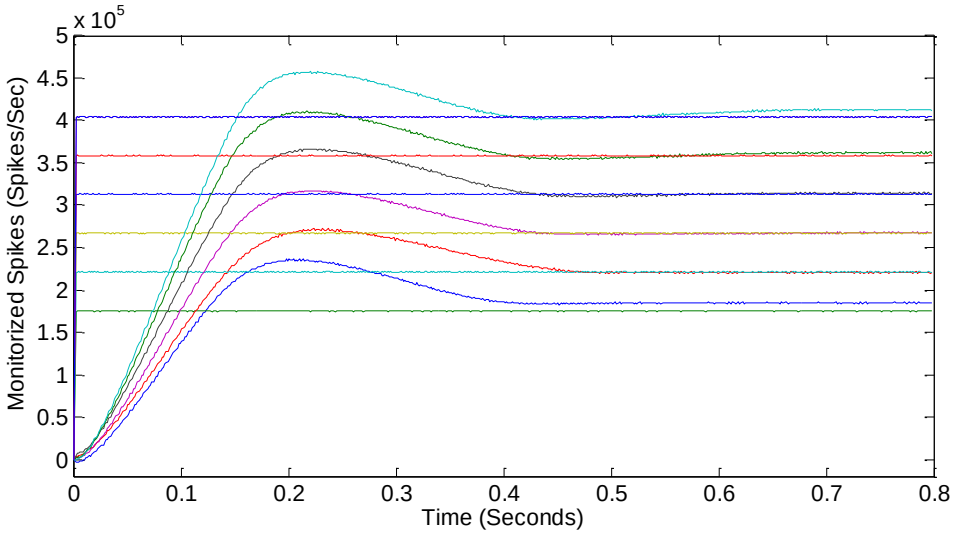


Figura 5.46: Respuesta del controlador PID de posición basado en spikes ante distintas referencias de posición

5.6. Integración de los controladores basados en spikes con otros sistemas pulsantes

A lo largo de este trabajo hemos ido generando los spikes de referencia para los controladores, tanto de velocidad como de posición, usando generadores sintéticos de spikes, sin embargo, para un funcionamiento autónomo de estos controladores, las referencias proporcionadas al controlador deberían provenir desde el exterior de la AER-Robot, no desde un PC como hasta ahora. En este apartado vamos a proponer algunas alternativas para integrar los controladores diseñados con otros sistemas basados en los modelos pulsantes, como son los sistemas AER. De manera que la información proveniente de un sistema de sensado y procesamiento basado en AER, será recibida a través del puerto AER de entrada de la AER-Robot, debiendo el controlador basado en spikes actuar en consecuencia.

En el caso de recibir las referencias de los controladores representadas en direcciones AER, las direcciones AER pueden codificar las referencias de diversas maneras. La forma más inmediata es asignarle a cada referencia de cada controlador un par de direcciones AER, una positiva y otra negativa, de tal manera que la frecuencia de los eventos de una misma dirección AER sea directamente la frecuencia de los spikes aplicados a la referencia del controlador con dicha dirección asignada. En la Figura 5.47 mostramos una posible arquitectura de una entidad receptora de eventos AER para dichos fines. A la izquierda de la figura se encuentra una entidad encargada de implementar el protocolo AER de entrada, pudiendo ser este protocolo el usado en el proyecto CAVIAR [Häfliger04] u otras implementaciones de la comunicación AER [Zamarreño08][Fasnacht08][Miro06][Miro07]. Tras recibir los eventos AER de entrada, la dirección AER será enviada a un decodificador de direcciones, en el centro, cuya salida será un spike dirigido al controlador indicado por la dirección. De esta manera tenemos una comunicación directa entre los spikes disparados desde otros sistemas AER y los controladores, sin más paso intermedio que la comunicación AER.

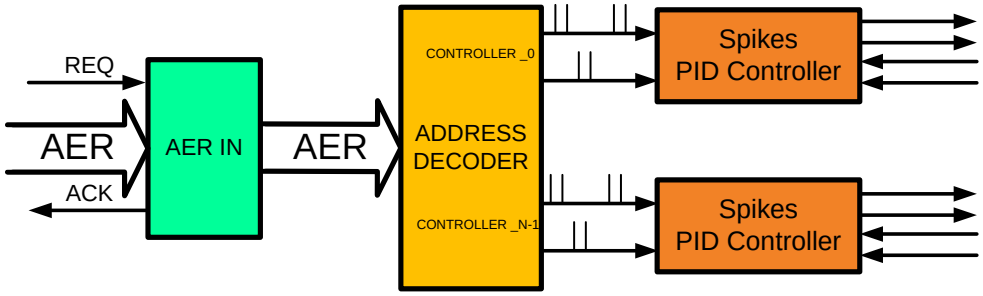


Figura 5.47: Arquitectura de la entrada AER de los controladores con conectividad directa

Como acabamos de comentar, para codificar la dirección de cada referencia basta con el uso de dos direcciones distintas, una para los spikes positivos y otra para los negativos, así que debemos descomponer la dirección AER en dos campos: la referencia a la que va dirigido el spike y su signo. En el caso de una dirección de 16 bits, podríamos reservar 15 bits para la dirección del controlador a la que va dirigido el evento, y un bit para su signo. Pudiendo tener un espacio de direcciones enrome de 2^{15} controladores:

Bits del 15 al 1	Bit 0
Dirección del controlador	0 - positivos 1 - negativos

El uso de este mecanismo para la comunicación de las referencias tiene el inconveniente de necesitar una tasa sostenida de eventos AER dirigidos a cada referencia, ya que si queremos que un motor rote a una determinada velocidad o alcance una determinada posición, hemos de recibir una frecuencia de eventos AER proporcional a dicha velocidad o posición. Debiendo la entidad de eventos AER soportar una tasa de eventos de entrada al menos tan alta como el número de controladores por la frecuencia de spikes máxima de cada uno de ellos:

$$Tasa\ de\ Eventos\ AER\ de\ entrada \geq Numero\ Controladores * f_{spikesMaxima}$$

Ecuación [5-9]

La información de los eventos AER no sólo reside en la frecuencia de cada dirección, si no que la propia dirección AER puede contener información. De este modo, otra posibilidad es codificar la referencia en sí dentro de cada dirección, es decir, asignar cada velocidad o posición del motor con una dirección AER. De tal manera que cada evento AER contendrá dos campos, la referencia a la que va dirigida, y el valor que debe tener. Por ejemplo, con direcciones AER de 16 bits podríamos reservar 10 bits para los valores de las referencias, y los 6 restantes para la dirección de dichas referencias, obteniendo un espacio de direcciones con 64 controladores y 512 posibles valores con signo para cada referencia:

Bits del 15 al 10	Bits del 9 al 0
Dirección de controlador	Valor de la referencia con signo

Una posible implementación de una entidad que implemente este mecanismo es mostrada en la Figura 5.48. En ella encontramos la entidad encargada de implementar el protocolo AER de recepción a la izquierda, la dirección de los eventos AER recibidos será enviada a un decodificador de direcciones, el cual determinará la identidad del controlador al que va dirigido el evento AER, almacenando en un registro el valor de la referencia contenida en la dirección. De tal manera que tendremos una serie de registros que almacenarán el valor de referencia que se desea proporcionar a cada controlador de manera individual. Sólo resta convertir estos valores a spikes, para ello vamos usar generadores sintéticos de spikes Bit-Wise, teniendo tantos generadores como controladores, y estando sus salidas conectadas a cada controlador.

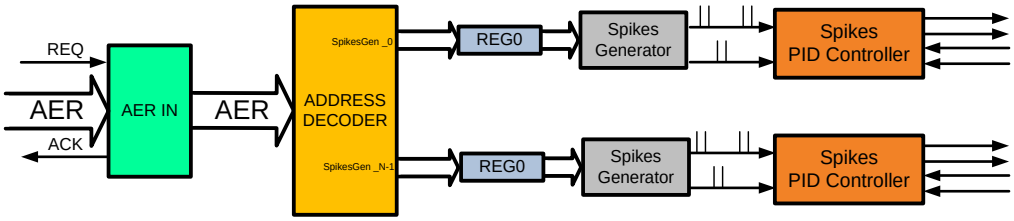


Figura 5.48: Arquitectura de la entrada AER de los controladores en los que la dirección codifica la referencia

Esta codificación tiene la ventaja de que no necesita mantener una tasa de eventos AER tan elevada como en el caso anterior, es decir, si queremos que un motor se mueva a una determinada posición no necesitamos estar enviándole eventos AER continuamente, solamente tendremos que enviarle un evento con el valor de la posición que deseamos que alcance. De esta manera el valor de la referencia codificada en cada evento AER se quedará almacenado en un registro interno, y no necesitaremos mandarle ningún evento más hasta que el motor deba de cambiar de posición. Gracias a esta característica de la codificación, podríamos aligerar en gran medida el tráfico AER entre la AER-Robot y el elemento AER que le proporcione las referencias de los motores.

Desde el punto de vista del consumo hardware, resulta evidente que mediante la codificación AER directa el hardware necesario es mucho más simple que en el caso en que la dirección codifica el valor de la referencia. Ya que en el segundo caso necesitaremos registros para almacenar el valor de las referencias, y tantos generadores sintéticos de spikes bit-wise como controladores. En definitiva a la hora de la elección final del uso de uno u otro mecanismo deben considerarse tres cuestiones, la carga del bus AER, la amplitud del espacio de direcciones necesarias, y la cantidad de hardware disponible. Sin embargo, en el mundo real, quién en realidad elegirá el uso de una codificación AER u otra, serán las características del elemento AER que gestione las referencias de los controladores.

6. Aplicación concreta de los controles en lazo cerrado basados en pulsos: Eddie

A lo largo de los capítulos anteriores hemos ido exponiendo el diseño de controladores PID basados en spikes. Una vez expuestos estos controladores, y como una posible aplicación concreta de los controles PID basados en spikes presentados, hemos construido una plataforma de demostración: Eddie. Eddie es un robot móvil diferencial, con dos ruedas motrices traseras, y una rueda libre delantera, controlado únicamente con spikes, con los controles expuestos en los capítulos anteriores, pero implementando controles más complejos como expondremos más adelante[Jimenez08b]. En la Figura 6.1 a la izquierda vemos una fotografía de Eddie, en la que podemos ver las ruedas traseras, junto con el eje de los motores, la AER-Robot que es el corazón del robot, un receptor de radio R/C (en negro en el centro, y usado para aplicaciones autónomas), y las baterías que lo alimentarán.

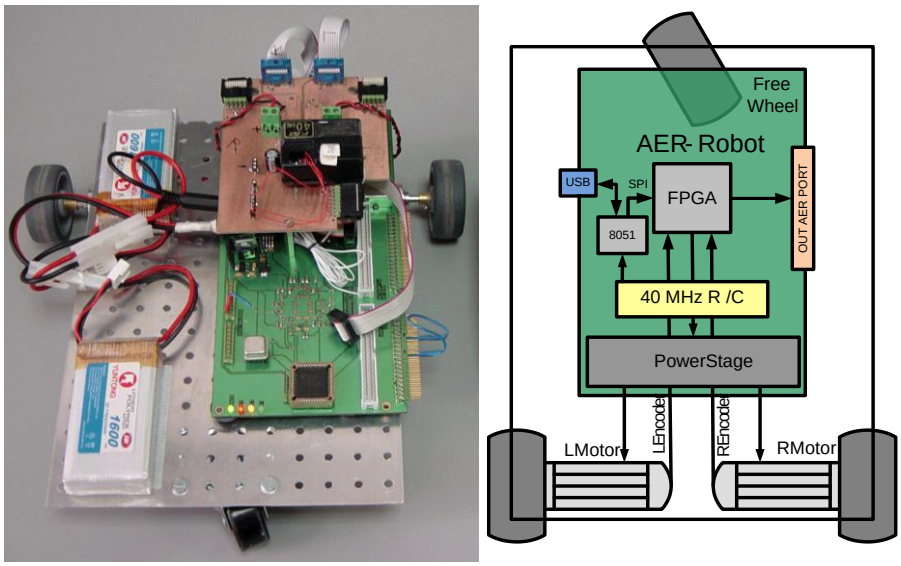


Figura 6.1: Fotografía de Eddie y su diagrama de bloques

A la derecha de la Figura 6.1 podemos observar el diagrama de bloques global del sistema. En el interior de la FPGA están situados los controles basados en spikes, y estos son gestionados desde el microcontrolador a través del puerto SPI, como ya hicimos en el capítulo anterior. Cambiando el firmware del microcontrolador podemos controlar a Eddie desde diversas fuentes: el puerto USB del microcontrolador, para su excitación, monitorización y análisis (reutilizando el mismo firmware del capítulo anterior); así como desde el módulo de radio de 40MHz, para poder controlarlo desde una emisora de radio control.

6.1. Definición de las ecuaciones cinemáticas y la arquitectura de control de robots diferenciales

Conceptualmente hablando, este tipo de robots móviles sólo se pueden desplazar por un plano, siendo las variables de bajo nivel a controlar la velocidad individual de cada motor, y las consignas o referencias proporcionadas desde un nivel superior, como son la velocidad longitudinal o lineal y la velocidad lateral o de giro del robot sobre dicho plano [Diaz97]. Las ecuaciones del movimiento cinemático de este modelo de robot móvil han sido ampliamente

abordas desde puntos de vista generales en una gran cantidad de trabajos [Rajago97] [Alexan90], en la Figura 6.2 mostramos un diagrama conceptual de un robot móvil diferencial. En la figura representamos al robot visto desde arriba, con sus dos ruedas motrices traseras y la rueda libre delantera. Se define la distancia R como el diámetro de cada rueda trasera, y D como la distancia entre los centros de ambas ruedas.

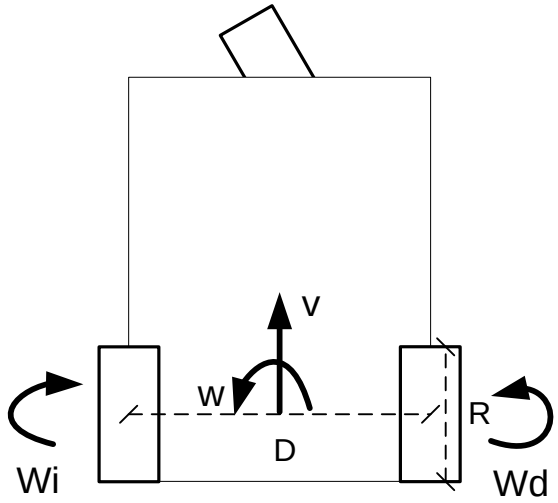


Figura 6.2: Diagrama conceptual de un robot móvil diferencial

Según se expone en [Diaz97], cuando trabajamos con este modelo de robots móviles diferenciales, la velocidad absoluta del robot es proporcional a la media de la velocidad de las ruedas traseras. De tal manera que si ω_d y ω_i son las velocidades angulares de cada rueda (en rad/sec), la velocidad lineal del robot (en m/sec) puede ser calculada acorde a la siguiente ecuación:

$$v = \frac{R(\omega_d + \omega_l)}{2}$$

Ecuación [6-1]

Además la velocidad de giro del robot (en rad/sec) será la multiplicación del cociente del diámetro de las ruedas, R, y la distancia entre los centros de las ruedas, D, por la diferencia de velocidad de las ruedas (en rad/sec), tal y como exponemos a continuación:

$$\omega = \frac{R(\omega_d - \omega_l)}{D}$$

Ecuación [6-2]

De las ecuaciones anteriores se deduce que el control de ambas de variables está muy acoplado entre ellas (velocidad lineal y de giro), dependiendo directamente una de la otra, y dependientes de las velocidades de los dos motores.

A lo largo de la literatura existen numerosas propuestas del diseño de la arquitectura de control de este modelo de robot, una de ellas es la que se propone en [Ollero01], y aunque no de manera exacta, será la implementada en el interior de Eddie. La Figura 6.3 muestra la arquitectura conceptual propuesta para estos fines de control. Este modelo trata de ir generando referencias de los controladores de velocidad y giro para que el robot tome en cada

instante la dirección apropiada. Esta arquitectura se divide en dos controladores, uno encargado de la dirección o giro del robot, a la izquierda, y otro responsable de la velocidad del robot, a la derecha. Sin embargo, tal y como cita textualmente la fuente mencionada, en el caso los robots móviles diferenciales "los bucles de control están fuertemente acoplados", siendo necesario tener en cuenta esta característica en el diseño del controlador. En la parte superior de la figura se muestran los elementos de más alto nivel, compuestos por los generadores de consignas o referencias de velocidad, lineal y de giro, del robot. Las salidas de estos elementos están dirigidas a dos controladores, uno para controlar cada referencia, proporcionando las señales de control a los actuadores del robot, que serán las etapas de potencia de los motores. Finalmente, la potencia aplicada a los actuadores robóticos harán que los motores comiencen a rotar, y en consecuencia, esta rotación se traducirá en el movimiento propio del robot, midiendo dicho movimiento con los sensores adecuados y realimentando todos los elementos de niveles superiores, para poder así hacer que el robot actúe en base a su estado.

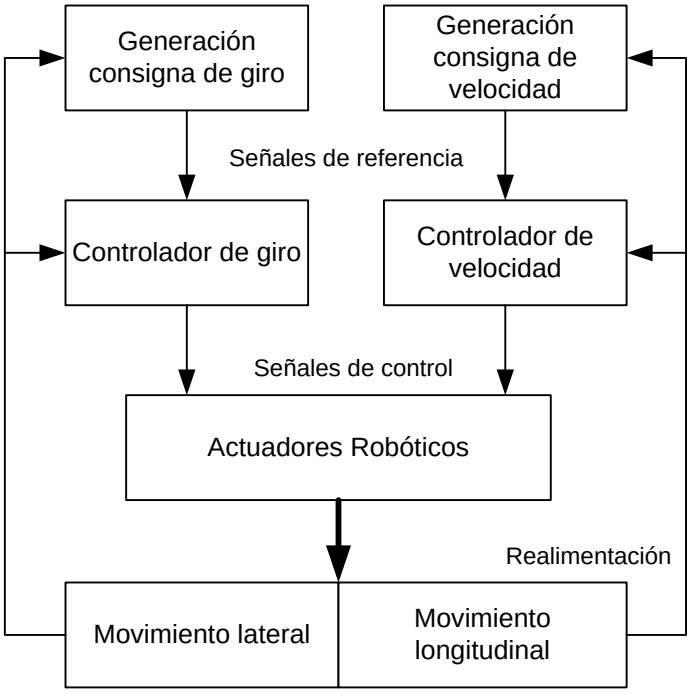


Figura 6.3: Arquitectura conceptual general para el control de un robot móvil diferencial

6.2. Control basado en spikes para robots diferenciales

Una vez expuestas las ecuaciones de movimiento de Eddie y la arquitectura de control conceptual de control usada comúnmente para robots móviles diferenciales, en este apartado vamos a exponer el diseño de un controlador PID basado en spikes que gestione sus variables de control mediante la codificación de spikes en frecuencia. Conceptualmente, podemos representar a Eddie como un sistema MIMO (Multiple-Inputs / Multiple-Outputs System), donde las entradas del sistema son las referencias de velocidad y giro, y la velocidad real de cada motor (realimentación). Las salidas del sistema son los spikes expandidos, lo cuales, tras atravesar la etapa de potencia, serán aplicados al motor.

En la Figura 6.4 mostramos el diagrama de bloques del control implementado para Eddie (aunque puede extenderse a otros robots móviles diferenciales), en él los spikes fluyen desde dos generadores sintéticos de spikes Bit-Wise (izquierda) hasta los motores (derecha), atravesando varios lazos de control hasta llegar a los Spikes Expansor y ser aplicados a los motores. Como acabamos de comentar, la velocidad del robot es la media de velocidad de las ruedas, y la velocidad de giro la diferencia entre ambas, así que vamos a usar dos generadores de spikes, uno para la referencia de velocidad del robot, y otro para generar los spikes de referencia de la velocidad de giro. La idea es que un generador de spikes nos proporcione los spikes de la referencia de velocidad a la que queremos que se mueva el robot, restando a una rueda y sumando a la otra los spikes provenientes del generador sintético de la referencia de giro, obteniendo así las referencias de velocidad individual de cada motor. Una vez obtenidas ambas referencias de velocidad, cada una de ellas es aplicada a la entrada del controlador PID basado en spikes de cada motor. Además un tercer lazo de control ha sido introducido, el cual puede ser deshabilitado, encargado de medir la diferencia de velocidad de los motores, y restarla con la referencia de velocidad de giro, obteniendo así el error de velocidad de giro cometido en cada momento, sumando o restando a los spikes de referencia de velocidad individual de cada motor este error. Con este lazo nos aseguraremos que la diferencia de velocidad entre las ruedas es la correcta. Sin embargo se ha de tener en cuenta que como la referencia de velocidad de giro es restada a un motor y sumada al otro, cuando calculamos la diferencia de velocidad entre los motores, ésta será el doble respecto a la referencia. En resumen, Eddie tiene tres lazos de control, uno para cada motor individualmente, y un tercero para la velocidad de giro del robot. Además, para poder desactivar el lazo de control de giro se ha insertado una señal de habilitación (Turn Loop Enable, TLE), de tal manera que cuando no esté habilitado no disparará ningún spike, quedando el lazo de control central deshabilitado.

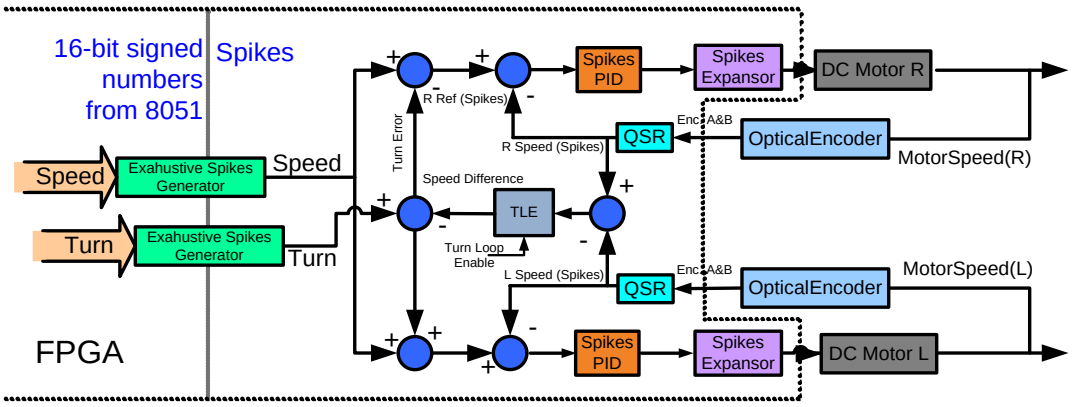


Figura 6.4: Bloques de control de Eddie

Hemos introducido otro cambio sustancial en los controladores PID basados en spikes respecto a los anteriormente presentados, ya que hemos suprimido los bancos de Integrate & Generate y de derivadores de spikes. Hemos prescindido de los bancos ya que fueron diseñados en el capítulo anterior para poder modificar el valor del número de bits de estos componentes, debido a que este parámetro había que fijarlo en tiempo de síntesis, sin embargo, Eddie no necesita modificar este parámetro para su funcionamiento, en consecuencia, simplemente hemos sintetizado un Integrate & Generate de 16 bits y un derivador de spikes de 20 bits en cada controlador PID.

En la Tabla 6-1 mostramos el sumario del informe de la síntesis del control basado en spikes de Eddie. En ella mostramos el uso de recursos de la FPGA, de forma resumida vemos que el controlador de Eddie consume 1802 slices, de los 3584 que contiene nuestra Spartan 3, un 50%, de estos slices el 13% son usados por la memoria RAM del monitor AER. Pudiendo operar como máximo a 50.98MHz. El controlador de prueba mostrado en el capítulo anterior necesitaba 1737 slices para un solo motor, sin embargo, el controlador de Eddie es capaz de controlar dos motores a partir de dos referencias distintas, incluyendo para ello un mayor número de elementos, necesita solamente 65 slices más. Esto es debido a dos factores, el primero de ellos es que el consumo hardware de los elementos periféricos del controlador (monitor AER y controlador SPI) se va diluyendo a medida que incluimos más elementos de procesamiento de spikes, y el segundo factor es la desaparición de los bancos de Integrate & Generate y derivadores de spikes, añadiendo ahora sólo un elemento de cada tipo, en vez de un banco con 4 de ellos.

Tabla 6-1 Resultados de la síntesis del controlador de EDDIE

```
Device Utilization Summary:

   Number of BUFGMUXs                        1 out of 8      12%

   Number of External IOBs                  37 out of 141    26%

      Number of LOCed IOBs                  37 out of 37    100%

   Number of Slices                       1802 out of 3584   50%

      Number of SLICEMs                    248 out of 1802   13%
```

6.2.1. Mapa de direcciones internas de Eddie

En el interior de Eddie la comunicación entre el microcontrolador y la FPGA se realiza gracias al puerto SPI, al igual que en el capítulo anterior. Sin embargo, en este caso hemos tenido que modificar el espacio de direcciones internas, adaptándolo a los componentes internos del robot. En la Tabla 6-2 mostramos el mapa de direcciones interno de Eddie. En la primera posición, 0x00, nos encontramos con el registro de control del robot, mediante el cual podemos encender / apagar los LEDS de la AER-Robot, modificar los controladores PID haciendo uso de las señales de reset de los términos integral y derivativo, y fijar el modo de funcionamiento de Eddie, habilitando o deshabilitando el lazo central de control de velocidad de giro del robot. En las posiciones 0x01 y 0x02 podemos encontrar los generadores sintéticos de spikes, generando las referencias de velocidad y giro del sistema. A continuación en las tres siguientes posiciones encontramos los elementos de control del motor derecho. En primer lugar, en la posición 0x03, encontramos el tiempo del ancho de spike que el *SpikesExpansor* aplicará a los spikes provenientes del controlador antes de aplicarlos al motor. A continuación, están situados en las posiciones 0x04 y 0x05 los divisores de frecuencia del Integrate & Generate y del derivador de spikes respectivamente. En las tres siguientes posiciones, desde la

dirección 0x06 hasta 0x08, se encuentra esta misma estructura, ancho de spikes y divisores de frecuencia, pero para el motor izquierdo, pudiendo así modificar independientemente los parámetros de cada motor. Todos los tiempos de hold de los Hold & Fire usados están situados en las direcciones 0x09 y 0x0A, pudiendo así ser todos modificados en conjunto. Finalmente, nos encontramos con las posiciones del monitor AER masivo, a partir de la dirección 0x10 hasta la posición 0x1F, el cual será descrito en mayor detalle en el siguiente apartado.

Tabla 6-2: Espacio de direcciones interno de EDDIE

Dirección SPI	Entidad	Comentarios
0x00	Registro de control	[5]-TLE [4]-SD_RST [3]-I&G_RST [2-0]-LEDs
0x01	Generador de spikes Bit-Wise de referencia de velocidad	Número de 16bits con signo
0x02	Generador de spikes Bit-Wise de referencia de giro	Número de 16bits con signo
0x03	SpikesExpansor Derecho	Ancho de spike
0x04	Integrate & Generate Derecho	Divisor de frecuencia del Integrate & Generate
0x05	Derivador de spikes Derecho	Divisor de frecuencia del derivador de spikes
0x06	SpikesExpansor Izquierdo	Ancho de spike
0x07	Integrate & Generate Izquierdo	Divisor de frecuencia del Integrate & Generate
0x08	Derivador de spikes Izquierdo	Divisor de frecuencia del derivador de spikes
0x09	Todos los Hold & Fire	+0 Tiempo de hold de la entrada U +1 Tiempo de hold de la entrada Y
0x10-0x1F	Monitor AER masivo	Dirección AER de cada spike

6.2.2. Monitorización de Eddie a través de la representación AER

Para monitorizar la actividad de los spikes en el interior de Eddie, hemos recurrido de nuevo a la representación AER, de tal modo que vamos a dar un número, o dirección a cada señal de spikes del sistema. Hemos usado exactamente el mismo monitor AER que en el capítulo anterior, pero cambiando los spikes a monitorizar. Como el controlador de Eddie tiene un elevado número de señales a monitorizar, unas 26, el ancho de banda del bus AER paralelo no es suficiente para soportar tanto tráfico AER, sólo han sido monitorizados los spikes pertenecientes a las 8 señales más relevantes del interior de Eddie. Estas señales han sido, por un lado, las consignas de velocidad y giro, y por otro lado las referencias, velocidad y salida del controlador PID de cada uno de los motores de manera individual, tal y como mostramos en la Tabla 6-3. Además en la tabla encontramos la dirección SPI de cada una de las direcciones AER de los spikes monitorizados, pudiendo modificar la dirección de cada uno de ellos en base a nuestras necesidades.

Tabla 6-3: Espacio de direcciones detallado del monitor AER de EDDIE

Dirección SPI	Dirección AER (direcciones pares para eventos positivos, e impares para los negativos)
0x10+0x00	Referencia de velocidad
+0x02	Referencia de Giro
+0x04	Refrencia individual de velocidad del motor Derecho
+0x06	Velocidad del motor Derecho
+0x08	Spikes de control del motor Derecho
+0x0A	Refrencia individual de velocidad del motor Izquierdo
+0x0C	Velocidad del motor Izquierdo
+0x0E	Spikes de control del motor Izquierdo

6.3. Excitación y análisis de las respuestas de Eddie

Al igual que en el apartado anterior, vamos a excitar a Eddie con diferentes señales de entrada, para poder monitorizar y analizar la respuesta del control que alberga en su interior. Con el fin de monitorizar la actividad interna de Eddie vamos a volver a fijarle los parámetros de configuración del control, como en el capítulo anterior, y a excitarlo con diversas señales de referencia de entrada, tanto para velocidad como para giro. De forma que a través del puerto USB del 8051, vamos a ir enviando al controlador PID basado en spikes del interior de la FPGA, los valores que deseemos desde MATLAB, monitorizando la respuesta del sistema mediante el uso de una USBAERmini2.

Para los experimentos que vamos a realizar a continuación hemos elegido un juego de parámetros de los controladores en base al estudio realizado en capítulo anterior, usando los mismos parámetros para ambos motores, y mostrados en la Tabla 6-4.

Tabla 6-4: Parámetros usados para los experimentos realizados sobre Eddie

Parámetro	Valor
Ancho de los spikes	5.12uSec
Número de bits del integrador	16bits
Divisor de frecuencia del integrador	20
Número de bits del derivador	18bits
Divisor de frecuencia del derivador	30

6.3.1. Análisis de Eddie con el lazo de control de giro habilitado

En primer lugar, manteniendo el lazo central de giro habilitado, vamos a excitar a Eddie con una señal en escalón en la referencia de velocidad, cuyos resultados se muestran en la Figura 6.5. En la figura vemos las señales de spikes monitorizadas que se especificaban en la Tabla 6-3. En primer lugar encontramos la referencia de velocidad de giro fija a 0, a continuación vemos la referencia velocidad así como las referencias individuales de cada

motor, coincidiendo todas ellas en este caso. También se incluyen las respuestas de ambos motores, mostrando un comportamiento subamortiguado y con un error en régimen permanente nulo. Ambas señales son tan similares que prácticamente se superponen. Esta similitud ha sido lograda por la introducción del tercer lazo de control, tal y como veremos más adelante. Finalmente mostramos las señales de control, salidas de los controlados PID basados en spikes, aplicada a cada motor, estas señales no son iguales, ya que cada una intenta compensar a su motor para que mantenga la misma velocidad que el otro, y no cabe decir, que aunque dos motores sean iguales, es muy poco probable que se comporten exactamente de la misma manera.

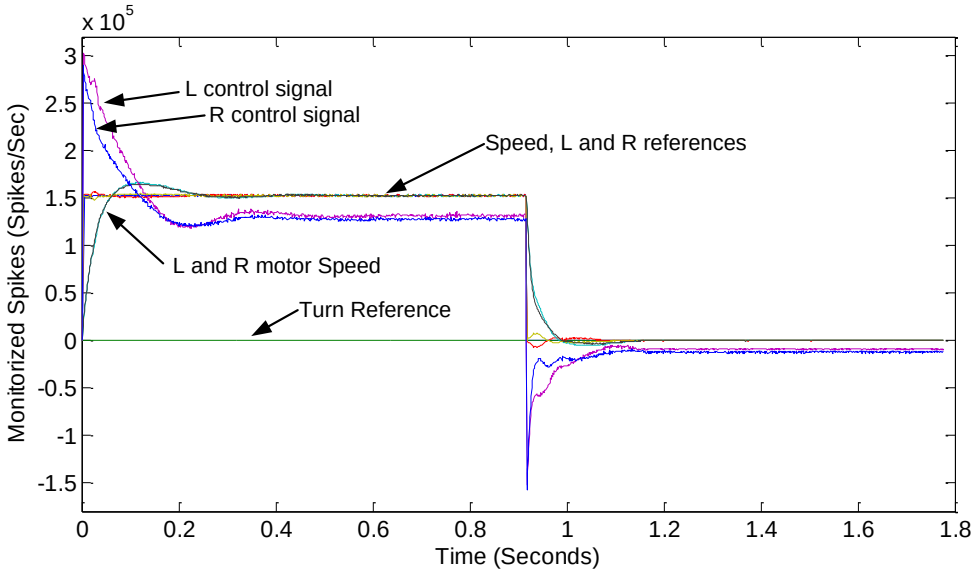


Figura 6.5: Respuesta Eddie ante una referencia en escalón de velocidad y el control de giro habilitado

A continuación hemos procedido a excitar al sistema con escalón de giro, cuya respuesta se muestra en la Figura 6.6. En ella vemos como los motores muestran velocidades, y errores, simétricos, ya que el giro en este robot consiste en la diferencia de velocidad entre las ruedas. De manera, que al estar la referencia de velocidad fija a 0, la respuesta de los motores es girar a la misma velocidad pero en sentido contrario. Mostrando nuevamente un comportamiento subamortiguado. Sin embargo, el error en régimen permanente es nulo respecto a la referencia individual del motor, pero no respecto a la referencia de la velocidad de giro.

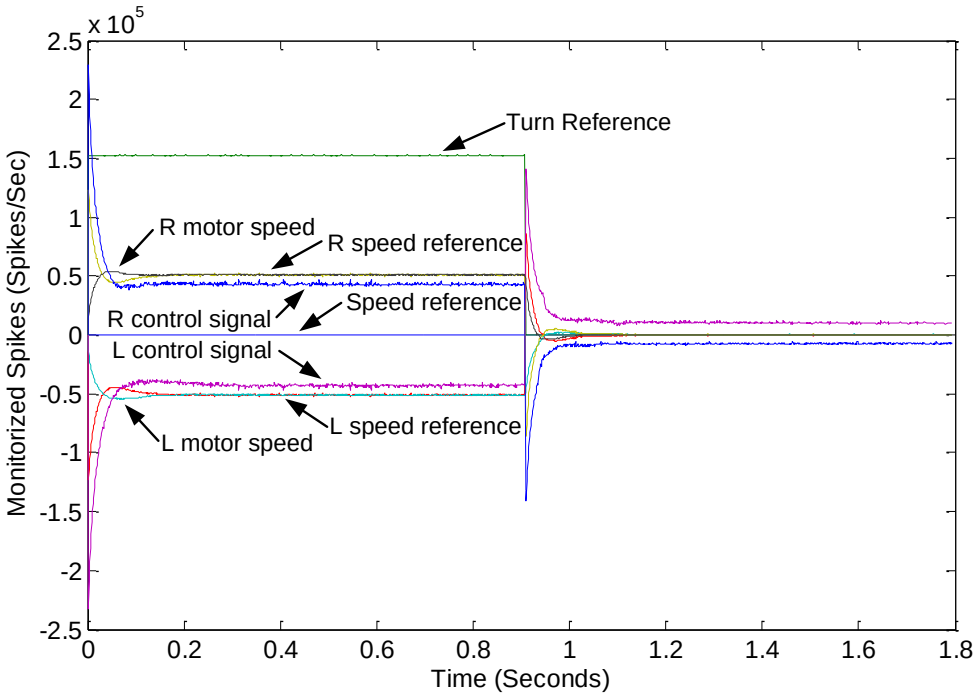


Figura 6.6: Respuesta Eddie ante una referencia en escalón de giro y el control de giro habilitado

Como puede observarse en la figura anterior, el error en régimen permanente de la diferencia de giro es elevadísimo. Esto es debido a que desde el punto de vista de la referencia de giro, o referencia de diferencia de velocidad entre los motores existe un doble lazo de control, uno que controla la diferencia de velocidad, y otro que controla la velocidad individual de cada motor. En primer lugar, al restar la velocidad de ambos motores estamos obteniendo el doble de la diferencia de la velocidad de los motores:

$$f_{MotorTurnDifference} = f_{SpeedRMotor} - f_{SpeedLMotor}$$
$$= f_{SpeedRef} + f_{TurnRef} - (f_{SpeedRef} + f_{TurnRef}) = 2 * f_{TurnRef}$$

Ecuación [6-3]

Dado que la ganancia de cada motor con su referencia individual es 1, gracias al uso de un término integral en el controlador, dado que podemos modelar la ganancia de realimentación como 2, obtenemos la ganancia equivalente de la velocidad de giro del motor:

$$K_{EddieTurn} = \frac{K_{ClosedLoop}}{1 + K_{FeedBack}K_{ClosedLoop}} \approx \frac{1}{1 + 2 * 1} = 0.33$$

Ecuación [6-4]

De tal manera que mediante la introducción de este lazo de control encontramos un error en régimen permanente cercano al 67%. Sin embargo, este hecho deja constancia de la aplicación del principio de superposición en nuestros controles basados en spikes.

El experimento que nos resta por realizar en este apartado es la excitación simultánea de Eddie con dos escalones, uno de velocidad y otro de giro. La Figura 6.7 muestra la respuesta de Eddie ante esta situación. En ella se puede apreciar como la referencia de velocidad es sumada a la referencia de giro para obtener la referenciad el motor derecho, así como restada

para obtener la referencia del motor izquierdo. Ambos motores siguen respectivamente sus referencias individuales, cometiendo un error en régimen permanente respecto a la velocidad de giro del robot.

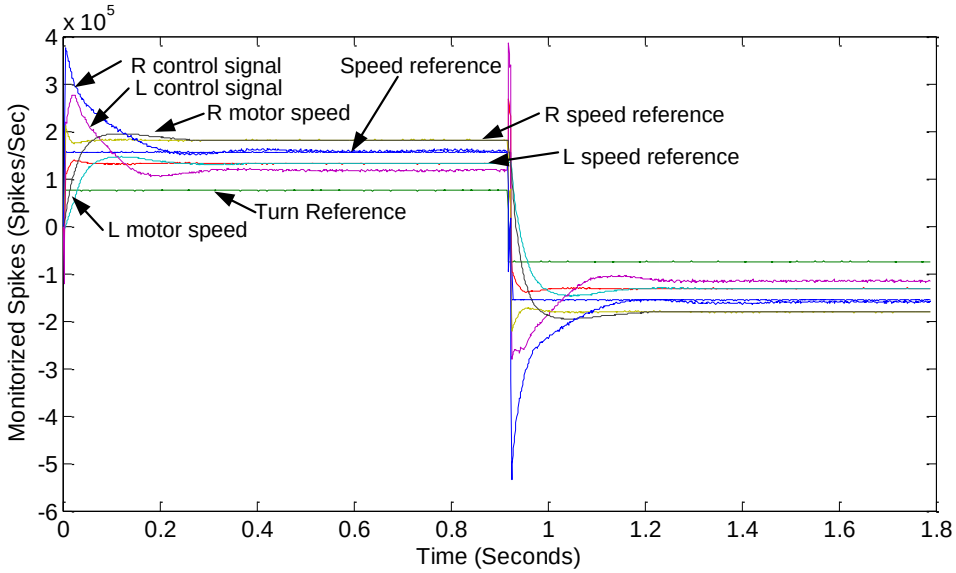


Figura 6.7: Respuesta Eddie ante una referencia escalones de velocidad y giro con el control de giro habilitado

6.3.2. Análisis de Eddie con el lazo de control de giro deshabilitado

En este apartado vamos a analizar el comportamiento de Eddie cuando desactivamos el lazo de control de giro. En primer lugar hemos excitado a Eddie con un escalón en velocidad. En la Figura 6.8 pueden verse los resultados obtenidos. Esta figura es muy semejante a la del caso anterior, ya que el lazo central de control de giro sólo afecta al giro, no a la velocidad principal del robot. La única diferencia con la respuesta del robot en el mismo caso del apartado anterior radica en que las velocidades de los motores no son tan semejantes. Los motores no están rotando siempre a la misma velocidad, ya que en el caso anterior el lazo de control de giro hacía que sus velocidades fueran muy semejantes, prácticamente coincidentes.

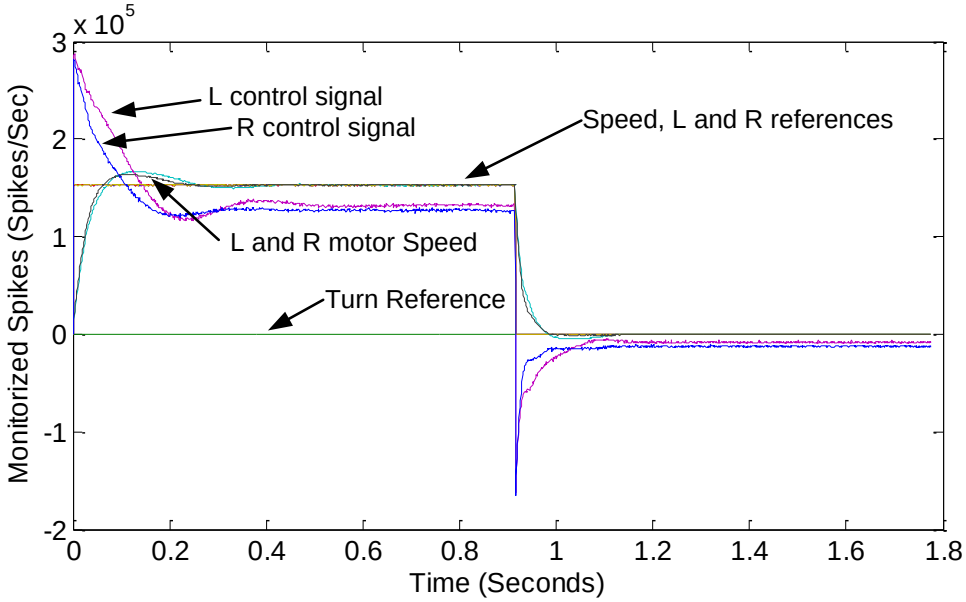


Figura 6.8: Respuesta Eddie ante una referencia de velocidad en escalón y el control de giro deshabilitado

En este punto, al igual que en el apartado anterior vamos a proporcionarle a Eddie un escalón de giro, mostrado en la Figura 6.9. En ella se puede apreciar como las velocidades de los motores ahora no sólo alcanzan sus referencias individuales, sino también la referencia de la velocidad de giro del robot, a diferencia del sistema con el lazo de control de giro habilitado expuesto en el apartado anterior. Eliminándose el error en régimen permanente acaecido con anterioridad.

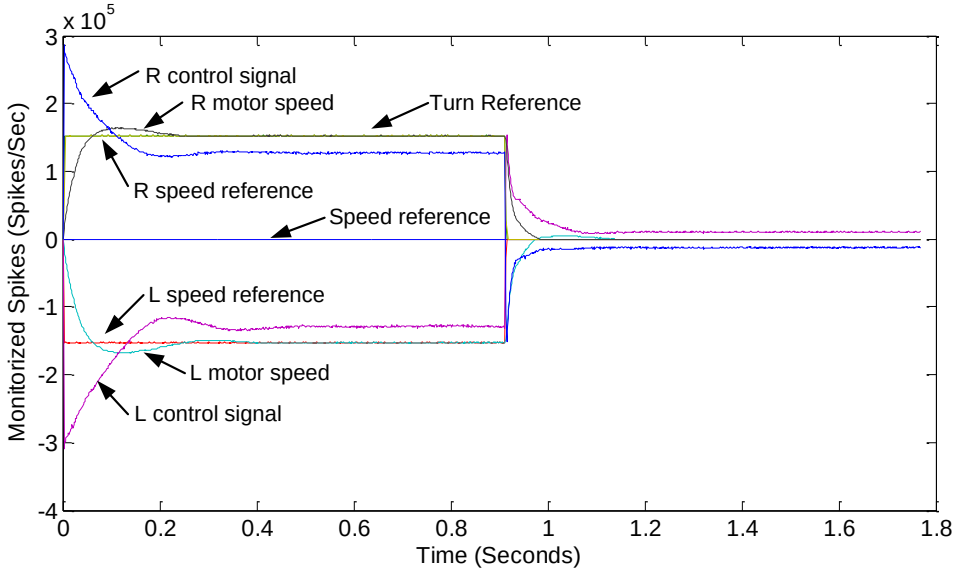


Figura 6.9: Respuesta Eddie ante una referencia de giro en escalón y el control de giro deshabilitado

Finalmente, para concluir con el análisis de Eddie vamos a excitarlo con dos escalones, uno en velocidad y otro de giro. La Figura 6.10 muestra la respuesta de Eddie. A diferencia de los resultados obtenidos en el apartado anterior, los motores alcanzan todas las referencias y desaparecen los errores en régimen permanente.

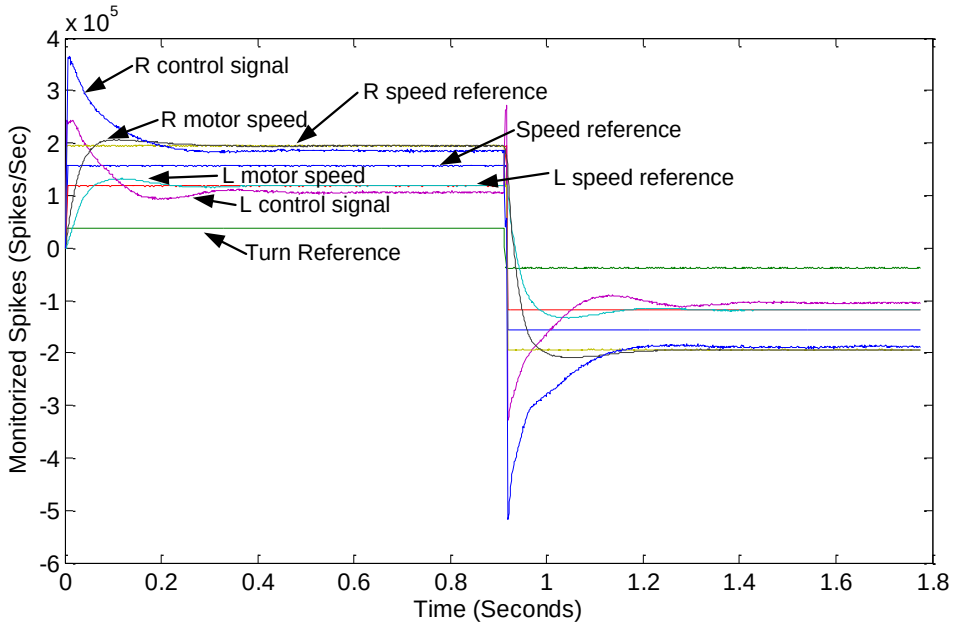


Figura 6.10: Respuesta Eddie ante una referencia en escalón de velocidad y giro con el lazo de control de giro deshabilitado

6.4. Control desde emisora de Radio / Control

Para que Eddie pueda navegar por el mundo y no dependa de un PC con un puerto USB, hemos sustituido a este último por un receptor de radio común de aeromodelismo. El receptor elegido usa una modulación PPM (Pulse Position Modulation), funciona a 40MHz, e incluye 4 canales. Para decodificar los canales del receptor hemos conectado la señal recibida a través de la radio antes de ser desmultiplexada al microcontrolador. Siendo la única tarea de este último el desmultiplexar los canales, escalarlos, y enviárselos a la FPGA a través del puerto SPI al generador sintético de referencias adecuado (velocidad y giro). En la Figura 6.11 mostramos una captura de la señal aplicada al microcontrolador. En este tipo de modulación nos encontramos el siguiente patrón: Un tiempo de sincronización a '1' muy elevado, del orden de milisegundos, y a continuación una serie de pulsos, siendo el tiempo a '1' variable, estando la información contenida en este tiempo, y un tiempo a '0' fijo entre ellos, usado para separar los canales entre sí. Siendo la posición de cada pulso el número del canal en cuestión, es decir, el primer pulso corresponde al canal 1, el segundo al canal 2, y así sucesivamente.

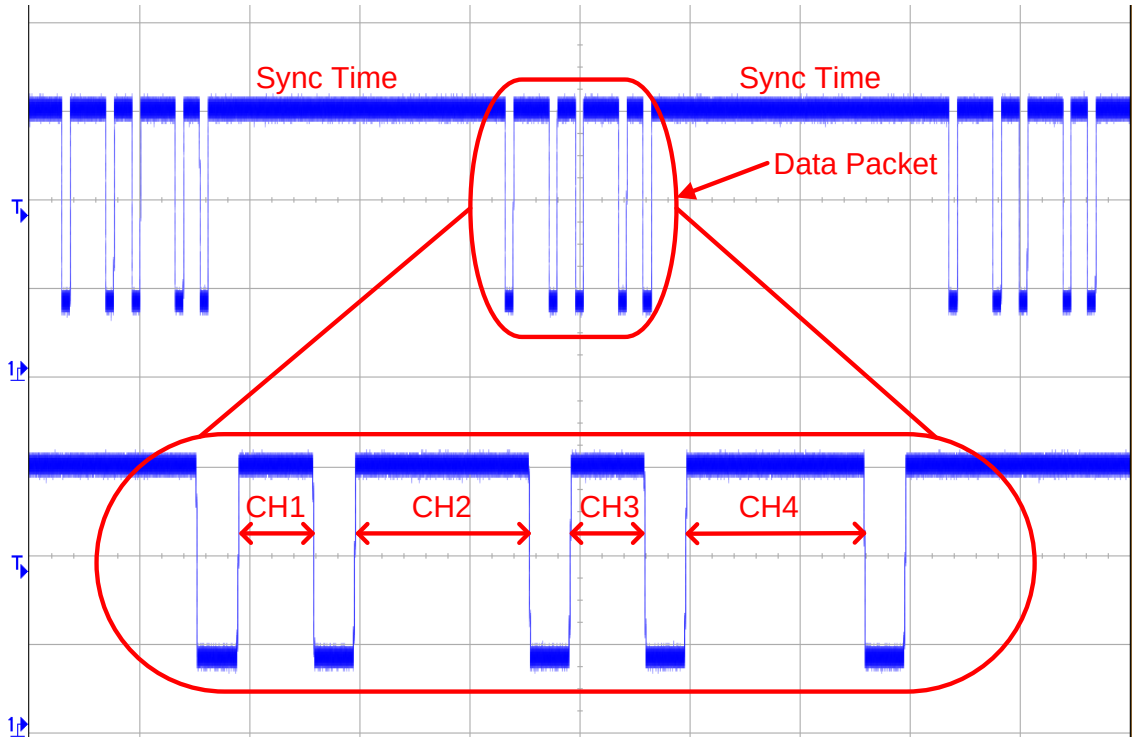


Figura 6.11: Señal modulada en PPM recibida por el microcontrolador

Para decodificar la información de cada canal hemos de medir el tiempo de los pulsos a '1' recibidos, con este objetivo hemos utilizado un dispositivo incluido en nuestro microcontrolador [C8051F320] llamado el "Programable Counter Array" (PCA), cuyo diagrama de bloques se muestra en la Figura 6.12. Como su nombre indica, el PCA es un conjunto de contadores / temporizadores programables que pueden ser usados con múltiples propósitos. En nuestro caso vamos a usar sólo un módulo que va a ser configurado en modo edge-triggered (disparo por flanco), mostrado en la Figura 6.13. El PCA en este modo mide el tiempo entre pulsos, y además nos dispara una interrupción en cada flanco de la señal PPM. De tal manera que cada vez que se dispare la interrupción encontraremos en un registro determinado el tiempo exacto en el que se produjo la interrupción. Así que podemos ir reconstruyendo el valor de la señal PPM comparando el tiempo actual con el de la última vez que se disparó la interrupción, y anotando el tiempo actual para ser comparado con la próxima marca de tiempo en la siguiente interrupción. Gracias a esta técnica podemos almacenar el tiempo entre pulsos con una precisión muy elevada. Sólo nos resta estar sincronizados con la señal PPM para saber en qué posición se encuentra cada pulso, para poder así decodificar el canal adecuado. Esta cuestión se resuelve esperando a que el tiempo medido por la PCA sea mayor que cierto umbral suficientemente elevado temporalmente hablando, asegurándonos así que va a comenzar una nueva transferencia y que el primer pulso corresponde al primer canal.

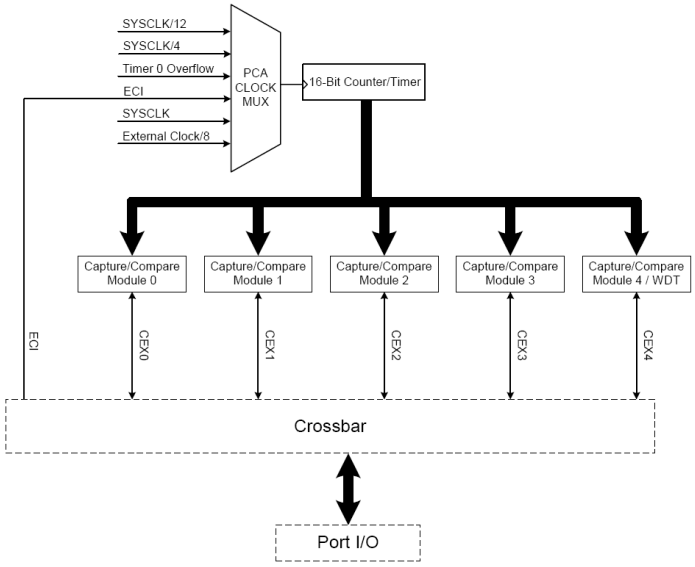


Figura 6.12: Diagrama de bloques del PCA incluido en el C8051F320

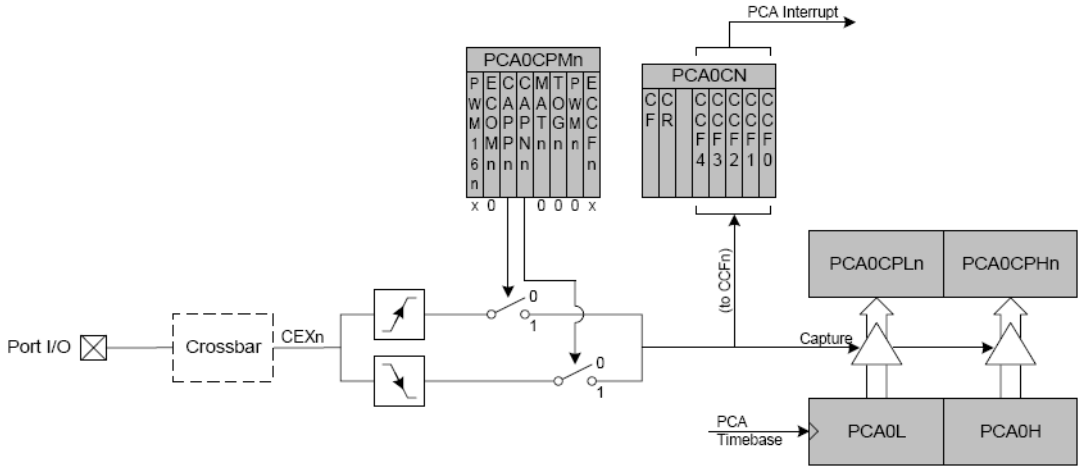


Figura 6.13: Estructura de un contador del PCA modo edge-triggered

Una vez descodificados los distintos canales, sólo resta escalar la medida del tiempo de cada canal al rango de operación de Eddie, en este caso particular un número entre ±350, y enviarlo a través del bus SPI a la dirección de la FPGA correcta, 0x01 si acabamos de recibir la información del canal de velocidad, ó 0x02 si el canal es el de giro. Además, podemos interceptar mensajes dirigidos al microcontrolador, como por ejemplo, el cambio de marcha, es decir, indicarle al microcontrolador que convierta los valores del canal de velocidad en base al valor de otro canal. O enviarle comandos más complejos, como aumentar / disminuir el ancho de spike, resetear la FPGA, o activar / desactivar el lazo de control central. Pudiendo finalmente controlar así a Eddie como si fuera un coche de radio control, cómodamente desde una emisora estándar, y alterando su comportamiento según precise el experimentador.

7. Diseño y análisis de elementos para el procesado de spikes basados en la realimentación del Integrate & Generate

En capítulos anteriores se han ido mostrando una serie de mecanismos y elementos que permiten adaptar los métodos de control clásicos a sistemas con representación y procesado pulsante. Dada la proximidad a nivel teórico de la disciplina del control automático con la del procesamiento de señales, y ya que hemos diseñado bloques para el control relativamente equivalentes a los controles analógicos, en este capítulo vamos darle un enfoque distinto a los elementos ya diseñados, orientándolos al procesamiento de señales basadas en spikes, PSS, o en inglés "Spikes-based Signal Processing", SSP. Generalizaremos y extenderemos la teoría sobre el procesado mediante la representación pulsante de la información.

Los sistemas de procesamiento de señales digitales clásicos, DSP, se utilizan para realizar procesamientos matemáticos sobre señales digitales [Mitra93]. Los procesamientos más típicos son la transformada de Fourier discreta [Mitra93], los filtros de respuesta infinita a impulso (IIR) [Ifeachor02] , los filtros de respuesta finita a impulso (FIR) [Ifeachor02], o cualquier función de transferencia en dominio Z en general[Oppemheim92]. Todos estos procesamientos se basan en tomar un sub conjunto de las muestras digitales de entrada, y particularmente en los filtros IIR también de salida, multiplicarlas por una serie de coeficientes y sumar el resultado de las multiplicaciones, obteniendo así una nueva muestra de salida de la señal procesada. Estos sistemas representan las señales como muestras digitales, las cuales pueden estar codificadas de diversas maneras, como por ejemplo números enteros, en coma fija o coma flotante, necesitando diversos anchos de bits para representar las muestras dependiendo de la precisión deseada. Además complejas unidades aritméticas hardware son necesarias para realizar las operaciones de multiplicación y suma, conocidas comúnmente como multiplicate & accumulate (MAC). Dada la complejidad de estas unidades, sólo unas pocas pueden ser implementadas dentro de un DSP, siendo muy común encontrar sólo una [TMS320C672x]. Apareciendo así la necesidad de reutilizarlas en el tiempo, y convertir los procesamientos de señales en algoritmos secuenciales que vayan cambiando los datos de entrada de estas unidades, realizando un procesamiento iterativo. El problema surge cuando queremos realizar un procesado de varias señales dentro de un DSP, ya que el procesado de cada muestra requiere un tiempo de computación nada despreciable, necesitando mantener un ritmo de procesado de muestras superior, o al menos equivalente, a la frecuencia de muestreo de cada una de las señales de entrada, teniendo en cuenta que el comportamiento algorítmico del procesado de la señal introduce instrucciones de cabecera, u overhead, que hacen que se decremente la eficiencia del DSP. Finalmente estos procesados pueden necesitar importantes cantidades de memoria, para almacenar el "historial" de las muestras, tanto de entrada como de salida en el caso de los filtros IIR, así como los coeficientes asociados a cada procesamiento en particular, además de necesitar una memoria en la que almacenar el algoritmo de procesamiento.

Si retomamos los modelos neuronales pulsantes, los procesamientos sobre señales de spikes que vamos a implementar tienen características completamente opuestas a los tradicionales DSP, siendo más parecidos a cómo funciona el sistema nervioso de los seres

vivos. Por ejemplo, la memoria no está centralizada como en los DSP, sino distribuida en los elementos de computación, pudiendo afirmar que la memoria, tanto de código como de datos, son los elementos de procesado en sí, además gracias a la representación de la información en spikes no son necesarios complejas unidades aritméticas, la operación más compleja que vamos a realizar es añadir 1 a un registro, y además, gracias a la simplicidad del hardware, lo vamos a replicar tantas veces como lo necesitemos para un procesado concreto, obteniendo un sistema de procesamiento de señales masivamente paralelo.

Un filtro analógico es un sistema que amplifica o atenúa en cierta medida determinadas componentes frecuenciales de una señal analógica [Oppemheim98] [Williams81]. Dado que la amplitud de las señales codificadas como spikes se encuentra en la frecuencia de dichos spikes, los cambios de la frecuencia de los spikes serían los cambios en amplitud de la señal que representan. En consecuencia, y aunque resulte algo enrevesado, podríamos definir como un filtro frecuencial de spikes aquel que filtre la frecuencia de los cambios de la frecuencia de los spikes. Por ejemplo, un filtro paso de baja de spikes funcionará atenuando las componentes de alta frecuencia en los cambios de la frecuencia de los spikes que la componen, proporcionando a su salida una nueva secuencia de spikes, la cual solamente contendrá componentes de baja frecuencia de la señal original que representan.

En los próximos apartados vamos a realimentar el Integrate & Generate usando un Hold & Fire, ambos elementos presentados en los capítulos anteriores, para crear distintos tipos de filtros frecuenciales de spikes, así como simularlos para poder analizarlos. Los distintos tipos de filtros han sido diseñados de forma incremental, de tal forma que partiendo del filtro básico de paso de baja, formado por un Integrate & Generate y un Hold & Fire, se han diseñado filtros cada vez más complejos. En primer lugar, vamos a presentar al filtro de spikes paso de baja con ganancia unitaria, es decir, un filtro que atenuará las componentes frecuenciales más altas, permitiendo el paso de las bajas frecuencias con una ganancia igual a 1 ó 0dB. A continuación vamos a introducir un nuevo componente, el divisor de spikes, para crear filtros con ganancia superior a 0dB y filtros con ganancia completamente ajustable. Finalmente, basándonos en los filtros paso de baja con ganancia ajustable vamos a diseñar filtros de spikes paso de alta y paso de banda.

7.1. Diseño de filtros de paso de baja basados en spikes

En este apartado vamos a abordar el diseño de filtros de spikes de paso de baja, siendo su objetivo filtrar los cambios de alta frecuencia en la frecuencia de los spikes, tal y como si lo hiciéramos en la amplitud de la señal analógica tomada como referencia. Como acabamos de comentar, vamos a comenzar exponiendo el diseño de filtros paso de baja basados en spikes, con una ganancia en la banda de paso unitaria o de 0dB, para a continuación ir perfeccionándolos hasta la obtención de filtros con ganancia completamente ajustable.

7.1.1. Diseño de filtros de paso de baja basados en spikes con ganancia unitaria

En primer lugar vamos a proceder al diseño de un filtro de spikes paso de baja con ganancia unitaria. Este es el caso del filtro más simple, y más a delante mostraremos cómo diseñar filtros con ganancias variables, así como sus implicaciones. Para diseñar este filtro paso de baja vamos realimentar un Integrate & Generate gracias a la ayuda de un Hold & Fire [Jimenez10a]. El diagrama de bloques interno del filtro básico se encuentra en la Figura 7.1, donde se muestra cómo se han conectado ambos componentes, estando el Integrate & Generate realimentado negativamente con la ayuda de un Hold & Fire. La entrada del Integrate & Generate es la diferencia entre la frecuencia de los spikes (la entrada *SpikesIn)* y la propia salida del Integrate & Generate. De tal manera que el Integrate & Generate integrará la diferencia entre el valor integrado y la entrada, y su salida serán los spikes filtrados.

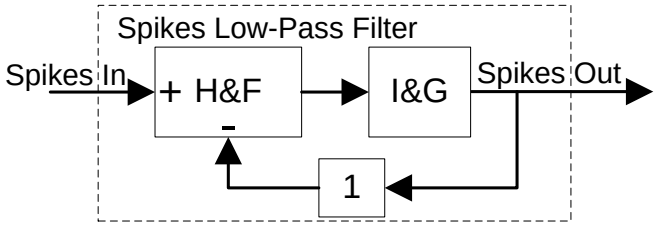


Figura 7.1: Diagrama de bloques del filtro paso de baja de spikes con ganancia unitaria

Aplicando la teoría de sistemas, si tomamos la función de transferencia en el domino S del Integrate & Generate, y calculamos el sistema equivalente tras realimentar el Integrate & Generate con ganancia unitaria, usando la Ecuación [4-10] obtenemos:

$$F_{SLPF}(s) = \frac{F_{outSpikes}(s)}{F_{inputSpikes}(s)} = \frac{I\&G(s)}{1 + 1 * I\&G(s)} = \frac{K_{BWSpikesGen}/_{s}}{1 + K_{BWSpikesGen}/_{s}} = \frac{K_{BWSpikesGen}}{s + K_{BWSpikesGen}}$$

Ecuación [7-1]

El sistema obtenido es equivalente a un filtro paso de baja [Pallás99] con un polo en $K_{BWSpikesGen}$ y con ganancia unitaria:

$$K_{SLPF} = \frac{K_{BWSpikesGen}}{K_{BWSpikesGen}} = 1$$

Ecuación [7-2]

Siendo su frecuencia de corte en radianes por segundo equivalente a la ganancia del generador de spikes sintéticos Bit-Wise, incluido en el interior del Integrate & Generate. La cual podemos calcularla usando la Ecuación [2-22]:

$$\omega_{cut-off} = K_{BWSpikesGen} = \frac{F_{CLK}}{2^{n-1}(intFreqDiv + 1)}$$

Ecuación [7-3]

Donde n es el número de bits, e intFreqDiv el divisor de frecuencia del Integrate & Generate situado en el interior del filtro.

Para analizar el comportamiento de este filtro lo hemos simulado con Simulink junto con Xilinx SystemGenerator, tal y como hicimos en los primeros capítulos con el controlador basado en spikes. Para ello, hemos diseñado un escenario de simulación, el cual será reutilizado más adelante para simular el resto de filtros diseñados. El escenario de simulación diseñado es mostrado en la Figura 7.2, en la que pueden observarse las entradas del sistema a la izquierda, en las que un generador sintético de spikes conectado a la salida de un generador de funciones, generará los spikes que excitarán al filtro. A continuación encontramos el filtro de spikes, en este caso el filtro paso de baja de spikes con ganancia unitaria, al que someteremos a diversos estímulos para analizar su respuesta. Finalmente hallamos un bloque para la comunicación de los resultados de las simulaciones al espacio de trabajo de MATLAB, el cual nos permitirá analizar las respuestas del filtro.

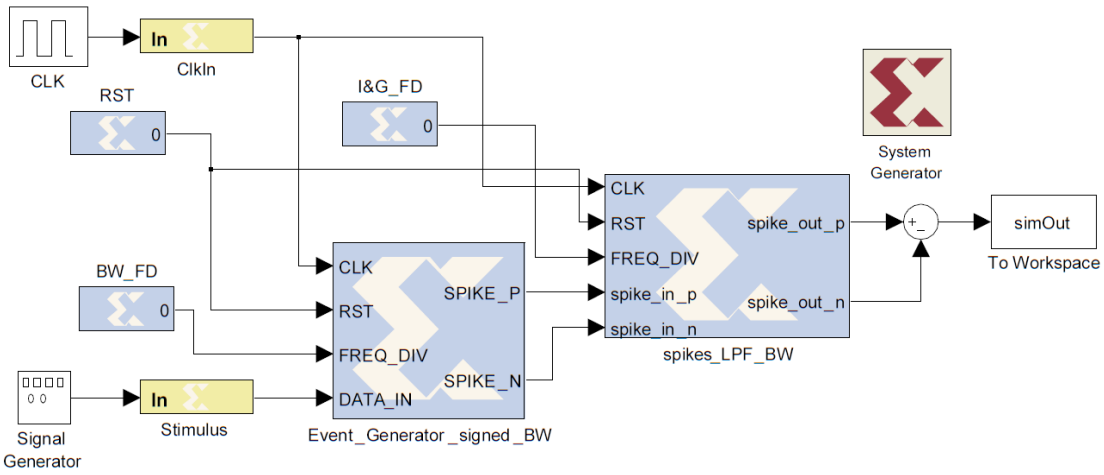


Figura 7.2: Escenario de simulación diseñado para los filtros de spikes

Acorde con la Ecuación [7-5] el polo del filtro puede ajustarse en base a dos parámetros, el número de bits del Integrate & Generate, y el divisor de frecuencia de su generador interno. Así que hemos realizado una serie de simulaciones para comprobar el efecto de ambos parámetros en la respuesta del filtro.

En primer lugar hemos procedido a comprobar la respuesta temporal del filtro. Para ello lo hemos estimulado, partiendo del reposo, con una secuencia de spikes con frecuencia constate, como si de una señal de entrada en escalón se tratase. La Figura 7.3 muestra en su parte superior la frecuencia de los spikes de entrada del filtro, en verde, y la reconstrucción de los spikes de la salida del filtro, además en la parte inferior de la figura podemos encontrar los spikes componentes de ambas señales. Los spikes de salida de filtro aumentan exponencialmente hasta alcanzar la frecuencia de los spikes presentes a su entrada, tal y cabría esperar de un filtro paso de baja con la amplitud de una señal.

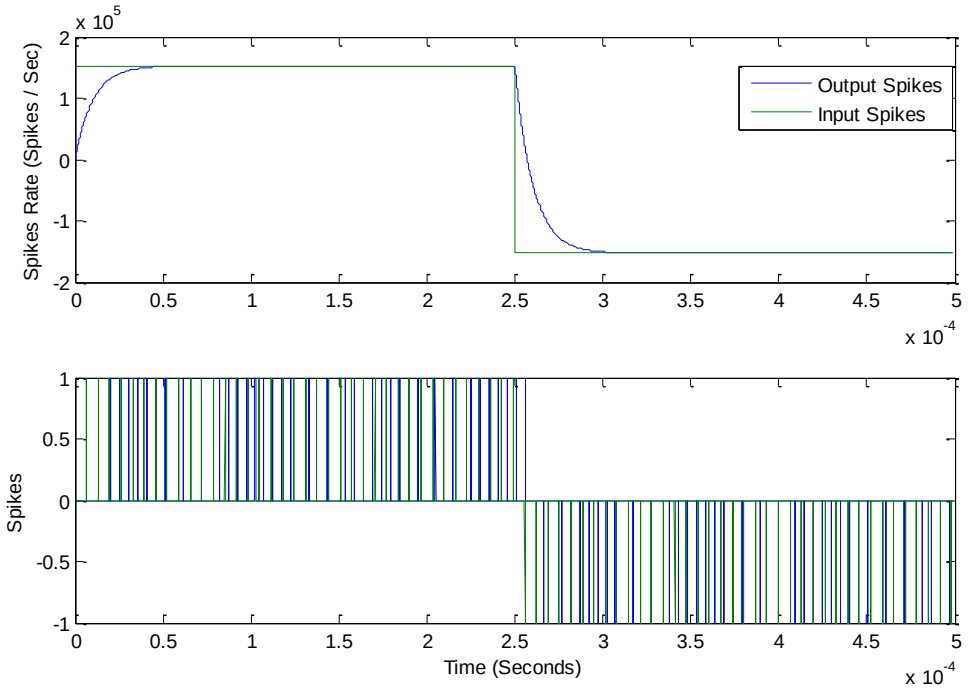


Figura 7.3: Respuesta del filtro paso de baja de spikes ante una entrada en escalón

Para comenzar con el análisis del filtro hemos fijado el divisor de frecuencia a 0, y hemos modificado el número de bits entre 9 y 12. Los parámetros utilizados, así como el tiempo de subida de la salida del filtro teórica esperada podemos encontrarlos en la Tabla 7-1. La respuesta temporal del filtro para los diversos parámetros asignados es mostrada en la Figura 7.4, en la que se muestra la reconstrucción de la frecuencia de los spikes de salida del filtro, en trazos continuos, así como la respuesta teórica de un filtro paso de baja ideal de semejantes características, en trazos discontinuos. En ella se aprecia cómo la frecuencia de los spikes de salida crece exponencialmente hasta llegar a la misma frecuencia de los spikes aplicados a la entrada del filtro, comportándose de forma semejante a un filtro de primer orden, en la que el tiempo de subida de la señal a su salida depende de la inversa de la frecuencia de corte del filtro (4Ц) [Williams81]. Además, incrementando el número de bits disminuimos la frecuencia de corte del filtro, y en consecuencia, aumentamos el tiempo de subida de la frecuencia de los spikes a la salida del filtro.

Tabla 7-1: Tiempo de subida del filtro de spikes paso de baja frente al número de bits del Integrate & Generate

Número de bits	Divisor de frecuencia	Tiempo de subida(Segundos)
9	0	0.2*10-4
10	0	0.4*10-4
11	0	0.8*10-4
12	0	1.6*10-4

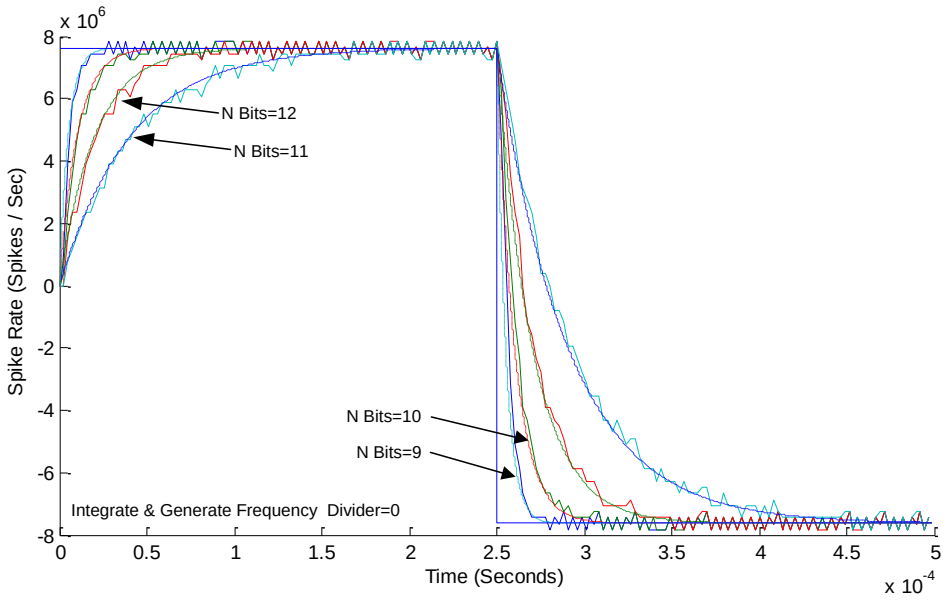


Figura 7.4: Respuestas del filtro paso de baja de spikes con diversos números de bits

A continuación hemos repetido el experimento anterior, pero manteniendo fijo el número de bits a 10, y modificando el divisor de frecuencia entre 0 y 8, con incrementos de 2, tal y como exponemos en la Tabla 7-2. Los resultados son mostrados en la Figura 7.5, y son análogos a los obtenidos con anterioridad. Al igual que en el caso anterior, al decrementar la ganancia del Integrate & Generate, también decrementamos la frecuencia del corte del filtro, aumentando en consecuencia el tiempo de subida de la salida del filtro. Sin embargo, hay que destacar los dos últimos casos, cuando el divisor de frecuencia vale 6 y 8, en ambos casos la salida del filtro se satura a una determinada frecuencia de los spikes de salida. Esto se debe a la saturación del Integrate & Generate, cuya frecuencia máxima es dependiente del divisor de frecuencia tal y como exponía la Ecuación [4-14].

Tabla 7-2: Tiempo de subida del filtro de spikes paso de baja frente al divisor de frecuencia del Integrate & Generate

Número de bits	Divisor de frecuencia	Tiempo de subida (Segundos)
10	0	0.4*10-4
10	2	1.2*10-4
10	4	2*10-4
10	6	2.8*10-4
10	8	3.6*10-4

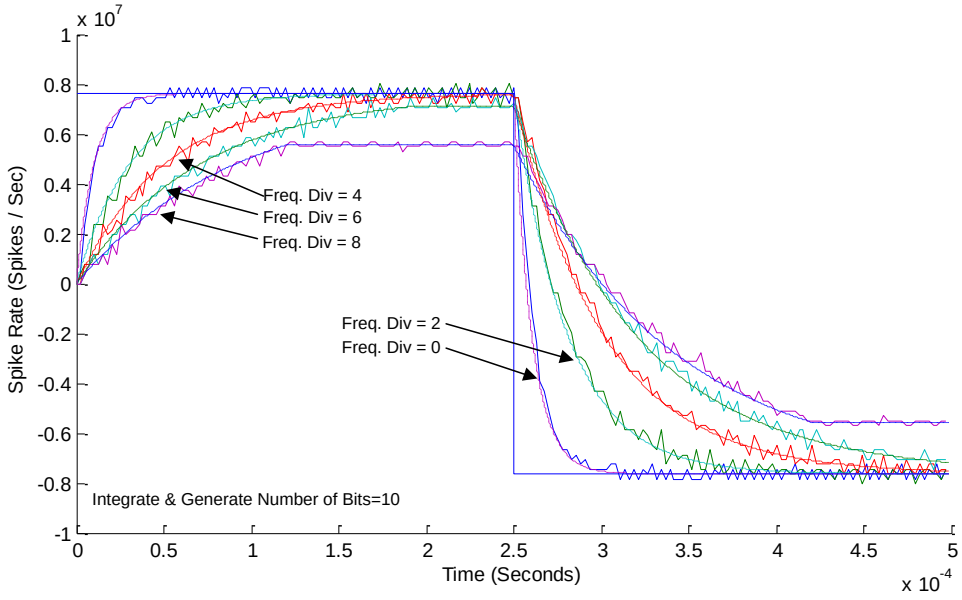


Figura 7.5: Respuestas del filtro paso de baja de spikes con diversos divisores de frecuencia

Tras analizar la respuesta temporal del filtro, vamos a proceder a estudiar su respuesta frecuencial. Con este propósito hemos realizado una serie de simulaciones, en las que el filtro ha sido excitado con una secuencia de spikes cuya frecuencia cambia senoidalmente, y a diversas frecuencias de cambio. En otras palabras, hemos aplicado al generador de spikes señales senoidales, en amplitud, cambiando su frecuencia iterativamente, realizando así un barrido en frecuencia de la respuesta del filtro y anotando la amplitud del filtro y calculando su ganancia en cada caso. Las simulaciones han sido realizadas variando tanto el número de bits, en la Figura 7.6, como el divisor de frecuencia, en la Figura 7.7, del Integrate & Generate. Se obteniene una representación de la respuesta del filtro equivalente a un diagrama de Bode. Las tablas 7-3 y 7-4 muestran los parámetros utilizados así como la frecuencia de corte teórica del filtro. En ambas figuras encontramos la respuesta del filtro simulada, en línea continua, y la respuesta de un filtro ideal en similares condiciones, en línea discontinua. En primer lugar vemos cómo el filtro tiene una ganancia de 0dB en la banda de paso, además se observa el comportamiento del filtro paso de baja, proporcionando una ganancia de -3dB en la frecuencia de corte, y a partir de ahí decrementando la ganancia de la señal de salida en -20dB por década [Oppemheim98].

Tabla 7-3: Frecuencia de corte del filtro de spikes paso de baja frente al número de bits del Integrate & Generate

Número de bits	Divisor de frecuencia	Frecuencia de corte (Hz)
10	0	1.5*104
11	0	0.77*104
12	0	0.38*104
13	0	0.19*104
14	0	0.09*104

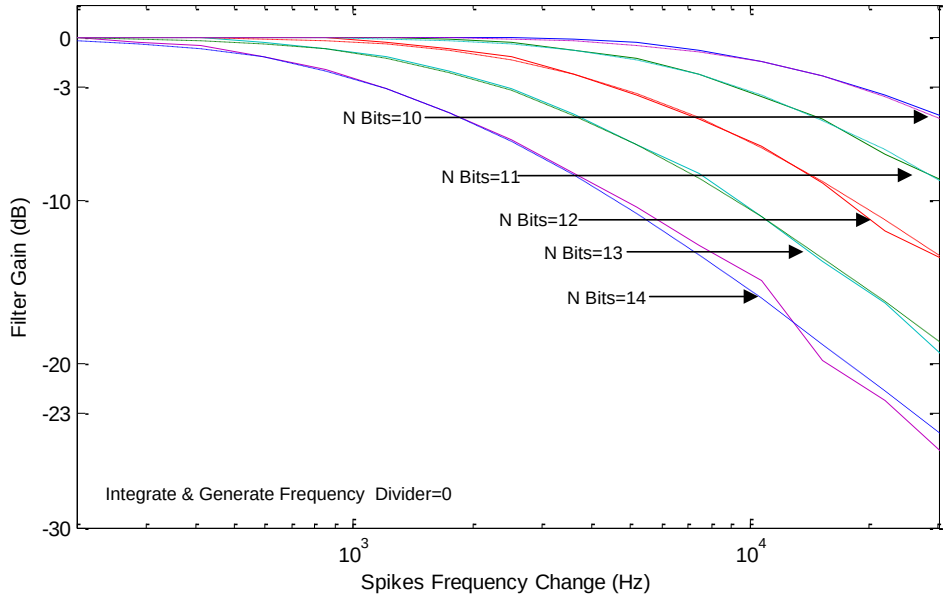


Figura 7.6: Respuesta frecuencial del filtro paso de baja de spikes con diversos números de bits

Tabla 7-4: Frecuencia de corte del filtro de spikes paso de baja frente al divisor de frecuencia del Integrate & Generate

Número de bits	Divisor de frecuencia	Frecuencia de corte (Hz)
10	0	1.5*104
10	1	0.77*104
10	2	0.51*104
10	3	0.38*104
10	4	0.31*104

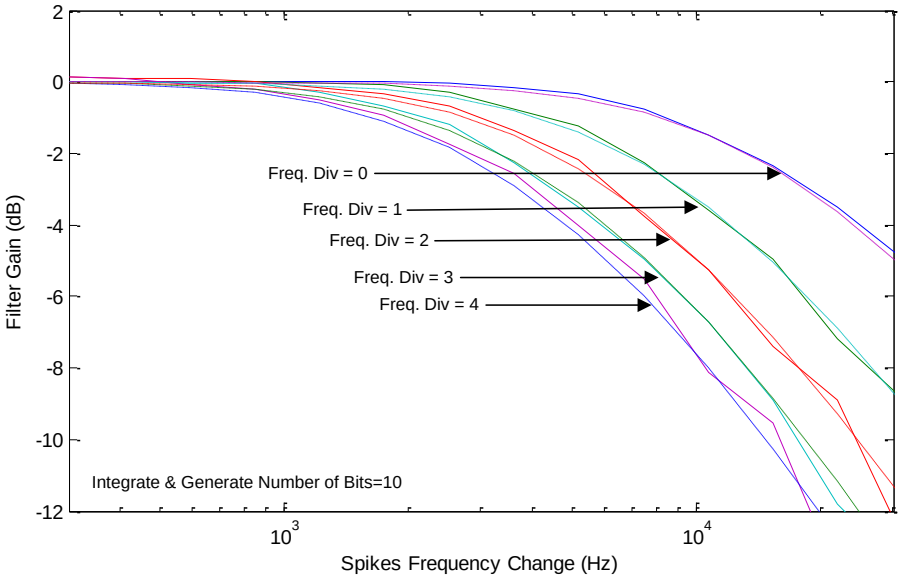


Figura 7.7: Respuesta frecuencial del filtro paso de baja de spikes con diversos divisores de frecuencia

7.1.2. Diseño de filtros de paso de baja basados en spikes con ganancia ajustable. Los divisores de spikes

Los filtros mostrados hasta ahora tienen el inconveniente de presentar una ganancia unitaria, o 1. Una nueva cuestión a abarcar es cómo diseñar filtros con ganancia diferente a 1, tal y como podemos hacer con los filtros tradicionales. A continuación vamos a exponer cómo diseñar filtros con ganancias superiores a 1, para en siguientes apartados diseñar filtros con ganancia completamente ajustable.

Para diseñar filtros con ganancias superiores a 1, nos vamos a apoyar de nuevo en la teoría clásica de sistemas [Oppenheim98], con el fin buscar modelos que nos permitan incrementar esta característica en nuestro filtro. Sea FB(s) la función de transferencia de un nuevo elemento situado en lazo de realimentación, es decir, entre la salida del Integrate & Generate y la entrada negativa del Hold & Fire, la función de transferencia del filtro equivalente sería:

$$\frac{F_{outSpikes}(s)}{F_{inputSpikes}(s)} = \frac{SI\&G(s)}{1 + FB(s) * I\&G(s)} = \frac{K_{BWSpikesGen}}{s + FB(s) * K_{BWSpikesGen}}$$

Ecuación [7-4]

Multiplicar la frecuencia de los spikes por un número entero puede ser una tarea muy ardua, sin embargo dividirla no resulta tan complicada. Entendiendo un divisor de spikes como un elemento cuya salida es una secuencia de spikes, y su frecuencia es la división entre un número entero y la frecuencia de los spikes de entrada, y que sin embargo, la distribución temporal de los eventos sea la más homogénea posible. La idea para obtener filtros con ganancias mayores a 1 es situar en la realimentación un elemento que realice dicha operación, tal y como muestra la Figura 7.8 con el nombre de *Spikes Div*.

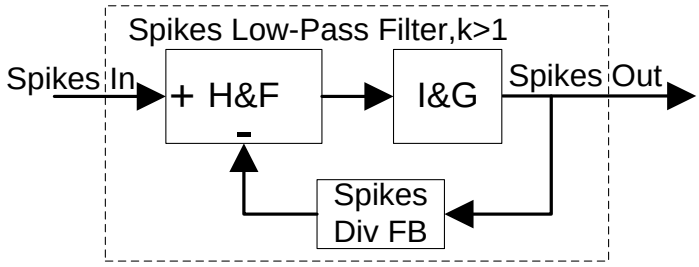


Figura 7.8: Diagrama de bloques del filtro paso de baja de spikes con ganancia mayor que 1

Acorde con la figura anterior la función de transferencia del divisor de spikes puede verse como un multiplicador que multiplica la frecuencia de señal de entrada entre [0 – 1], siendo 0 la ausencia total de spikes, y 1 representaría una salida con una frecuencia similar a la entrada. Sustituyendo la función de transferencia del divisor de spikes $k_{spikesDiv}$ en la ecuación anterior, obtenemos la nueva función de transferencia del filtro:

$$\frac{F_{outSpikes}(s)}{F_{inputSpikes}(s)} = \frac{SI\&G(s)}{1 + FB(s) * SI\&G(s)} = \frac{K_{BWSpikesGen}}{s + K_{spikesDiv} * K_{BWSpikesGen}}$$

Ecuación [7-5]

Calculando la ganancia del filtro k_{SLPF}:

$$K_{SLPF} = \frac{K_{BWSpikesGen}}{K_{spikesDiv} * K_{BWSpikesGen}} = \frac{1}{K_{spikesDiv}}$$

Ecuación [7-6]

Dado que $k_{spikesDiv}$ está comprendida entre [0 - 1], podemos ajustar la ganancia del filtro modificando este parámetro entre [1 - ∞] o en un caso real, entre 1 y la frecuencia máxima de saturación del filtro. Sin embargo, este parámetro no sólo afecta a la ganancia del filtro, sino también al polo del filtro, empujándolo más a la izquierda cuanta más ganancia le asignemos al filtro, así como manteniendo su valor original cuando le proporcionemos a $k_{spikesDiv}$ un valor de 1, tal y como muestra la siguiente ecuación:

$$\omega_{cut-off} = K_{spikesDiv} * K_{BWSpikesGen}$$

Ecuación [7-7]

A continuación vamos a proponer tres elementos que actuarán como divisores de spikes mediante el uso de diversas estrategias, la primera basada en la generación de números aleatorios, el divisor de spikes probabilístico, la segunda basada en el método bit-wise expuesto con anterioridad para la generación de spikes, y estrategia final basada en el uso de contadores digitales.

7.1.2.1. El divisor de spikes probabilístico

La primera opción por la que nos decantamos para diseñar e implementar un divisor de spikes fue usar técnicas probabilísticas. Dichas técnicas ya se han usado en el pasado para procesamiento de información AER, ya sea para generarlas [Linares03][Linares06b], como para realizar convoluciones [Paz08][Paz09a]. La idea de estos mecanismos es usar un generador de

números aleatorias, disparando un evento AER o no en base a la probabilidad asignada a dicho evento.

Aunque podrían realizarse diversas implementaciones sobre la decisión probabilística de dispará un spike o no, en nuestro caso, vamos a diseñar un componente que sólo permita que lo atraviesen los spikes a su entrada durante un cierto porcentaje del tiempo. La Figura 7.9 muestra el diagrama de bloques del circuito diseñado para el derivador de spikes probabilístico. En la parte superior de la figura encontramos un registro de generación de número aleatorios, conocido como registro de desplazamiento realimentado a la izquierda, o en inglés *left shifted feedback register* (LFSR), cuya descripción detallada se puede encontrar en [LFSR]. Hemos optado por un LFSR de 32 bits, aprovechando solamente los 16 bits inferiores, intentando así obtener números aleatorios con la mínima correlación posible [Clemente01]. Estos 16 bits inferiores están conectados a un comparador digital, el cual compara dicho valor aleatorio con el coeficiente de división asignado, *spikesDiv* de 16 bits, habilitando los buffers situados en la parte inferior de la figura sólo cuando la salida del LFSR sea menor que la señal *spikesDiv*. Pudiéndose calcular la proporción de tiempo que estará la salida de los spikes habilitada como el cociente entre *spikesDiv* y número de bits de LFSR utilizados:

$$t_{bufferOn} = \frac{spikesDiv}{2^{16}}$$

Ecuación [7-8]

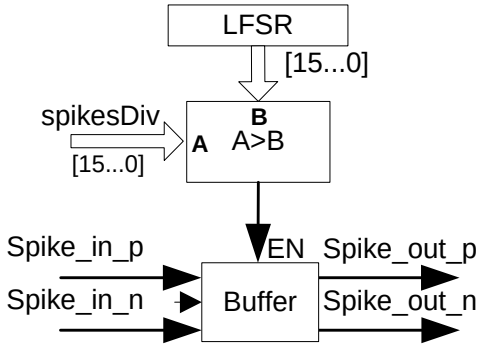


Figura 7.9: Esquemático del divisor de spikes probabilístico

Y en consecuencia, la frecuencia de los spikes de salida se vería afectada por dicho cociente, comprendido entre 0 y 1:

$$f_{spikesDivOut} = t_{bufferOn} * f_{spikesDivIn}$$

Ecuación [7-9]

A continuación el divisor de spikes probabilístico ha sido incluido en la realimentación del filtro paso de baja, y hemos procedido a simularlo. La Figura 7.10 muestra la salida del filtro ante una entrada en escalón, en ella presentamos la reconstrucción de la salida del filtro, en trazo continuo así como la respuesta teórica ideal que debería mostrar el filtro, en trazos discontinuos. Para la simulación hemos fijado la ganancia del Integrate & Generate con 9 bits y sin divisor de frecuencia, 0, así como variando de 10% en 10% la probabilidad de que un spike pueda atravesar el divisor de spikes probabilístico. De esta figura hay que comentar varios

detalles. En primer lugar se puede observar cómo cuando tenemos una probabilidad del 100%, la salida del filtro tiene una ganancia unitaria de nuevo, ya que al no dividir los spikes volvemos a tener un filtro equivalente al anterior, sin embargo, cuando empezamos a disminuir la probabilidad, dividiendo en mayor medida la señal de la realimentación, la ganancia del filtro empieza a aumentar inversamente proporcional a la probabilidad. Por ejemplo, con un 50% (0.5) obtenemos una ganancia de 2, y con un 10% (0.1) una ganancia de 10. En siguiente lugar también se aprecia cómo la ganancia de la realimentación afecta al polo, ya que si observamos el tiempo de subida de cada respuesta, el tiempo de respuesta va creciendo acorde con la ganancia del filtro. También presenta el caso de la saturación del filtro a la máxima frecuencia del Integrate & Generate que lleva en su interior, en este caso a 50M Spikes/Sec. Y en último lugar, la respuesta no es limpia e incluye una elevada cantidad de "ruido" introducido por el LFSR, debido a la falta de homogeneidad de los valores generados.

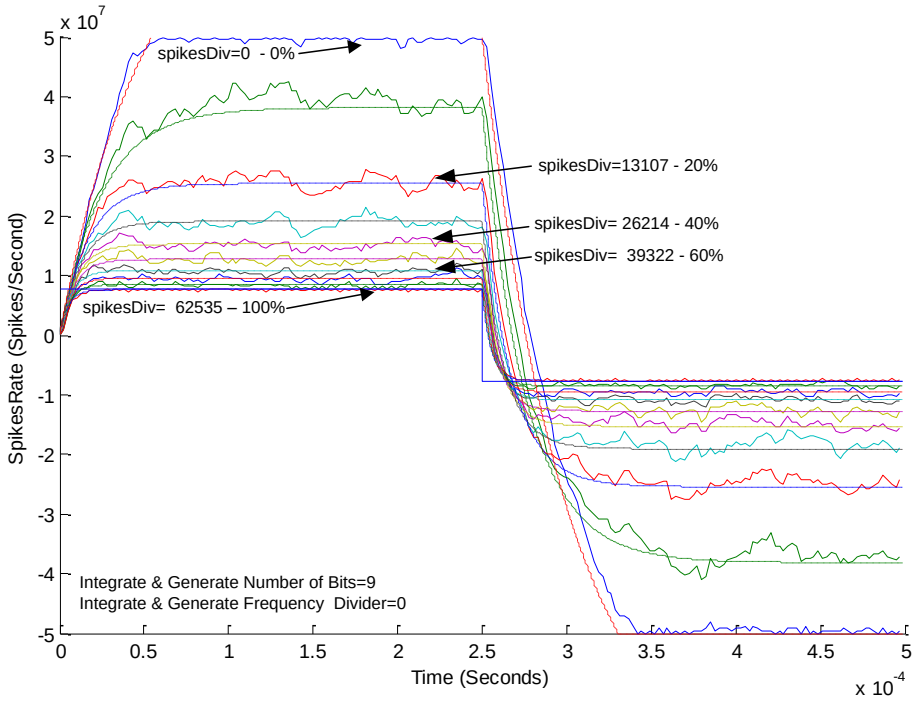


Figura 7.10: Respuesta ante escalón del filtro paso de baja con un divisor probabilístico en la realimentación

7.1.2.2. Divisor de spikes basado en el método Bit-Wise

Tal y como expusimos en el Capítulo 2, el generador bit-wise generaba spikes distribuidos lo más homogéneamente posible a lo largo del tiempo [Paz09a]. A partir de las características del generador bit-wise surge la idea de usarlo como divisor de spikes. La idea del divisor de spikes basado en el método bit-wise radica en usar un contador, cuyas entradas se incrementan con cada spike que es recibido en su puerto de entrada, invirtiendo bit a bit la salida del contador y comparándola con el valor del divisor de spikes, habilitando finalmente, o no, un buffer situado a su salida. De esta manera mantenemos abierto el "tránsito de spikes" un número de veces inversamente proporcional al valor del divisor de frecuencia de los spikes.

En la Figura 7.11 se muestran los componentes internos del divisor de spikes basado en el método bit-wise. En la parte superior de la figura encontramos el contador digital, el cual se incrementará cada vez que reciba un spike, ya sea negativo o positivo. La salida de dicho contador es invertida bit a bit y comparada con el valor del divisor de spikes, *spikesDiv*, en la parte central de la figura. Finalmente, la salida del comparador habilitará un buffer que permitirá el tránsito de los spikes presentes en su entrada.

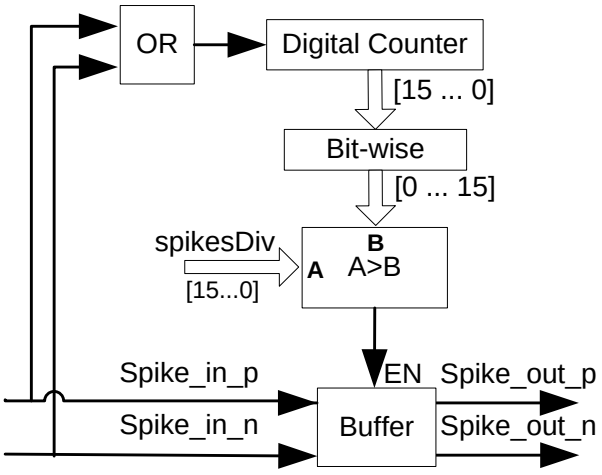


Figura 7.11: Esquemático del divisor de spikes basado en el método bit-wise

De tal manera, que al igual que en el caso anterior, la proporción de tiempo que el buffer estará dispuesto a dejar pasar un spike, será el cociente entre el valor *spikesDiv* y la potencia del número de bits del contador del divisor de spikes. En consecuencia, la frecuencia de los spikes de salida se vería afectada por dicho cociente, comprendido entre 0 y 1:

$$f_{spikesDivOut} = \frac{spikesDiv}{2^{16}} * f_{spikesDivIn}$$

Ecuación [7-10]

A continuación, con el fin de analizar el comportamiento de este componente, lo hemos incluido en la realimentación del filtro paso de baja, y hemos procedido a simularlo con los parámetros mostrados en la Tabla 7-5. La Figura 7.12 muestra la salida del filtro ante una entrada en escalón, en ella hemos representado la reconstrucción de la salida del filtro, en trazo continuo así como la respuesta teórica ideal que debería mostrar el filtro. Para la simulación hemos fijado la ganancia del Integrate & Generate con 9 bits y sin divisor de frecuencia, 0, y hemos ido variando de 10% en 10% señal *spikesDiv*. Al igual que en el apartado anterior, hemos de comentar varias cuestiones acerca de estos resultados, la primera y más importante con respecto al método anterior, es la limpieza de las respuestas obtenidas, gracias a la mejor distribución temporal de los spikes proporcionada por método. Además, de nuevo se observa claramente cómo la ganancia del filtro es afectada por este parámetro, evidenciando una elevadísima aproximación a la respuesta ideal del filtro, así como que el valor del divisor de spikes desplaza el polo del filtro, aumentando el tiempo de subida del filtro con el incremento de la ganancia.

Tabla 7-5: Ganancia y tiempo de subida del filtro de spikes paso de baja frente al valor divisor de spikes de la realimentación

Numero de bits	Divisor de frecuencia	Divisor de spikes	Ganancia del filtro	Tiempo de subida (Segundos)
9	0	0.0	Infinita	Infinita
9	0	0.1	10	2.04*10-4
9	0	0.2	5	1.02*10-4
9	0	0.3	3.33	0.68*10-4
9	0	0.4	2.5	0.51*10-4
9	0	0.5	2	0.41*10-4
9	0	0.6	1.66	0.34*10-4
9	0	0.7	1.42	0.29*10-4
9	0	0.8	1.25	0.22*10-4
9	0	0.9	1.11	0.2*10-4
9	0	1.0	1.0	0.16*10-4

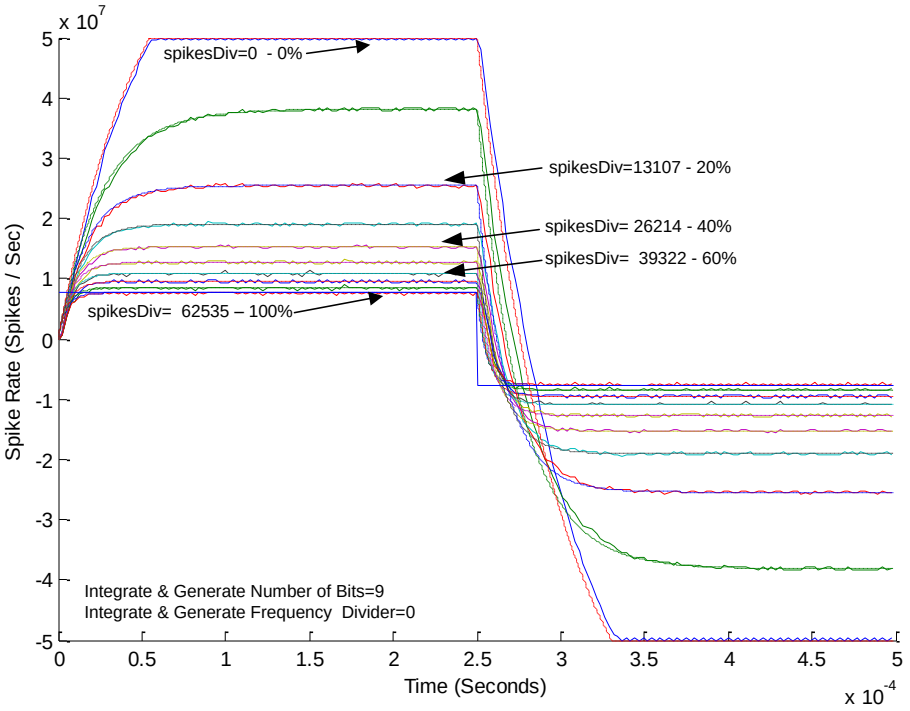


Figura 7.12: Respuesta ante escalón del filtro paso de baja con un divisor basado en el método bit-wise en la realimentación

Para conocer la respuesta frecuencial de este filtro, hemos realizado una secuencia de simulaciones, mostradas en la Figura 7.12. En ellas vemos la respuesta frecuencia ante diversos valores del divisor de spikes, *spikesDiv*. Si observamos la figura cuando *spikesDiv* tiene un valor muy alto, la ganancia de la banda de paso de las primeras respuestas aumenta inversamente proporcional al valor de *spikesDiv*. Sin embargo, la frecuencia de corte también se ve alterada por este parámetro, empujando el polo a la derecha a medida que aumentamos su valor.

Cuando *spikesDiv* tiene un valor muy pequeño muy pocos spikes atraviesan el divisor, haciendo que el Integrate & Generate se comporte casi como un integrador puro, saturándose a su máxima frecuencia. A medida que los cambios de los spikes de entrada aumentan de frecuencia, comienza a comportarse como un filtro paso de baja, gracias a que al comenzar a filtrar spikes su frecuencia de salida baja del nivel de saturación. Estando todos los parámetros usados en las simulaciones de este filtro mostrados en la Tabla 7-6.

Tabla 7-6: Ganancia y frecuencia de corte del filtro de spikes paso de baja frente al valor divisor de spikes de la realimentación

Numero de bits	Divisor de frecuencia	Divisor de spikes	Ganancia del filtro (dB)	Frecuencia de corte (Hz)
12	0	0	Infinita	0
12	0	0.2	13.97	0.77*103
12	0	0.4	7.95	1.55*103
12	0	0.6	4.4	2.33*103
12	0	0.8	1.94	3.1*103
12	0	1.0	0	3.88*103

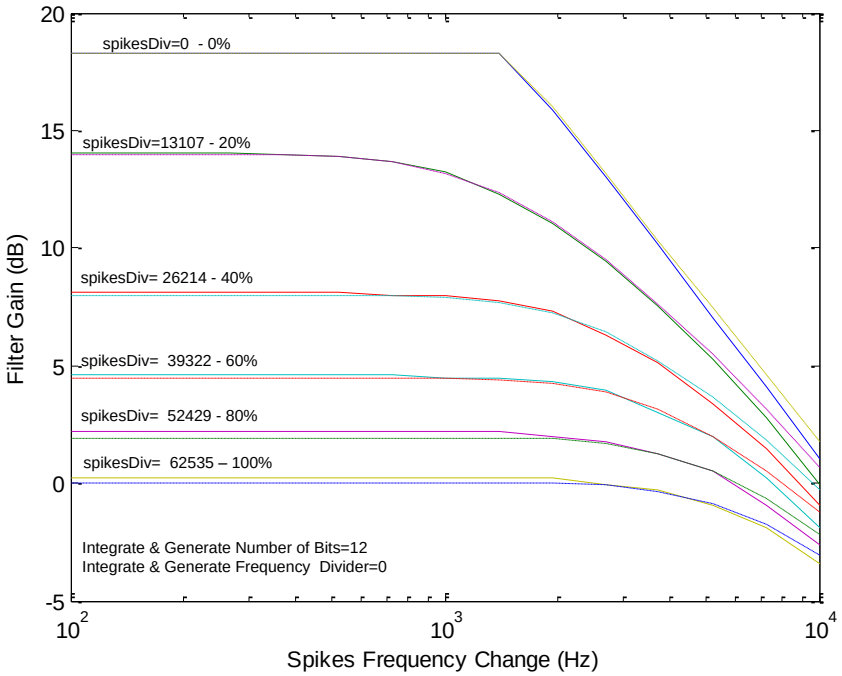


Figura 7.13: Diagrama de Bode del filtro paso de baja con un divisor en la realimentación para diversos valores del divisor

7.1.2.3. Divisor de spikes basado en contadores digitales

El último divisor de spikes que vamos a proponer es el divisor basado en contadores con tamaño o módulo programable. Es el mecanismo más intuitivo de hacer la división de un tren de pulsos. Consiste en un contador más un comparador como se muestra en la

Figura 7.14. La idea es acumular en el contador el número de spikes que se reciben por el puerto de entrada, y en caso de que la cuenta sea mayor que el valor de la señal spikesDiv, la señal a la salida del comparador inicializará el contador digital. Además la salida del comparador será el propio spike de salida, ya que estará solamente un ciclo activada, debido a que al ciclo de reloj siguiente a la activación de esta señal el contador digital se pondrá a "0", dejando de ser la entrada B del comparador mayor que la entrada A.

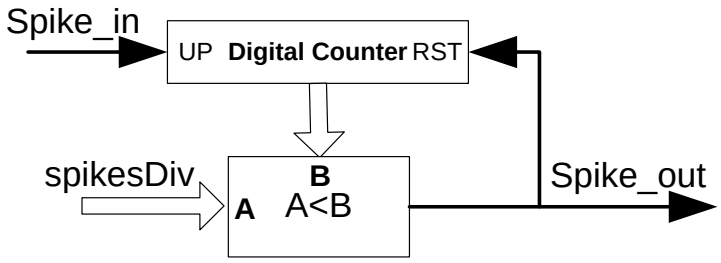


Figura 7.14: Diagrama de bloques del divisor de spikes basado en contador sin signo

Este método es algo más simple que divisor de spikes basado en el método bit-wise, ya que no requiere la inversión de los bits del contador. Para poder utilizar este mecanismo con signo se propone replicar el sistema de la Figura 7.14, utilizando un contador y un comparador para el canal de los spikes positivos y otro conjunto igual para los negativos. Evidentemente este divisor tiene un comportamiento prácticamente igual al anteriormente propuesto basado en el bit-wise.

7.1.3. Diseño de filtros de paso de baja basados en spikes con ganancia completamente ajustable.

Finalmente vamos a diseñar un filtro de spikes paso de baja con ganancia totalmente ajustable, para ello se ha añadido un divisor de spikes a la salida del filtro de spikes con ganancia superior a 1. El diagrama de bloques de este filtro se puede encontrar en la Figura 7.15, en el que puede verse el nuevo divisor de spikes localizado a la salida del filtro. De esta manera podemos elevar la ganancia del filtro gracias al divisor situado en la realimentación, y proceder a disminuirla con el divisor de spikes situado a la salida del filtro.

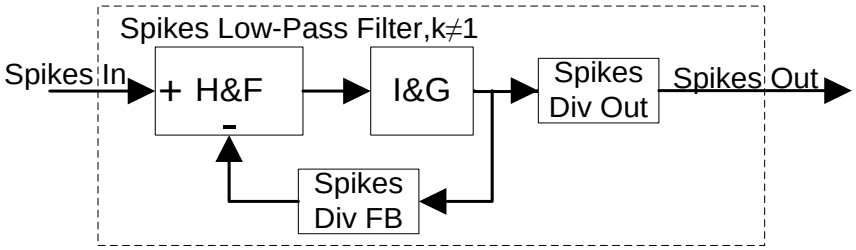


Figura 7.15: Diagrama de bloques del filtro paso de baja de spikes con ganancia ajustable

Sea $K_{BWSpikesGen}$ la ganancia del Integrate & Generate, $K_{spikesDivFB}$ la ganancia del divisor de spikes del lazo de realimentación, y $K_{spikesDivOut}$ la ganancia del divisor de spikes situado a la salida del filtro. Podemos calcular la función de transferencia del filtro de spikes paso de baja

con ganancia ajustable como el producto de la función de transferencia del filtro paso de spikes con ganancia superior a 1 y la ganancia del divisor de spikes situado a la salida de filtro. Se ha de tener en cuenta que las ganancias de los divisores de spikes son siempre menores que 1.

$$F_{SLPF}(s) = \frac{K_{spikesDivOut} * K_{BWSpikesGen}}{s + K_{spikesDivFB} * K_{BWSpikesGen}}$$

Ecuación [7-11]

Calculando la ganancia del filtro, k_{SLPF}, como el cociente entre ambas ganancias de los divisores de spikes:

$$K_{SLPF} = \frac{K_{spikesDivOut} * K_{BWSpikesGen}}{s + K_{spikesDivFB} * K_{BWSpikesGen}} = \frac{K_{spikesDivOut}}{K_{spikesDivFB}} = \frac{\frac{spikesDiv}{2^{16}}}{\frac{spikesDiv_{FB}}{2^{16}}} = \frac{spikesDiv_{Out}}{spikesDiv_{FB}}$$

Ecuación [7-12]

Sin embargo, la frecuencia del polo sigue dependiendo de $K_{spikesDivFB}$ y $K_{BWSpikesGen}$:

$$\omega_{cut-off} = K_{spikesDivFB} * K_{BWSpikesGen}$$

Ecuación [7-13]

Simulando la respuesta ante un escalón de este filtro para diversos pares de divisores, con diversas relaciones gananciales, hemos obtenido la Figura 7.15. En dicha figura se observa cómo a la respuesta de la reconstrucción de los spikes es muy semejante a la respuesta teórica de un filtro ideal en semejantes condiciones. Además puede verse claramente como la ganancia del filtro es acorde al cociente entre los divisores de spikes, según la Ecuación [7-12], pero el tiempo de respuesta del filtro presenta un polo dependiente de su ganancia, $K_{BWSpikesGen}$ y de la ganancia del divisor de spikes situado en el lazo de realimentación, $K_{spikesDivFB}$, tal y como predecía la Ecuación [7-13]. Los parámetros utilizados en estas simulaciones pueden encontrarse en la Tabla 7-7, así como la ganancia teórica esperada y el tiempo de subida de la señal de salida.

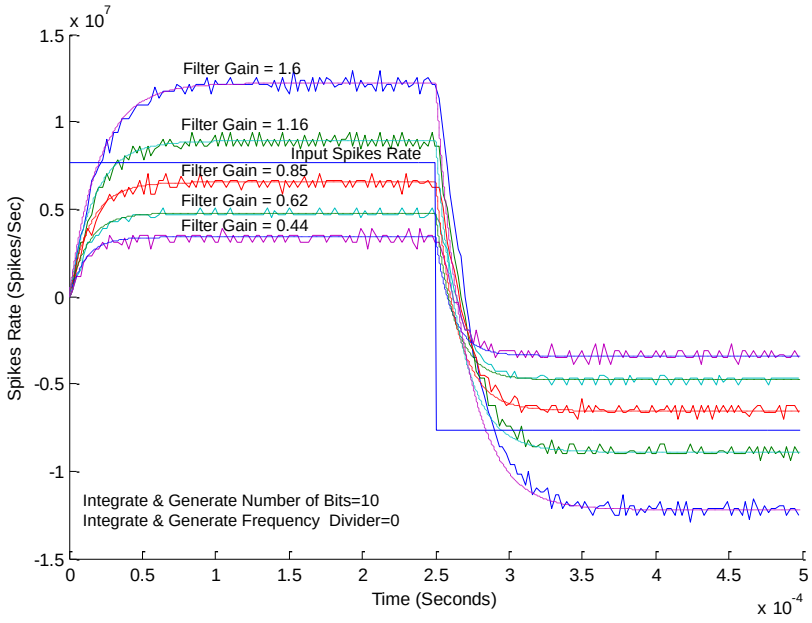


Figura 7.16: Respuesta ante escalón del filtro paso de baja con ganancia ajustable

Tabla 7-7: Ganancia y tiempo de subida del filtro de spikes paso de baja frente a los valores de los divisores de spikes

Número de simulación	Número de bits	Divisor de frecuencia	Divisor de spikes de salida	Divisor de spikes de realimentación	Ganancia	Tiempo de subida (Segundos)
1	10	0	0.8	0.5	1.6	6.82*10-5
2	10	0	0.7	0.6	1.16	5.85*10-5
3	10	0	0.6	0.7	0.85	5.12*10-5
4	10	0	0.5	0.8	0.62	4.55*10-5
5	10	0	0.4	0.9	0.44	3.84*10-5

Finalmente hemos realizado una serie de simulaciones para determinar la respuesta frecuencial del filtro de spikes paso de baja con ganancia ajustable. Dichas simulaciones se han realizado con diversos pares de ganancias de los divisores de spikes, y son mostradas en la Figura 7.17. En ella se observa claramente la influencia de $K_{spikesDivFB}$ en el polo del filtro, así como la ganancia se encuentra por encima de 0dB cuando la proporción $K_{spikesDivOut}$ y $K_{spikesDivFB}$ es superior a 1, e inferior a 0dB cuando dicha proporción es menor que 1. Además en la Tabla 7-8 se muestran los parámetros de la simulación, así como la ganancia y frecuencia de corte del filtro teóricas esperadas.

Tabla 7-8: Ganancia y frecuencia de corte del filtro de spikes paso de baja frente al valor divisor de spikes de la realimentación

Número de simulación	Número de bits	Divisor de frecuencia	Divisor de spikes de salida	Divisor de spikes de realimentación	Divisor de spikes	Ganancia del filtro (dB)	Frecuencia de corte (Hz)
1	9	0	0.8	0.5	0	4.82	1.22*103
2	9	0	0.7	0.6	0.2	1.33	1.46*103
3	9	0	0.6	0.7	0.4	-1.33	1.70*103
4	9	0	0.5	0.8	0.6	-4.82	1.95*103

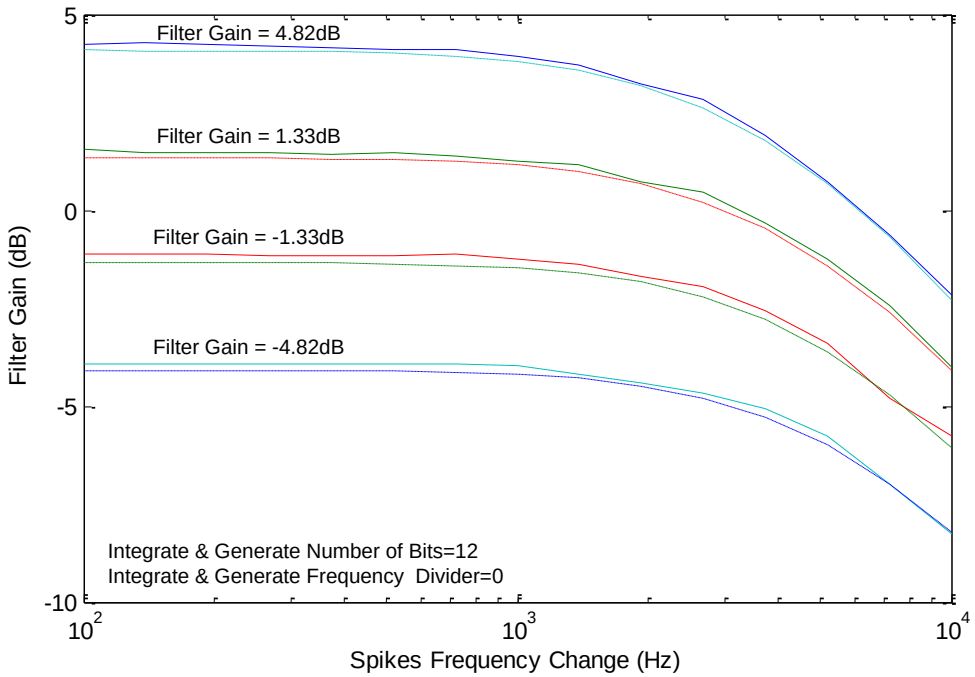


Figura 7.17: Diagrama de Bode del filtro paso de baja con ganancia ajustable

7.2. Diseño de filtros de paso de alta basados en spikes

El componente presentado con anterioridad en el Capítulo 4.3 como el derivador de spikes, no es en realidad más que un filtro paso de alta, en este apartado vamos a proponer otra arquitectura alternativa para el diseño del filtro de paso de alta de spikes. Para este filtro vamos a utilizar un filtro de spikes de paso de baja con ganancia ajustable presentado con anterioridad, la idea es restar de la señal de entrada, la cual contiene todas las componentes espectrales, la señal tras filtrarla con un filtro paso de baja, tal como se muestra en la Figura 7.18. Esta arquitectura para el filtro paso de alta ha sido elegida ya que gracias al filtro paso de baja con ganancia ajustable muestra una gran versatilidad, a diferencia de la usada para el derivador del spikes.

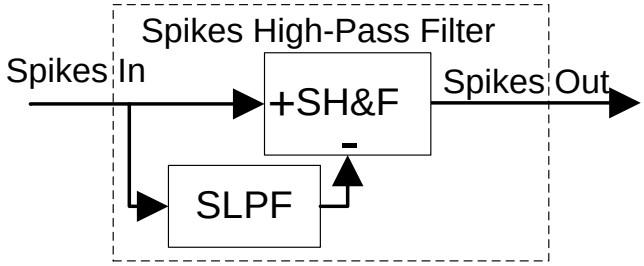


Figura 7.18: Diagrama de bloques del filtro paso de alta de spikes basado en el filtro paso de baja

La función de transferencia equivalente en el domino S de este filtro con dicha arquitectura puede calcularse como:

$$F_{SHPF}(s) = 1 - F_{SLPF}(s) = 1 - \frac{K_{spikesDivOut} * K_{BWSpikesGen}}{s + K_{spikesDivFB} * K_{BWSpikesGen}} =$$
$$= \frac{s + K_{spikesDivFB} * K_{BWSpikesGen} - K_{spikesDivOut} * K_{BWSpikesGen}}{s + K_{spikesDivFB} * K_{BWSpikesGen}} =$$
$$= \frac{s + (K_{spikesDivFB} - K_{spikesDivOut}) * K_{BWSpikesGen}}{s + K_{spikesDivFB} * K_{BWSpikesGen}}$$

Ecuación [7-14]

De tal forma que dependiendo de la relación entre la ganancia de realimentación y la ganancia de salida del filtro de spikes, podemos encontrarnos diversos sistemas equivalentes [Williams81]:

a) Si $K_{spikesDivFB} > K_{spikesDivOut}$, nos encontramos con un polo y un cero reales negativos, los cuales pueden ser descompuestos de la siguiente manera:

$$F_{SHPF}(s) = \frac{s + \beta}{s + \alpha} = \frac{s}{s + \alpha} + \frac{\beta}{s + \alpha}$$

Donde $\alpha = K_{spikesDivFB} * K_{BWSpikesGen}$

, y $\beta = (K_{spikesDivFB} - K_{spikesDivOut}) * K_{BWSpikesGen}$

b) Si $K_{spikesDivFB} = K_{spikesDivOut}$, nos encontramos filtro paso de alta, con un cero en el origen y un polo real negativo:

$$F_{SHPF}(s) = \frac{s}{s + \alpha}$$

Donde $\alpha = K_{spikesDivFB} * K_{BWSpikesGen}$

c) Si $K_{spikesDivFB} < K_{spikesDivOut}$, nos encontramos con dos polos reales negativos, y podríamos descomponer la función de transferencia en dos:

$$F_{SHPF}(s) = \frac{s - \beta}{s + \alpha} = \frac{s}{s + \alpha} + \frac{-\beta}{s + \alpha}$$

Donde $\alpha = K_{spikesDivFB} * K_{BWSpikesGen}$

y $\beta = (K_{spikesDivFB} - K_{spikesDivOut}) * K_{BWSpikesGen}$

En este caso nos encontramos con la superposición de un filtro. Por un lado tendremos las componentes de alta frecuencia de la señal, y por otro las de baja frecuencia pero con el signo invertido. Siendo la salida del filtro en este caso la suma de ambas respuestas.

A continuación hemos procedido a simular el comportamiento del filtro de paso de alta de spikes para diversas relaciones de los divisores de spikes. Los parámetros usados en las simulaciones pueden encontrarse en la Tabla 7-9, así como los resultados en la Figura 7.19. En la figura vemos cómo varía el comportamiento de la salida del filtro en base a la relación entre las ganancias de realimentación y de salida del filtro paso de baja. En las tres primeras simulaciones nos encontramos en el caso c), cuando $K_{spikesDivFB} < K_{spikesDivOut}$, el filtro se comporta como si en realidad fueran dos filtros. Por un lado vemos con signo positivo el "impulso" filtrado por la componente de alta frecuencia, además restada, está la señal original tras ser filtrada por la componente que actúa como un filtro paso de baja. A continuación, en la cuarta simulación $K_{spikesDivFB}$ es igual $K_{spikesDivOut}$, encontrándonos en el caso b), el filtro paso de alta puro. En esta ocasión la respuesta del filtro es análoga a la de un filtro paso de alta, y además no muestra ninguna componente de baja frecuencia, logrando una ganancia nula en la banda continua, 0Hz. Finalmente, en las tres últimas simulaciones $K_{spikesDivFB} > K_{spikesDivOut}$, encontrándonos en el caso a), en el que la salida del filtro es la adición entre un filtro paso de baja y un filtro paso de alta. De nuevo en la figura puede apreciarse cómo la salida del filtro es equivalente a dichas respuestas superpuestas, formada por un "impulso" aportado por la componente perteneciente al filtro paso de alta, y una señal con crecimiento exponencial aportada por el filtro paso de baja.

Tabla 7-9: Ganancia del filtro de spikes paso de alta frente a los valores de los divisores de spikes

Número de simulación	Número de bits	Divisor de frecuencia	Divisor de spikes de salida	Divisor de spikes de realimentación	Ganancia 0Hz
1	10	0	1.0	0.4	-1.5
2	10	0	0.9	0.5	-0.8
3	10	0	0.8	0.6	-0.33
4	10	0	0.7	0.7	0
5	10	0	0.6	0.8	0.25
6	10	0	0.5	0.9	0.4
7	10	0	0.4	1.0	0.6

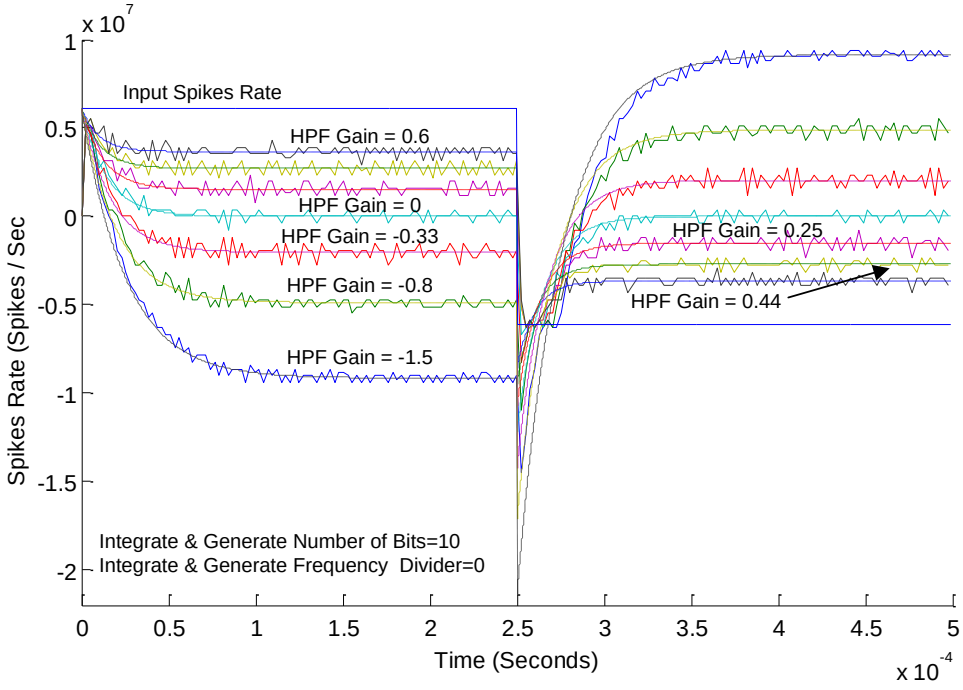


Figura 7.19: Respuesta a escalón del filtro de spikes paso de alta con distintas relaciones entre los divisores de spikes

Tras simular las respuestas temporales del filtro en cada caso, vamos a disponernos a comprobar su comportamiento frecuencial. Con este objetivo hemos realizado un barrido de frecuencia a su entrada, calculando su diagrama de Bode. En la Tabla 7-10 mostramos los parámetros utilizados en las simulaciones, y el diagrama de Bode del filtro paso de alta basado en spikes en la Figura 7.20. En la figura se ven de nuevo claramente los tres casos anteriores, en los tres primeros casos la ganancia del divisor de spikes de la realimentación es menor que la ganancia del divisor de spikes de la salida del filtro paso de baja, en consecuencia, este filtro tendrá una ganancia mayor que uno, encontrándonos en el caso a), mostrando en el diagrama de Bode la componente aportada por el polo, atenuada a continuación por el cero. En el siguiente caso ambas ganancias son iguales, y en consecuencia el filtro paso de baja tiene ganancia 1, en este caso el filtro se comporta como un filtro paso de alta ideal, b), atenuando hasta el infinito la componente continua de señal, y dejando pasar las altas frecuencias con una ganancia de 0dB. Finalmente en los tres últimos casos, la ganancia de la realimentación es mayor que la ganancia de salida de filtro paso de baja, teniendo una ganancia menor de 0dB, encontrándonos en la situación c), en la que el filtro vuelve a componerse de un polo y un cero, pero en este caso, con una fase de 180º debido al signo negativo de la componente de baja frecuencia del filtro.

Tabla 7-10: Ganancia y frecuencia de corte del filtro de spikes paso de alta frente al valor divisor de spikes de la realimentación

Número de simulación	Número de bits	Divisor de frecuencia	Divisor de spikes de salida	Divisor de spikes de realimentación	Ganancia del filtro en la banda de continua (dB)	Frecuencia de corte (Hz)
1	14	0	1.0	0.4	3.5218	1.22*103
2	14	0	0.9	0.5	-1.9382	1.52*103
3	14	0	0.8	0.6	-9.6297	1.83*103
4	14	0	0.7	0.7	$-\infty$	2.13*103
5	14	0	0.6	0.8	-12.0412	2.44*103
6	14	0	0.5	0.9	-7.1309	2.74*103
7	14	0	0.4	1.0	-4.4370	3.05*103

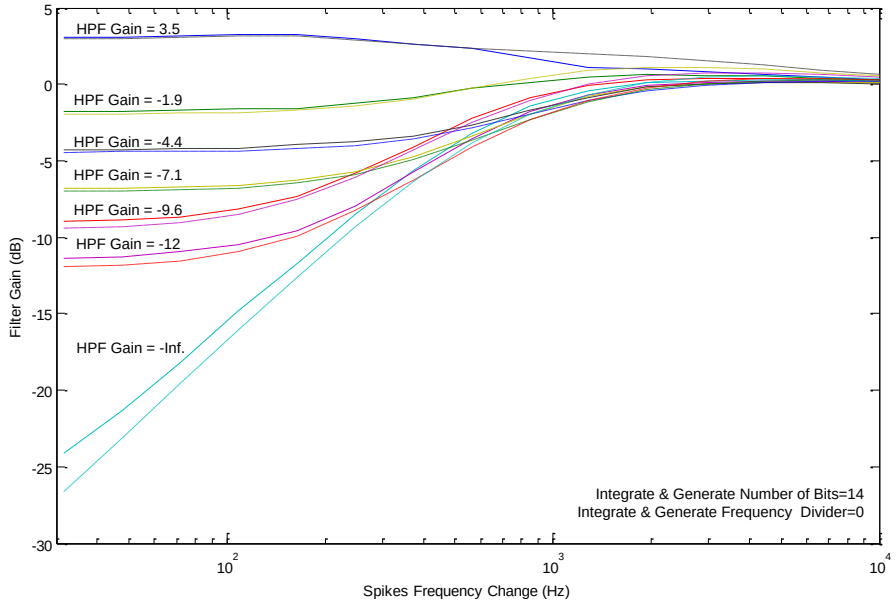


Figura 7.20: Diagrama de Bode del filtro de spikes paso de alta con distintas relaciones entre los divisores de spikes

7.3. Diseño de filtros de paso de banda basados en spikes con bajo factor Q

En este apartado vamos a diseñar un filtro paso de banda basado en spikes con un factor de calidad (Q) bajo, inferior a 0.5. Este filtro resulta muy interesante, ya que será el corazón de la cóclea sintética basada en la representación AER que presentaremos en el próximo capítulo. Los filtros paso de banda son aquellos que sólo permiten el paso de las componentes frecuenciales de una señal comprendidas entre dos bandas de frecuencia. Trasladado a los spikes, sólo permitiría el paso de los spikes cuyas componentes frecuenciales de los cambios de su frecuencia estén en la banda de paso del filtro. Hemos diseñado un filtro de spikes paso de banda restando la salida de dos filtros de spikes paso de baja con ganancia ajustable con distintas frecuencias de corte, usando un Hold & Fire, tal y como mostramos en

la Figura 7.21. De manera que el filtro con la frecuencia de corte más alta será la entrada positiva del Hold & Fire, conteniendo todas las componentes espectrales de la señal de entrada a excepción de las atenuadas. La entrada negativa del Hold &Fire será el filtro con la frecuencia de corte más baja, de manera que este filtro dejará pasar las frecuencias bajas, las que queremos atenuar con el filtro paso de banda, que serán restadas del filtro con mayor ancho de banda, obteniendo así sólo la banda situada entre ambas frecuencias de corte.

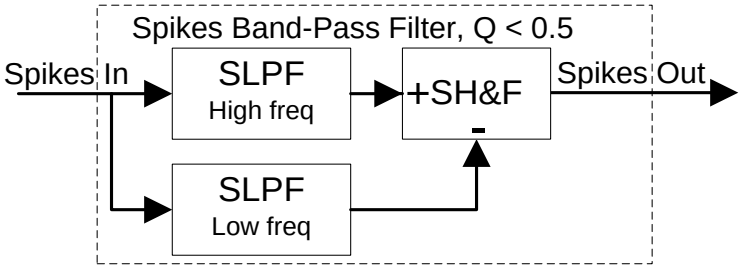


Figura 7.21: Diagrama de bloques del filtro paso de banda de spikes

Si desarrollamos la función de transferencia del filtro paso de banda usando la Ecuación [7-10], obtenemos:

$$F_{SBPF}(s) = F_{SLPF_{HF}}(s) - F_{SLPF_{LF}}(s) = \frac{K_{spikesDivHF} * K_{BWSpikesGenHF}}{s + K_{spikesDivFBHF} * K_{BWSpikesGenHF}} - \frac{K_{spikesDivLF} * K_{BWSpikesGenLF}}{s + K_{spikesDivFBLF} * K_{BWSpikesGenLF}}$$

Ecuación [7-15]

Agrupando los parámetros de los filtros, podemos decir que:

$$\alpha = K_{spikesDivOutHF} * K_{BWSpikesGenHF}$$
$$\beta = K_{spikesDivFBHF} * K_{BWSpikesGenHF}$$
$$\gamma = K_{spikesDivOutLF} * K_{BWSpikesGenLF}$$
$$\delta = K_{spikesDivFBLF} * K_{BWSpikesGenLF}$$

Ecuación [7-16]

Siendo β y δ las frecuencias de corte del filtro de alta y baja frecuencia respectivamente. De tal manera que el filtro de banda equivalente resultante tiene la siguiente forma:

$$F_{SBPF}(s) = \frac{\alpha}{s + \beta} - \frac{\gamma}{s + \delta} = \frac{\alpha(s + \delta) - \gamma(s + \beta)}{(s + \beta)(s + \delta)} = \frac{(\alpha - \gamma)s + \alpha\delta - \gamma\beta}{s^2 + (\beta + \delta)s + \beta\delta}$$

Ecuación [7-17]

Teniendo en cuenta que la función de transferencia de un filtro paso de banda ideal de segundo orden presenta la siguiente estructura [Pallás99]:

$$F_{BPF}(s) = k_{BPF} \frac{\frac{\omega}{Q}s}{s^2 + \frac{\omega}{Q}s + \omega^2}$$

Ecuación [7-18]

Donde ω representa la banda de paso del filtro y Q el factor de calidad. Nuestro filtro será equivalente al filtro paso de banda ideal en el caso de que cumpla las siguientes condiciones:

$$\alpha\delta = \gamma\,\beta \text{ y además } \alpha > \gamma$$

Sustituyendo estos valores por los parámetros de los filtros paso de baja basados en spikes componentes del filtro paso de banda según la Ecuación [7-16] obtenemos que:

$$\alpha\delta = \gamma\,\beta => K_{spikesDivOutHF} * K_{BWSpikesGenHF} * K_{spikesDivFBLF} * K_{BWSpikesGenLF}$$

$$=> K_{spikesDivFBHF} * K_{BWSpikesGenHF} * K_{spikesDivOutLF} * K_{BWSpikesGenLF}$$

$$=> K_{spikesDivOutHF} * K_{spikesDivFBLF} = K_{spikesDivFBHF} * K_{spikesDivOutLF}$$

$$=> \frac{K_{spikesDivOutHF}}{K_{spikesDivFBHF}} = \frac{K_{spikesDivOutLF}}{K_{spikesDivFBLF}}$$

Apoyándonos en la Ecuación [7-12], en la que calculábamos la ganancia del filtro de spikes paso de baja en base a sus divisores de frecuencia, la condición antes expuesta queda reducida a que la ganancia de ambos filtros sea la misma:

$$\frac{K_{spikesDivOutHF}}{K_{spikesDivFBHF}} = \frac{K_{spikesDivOutLF}}{K_{spikesDivFBLF}} => K_{SLFP\_HF} = K_{SLFP\_LF}$$

Ecuación [7-19]

Tras satisfacer estas condiciones, obtenemos un filtro de spikes paso de banda equivalente al filtro paso de banda ideal, en el que la frecuencia central está determinada por:

$$\omega = \sqrt{\beta\delta}$$

Ecuación [7-20]

El factor de calidad del filtro, Q, puede ser calculado de manera que:

$$\beta + \delta = \frac{\omega}{Q} => Q = \frac{\sqrt{\beta\delta}}{\beta + \delta}$$

Ecuación [7-21]

Así como la ganancia del filtro, K_{SBPF}:

$$\alpha - \gamma = k_{BPF}\frac{\omega}{Q} = k_{BPF}(\beta + \delta) => k_{BPF} = \frac{\alpha - \gamma}{\beta + \delta}$$

Ecuación [7-22]

Como nos muestra la Ecuación [7-19], una de las condiciones iniciales impuestas consiste en que ambos filtros paso de baja tengan la misma ganancia. Además en el caso particular en que la ganancia de ambos filtros sea 1 ó 0dB, los divisores de spikes de salida y realimentación de cada filtro deberán ser iguales, haciendo que $\beta = \alpha$ y $\delta = \gamma$. Teniendo en cuenta que β y δ son las frecuencias de corte de ambos filtros (ω_{HF} y ω_{LF} respectivamente), podemos expresar la Ecuación [2-22] en función de las frecuencias de corte de cada filtro en este caso particular como:

$$k_{BPF} = \frac{\alpha - \gamma}{\beta + \delta} = \frac{\omega_{HF} - \omega_{LF}}{\omega_{HF} + \omega_{LF}}$$

Ecuación [7-23]

Constatando así que la ganancia del filtro paso de banda, cuando los filtros paso de baja tienen ganancia 1, depende exclusivamente la distancia entre las frecuencias de corte de dichos filtros.

Expuestas las ecuaciones que gobiernan el comportamiento del filtro paso de banda basado en spikes, a continuación vamos a exponer un ejemplo práctico de cómo ajustar los parámetros del filtro para una frecuencia central, ω, y factor de calidad, Q, deseados. En primer lugar vamos a usar la Ecuación [7-21] para obtener el valor de la frecuencia de corte del filtro de alta frecuencia, β, en base a la diferencia entre el cociente de la frecuencia central del filtro y el factor de calidad, y la frecuencia de corte del filtro de baja frecuencia, δ. De tal manera que el valor ideal de β será:

$$\beta + \delta = \frac{\omega}{Q} \Rightarrow \beta = \frac{\omega}{Q} - \delta$$

Ecuación [7-24]

A continuación, combinando la Ecuación [7-24], recién obtenida, con la Ecuación [7-20], podemos despejar el valor de δ en base la frecuencia de central del filtro y el factor de calidad.

$$\omega = \sqrt{\left(\frac{\omega}{Q} - \delta\right) * \delta} \Rightarrow \delta^2 - \frac{\omega}{Q} * \delta + \omega^2 = 0$$

Ecuación [7-25]

Despejando δ de la ecuación anterior, tal y como se tratase de un polinomio de segundo orden, obtenemos la expresión de su valor. Sin embargo, podríamos obtener dos soluciones para δ, una por encima de la frecuencia de central, y otra por debajo. Para que se cumpla la condición inicial sólo vamos a usar el valor de δ por debajo de la frecuencia central.

$$\delta = \frac{\frac{\omega}{Q} - \sqrt[2]{\left(\frac{\omega}{Q}\right)^2 - 4 * \omega^2}}{2}$$

Ecuación [7-26]

Nótese que para obtener soluciones reales de δ, el valor del interior de la raíz cuadrada han de ser mayor que 0, de manera que el máximo factor de calidad alcanzable con estos filtros es inferior a 0.5, tal y como muestra la siguiente ecuación:

$$\left(\frac{\omega}{Q}\right)^2 - 4 * \omega^2 > 0 \Rightarrow \left(\frac{\omega}{Q}\right)^2 > 4 * \omega^2 \Rightarrow \frac{1}{Q^2} > 4 \Rightarrow Q < 0.5$$

Ecuación [7-27]

Dado que los dos filtros paso de baja de spikes usados para obtener este filtro están compuestos por dos polos reales, el filtro paso de banda equivalente conservará ambos polos

reales. Debido a este hecho, el factor Q no puede ser superior a 0.5, o visto desde el punto de vista del control [Ogata04] será un sistema de segundo orden sobreamortiguado, con un coeficiente de amortiguación mayor que 1, y sin frecuencia de resonancia. Para diseñar filtros paso de banda con mayores coeficientes de calidad deben de ser desarrollados nuevas topologías de filtros de spikes con polos con componentes imaginarias, como expondremos en siguiente apartado.

Una vez conocida δ, haciendo uso de la Ecuación [7-24] podemos calcular fácilmente el valor de β, simplemente sustituyendo el valor de δ en la ecuación.

Determinadas las frecuencias de cortes de ambos filtros, β y δ, hemos de ajustar los parámetros de ambos filtros paso de baja para que sus frecuencias de corte coincidan con β y δ respectivamente. Acorde con la Ecuación [7-11] cada filtro posee 4 parámetros a ajustar: el número de bits y el divisor de frecuencia del Integrate & Generate, y los divisores de spikes de la realimentación y de la salida del filtro. Así que vamos a ir ajustando uno a uno los parámetros del filtro hasta diseñar los filtros adecuados que conformarán el filtro paso de banda basado en spikes.

En primer lugar vamos a ajustar las frecuencias de corte de cada filtro de manera individual, repitiendo el mismo proceso que vamos a exponer a continuación para cada filtro. Según la Ecuación [7-11] la frecuencia de corte de un filtro paso de baja basado en spikes depende de la ganancia del Integrate & Generate y del valor del divisor de spikes de la realimentación del filtro. Vamos a comenzar ajustando la ganancia del Integrate & Generate, ya que su rango no es muy limitado, para más adelante proporcionar el ajuste preciso con el divisor de spikes. Para ello vamos suponer que el divisor de spikes vale 1, y además hemos de tener en cuenta que la ganancia del Integrate & Generate depende de dos parámetros, el número de bits y el divisor de frecuencia, tal y como exponíamos en la Ecuación [4-9]. Para seleccionar ambos parámetros hemos implementado un pequeño algoritmo de búsqueda expuesto a continuación. Este algoritmo comienza buscando el número de bits adecuado de Integrate & Generate, así que supone que el divisor de frecuencia vale 0, y va aumentando el número de bits hasta que la ganancia del Integrate & Generate es menor que la frecuencia de corte deseada, siendo el primer valor que cumple esta condición el mínimo número de bits a utilizar. A continuación despejamos de la Ecuación [4-9] el divisor de frecuencia para obtener la ganancia del Integrate & Generate deseada.

```
function [nBits,freqDiv]=freq2IG(Fclk,freq)

freqDiv=0;
nBits=3;
while IG_gain(Fclk,nBits,freqDiv)>freq
    nBits=nBits+1;
end
freqDiv=(Fclk/(freq*2^(nBits-1)))-1;
```

Según este par de valores, sin obtener aún la ganancia del divisor de spikes de la realimentación, la frecuencia de corte del filtro está perfectamente ajustada a la frecuencia

proporcionada. Sin embargo, esta situación sólo se da en condiciones ideales. En la realidad sucede que el divisor de frecuencia es un número real, no entero, con parte decimal, y que además nuestro Integrate & Generate sólo acepta divisores de frecuencia como números enteros, debido a este hecho la frecuencia de corte conseguida en la realidad sólo sería aproximada a la deseada. Para paliar esta problemática vamos a hacer uso del divisor de spikes, ya que representa a un número entre [0-1] con 16 bits, teniendo 2^{16} posibles valores, aumentando en gran medida la precisión de ajuste del filtro paso de baja. Lo que vamos a hacer es igualar la ganancia del Integrate & Generate ideal, con el producto del Integrate & Generate real por la ganancia de realimentación. Siendo única la diferencia entre ambas ganancias del Integrate & Generate el valor del divisor de frecuencia, el valor real con decimales para la ganancia ideal, y la parte entera del divisor de frecuencia para la ganancia real. Despejando la ganancia del divisor de spikes de la realimentación obtenemos:

$$K_{BWInt\&GenIdeal} = K_{BWInt\&GenReal} * K_{spikesDivFB}$$

$$\frac{F_{CLK}}{2^{n-1}(FreqDiv_{ideal}+1)} = \frac{F_{CLK}}{2^{n-1}(\lfloor FreqDiv_{ideal} \rfloor + 1)} * K_{spikesDivFB}$$

$$K_{spikesDivFB} = \frac{(\lfloor FreqDiv_{ideal} \rfloor + 1)}{(FreqDiv_{ideal}+1)}$$

Ecuación [7-28]

Finalmente hemos conseguido ajustar de manera precisa la frecuencia de corte de cada filtro, sólo resta ajustar cada una de sus ganancias, ya que al asignar un valor distinto de uno a la ganancia del divisor de spikes de la realimentación, estamos proporcionándole a cada filtro una ganancia distinta, y esta situación incumple una de las condiciones establecidas con anterioridad para los filtros paso de banda basados en spikes. En definitiva, hemos de calcular la ganancia del divisor de spikes de salida de cada filtro para que ambos tengan el mismo valor. El caso más simple que nos encontramos es cuando asignamos una ganancia de 1 o 0dB a cada filtro. Según la Ecuación [7-12] para que un filtro tenga ganancia 1, la ganancia del divisor de spikes de la salida del filtro debe valer lo mismo que la ganancia del divisor de spikes de la realimentación del filtro:

$$K_{SLPF} = \frac{K_{spikesDiv}}{K_{spikesDivFB}} = 1 => K_{spikesDiv} = K_{spikesDivFB}$$

Ecuación [7-29]

Para comprobar el correcto funcionamiento del filtro paso de banda basado en spikes, y además poner a prueba el mecanismo de ajuste del filtro que acabamos de exponer, vamos a diseñar filtros para diversas frecuencias de centrales y factores de calidad, simulando cada juego de parámetros, y calculando el diagrama de Bode de la respuesta del filtro. En primer lugar vamos a comenzar diseñando filtros paso de banda con un factor de calidad fijo, 0.4, y diversas frecuenciales centrales, desde 1kHz hasta 5kHz. Hemos comenzado calculando los parámetros para cada caso, mostrados en la Tabla 7-11, y a continuación los hemos simulado calculando su diagrama de Bode, cuyos resultados pueden encontrarse en la Figura 2.1. En la

figura mostramos la respuesta frecuencial de cada filtro, en trazo continuo, y la respuesta de filtros equivalentes ideales, en trazo discontinuo. Se puede observar claramente que el filtro se comporta como un filtro paso de banda, atenuando las componentes frecuenciales lejanas a la frecuencia central del filtro, y sólo dejando pasar las componentes en las cercanas a dicha frecuencia central.

Tabla 7-11: Parámetros de los filtros paso de banda basados en spikes para distintas frecuencias centrales de paso

Filtro de alta frecuencia					Filtro de baja frecuencia					Filtro Paso de banda	
N bits	Div. Freq.	Div. salida	Div. realimentación	Freq. Corte	N bits	Div. Freq.	Div. salida	Div. realimentación	Freq. Corte	Freq. Corte	Q
13	0	0.9714	0.9714	1.8873kHz	15	0	0.9714	0.9714	471.8135	1kHz	0.4
12	0	0.9714	0.9714	2.5163kHz	14	0	0.9714	0.9714	629.0847	2kHz	0.4
11	0	0.6476	0.6476	3.7745kHz	14	0	0.6476	0.6476	943.6271	3kHz	0.4
11	0	0.7771	0.7771	6.0392kHz	13	0	0.7771	0.7771	1.5098kHz	4kHz	0.4
11	0	0.9714	0.9714	7.5490kHz	13	0	0.9714	0.9714	1.8873kHz	5kHz	0.4

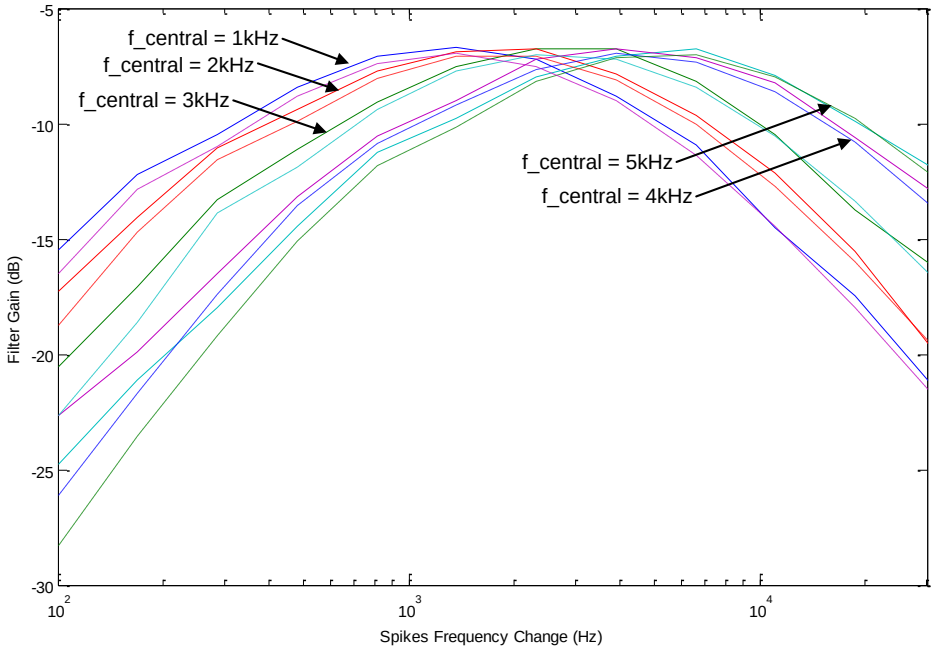


Figura 7.22: Diagrama de Bode del filtro paso de banda basado en spikes para diversas frecuencias centrales

Como siguiente experimento hemos fijado la frecuencia central del filtro a 2kHz, y hemos ido cambiando el factor de calidad Q, desde 0.25 hasta 0.4. Obteniendo así diversos filtros con la misma frecuencia central, pero con distintos anchos de banda. En la Tabla 7-12 pueden encontrarse los parámetros usado para cada filtro, así como los resultados de las simulaciones en la Figura 7.23. En la figura una vez más mostramos la respuesta frecuencial del filtro simulado, en trazo continuo, frente a la respuesta de un filtro ideal, en trazo discontinuo. Nuevamente el filtro se comporta como un filtro paso de banda, sin embargo, en esta ocasión

todos los filtros están centrados a la misma frecuencia, pero el ancho de banda, o la calidad del filtro, se ve modificada de acuerdo con el factor Q de calidad, de manera que cuanto menor es dicho valor, más frecuencias podrán atravesar nuestros filtros. Sin embargo, al aumentar el factor Q la banda de paso del filtro es más estrecha, atenuando más eficientemente las componentes frecuenciales fuera de la banda de paso. Además se ve cómo la calidad del filtro afecta a la frecuencia de manera inversamente proporcional, es decir, al aumentar el factor Q la ganancia del filtro disminuye, ya que las frecuencias de corte de los filtros paso de baja se aproximan cada vez más, rechazando mejor las componentes frecuenciales fuera de la banda, pero también atenuándose mutuamente en la banda de paso, tal y como predecía la Ecuación [7-23].

Tabla 7-12: Parámetros de los filtros paso de banda basados en spikes para distintos factores de calidad

Filtro de alta frecuencia					Filtro de baja frecuencia					Filtro Paso de banda	
N	**Div. Freq.**	**Div. salida**	**Div. realimentación**	**Freq. Corte**	**N**	**Div. Freq.**	**Div. salida**	**Div. realimentación**	**Freq. Corte**	**Freq. Corte**	**Q**
11	0	0.9	0.9	7kHz	16	0	0.5238	0.5238	127.202	2kHz	0.25
12	0	0.5697	0.5697	2.213kHz	15	0	0.8282	0.8282	402.2789	2kHz	0.3
12	0	0.7031	0.7031	2.732kHz	15	0	0.6710	0.6710	325.9153	2kHz	0.35
12	0	0.8595	0.8595	3.34kHz	15	0	0.5489	0.5489	266.6076	2kHz	0.4

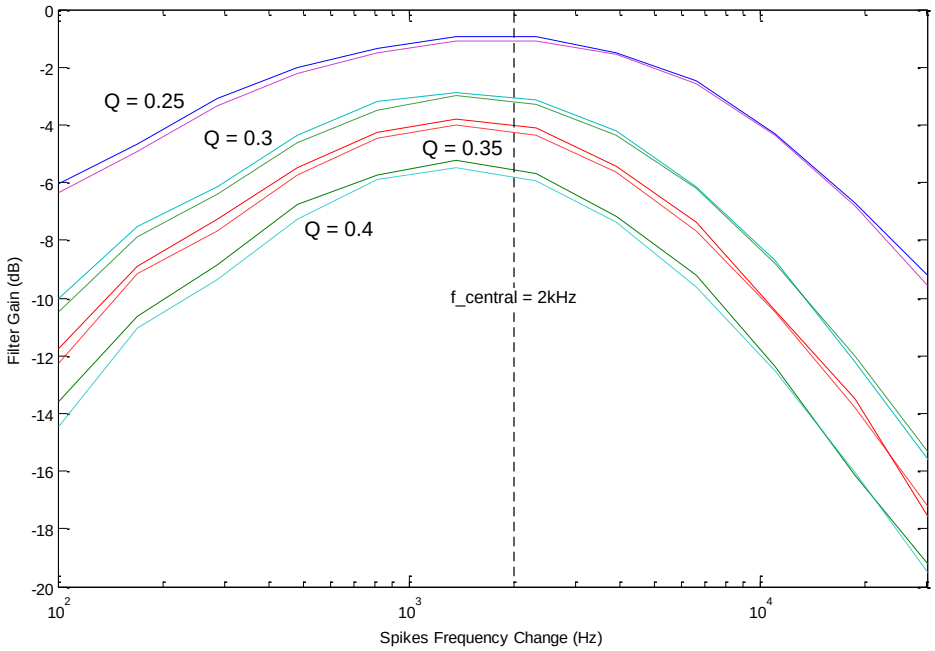


Figura 7.23: Diagrama de Bode del filtro paso de banda basado en spikes para diversos factores de calidad

Para concluir el análisis de este filtro, vamos a proceder a analizar su respuesta temporal. Con este fin hemos utilizado un filtro paso de banda centrado en 3kHz y con un factor de calidad de 0.4, presentado en la Tabla 7-11. El generador sintético de spikes situado a

la entrada del filtro ha sido excitado con una señal senoidal a 3kHz con interferencias, estando dichas interferencias compuestas por un tono a 40kHz y otro a 300Hz. La Figura 7.24 nos muestra los resultados de las simulaciones, en ella encontramos la señal de excitación de los spikes del filtro, en azul, así como la reconstrucción de los spikes de salida del filtro, en negro, junto con la respuesta teórica del filtro, en rojo y con trazo discontinuo. La frecuencia de los spikes de la salida del filtro se aproxima en gran medida a los esperados del filtro ideal, además las interferencias han sido rechazadas en gran medida, predominando las componentes frecuenciales localizadas en la banda de paso del filtro.

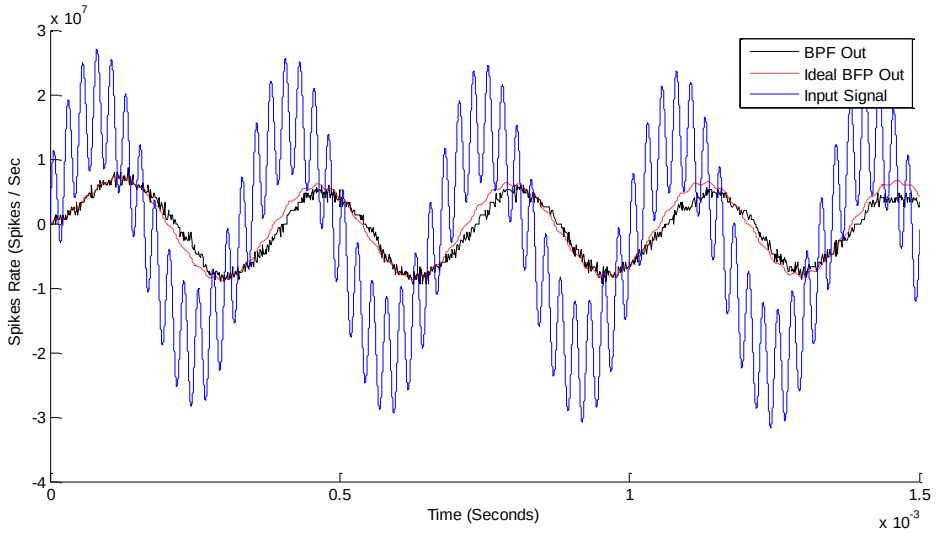


Figura 7.24: Respuesta temporal del filtro paso de banda basado en spikes para una señal de entrada con interferencias

En la figura anterior también puede apreciarse la desviación en fase introducida por el filtro paso de banda. Sin embargo, a lo largo de este trabajo no hemos analizado el efecto de los filtros de spikes sobre las fases de las señales. Este análisis ha sido imposible debido a limitaciones encontradas con el simulador, ya que para analizar la fase de las señales de baja frecuencia nos hemos visto obligados a realizar simulaciones muy largas, de más de un ciclo de la señal original. Necesitando de esta manera una cantidad de tiempo de simulación y de memoria elevadísima, topándonos con los límites de uso de memoria de MATLAB. Sin embargo, dada la concordancia de las respuestas temporales de los filtros de spikes con los teóricos, podemos suponer que la desviación en fase de los filtros basados en spikes es acorde a la esperada de los filtros analógicos ideales.

7.4. Diseño de filtros de paso de banda basados en spikes con elevado factor Q

Finalmente, vamos a diseñar un filtro de spikes paso de banda de segundo orden con un factor Q elevado, este filtro se comportará de la misma manera que el filtro paso de banda basado en spikes presentado en el apartado anterior, permitiendo el paso de las componentes frecuenciales próximas a la frecuencia central del filtro y atenuando las componentes

frecuenciales distantes a la frecuencia central. El filtro paso de banda basado en spikes presentado en el capítulo anterior tenía una limitación en el factor de calidad del filtro, no pudiendo conseguir un factor Q superior a 0.5, siendo este valor un factor de calidad muy bajo. Este hecho era debido a que el filtro de spikes paso de banda estaba compuesto por dos filtros paso de baja con polos reales. Sin embargo, para obtener factores de calidad mayores que 0.5 es necesario introducir polos complejos en la función de transferencia del filtro, lo cual es imposible mediante el uso de la técnica de restar la salida de dos filtros paso de baja basados en spikes. Inspirándonos en la forma en que los circuitos analógicos implementan los filtros paso de banda [Pallas99], en los que la propia salida del filtro paso de banda se suma de manera ponderada a la realimentación del filtro, hemos diseñado un nuevo filtro paso de banda basado en spikes con un alto factor de calidad. La Figura 7.25 muestra los componentes internos de este nuevo filtro, el cual se basa en realimentar la salida de un Integrate & Generate con dos elementos, otro Integrate & Generate, integrando la salida del filtro, y un divisor de spikes, el cual realimenta en paralelo el filtro con una proporción de la salida del propio filtro.

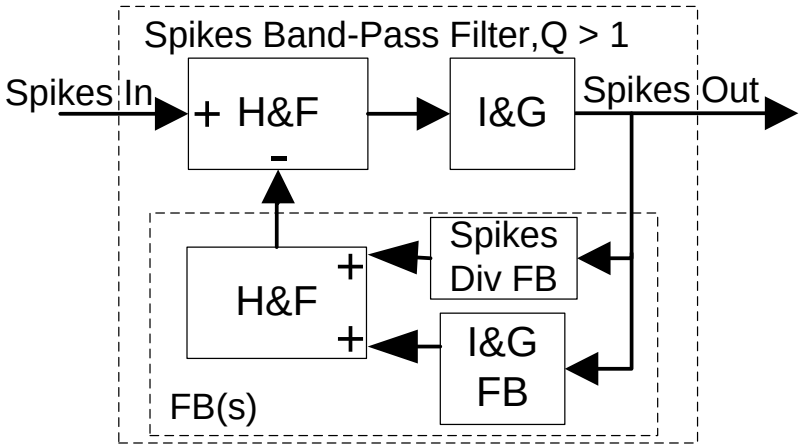


Figura 7.25: Diagrama de bloques del filtro de paso de banda de alta calidad basado en spikes

En primer lugar vamos a modelar la función de transferencia de los elementos situados en la realimentación del filtro, *FB(s)*, de manera que la función de transferencia equivalente conjunta de ambos elementos puede calcularse como:

$$FB(s) = K_{spikesDiv} + I\&G(s)_{FB} = K_{spikesDiv} + \frac{K_{I\&G\_FB}}{s} = \frac{K_{spikesDiv} * s + K_{I\&G\_FB}}{s}$$

Ecuación [7-30]

De forma que la función de transferencia equivalente del filtro propuesto puede obtenerse de la siguiente manera:

$$BPF(s)_{HQ} = \frac{I\&G(s)}{1+FB(s)*I\&G(s)} = \frac{\frac{K_{I\&G}}{s}}{1+\frac{K_{spikesDiv}*s+K_{I\&G_{FB}}}{s}*\frac{K_{I\&G}}{s}} = \frac{K_{I\&G} * s}{s^2 + K_{I\&G} * K_{spikesDiv} * s + K_{I\&G} * K_{I\&G\_FB}}$$

Ecuación [7-31]

Si comparamos la función de transferencia obtenida con la de un filtro paso de banda analógico ideal, presentado con anterioridad en la Ecuación [7-18], podemos realizar una

equivalencia paramétrica entre ambas funciones de transferencia, posibilitándonos de esta forma el cálculo de los parámetros del filtro de spikes paso de banda en base a las ganancias de los elementos incluidos en su interior. En consecuencia, la frecuencia central, factor de calidad y ganancia del filtro, pueden calcularse de la siguiente manera:

$$\omega = \sqrt{K_{I\&G} * K_{I\&G\_FB}}$$

Ecuación [7-32]

$$\frac{\omega}{Q} = K_{I\&G} * K_{spikesDiv} => Q = \frac{\sqrt{K_{I\&G} * K_{I\&G\_FB}}}{K_{I\&G} * K_{spikesDiv}}$$

Ecuación [7-33]

$$k_{BPF}\frac{\omega}{Q} = K_{I\&G} => \ k_{BPF} = \frac{K_{I\&G}}{K_{I\&G} * K_{spikesDiv}} = \frac{1}{K_{spikesDiv}}$$

Ecuación [7-34]

En el caso particular que $K_{I\&G} = K_{I\&G\_FB}$, podemos simplificar las ecuaciones anteriores como:

$$\omega = \sqrt{K_{I\&G} * K_{I\&G}} = \ K_{I\&G}$$

Ecuación [7-35]

$$Q = \frac{K_{I\&G}}{K_{I\&G} * K_{spikesDiv}} = \frac{1}{K_{spikesDiv}}$$

Ecuación [7-36]

Así la frecuencia central del filtro dependerá de la ganancia de ambos Integrate & Generate. Sin embargo, el factor de calidad Q será inversamente proporcional al valor del divisor de spikes, el cual tendrá un rango comprendido entre [0,1], estando el factor de calidad comprendido entre [∞,1]. En consecuencia, con el uso de este mecanismo podremos diseñar filtros paso de banda con un elevado factor de calidad, ampliando las prestaciones de los filtros presentados anteriormente. Esta cualidad se debe a que los polos de este filtro son polos complejos conjugados, con una componente real y otra imaginaria. Si calculamos el valor de los polos despejando el denominador de la Ecuación [7-31], suponiendo que ambos Integrate & Generate tienen la misma ganancia, obtenemos que:

$$s^2 + K_{I\&G} * K_{spikesDiv} * s + (K_{I\&G})^2 = 0$$

Ecuación [7-37]

Resolviendo esta ecuación como si de un polinomio de segundo orden se tratase, podemos calcular analíticamente los valores de los polos del filtro:

$$s = \frac{-K_{I\&G} * K_{spikesDiv} \pm \sqrt{\left(K_{I\&G} * K_{spikesDiv}\right)^2 - 4 * (K_{I\&G})^2}}{2}$$

Ecuación [7-38]

os polos del filtro tendrán una componente imaginaria siempre y cuando la raíz cuadrada de la Ecuación [7-38] no tenga una solución real:

$$\left(K_{I\&G} * K_{spikesDiv}\right)^2 - 4 * (K_{I\&G})^2 < 0 \quad => \quad \left(K_{I\&G} * K_{spikesDiv}\right)^2 < 4 * (K_{I\&G})^2$$
$$=> \quad K_{spikesDiv} < 2$$

Ecuación [7-39]

Según la ecuación anterior, los polos del filtro serán polos complejos conjugados siempre y cuando $K_{spikesDiv}$ < 2, y puesto que $K_{spikesDiv}$ es un número comprendido entre 0 y 1, tal y como mostraba la Ecuación [7-10], independientemente de su valor obtendremos siempre un par de polos complejos conjugados. Si hacemos una analogía entre este filtro y los sistemas de control tradicionales [Ogata04], el filtro paso de banda basado en spikes de alta calidad se comporta como un sistema subamortiguado de segundo orden, presentando una frecuencia de resonancia en la frecuencia central del filtro, así como un factor de calidad comprendido entre [∞,1]. Cabe destacar un aspecto importante de todos los sistemas que estamos diseñando en este trabajo, suponer $\mathrm{K_{I\&G}} = \mathrm{K_{I\&G\_FB}}$ en un sistema analógico clásico puede ser una aproximación más o menos real por la imposibilidad de tener esa igualdad de forma exacta debido a las fuentes de error en dichos sistemas analógicos. Sin embargo en nuestro caso los sistemas planteados están basados en contadores que podrán ajustarse de forma adecuada hasta conseguir la identidad deseada. Por otra parte el uso de estos contadores hace que no sean posibles todos los valores posibles debido a la resolución de los mismos y a que manejan números enteros. Sin embargo, se podrían añadir dos divisores de spikes para mejorar la precisión con la que pueden ser ajustadas ambas ganancias, pero no han sido añadidos en este filtro para no complicar el modelo a nivel paramétrico, permitiendo de esta manera un estudio más directo de sus cualidades.

El filtro de spikes paso de banda de alta calidad ha sido simulado con el fin de analizar su comportamiento. A continuación vamos a realizar una serie de simulaciones en las que incluiremos un filtro paso de banda con la frecuencia central en 3.8kHz, pero iremos modificando iterativamente el factor de calidad del filtro. La Tabla 7-13 muestra los parámetros usados en las simulaciones, ambos Integrate & Generate han sido configurados de la misma manera, con el mismo número de bits y divisor de frecuencia, sin embargo, el valor del divisor de spikes añadido en la realimentación va aumentando en cada caso de simulación. En consecuencia la frecuencia central del filtro será siempre la misma, ya que sólo depende de la ganancia de los Integrate & Generate, sin embargo, los valores del factor Q y de la ganancia del filtro son diferentes en cada caso, ya que ambos parámetros dependen exclusivamente de la ganancia del divisor de spikes en la realimentación.

Tabla 7-13: Parámetros de los filtros paso de banda basados en spikes de alta calidad con distintos factores Q

N bits	Div. Freq.	N bits Realim.	Div. Freq. Realim.	Spikes Div. Realim.	Ganancia (dB)	Freq. Central	Q
12	0	12	0	0.03	15	3885.6HZ	31.62
12	0	12	0	0.1	10	3885.6HZ	10
12	0	12	0	0.31	5	3885.6HZ	3.16
12	0	12	0	1	0	3885.6HZ	1

En primer lugar vamos a realizar una serie de simulaciones para analizar la respuesta frecuencial del filtro paso de banda de alta calidad basado en spikes. La Figura 7.26 muestra el diagrama de Bode del filtro paso de banda de alta calidad basado en spikes para los distintos casos presentados en la Tabla 7-13. Se puede observar que los filtros se encuentran centrados a una frecuencia de 3.8kHz, tal y como la Ecuación [7-35] predecía, además puede apreciarse como el factor de calidad del filtro va aumentando de manera inversamente proporcional con la ganancia del divisor de spikes de la realimentación acorde con la Ecuación [7-36], finalmente también puede apreciarse el efecto de la ganancia del divisor de spikes en la ganancia del filtro, siendo equivalente al factor Q del filtro como demostraba la Ecuación [7-34]. Sin embargo, la banda de paso no llega efectivamente a esas ganancias tan elevadas, se encuentran algo por debajo, este efecto lo achacamos a la saturación de los Integrate & Generate, ya que aunque el filtro puede conseguir ganancias infinitas de manera ideal, los sistemas reales tienen niveles de saturación, no pudiendo ir más allá.

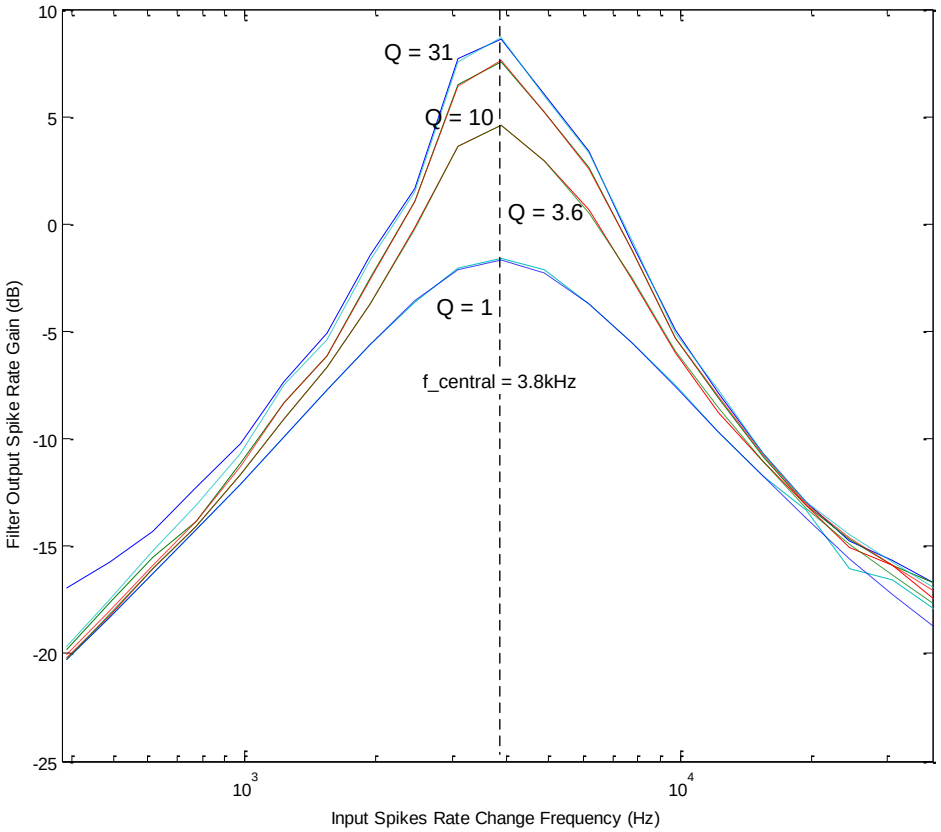


Figura 7.26: Diagrama de Bode del filtro paso de banda basado en spikes con alto factor de calidad

A continuación hemos realizado una serie de simulaciones temporales para poder observar el comportamiento transitorio del filtro a lo largo del tiempo. En la Figura 7.27 podemos encontrar la respuesta temporal del filtro paso de banda basado en spikes de alta calidad ajustado acorde con los parámetros expuestos con anterioridad en la Tabla 7-13. En este caso los hemos estimulado con una señal en escalón, proporcionándoles una frecuencia de spikes de entrada constante. La respuesta del filtro contiene una “sobreoscilación”, pero rechazando

la banda continua. Esta oscilación se debe a que un escalón ideal está compuesto por infinitas componentes frecuenciales [Oppenheim92], de tal manera que el filtro paso de banda solo deja pasar las componentes espectrales de los spikes localizados alrededor de su frecuencia central, y además amplificándolas debido a la resonancia, traduciéndose en la oscilación apreciada en la figura. La frecuencia de oscilación coincide con la frecuencia de resonancia o central del filtro, y además la amplitud de la oscilación aumenta acorde con el factor Q, debido a que la ganancia en la banda de paso es equivalente a dicho factor.

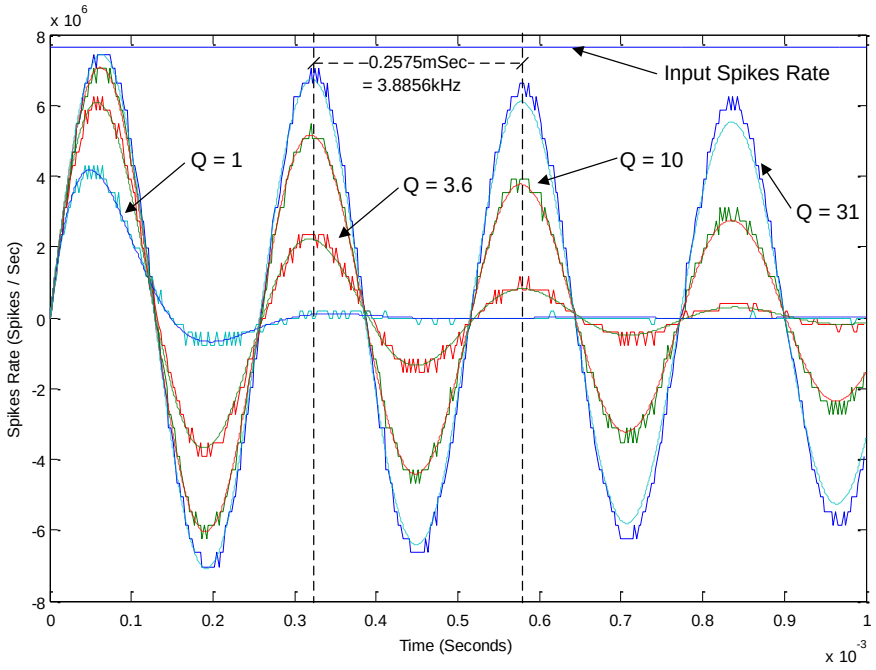


Figura 7.27: Respuesta ante escalón del filtro paso de banda basado en spikes con alto factor de calidad

7.5. Consumo hardware y rendimiento de los filtros basados en spikes

Para concluir con el estudio de los filtros de spikes, vamos a realizar un análisis de los recursos hardware necesarios para implementar estos filtros en una FPGA, así como el rendimiento computacional de los mismos, comparando el número de operaciones equivalentes llevadas a cabo por los filtros de spikes frente a los tradicionales filtros digitales. Con este propósito hemos sintetizado los diversos filtros expuestos, y tomado nota del número de slices necesarios para implementar cada filtro así como su frecuencia máxima. En la Tabla 7-14 podemos encontrar los resultados de estas síntesis, y además encontramos el número máximo de filtros que se pueden alojar en una FGPA de la familia Spartan3 400, así como la tasa de operaciones equivalentes de los filtros de spikes frente a los filtros discretos tradicionales.

Para sintetizar los filtros hemos fijado el número de bits a 12, como caso medio, en la que queda evidenciado cómo el consumo de slices de los filtros aumenta acorde con la complejidad del filtro, consumiendo el filtro paso de baja básico 65 slices y necesitando más del doble de esta cantidad para implementar un filtro paso de baja con ganancia ajustable

usando el divisor de spikes bit-wise. También resulta destacable la poca diferencia de consumo que hay entre un filtro paso de baja con un divisor de spikes basado en un LFSR y el basado en el método bit-wise, sólo 5 slices. Los filtros paso de alta y paso de banda muestran un consumo de slices coherente, es decir, el filtro paso de alta incluye un filtro paso de baja con ganancia ajustable y un Hold & Fire, consumiendo 14 slices más que el filtro paso de baja incorporando, dejando constancia que los 14 slices son consumidos por el Hold & Fire. El filtro que más consumo presenta es el de paso de banda con factor Q<0.5, ya que para esta síntesis ambos filtros tienen 12 bits, incorporando dos filtros paso de baja y un Hold & Fire. Sin embargo no necesita más del doble de slices como cabía presumir, sino que consume algunos menos, este efecto lo achacamos a la optimización que realiza la herramienta de síntesis, ya que al tener ambos filtros el mismo número de bits pueden compartir algunos elementos internos, como por ejemplo los contadores de ciclo de los generadores bit-wise del interior del los Integrate & Generate. Sin embargo, el filtro paso de banda basado en spikes de alta calidad tiene un consumo mucho más reducido que el de baja calidad, consumiendo sólo 12 slices más que filtro paso de alta, este hecho es debido a que necesitas muchos menos elementos que si de dos filtros se tratase, ahorrando el uso de 3 divisores de spikes. En cuanto a la frecuencia de operación, el filtro paso de baja básico es el más rápido, ya que es el que menos niveles de lógica necesita. Sin embargo, conforme vamos aumentando la complejidad del filtro, se van incorporando niveles lógicos, decrementando, en consecuencia, la velocidad de operación de los filtros. Resulta interesante la gran diferencia que hay entre los filtros con distintos divisores de spikes, a diferencia del consumo hardware que es muy parejo, ya que el divisor de spikes que usa el LFSR es más rápido, por incorporar un hardware más simple. Sin embargo el divisor basado en el método bit-wise es más lento, pero proporciona los mejores resultados.

De nuevo el cálculo de las operaciones realizadas por el filtro es muy complicado de hacer, casi imposible, ya que las operaciones que realizan estos filtros dependen de los spikes que se disparen, sin embargo, vamos a compararlos con filtros homólogos digitales, calculando así el número de operaciones equivalentes realizadas por los filtros, de la misma que se realiza en [Linares09] para medir el rendimiento de convolucionadores AER. Un filtro digital se puede representar de forma genérica según la Ecuación [7-40] [Oppemheim92], según esta ecuación, cada muestra de salida, y(n), está compuesta por dos sumatorios, en el primero se multiplica el histórico de las muestras de entrada, x, por una serie de constantes b, y en el segundo se realiza la misma operación con el histórico de las muestras de salida (y), y con sus propios coeficientes (a), siendo N y M los órdenes del filtro.

$$y(n) = \sum_{k=0}^{N} b_k * x(n-k) - \sum_{k=1}^{M} a_k * y(n-k)$$

Ecuación [7-40]

La ecuación anterior coincide con la de un filtro de respuesta infinita o IIR, sin embargo en el caso particular en el que sólo usamos las muestras de entrada, obtenemos un filtro de respuesta finita o FIR, siendo estos el modelo de filtro que vamos a utilizar para calcular las operaciones equivalentes de los filtros basados en spikes. Un filtro FIR responde a la Ecuación [7-41] [Mitra93], en la que N representa el orden del filtro.

$$y(n) = \sum_{k=0}^{N} b_k * x(n-k)$$

Ecuación [7-41]

En el caso particular de un filtro de orden 1, el filtro responderá a la Ecuación [7-42]. Para implementar un filtro FIR de primer orden acorde a dicha ecuación, necesitamos realizar dos multiplicaciones y una suma, si asignamos un peso clásico de 1-4-9 a las operaciones de suma, multiplicación y división respectivamente [Patterson09], necesitamos un total ideal de 9 operaciones para filtrar cada muestra. El número de operaciones necesarias es ideal ya que en él no vamos a tener en cuenta las operaciones necesarias en un procesador real de acceso a registros y memoria, cálculo de índices y rotación de muestras. Sin embargo vamos a usar este valor como referencia ya que nuestros filtros no realizan ninguna de estas operaciones, además vamos a considerar las 9 operaciones para todos los filtros paso de baja, ya que todos estos filtros se pueden codificar con el mismo algoritmo.

$$y(n) = b_0 * x(n) + b_1 * x(n-1)$$

Ecuación [7-42]

Para los filtros paso de alta vamos a considerar necesarias 10 operaciones, las 9 del filtro paso de baja más la resta necesaria, acorde a la Ecuación [7-43].

$$y(n) = 1 - (b_0 * x(n) + b_1 * x(n-1))$$

Ecuación [7-43]

Finalmente los filtros paso de banda incorporan dos filtros paso de baja, necesitando 18 operaciones, y además una operación más para restar ambos filtros, resultando un total de 19 operaciones.

$$y(n) = (b_{HF0} * x(n) + b_{HF1} * x(n-1)) - (b_{LF0} * x(n) + b_{LF1} * x(n-1))$$

Ecuación [7-44]

En la última columna de la Tabla 7-14 mostramos la tasa de operaciones equivalentes de cada filtro individualmente, así como las operaciones equivalentes alcanzables por una FPGA llena de filtros de mismo tipo y con las mismas características. El filtro paso de baja básico puede realizar más de 1.2Giga operaciones por segundo (Gops), alcanzando una FPGA completa más de 68Gops, sin embargo, esta tasa va disminuyendo a medida que el filtro se va volviendo más complejo, alcanzando el filtro paso de baja con ganancia ajustable 1Gops. Por otra parte el filtro paso de alta aumenta esta tasa hasta 1.1Gops, ya que realiza una operación más. Ambos filtros paso de banda alcanzan más de 2Gops, siendo los filtros que más operaciones equivalentes son capaces de realizar.

Tabla 7-14: Resumen de la síntesis y rendimiento de los filtros basados en spikes

Filtro basado en spikes	Número de bits	Slices	Máxima Frecuencia	Cantidad máxima de filtros en FPGA	Operaciones equivalentes por segundo. 1 filtro / máximo FPGA
Paso de baja	12	65	137.9MHz	55	1.24 / 68.26 Gops
Paso de baja con ganancia mayor que 1 – LFSR	12	100	126.8MHz	35	1.14 / 39.94 Gops
Paso de baja con ganancia mayor que 1 – Bit-Wise	12	105	113.2MHz	34	1.02 / 34.64 Gops
Paso de baja con ganancia ajustable	12	139	111.5MHz	25	1.00 / 24.90 Gops
Paso de alta	12	153	110.7MHz	23	1.11 / 25.46 Gops
Paso de banda Q<0.5	12,12	274	109.4MHz	13	2.078 / 27.02 Gops
Paso de banda Q>1	12,12	164	111.54MHz	21	2.12/44.52 Gops

8. Aplicación práctica de los filtros basados en spikes. La cóclea sintética

Uno de los campos en los que típicamente han sido usados los filtros frecuenciales, tanto analógicos como digitales, es en el diseño de sistemas de audio. Diversos centros de investigación han mostrado recientemente una intensa actividad referida al diseño de sensores y sistemas procesamiento de sonido de manera neuro-inspirada. En el apartado de los sensores de audio neuro-inspirados nos encontramos con diversos "oídos artificiales", o cócleas, la primera fue propuesta por Lyon y Mead en 1988 [Lyon88], encontrando las primeras implementaciones en [Watts92] y [Watts93], además podemos hallar varias implementaciones recientes en [Chan06], [Schaik05], [Hamilton08], [Hamilton09] y [Wen09], estando estas cócleas diseñadas como circuitos integrados AVLSI y ofreciendo una salida pulsante basada en la representación AER. Existe una propuesta de diseño de una cóclea digital basada en FPGA en [Wong06], estando compuesta por un banco de filtros digitales IIR, gobernados por un microcontrolador empotrado en la FPGA. También encontramos un ejemplo peculiar en [Mandal09], el cual propone una cóclea para señales con frecuencias del orden de 2GHz orientada a aplicaciones de cálculo de correlación entre señales. Las aplicaciones para las que se usan estas cócleas son muy diversas, existiendo desde aplicaciones de eco-localización [Yu09], de fusión audio-motora [Gomez07], y reconocimiento del habla [Chakrabartty2010].

Las cócleas AVLSI presentan algunos inconvenientes prácticos, en primer lugar el miss-match de los circuitos AVLSI les produce desviaciones en las frecuencias de los filtros internos, siendo muy difícil de calibrarlos y casi imposible disponer dos cócleas iguales para aplicaciones biaurales o stereo. En segundo lugar este tipo de circuitos tienen decenas de voltajes de polarización, o bias [Delbrück05], de sus transistores, siendo necesario proporcionarles una circuitería periférica de generación de todos los bias, con un elevado tamaño físico y consumo de potencia. Finalmente en tercer lugar, aunque de menor importancia, es el elevado coste que supone diseñar este tipo de sistemas. A parte, en este capítulo vamos exponer como hemos diseñado una cóclea sintética usando los filtros expuestos en el capítulo anterior, para ello vamos a comenzar describiendo brevemente el funcionamiento de las cócleas biológicas, para a continuación presentar la arquitectura de la cóclea sintética diseñada, así como el mecanismo de sintonización de sus parámetros, analizando finalmente su comportamiento de manera experimental en el mundo real. La cóclea que vamos a diseñar presenta algunas carencias respecto a las existentes ya que posee un menor número de canales. Sin embargo también presenta importante mejoras, por estar diseñada en una FPGA no tiene problemas de miss-match ni no son necesarios los voltajes de bias, además de un coste del orden de mil veces inferior. Pero no hay que perder de vista, como expondremos a continuación, que la carencia de canales puede solucionarse usando FPGAs con un mayor número de elementos lógicos, gracias a la escalabilidad de la cóclea diseñada.

8.1. Modelo biológico y electrónico de la cóclea

En [Watts93] puede encontrarse una descripción muy detallada del funcionamiento de una cóclea biológica. El sonido se desplaza a través del aire en forma de presión, la naturaleza ha conseguido obtener un sensor que es capaz de convertir la onda de presión sonora en información neuronal en nuestro cerebro, pero además no de manera absoluta, si no descomponiendo el sonido en sus componentes frecuenciales. Las cócleas de los seres vivos están formadas por un cilindro en forma de espiral, como si de un caracol se tratase, a lo largo de este cilindro nos encontramos con una membrana, la membrana basilar, rodeando la membrana basilar se encuentran unos diminutos pelos conocidos como pelos internos, o inner-hair, siendo cada uno de estos pelos sensibles al movimiento de la membrana basilar. La Figura 8.1 muestra una ilustración simplificada de la forma de la cóclea, así como a las frecuencias a las que son sensibles su diversas regiones, de tal manera que los sonidos más agudos, a 20kHz son captados a la entrada de la cóclea, y paulatinamente las diversas regiones van captando sonidos con frecuencia descendente hasta llegar a los 20Hz audibles en el caso de los seres humanos.

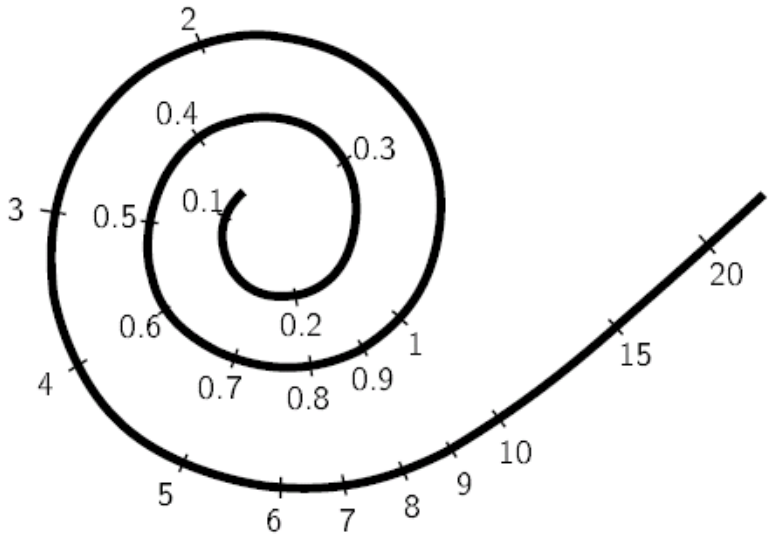


Figura 8.1: Estructura de una cóclea biológica junto con la localización de los pelos internos y las frecuencias de sensibilidad en kHz.

La membrana basilar se comporta como un filtro paso de banda, de tal manera que las ondas de sonido conforme la van atravesando longitudinalmente se ven atenuadas físicamente a lo largo de la membrana, pero sin embargo, hacen resonar una región concreta de la membrana basilar dependiendo de su frecuencia. De tal manera que al resonar la membrana los pelos internos situados debajo de esa zona serán golpeados. En la Figura 8.2 mostramos un modelo de la respuesta de la membrana basilar una cóclea biológica en base a la distancia desde el inicio de la membrana. De esta manera, las cócleas biológicas son capaces de aislar las componentes frecuenciales de las ondas sonoras, no solo captando el sonido tal cual, si no descomponiéndolo de forma mecánica. Finalmente, los pelos internos están conectados a unas células conocidas como gangliones celulares, estas células tienen un líquido

en su interior y traduce la vibración en información de naturaleza electro-química que excitará diversas partes del cerebro.

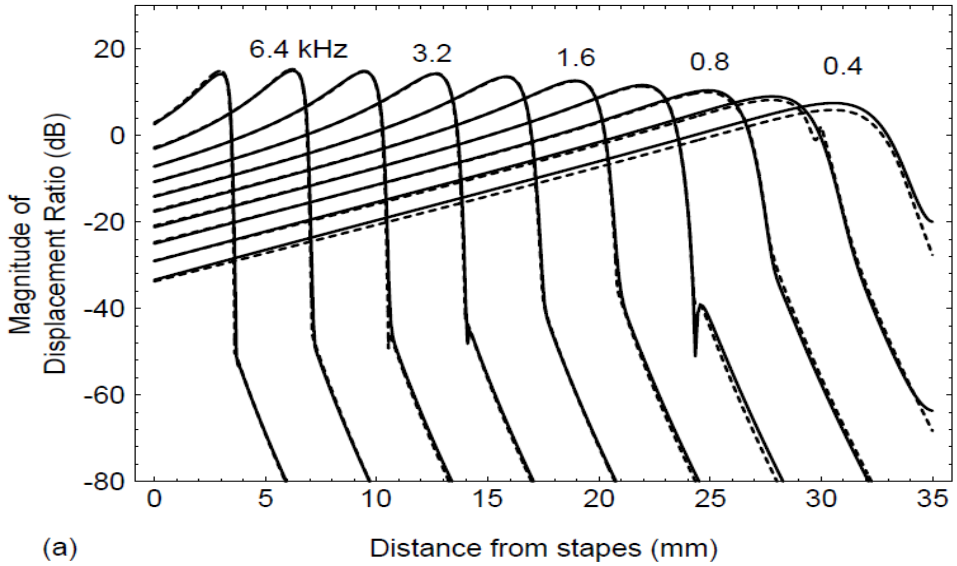


Figura 8.2: Frecuencias de paso frente a la longitud de los pelos internos de la cóclea

En [Schaik97] se propone un modelo para implementar una cóclea electrónica tratando de imitar a las cócleas biológicas. La idea de este modelo de cóclea electrónica es captar el sonido en forma de onda analógica, pudiendo provenir, por ejemplo, de un micrófono u otra fuente de audio, descomponerlo en sus componentes frecuenciales, y codificarlo en spikes. De tal manera que esta cóclea representará una capa de entrada de una red de procesamiento de sonido basada en spikes, codificando cada banda frecuencial del sonido en spikes, y comunicándolos a la siguiente capa. Con este propósito se propone un modelo de cóclea basada en bancos de filtros paso de baja analógicos de segundo orden, conectando los filtros paso de baja en cascada, y restando sus salidas para obtener las diferentes bandas de la cóclea, tal y como muestra la Figura 8.3, para a continuación finalmente convertir cada señal analógica a spikes con circuitos AVLSI.

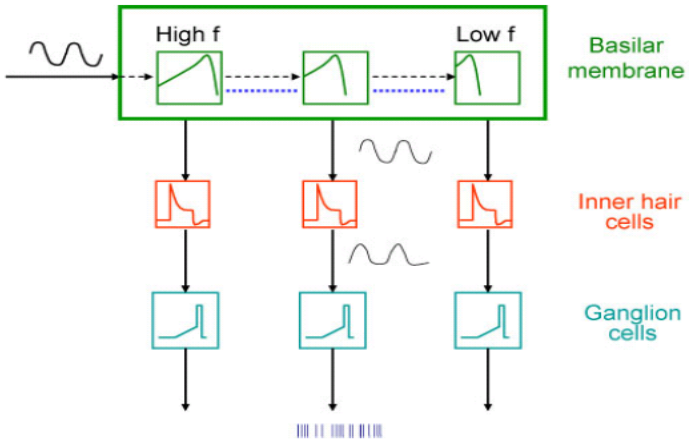


Figura 8.3: Diagrama de bloques del modela de la cóclea analógica

En [Chan07] se presenta el análisis de una cóclea analógica que implementa el modelo propuesto por [Schaik97]. Esta cóclea es una cóclea bi-aural o stereo, con un banco de filtros para el canal izquierdo y otro para el derecho, y con 32 canales en cada filtro. En la zona superior de la Figura 8.4 se muestra el diagrama de bloques de la cóclea diseñada por dichos autores. Tal y como se proponía en el modelo original, esta cóclea está compuesta por un banco de filtros analógicos, los cuales filtrarán el sonido analógico de entrada para su posterior conversión en spikes. El banco de filtros está compuesto por secciones idénticas, en la parte inferior de la figura, estando cada una de ellas compuesta por un filtro paso de baja analógico de segundo orden. La frecuencia de corte de los filtros paso de baja puede ajustarse gracias a una corriente de bias. Para sintonizar dichas frecuencias de corte se colocaron dentro del chip una línea resistiva en el silicio, conectando los filtros analógicos en distintos puntos de la línea resistiva, como si de un potenciómetro se tratara. Este mecanismo plantea problemas de miss-match, ya que la resistencia a lo largo de la línea puede variar debido a los procesos de fabricación de los chips, desviando las frecuencias de corte de los filtros, y perdiendo la distribución homogénea pretendida.

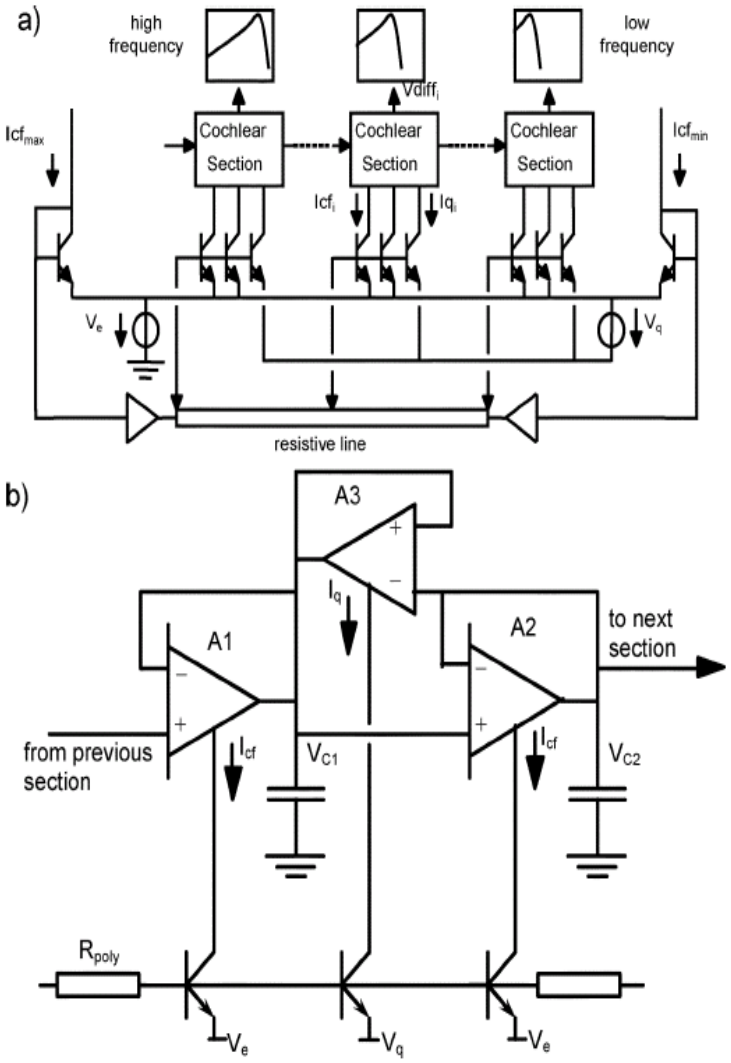


Figura 8.4: Banco y secciones individuales de los filtros de una cóclea analógica

La Figura 8.5 muestra los resultados obtenidos por los autores tras analizar la respuesta de una cóclea ya fabricada. La figura muestra la respuesta frecuencial de tres bandas del banco de filtros paso de banda en términos de spikes, es decir, la frecuencia que tienen los spikes de estos tres canales tras ser excitados por un barrido de señales senoidales. La figura evidencia como la falta de homogeneidad en la sintonización de las frecuencias de corte de los filtros, así como la desviación entre la ganancia de los filtros paso de baja, hacen que los filtros paso de banda tengan algunos problemas de off-set y distintas ganancias.

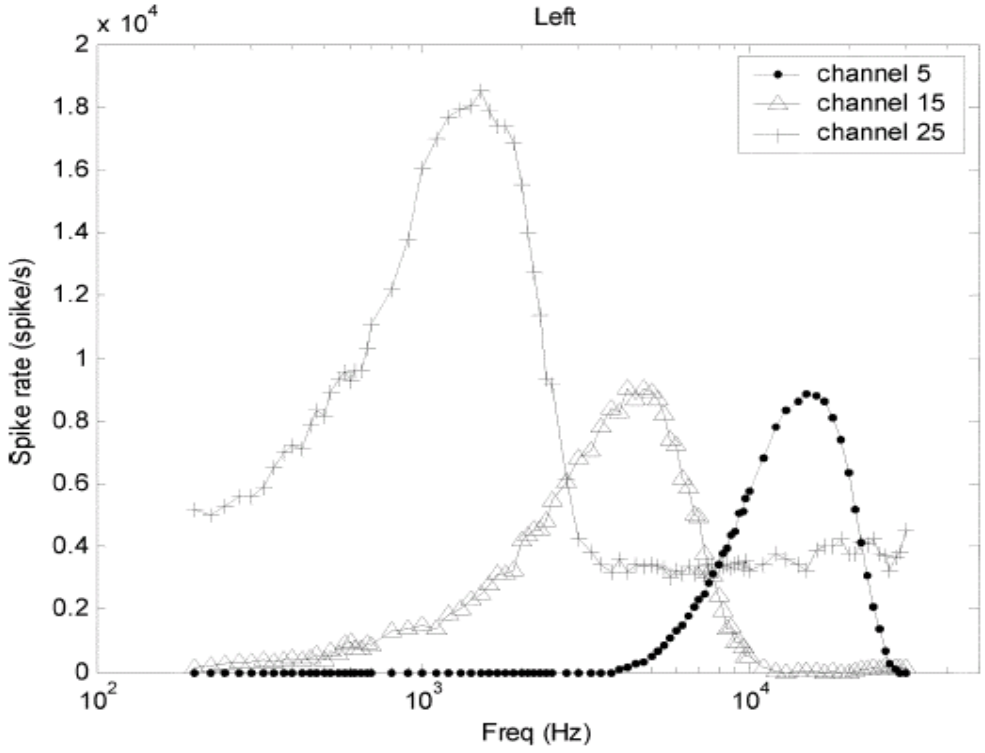


Figura 8.5: Respuesta frecuencial de los bancos de filtros paso de banda de una cóclea analógica

Finalmente los autores han presentado la salida AER de la cóclea. La Figura 8.6 muestra la salida AER del canal izquierdo de la cóclea tras capturar la voz humana durante un segundo. El eje X de la figura representa el tiempo de captura de la salida de la cóclea, y el eje Y la dirección AER del canal que dispara un spike, de manera que, por ejemplo, cada vez que se dispare el canal 24 se añadirá un punto verde en el número 24 del Y de la figura. Resulta curioso como parece que los spikes disparados están como "tumbados", esto es debido a la fase que se va acumulando a lo largo de las secciones de la cóclea se va incrementando, así como tardando cierto tiempo en propagarse a lo largo de las secciones.

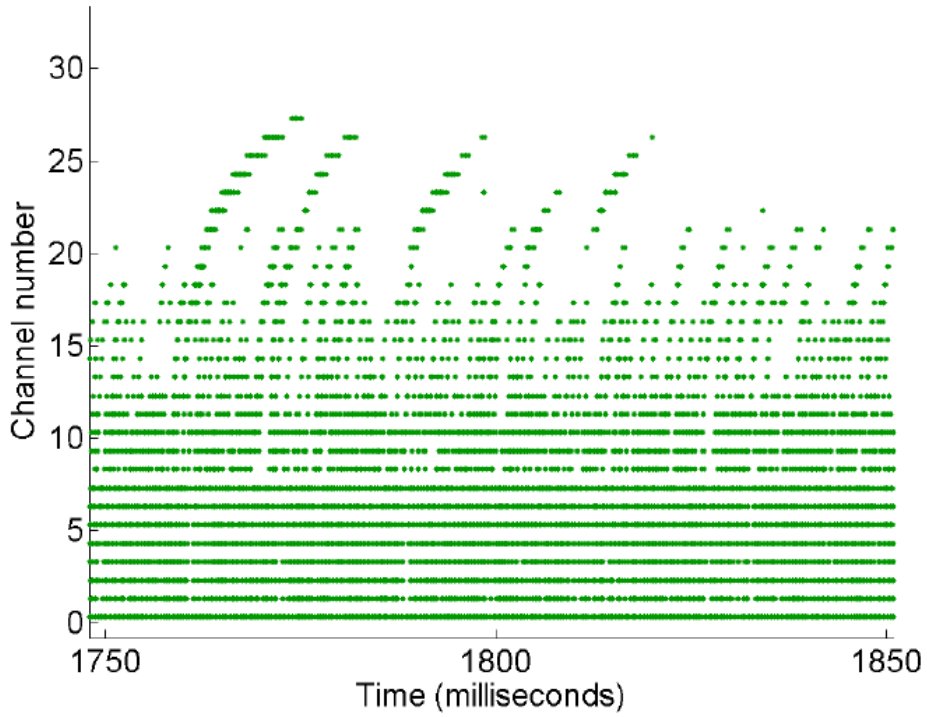


Figura 8.6: Cocleograma de una cóclea analógica de 32 canales

8.2. Propuesta de la arquitectura de la cóclea sintética

Las cócleas hasta ahora presentadas realizaban un procesamiento del audio tanto analógico como digital, para después convertir la información de cada canal de la cóclea a spikes o eventos AER. En este apartado vamos a proponer una arquitectura de carácter general para la implementación de cócleas que filtren directamente el sonido codificado como spikes, de esta manera conseguiremos combinar las ventajas de las cócleas analógicas, ofreciendo una alta escalabilidad y alta velocidad de procesamiento, y las cócleas digitales, ofreciendo una alta inmunidad al ruido y el miss-match. En la Figura 8.7 mostramos el diagrama de bloques de la arquitectura de la cóclea stereo basada en spikes, la idea de esta cóclea es que el sonido analógico se ha digitalizado gracias a un conversor analógico digital, como puede ser el PCM1804 [PCM1804], el cual transfiere el sonido digitalizado a una FPGA usando el bus de comunicación de sonido I2S [I2S], a la izquierda de la figura. Una vez en el interior de la FPGA nos encontramos con un módulo encargado de recibir el sonido a través del bus I2S y transferirlo a dos generadores de spikes, uno para el canal izquierdo y otro para el canal derecho. La salida de los generadores de spikes será una secuencia de spikes que codificarán el sonido de entrada en la frecuencia de los spikes. A continuación los spikes generados son usados como la entrada de dos bancos de filtros de spikes paso de banda, con distintas bandas de paso o canales, detallados a continuación. Finalmente la salida de los bancos de filtros es dirigida hacia un monitor AER, el cual codificará cada spike como un evento AER y lo transmitirá a la primera capa de procesamiento de audio basado en spikes.

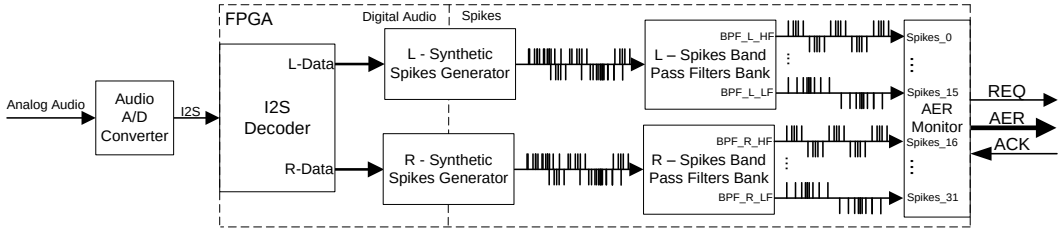


Figura 8.7: Diagrama de bloques de la arquitectura de la cóclea sintética

Para implementar el banco de filtro paso de banda hemos usado el mismo mecanismo que las cócleas presentadas anteriormente, de manera que vamos a conectar tantos filtros paso de baja de spikes de segundo orden como sean necesarios en cascada, siendo la salida de cada filtro paso de baja la entrada del siguiente, además el primer filtro es el que tiene la frecuencia de corte más alta, propagándose la salida entre los filtros con menores frecuencias de corte. Para obtener los spikes de la banda correspondiente basta con restar la salida de los filtros consecutivos usando un Hold & Fire. En la Figura 8.8 mostramos un banco de filtros de 3 canales, el cual necesita 4 filtros paso de banda de spikes de segundo orden. Para implementar los filtros de segundo orden simplemente hemos conectado dos filtros de primer orden consecutivamente.

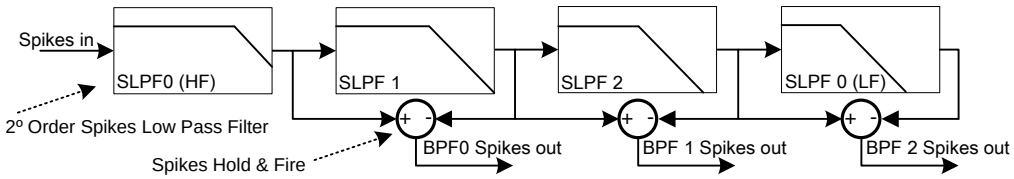


Figura 8.8: Ejemplo de un banco de filtros paso de banda con topología en cascada

Para simplificar la implementación del banco de filtro y hacerlo completamente escalable, hemos dividido el banco de filtros en secciones. Cada sección tiene la forma mostrada en la Figura 8.9, incluyendo un filtro paso de baja de spikes de segundo orden y un Hold & Fire. La idea es que los spikes provenientes de la sección anterior sean aplicados a la entrada del filtro paso baja la sección actual, de manera que salida del filtro paso de banda será la resta entre los spikes de entrada y la salida del filtro incluido en la sección, además la salida del filtro de la sección será propagada hacia la siguiente sección, para que repita el proceso. Pudiendo así añadir tantas secciones de filtros como se necesiten, pero teniendo en cuenta que hace falta un filtro paso de baja más que canales paso de banda tenga la cóclea. De manera que la entrada de la primera celda, será la salida del generador de spikes, y la salida del Hold & Fire será ignorada. Sin embargo, en la última celda no conectaremos la salida del filtro paso de baja, ya que no habrá ningún filtro esperando sus spikes.

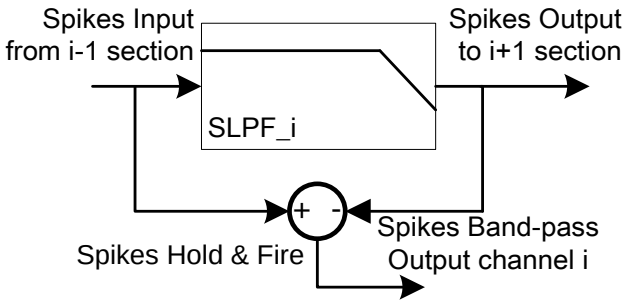


Figura 8.9: Diagrama de bloques de una celda aislada del banco de filtros paso de banda

8.3. Mecanismo de sintonización de la cóclea sintética

Sea $\{\omega_{BFP}\}$ el conjunto de las frecuencias centrales de cada filtro paso de banda, compuesto por N frecuencias, e ω_{BFP_i} la i-ésima frecuencia central, nuestro objetivo será buscar un nuevo conjunto, $\{\omega_{LFP}\}$, compuesto por las frecuencias de corte los N+1 filtros paso de banda incluidos en banco de filtros, siendo ω_{LFP_i} la frecuencia de corte del i-ésimo filtro paso de banda. Si adoptamos un punto de vista ideal, en el que los filtros no están conectados en cascada, y nos apoyamos en la Ecuación [7-20], podemos calcular la frecuencia central del i-ésimo filtro paso de banda como el producto entre las frecuencias de corte de los filtros i e i+1.

$$\omega_{BPF\_i} = \sqrt{\omega_{LPF\_i-1} * \omega_{LPF\_i}}$$

Ecuación [8-1]

Hemos de fijar la ganancia de cada filtro paso de baja a 1, ya que al estar conectados en cascada, si los filtros paso de baja tuvieran una ganancia distinta de 1 irían amplificando y/o atenuando la señal de entrada que se propagaría entre los filtros paso de baja, obteniendo filtros equivalentes muy distintos a los filtros paso de banda que pretendemos obtener, tal y como expusimos en el capítulo anterior. Teniendo en cuenta esta restricción podríamos calcular la función de transferencia equivalente de cada filtro paso de banda como la resta entre dos filtros paso de baja adyacentes, pero teniendo en cuenta que la función de transferencia de cada filtro paso de baja está compuesta por multiplicación de las funciones de transferencia de los filtros paso de baja con frecuencias de corte más altas, teniendo en cuenta que dada filtro paso de baja es de segundo orden, la función de transferencia equivalente de cada filtro paso de banda de la cóclea podría ser calculado de la siguiente manera:

$$BPF_i(S) = LPF_i(S) - LPF_{i+1}(S) = \prod_{k=1}^{i} LPF_k(S) - \prod_{k=1}^{i+1} LPF_k(S)$$
$$= \prod_{k=1}^{i} \frac{\omega_{LPF\_k}^2}{\left(s + \omega_{LPF\_k}\right)^2} * \left(1 - \frac{\omega_{LPF\_i+1}^2}{\left(s + \omega_{LPF_i+1}\right)^2}\right)$$

Ecuación [8-2]

Como la ganancia de los filtros paso de baja ha de ser unitaria, el problema del cálculo de los coeficientes de los filtros se reduce a la determinación de la frecuencia de corte.

Dado que los filtros están conectados en cascada, las frecuencias de corte de los filtros paso de baja han de estar situadas entre las bandas centrales adyacentes, dando lugar a la siguiente restricción:

$$\omega_{BPF\_i} < \omega_{LPF\_i+1} < \omega_{BPF\_i+1}$$

Ecuación [8-3]

Finalmente, el primer filtro paso de baja del banco, el de más alta frecuencia y referenciado como N+1, ha de tener una frecuencia de corte mayor que la última frecuencia central del filtro, la de más alta frecuencia, sin estar acotada y siendo la elección de este filtro crucial en el proceso de sintonización de la cóclea, ya que la salida de este filtro paso de baja será propagada a lo largo del resto de los filtros paso de baja, incidiendo directamente en su comportamiento. En consecuencia debemos acotar el la frecuencia de corte máxima que tiene el filtro paso de baja N+1.

$$\omega_{BFP\_N} < \omega_{LFP\_N+1} < Max.cut - off\ Freq.$$

Ecuación [8-4]

Usando estas ecuaciones podríamos obtener un sistema de ecuaciones para el cálculo de cada una de las frecuencias de corte de los filtros paso de baja:

$$\{\omega_{LFP}\} = \begin{cases} \omega_{LFP\_N+1} \in \left[\ \omega_{BFP\_N+1}, Max.cut - off\ Freq\ \right] \\ \omega_{LPF\_i} = \dfrac{\left(\omega_{BPF\_i}\right)^2}{\omega_{LPF\_i-1}} \end{cases}$$

Ecuación [8-5]

La ecuación anterior posee N+1 incógnitas, pero sólo N ecuaciones, estando el valor de la frecuencia de corte del filtro paso de baja N+1 simplemente acotada. Además el resto de frecuencias dependen directamente del valor elegido para la frecuencia de corte del primer filtro paso de baja.

8.3.1. Ajuste de filtros paso de banda basados en spikes median algoritmos genéticos

Dado que analíticamente resolver la Ecuación [8-5] podría ser una tarea muy tediosa, hemos optado por un método alternativo al analítico para resolverla, un algoritmo genético. Los algoritmos genéticos, o evolutivos [Michalewicz99], tratan de buscar soluciones a los problemas planteados usando técnicas evolutivas, planteando poblaciones de soluciones y permitiendo sólo la supervivencia de los mejores individuos para engendrar nuevas soluciones. El uso de esta técnica es ideal para resolver este tipo de problemas, ya que en este problema lo que tratamos es de ajustar con la mayor precisión posible un conjunto de valores acotados y muy dependientes entre ellos, además nos abre una reflexión sobre el uso de uso técnicas bio-inspiradas para el ajuste de sistemas neuro-inspirados, tal y como la naturaleza hace.

Para diseñar un algoritmo genético hemos de definir una población de soluciones, estando cada elemento de la población compuesto por un individuo, cuyo código genético porta su solución al problema. En nuestro caso particular el código genético de cada individuo estará formado por un vector de N+1 posiciones, donde N es el número de canales del banco de filtros, y cada posición contiene la frecuencia de corte de un filtro paso de baja. Además para cada individuo vamos a almacenar el error cometido respecto a las frecuencias centrales de los filtros paso de banda, de manera que la desviación, en tanto por 1, de las frecuencia central del filtro paso de banda número i, respecto a la frecuencia central ideal será calculado como:

$$Error_{Normalized\_i} = \frac{\left|\sqrt{\omega_{LFP\_i} * \omega_{LFP_i+1}} - \omega_{BFPideal\_i}\right|}{\omega_{BFPideal\_i}}$$

Ecuación [8-6]

Calculada la desviación de cada frecuencia central, estableceremos el error cometido por el individuo como la media aritmética de las desviaciones de todas las frecuencias centrales. Siendo el objetivo del algoritmo genético el minimizar este valor:

$$Error_{Normalized} = \frac{\sum_{i=1}^{N} Error_{Normalized\_i}}{N}$$

Ecuación [8-7]

La Figura 8.10 muestra el diagrama de flujo del algoritmo genético usado para el ajuste de la cóclea sintética, en primer lugar hay que definir los términos del problema, siendo estos el número de bandas, o canales, y la frecuencia central de cada una de ellas. Como ejemplo, hemos seleccionado un banco de 16 filtros y con las frecuencias centrales distribuidas logarítmicamente entre 200Hz y 15kHz, ya que este rango de frecuencia es comúnmente usado en aplicaciones de procesamiento de audio. A continuación hemos generado una población de individuos con frecuencias de corte aleatorias en el rango expuesto en la Ecuación [8-3], estando compuesta la población por 250 individuos. Una vez generada la población inicial, vamos a proceder a evaluarla, para ello le calcularemos el error cometido por cada individuo usando la Ecuación [8-7]. Una vez preparada la población inicial, nuestro algoritmo la "evolucionará" durante M generaciones, intentado minimizar el error de los individuos a medida que avanzan las generaciones. Para ello entraremos en un bucle, en el que en primer lugar ordenaremos la población de la generación anterior de forma ascendente en base al error cometido por cada individuo, de manera que los individuos con menor error quedarán situados en las posiciones más bajas de la población. A continuación tomaremos nota del mejor individuo, viendo si mejora a algún candidato anterior. Finalmente debemos cruzar los mejores individuos de la población actual, obteniendo de esta manera la siguiente generación, para implementar el cruce entre individuos seleccionaremos al azar dos individuos entre los mejores, engendrando entre ellos un nuevo individuo o hijo. Para engendrar un hijo hemos de asignarle un valor a sus genes calculado a partir de los genes de sus padres, en nuestro caso particular, el valor asignado a cada gen es la media aritmética del mismo gen que sus padres, incrementándolo en un pequeño valor aleatorio, simulando una mutación, para poder conseguir una mayor riqueza genética en nuestra siguiente generación. Además,

pera mejorar la variedad genética, en raras ocasiones tomamos como padre a uno de los peores individuos. Gracias a las mutaciones y a la selección de individuos con errores altos evitamos caer en problemas de mínimos locales, modelada en este caso como la endogamia de las poblaciones. Tras finalizar todas las generaciones, habremos almacenado el mejor individuo de todas las generaciones y será usado para ajustar la frecuencia de corte de los filtros paso de baja.

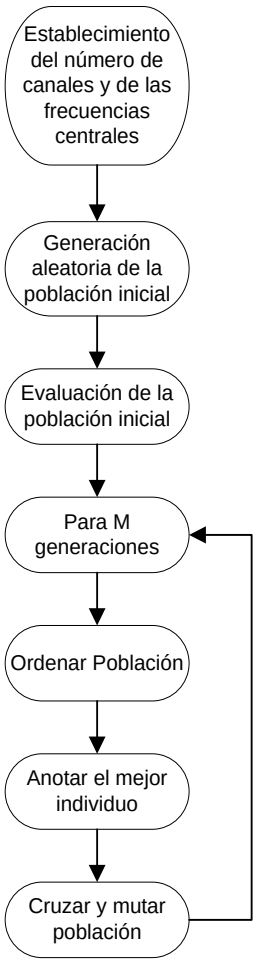


Figura 8.10: Diagrama de flujo del algoritmo genético para el ajuste de la cóclea

La Tabla 8-1 muestra los parámetros que han sido usados para la ejecución del algoritmo genético, comentando a continuación el resultado obtenido tras su ejecución. Como ejemplo de diseño de un banco de filtros hemos optado por diseñar un banco de filtros de 16 canales, para lo que necesitaremos calcular la frecuencia de corte de 17 filtros paso de baja. Para ello hemos creado una población de 250 individuos, inicializados con frecuencias de corte aleatorias, y se han evolucionado a lo largo de 80 generaciones. Para cruzar los individuos de cada generación hemos establecido una probabilidad de elegir a un padre de entre los mejores individuos de un 90%, seleccionando un padre de los peores elementos con el 10% de probabilidad restante. En caso de seleccionar un padre entre los mejores, este será escogido aleatoriamente entre el 30% de los mejores individuos de la población. Sin embargo, en caso de tener que escoger un padre entre los peores

individuos, este será elegido entre el 5% de los peores. Finalmente, al cruzar los individuos, sumaremos a las frecuencias de corte de los filtros un valor aleatorio entre -4.5Hz y 4.5Hz.

Tabla 8-1: Parámetros del algoritmo genético para el ajuste de una cóclea de 16 canales

Parámetro	Valor
Número de canales	16
Número de filtros paso de baja de segundo orden	17
Longitud de los genes de cada individuo	17
Cantidad de individuos en la población	250
Número de generaciones	800
Probabilidad de seleccionar un padre entre los mejores individuos	90%
Porcentajes de los mejores padres usados para el cruce de individuos	30%
Porcentajes de los peores padres usados para el cruce de individuos	5%
Mutación genética	[-4.5 – 4.5] Hz

La Figura 8.11 muestra los errores mínimos, máximos y medios cometidos por la población en cada generación a lo largo de la ejecución del algoritmo genético, encontrándose estos errores en tanto por uno. La población inicial tiene una desviación entorno al 10%. Sorprende este error tan bajo ya que las frecuencias de corte iniciales de los filtros paso de baja han sido elegidas aleatoriamente, sin embargo, hemos forzado frecuencias de partida acotadas entre las bandas de paso como expusimos anteriormente, hecho que ha permitido generar una población aleatoria con un bajo error de partida. En la figura también se aprecia como el error de las poblaciones va disminuyendo a medida que van avanzando las generaciones, minimizando muy rápidamente tanto el error cometido por el mejor individuo como el error de la población completa. En el caso que exponemos se ha encontrado una solución aceptable tras 800 generaciones, con poco más del 0.3% de error, pasadas unas 200 generaciones, sin embargo, aunque no se aprecie en la figura, el error mínimo no para de disminuir hasta alcanzar el 0.31% de error. Por otro lado, el error medio y máximo muestra un rizado, este rizado es debido a la inclusión de las mutaciones en el algoritmo, las cuales desvían levemente la frecuencia de los individuos al azar.

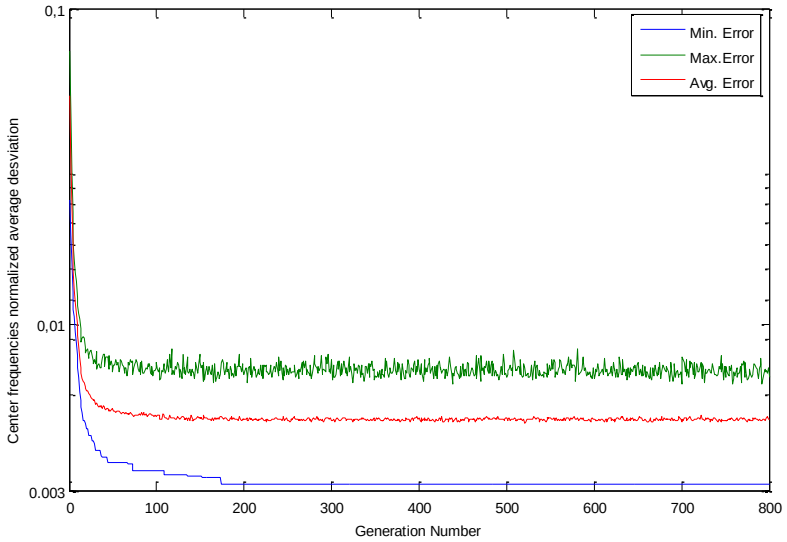


Figura 8.11: Error mínimo, máximo y medio cometido en cada generación

Como acabamos de comentar, tras la ejecución del algoritmo genético la mejor solución encontrada a nuestro problema tiene una desviación media respecto a las bandas centrales ideales del 0.31%. En la Tabla 8-2 mostramos la información contenida en el mejor individuo tras las 800 generaciones, este individuo contiene las 17 frecuencias de corte de los filtros paso de baja para la obtención de los 16 filtros paso de banda deseados. En la tabla no solo incluimos las frecuencias de corte, sino las frecuencias centrales de los filtros paso de banda, tanto equivalentes como ideales, así como la desviación porcentual que introducimos en la frecuencia central. Como puede apreciarse, el error cometido en general es muy pequeño, del orden de 10^{-2}, sin embargo los filtros de más alta frecuencia introducen una desviación muy elevada, llegando el canal de más alta frecuencia a mostrar más de un 3.6% de error.

Tabla 8-2: Resultados del algoritmo genético tras calcular una banco de filtros de 16 canales

Filtro paso de baja	Frecuencia de corte	Frecuencia equivalente paso de banda	Frecuencia central ideal paso de banda	Error relativo
1	19.073kHz	-	-	-
2	12.663kHz	15.541kHz	15kHz	3.6068%
3	9.903kHz	11.198kHz	11.248kHz	0.4443%
4	7.219kHz	8.455kHz	8.435kHz	0.2371%
5	5.547kHz	6.328kHz	6.325kHz	0.0398%
6	4.054kHz	4.742kHz	4.743 kHz	0.0194%
7	3.124kHz	3.559kHz	3.557kHz	0.0520%
8	2.277kHz	2.667kHz	2.667kHz	0.0199%
9	1.757kHz	2kHz	2kHz	0.0037%
10	1.28kHz	1.5kHz	1.5kHz	0.0022%
11	987Hz	1.124kHz	1.125kHz	0.0537%
12	718Hz	842Hz	843Hz	0.1675%
13	558Hz	633Hz	632Hz	0.0737%
14	403Hz	474Hz	474Hz	0.0176%
15	315Hz	356Hz	356Hz	0.2303%
16	226Hz	267Hz	267Hz	0.0070%
17	177Hz	200Hz	200Hz	0.0417%

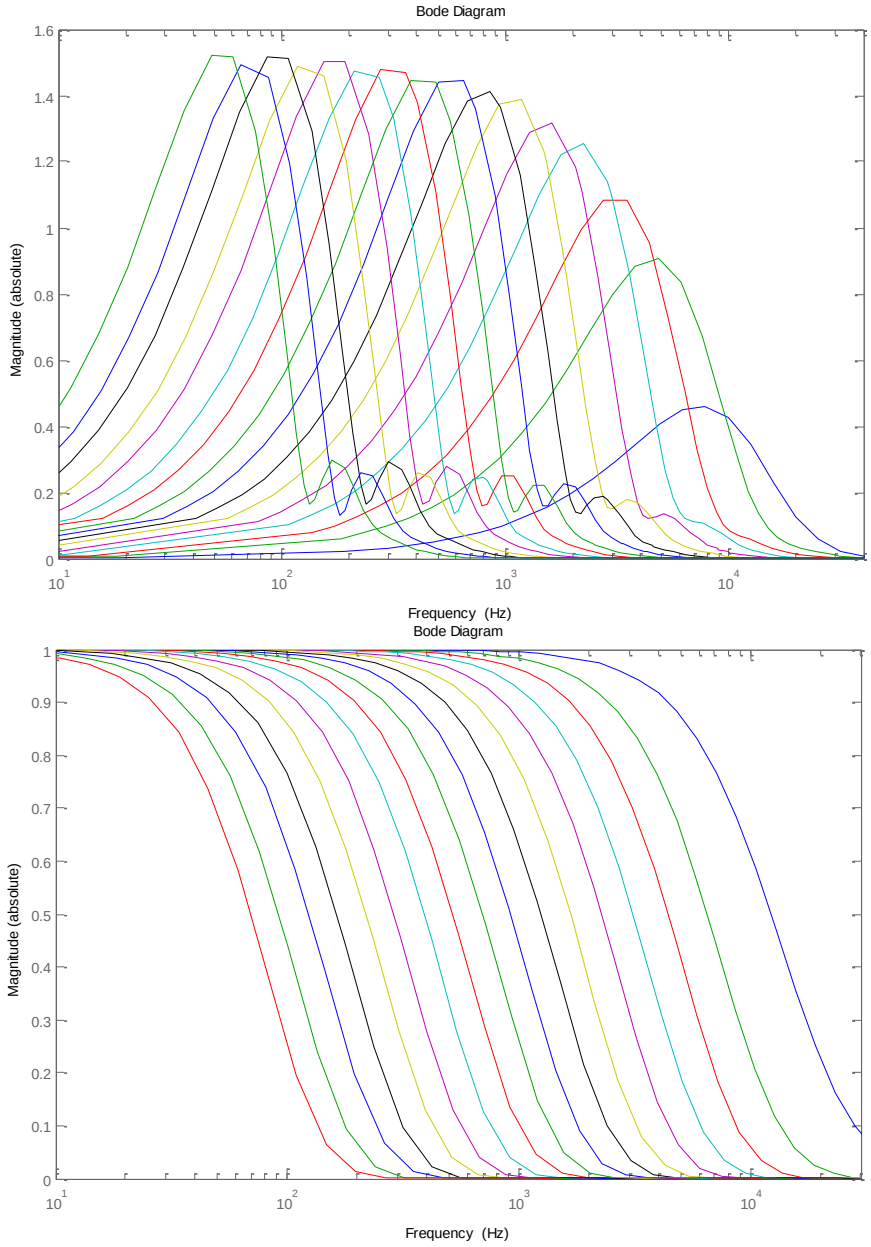


Figura 8.12: Diagramas de Bode de los filtros equivalentes paso de banda y los filtros paso de baja conectados en cascada

La Figura 8.12 muestra el diagrama de Bode ideal de los 16 filtros paso de banda equivalente, obtenidos tras la ejecución del algoritmo genético, en su parte superior, y en su parte inferior el diagrama de Bode de los filtros paso de baja incluidos en el banco de filtros. En el diagrama, los filtros paso de banda obtenidos están muy bien sintonizados, centrados con mucha precisión en la banda ideal, sin embargo su ganancia se va decrementando con la frecuencia, ya que tal y como vimos en el capítulo anterior, la ganancia del filtro depende de la distancia entre los filtros paso de baja. En la parte inferior de la figura mostramos el diagrama de Bode de los filtros paso de baja de segundo orden conectados en cascada. Se ve cómo el filtro de más alta frecuencia es el que tiene una pendiente de atenuación más pequeña que el

resto de filtros, y conforme nos vamos encontrando filtros en cascada, la pendiente de atenuación aumenta. Siendo ésta la mayor ventaja de esta arquitectura, ya que a medida que los spikes del sonido atraviesan más secciones de la cóclea, más atenuadas se encuentran las altas frecuencias, mostrando pendientes más abruptas. Este hecho incide radicalmente en la ganancia de las bandas, ya que las bandas de alta frecuencia se ven atenuadas no sólo por la distancia entre las frecuencias de corte, sino por la pendiente que ofrecen los mismos.

8.3.2. Ajuste del banco de filtros basados en spikes

Obtenidas las frecuencia de corte de los filtros paso de baja de spikes, sólo nos resta implementar el banco de filtros, para ello el último paso a dar será calcular los coeficientes propios de cada filtro paso de baja, para que su frecuencia de corte sea la obtenida a partir del algoritmo genético. Para esta tarea hemos usado el procedimiento de cálculo de los coeficientes de un filtro usando el algoritmo expuesto en el Apartado 7.3, así como la Ecuación [7-28] y la Ecuación [7-29]. Obteniendo de esta manera los parámetros de cada filtro paso de baja expuestos en la Tabla 8-3. Para calcular estos parámetros hemos intentado usar divisores de frecuencia pequeños, limitándolos a 2, resultando así filtros con número de bits comprendido entre 8 y 14, ajustando con precisión la frecuencia de corte del filtro con el divisor de spikes de la realimentación, y asignándole este mismo valor al divisor de spikes de la realimentación para obtener una ganancia unitaria.

Tabla 8-3: Parámetros de los filtros paso de baja obtenidos desde las frecuencias de corte

Filtro paso de baja	Número de bits	Divisor de frecuencia	Divisor de spikes de salida	Divisor de spikes de la realimentación
1	8	0	0.6242	0.6242
2	8	1	0.8288	0.8288
3	8	2	0.9722	0.9722
4	9	1	0.9449	0.9449
5	9	1	0.7261	0.7261
6	9	2	0.7961	0.7961
7	10	1	0.8177	0.8177
8	10	2	0.894	0.8940
9	11	1	0.92	0.9200
10	11	1	0.6703	0.6703
11	11	2	0.7752	0.7752
12	12	1	0.7522	0.7522
13	12	2	0.876	0.8760
14	13	1	0.8444	0.8444
15	13	2	0.9901	0.9901
16	14	1	0.9454	0.9454
17	14	1	0.7416	0.7416

Con el fin de comprobar la versatilidad del algoritmo diseñado, hemos calculado dos nuevos bancos de filtros paso de banda, uno de 24 canales y otro de 32, cuyos diagramas de Bode pueden encontrar en la Figura 8.13. Tal y como muestran las figuras, podemos calcular gracias al uso del algoritmo genético bancos de filtros de tantos canales como deseemos, obteniendo soluciones muy precisas en la mayoría de los casos. En los casos presentados la desviación media se sitúa entre 0.34% y 0.38% respectivamente, siendo aparentemente algo más elevada según vamos incrementando el número de filtros.

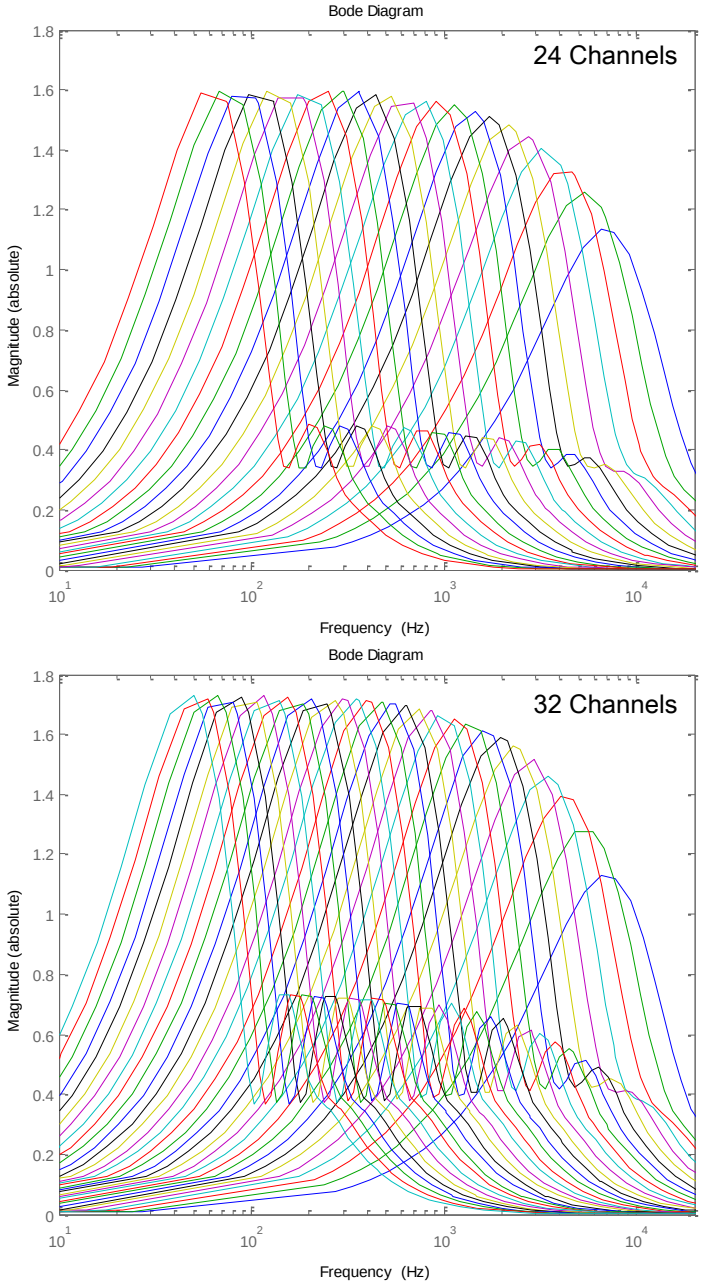


Figura 8.13: Diagramas de Bode de dos bancos de filtros, de 24 y 32 canales

8.4. Síntesis y rendimiento del banco de filtros

Para concluir con nuestra propuesta de una cóclea sintética, vamos a sintetizar los bancos de filtros calculados con el algoritmo genético, para poder realizar una estimación del consumo hardware que requerirán, así como poder observar su máxima velocidad de operación para calcular su rendimiento hardware. Dado que los resultados preliminares de la síntesis de los bancos de filtros mostraron un elevado consumo, decidimos pensar en FPGAs con una mayor capacidad, como son las FPGA de la familia Virtex-II de Xilinx [VIRTEX2], en particular la XC2V1000. La Tabla 8-4 muestra, entre otros datos, la cantidad necesaria de slices para implementar una sección de la cóclea individual y bancos de filtros de diversos números de canales, desde 16 hasta 32. El consumo de slices de cada banco aumenta casi linealmente con número de canales, sin embargo, la máxima frecuencia de operación oscila. Esta oscilación la acachamos a las optimizaciones que realiza el sintetizador, ya que al estar implementados los parámetros de los filtros como constantes, no se materializan en registros, sino en constantes, de esta manera el sintetizador puede simplificar muchos circuitos lógicos decarácter de general, como comparadores digitales, por circuitos combinacionales simples, compuestos por pocas puertas lógicas.

Para calcular el rendimiento de los filtros basados en spikes, vamos a calcular cuántas operaciones equivalentes realiza comparado con un DSP, el cual necesita una elevada cantidad de operaciones lógicas para realizar esta tarea, tal y como hicimos en el último apartado del capítulo anterior. Usando la Ecuación [7-41], podemos calcular la ecuación de un filtro FIR paso de baja de segundo orden de la siguiente manera:

$$y(n) = b_0 * x(n) + b_1 * x(n-1) + b_2 * x(n-2) + b_3 * x(n-3)$$

Ecuación [8-8]

De tal manera que cada sección de la cóclea tendrá que ejecutar esta operación y restarlo con el valor de la sección anterior, necesitando un total de 4 sumas y 4 multiplicaciones, equivaliendo en total a 20 operaciones. Realizando cada banco de la cóclea estas 20 operaciones por sección añadida, incluidas en la tabla. En la última columna de la tabla se muestra el número de operaciones equivalentes realizada por cada banco de filtro, mostrando una tasa de operaciones elevadísima, gracias a que todas estas operaciones se realizan en paralelo y a una eleva frecuencia.

Tabla 8-4: Consumo hardware de bancos de filtros de diversos números de canales

Número de canales	Consumo de slices	Máxima frecuencia de operación	Operaciones necesarias	Tasa de operaciones equivalentes
Sección individual del banco de filtros, 11bits	240	115.62MHz	20	23.12Gops
16	2992	103.81MHz	320	332.19Gops
20	3421	117.26MHz	400	469.04Gops
24	4009	120.52MHz	480	578.5Gops
28	4629	117.47MHz	560	657.83Gops
32	5118	108.32MHz	640	693.25Gops

9. Resumen, conclusiones y trabajos futuros.

Los diferentes elementos y mecanismos para el procesado de señales pulsantes que se han ido mostrando a lo largo de este trabajo se han obtenido gracias al paralelismo que hay entre éstos y los sistemas analógicos. En este capítulo vamos a resumir las características principales de los elementos de procesado pulsantes utilizados y su equivalencia con respecto a los sistemas analógicos tradicionales. En realidad, lo que estamos tratando es de realizar una computación pulsante a base de operaciones aritméticas mediante spikes. Para poder reflexionar de forma correcta sobre todo esto, vamos a sacar del olvido un campo como es el de la computación analógica, comprobaremos que prácticamente contamos ya con casi todos los elementos para emular un computador analógico, pero con mecanismos pulsantes. La ventaja de operar de esta forma es, por un lado, que en lugar de usar los clásicos amplificadores operacionales se pueden utilizar circuitos digitales bastante simples, y por otro, que la representación pulsante de la información está a medio camino entre las señales analógicas (afectadas por el ruido) y digitales (muestreadas y cuantizadas), sólo tomando las ventajas del uso de ambas.

9.1. Computadores analógicos y su equivalente pulsante.

La forma de operar de los computadores analógicos está basada en el modelado de sistemas reales y su analogía con los sistemas electrónicos (o mecánicos). De forma que si un sistema viene modelado por una ecuación diferencial dada, y si por analogía se obtiene un sistema electrónico con una descripción similar, se podría obtener una solución al primero sin más que construir el segundo, excitarlo y medir el resultado [Rummer69][Eterman60][Howe04a]. Los computadores analógicos también pueden realizar operaciones simples como la resolución de sistemas de ecuaciones [Rummer69]. Las posibilidades y limitaciones de los computadores analógicos están ampliamente estudiadas [Rummer69][Hower04a], no siendo nuestro objetivo analizarlas con detalle. Pretendemos simplemente analizar qué operaciones básicas pueden realizar estos dispositivos, así como comprobar si existe semejanza con los elementos pulsantes desarrollados en este trabajo. Vamos a obviar otros aspectos importantes como son los referentes a condiciones iniciales y escaladas.

Los computadores analógicos electrónicos utilizan básicamente señales analógicas caracterizadas por cuatro parámetros: amplitudes (tanto continua como alterna), frecuencia y fase, con los cuales se opera. Las operaciones clásicas que pueden realizar son: la suma, la inversión, la integración (respecto al tiempo), la multiplicación/división por una constante y la multiplicación, además de otras que pueden ser menos usadas como logaritmos, exponentes, divisiones y diferenciación respecto del tiempo. Todo ello basándose en unos pocos elementos electrónicos: resistencias, condensadores, diodos, etc., pero sobre todo amplificadores operacionales.

En este trabajo nos hemos centrado en sistemas lineales (focalizados para el control básico y la imitación de filtros), como es natural no hemos necesitado de cierto tipo de

operaciones que en otros ámbitos sí son necesarias, como puede ser la multiplicación. A continuación vamos a ir identificando los elementos pulsantes, desarrollados a lo largo de este trabajo, con las operaciones que pueden realizar un computador analógico. En el caso de la multiplicación vamos a proponer posibles soluciones aunque ninguna de ellas ha sido probada.

9.1.1. Inversión.

En el caso de los computadores analógicos la operación de inversión se realiza mediante el clásico inversor basado en un amplificador operacional, o AO [Rummer69]. En el caso del procesado pulsante basta cambiar los canales positivo y negativo (equivalente a cruzar un par de cables). La simpleza del inversor se debe a que en este trabajo hemos convenido que los pulsos de signo positivo viajan por un canal diferente al de los pulsos de signo negativo. Si el convenio fuera otro, como puede ser en el caso del AER el utilizar un bit de signo, la inversión no plantearía tampoco problema alguno, en sistemas pulsantes siempre es una operación muy simple y poco costosa desde cualquier punto de vista.

9.1.2. Suma y resta

En los computadores analógicos las operaciones de suma y resta se llevan a cabo por el clásico sumador basado en AO [Rummer69]. En el caso particular de una suma, mediante el uso de sistemas pulsantes caben dos posibilidades. Un sumador simple consistiría en una puerta “OR” en cada canal (positivo y negativo), con lo que el flujo resultante de pulsos en cada canal sería la suma de las entradas al sumador. El sistema funcionaría para señales con pulsos del mismo signo. Pero este planteamiento tiene un problema, debemos suponer que la representación de una señal implica que al mismo tiempo no puede ser positiva y negativa, es preciso que los pulsos de cada sumando puedan ser cancelados si su signo es el contrario y ocurren en un determinado entorno temporal, puesto que esto significa que ambas señales se contrarrestan. Para ello se ha diseñado el Hold & Fire (Capítulo 3), en el que seleccionando de forma adecuada el tiempo de hold se puede realizar sobre el flujo de pulsos de dos sumandos el balance de pulsos positivos y negativos. Lo que realmente se ha implementado con este dispositivo es un restador, pero debido a la inversión, su modelo es equivalente a un sumador de dos señales. Para realizar una suma de múltiples sumandos caben dos opciones: enlazar varios sumadores o implementar un Hold & Fire de múltiples entradas, lo cual es bastante simple e inmediato extendiendo el principio utilizado en este trabajo.

9.1.3. Integrador Temporal

En los computadores analógicos se recurre al clásico integrador basado en condensador y AO [Rummer69]. En nuestro caso hemos desarrollado el Integrate & Generate (Capítulo 4). Este dispositivo se puede implementar de múltiples formas, pero básicamente es un contador de pulsos cuyo valor muestra constantemente la integral que se va realizando más un elemento que vuelva a transformar dicho valor a tren de pulsos para obtener de esta forma una señal pulsante que representa la integral de la entrada.

9.1.4. Derivador Temporal.

En los computadores analógicos se suele recurrir de nuevo a un condensador más un AO, pero en configuración diferenciador [Rummer69]. En la computación analógica no suelen usarse los derivadores puesto, que estos son filtros paso de alta y amplifican el ruido de alta frecuencia. En consecuencia suele replantearse el problema para obviar el uso de derivadores por el riesgo que supone de inestabilidad [Eterman60]. En este trabajo se ha desarrollado un derivador a partir de la realimentación del integrador (Capítulo 4). Este derivador tiene un polo, lo que puede considerarse como que se le añade ciertas características de filtro paso de baja de orden uno, lo que mejora su respuesta desde el punto de vista del ruido de alta frecuencia. Aunque en este trabajo no se muestra un derivador sin polos, se podrían estudiar nuevas implementaciones de dicho elemento utilizando el Hold & Fire, de forma que las dos entradas del mismo se corresponderían con la señal a derivar, pero con una entrada retrasada con respecto a la otra.

9.1.5. Multiplicar por una constante

Para realizar dicha función los computadores analógicos se sirven de AO ajustando la ganancia mediante potenciómetros, es la misma arquitectura del sumador o el inversor [Rummer69]. En este trabajo dicha función se ha realizado mediante los divisores de pulsos (Capítulo 7). Es evidente que los divisores de pulsos permiten obtener la señal multiplicada por un número entre cero y uno, son divisores. Una forma de multiplicar por una constante N sería realizar un circuito que lanzase N pulsos a la salida por cada pulso que llegase a la entrada. Esto sin lugar a dudas es un multiplicador por una constante bastante simple, pero con un problema, hace que los pulsos de salida no se encuentren distribuidos de manera uniforme. Esto implica que dicho sistema introduce componentes de alta frecuencia que no estaban al principio, obsérvese que este comportamiento no lo tiene el divisor. Una forma de solucionar este problema es colocar un filtro paso de baja, como los diseñados en el Capítulo 7, a la salida de dicho multiplicador. Pero esto al fin y al cabo equivale a utilizar el filtro paso de baja de ganancia ajustable, diseñado en dicho Capítulo 7, para realizar la multiplicación por una constante, bastando con seleccionar una frecuencia de corte suficientemente alta para no perturbar la señal de entrada, su comportamiento sería el de una amplificador de ganancia N.

9.1.6. Multiplicador

En la computación analógica tradicional se han utilizado fundamentalmente tres tipos de diseños de multiplicadores [Howe05a]:

- Servo multiplier: Con servomotores que cambiaban la ganancia de un amplificador actuando sobre un potenciómetro, un multiplicando entraba al amplificador y el otro actuaba con el servo.

- Time division multiplier: La idea consiste en modular uno de los operandos en PWM, de forma que la anchura de pulso sea proporcional a la amplitud en cada instante de dicha

señal de entrada, mientras que el otro operando actúa sobre la altura del pulso, obsérvese que el área de dicho pulso representa la multiplicación de ambos operandos, habría que colocar a la salida del multiplicador un filtro paso de baja.

- Quarter-square multiplier: Estos multiplicadores están basados en la igualdad:

$$x * y = \frac{(x+y)^2 - (x-y)^2}{4}$$

Ecuación [9-1]

Por lo que obteniendo el cuadrado la multiplicación es inmediata. Para obtener el cuadrado se utilizan diodos conformadores de señal, ajustando el sistema para aproximar la función cuadrática por líneas.

Este elemento no ha sido desarrollado en este trabajo, puesto que los sistemas tratados no lo requerían. Debería plantearse como trabajo futuro, puesto que es un elemento indispensable para que este tipo de procesado pulsante pueda ser utilizado ampliamente.

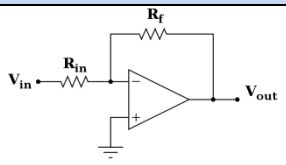
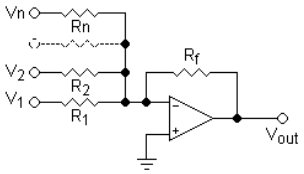
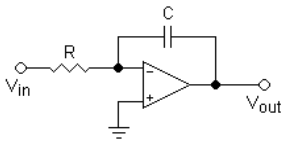
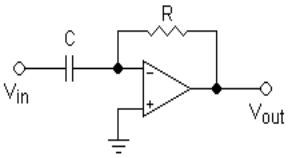
Una buena forma de empezar dicho planteamiento es utilizar sistemas de multiplicación equivalentes a los tradicionales de los computadores analógicos, expuestos en los párrafos anteriores. Así tenemos que basándose en el planteamiento del Servo Multiplier podemos actuar sobre la ganancia de un filtro paso de baja de ganancia ajustable (Capítulo 7) según uno de los multiplicandos, obteniéndose así un amplificador cuya ganancia está controlada por un operando. Esto implica que debemos traducir el flujo de pulsos de dicho operando en valores numéricos de la amplitud en cada instante, una forma de hacer esto consiste en medir el tiempo entre pulsos e ir cambiando el valor de la ganancia según dicha cifra, siempre y cuando la señal de entrada no tenga componentes de alta frecuencia (varíe más lentamente que el tiempo de respuesta del amplificador). Otra posibilidad es contar el número de pulsos durante un tiempo determinado y utilizar dicha cifra como ganancia, volviendo a poner el contador a cero en cada periodo, esto sería equivalente a realizar un muestreo periódico del valor medio de la señal, obsérvese que durante todo este trabajo se ha evitado dicha forma de proceder. Con respecto al signo en la salida no sería demasiado difícil de tratar de forma paralela a multiplicar el valor absoluto de la señal.

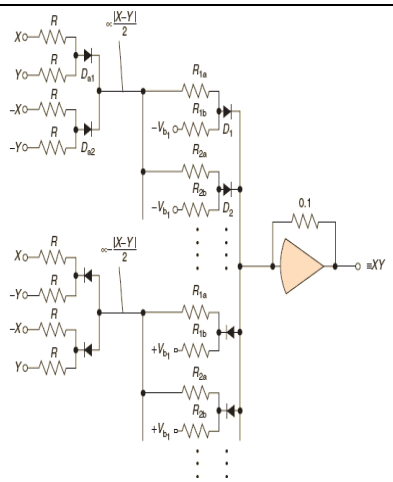
Basándose en el mecanismo time división multipliers, también se puede generar una multiplicación (división) de forma relativamente sencilla. Usando el generador de PWM del Capítulo 2 con uno de los operandos y controlando el paso o no de pulsos del otro operando mediante la señal de PWM también se obtiene un multiplicador, habría que colocar un filtro paso de baja a la salida. De la misma forma se podría utilizar el Spike Expansor para PFM descrito en el mismo capítulo.

En todo caso lo que podemos comprobar es que se pueden realizar operaciones con pulsos equivalentes a las realizadas en los computadores analógicos con elementos digitales muy simples, los cuales pueden implementarse en FPGAs de bajo coste, cuyo resumen podemos encontrar en la Tabla 9-1. La clave de toda esta analogía es la representación pulsante de las señales analógicas, que sin perder la información permite un procesado

parecido al analógico pero con elementos digitales. Como se ha indicado anteriormente la validez de la representación pulsante es clave para que se pueda llevar a cabo este tipo de procesado [Morgado04]. Por otra parte las ganancias del sistema junto con la pendiente de la recta de modulación a la hora de realizar la transformación de cada señal también es un factor de vital importancia. Resultando destacable que el estudio de la respuesta temporal y del ancho de banda de los elementos diseñados es importante para determinar las limitaciones que el procesado pulsante pueda tener.

Tabla 9-1: Operaciones básicas en computadores analógicos y sus equivalentes en procesador pulsante

	Computador Analógico	Computador Pulsante	Complejidad de implementación
Inversor		Cruce de canal positivo y negativo	No requiere hardware
Sumador		Hold & Fire	Biestables.
Integrador		Integrate & Generate	Contadores y comparadores
Derivador		Realimentación con Integrate & Generate	Igual que el integrador más un Hold & Fire.
Multiplicar por una constante	Igual que sumador (o inversor) eligiendo R1, R2,…Rn y Rf de forma apropiada.	Divisores y multiplicadores de spikes.	Igual que el derivador, con la adición de los contadores del divisor de spikes

Multiplicar (*Quarter-square multiplier [Howe04a]*)		Filtro paso de baja con ganancia ajustable controlada por multiplicador.	Idéntica a multiplicar por una constante, pero usando varios filtros.

9.2. Resumen

A lo largo del presente trabajo hemos propuesto, diseñado, implementado, simulado, y analizado diversos mecanismos para implementar controles basados en los modelos de las neuronas pulsantes. Para ello, en primer lugar, hemos diseñado e implementado elementos para actuar sobre motores de DC a partir de spikes. Se han implementado elementos basados en dos modulaciones distintas, la modulación PWM y la modulación PFM, siendo esta última coincidente con la usada por los modelos neuronales pulsantes más habituales (tipo AER). Además de diseñar e implementar ambos elementos, los hemos simulado junto con modelos de motores para poder así analizar las respuestas de un motor en diversos escenarios. Gracias a dichas simulaciones hemos podido analizar la interacción entre motores y los elementos implementados. Realizar diversas comparaciones y extrayendo de ellas las fortalezas y debilidades de los mecanismos propuestos.

El siguiente paso ha sido la propuesta, diseño, implementación, simulación y análisis de controles en lazo cerrado basados en spikes, comenzando con el diseño de simple controladores P, aumentando su complejidad hasta diseñar controlador PID basados en spikes. Para el desarrollo de controladores P basados en pulsos hemos propuesto dos mecanismos para restar dos señales de spikes, estos elementos han sido el Inter-Spike-Interval Difference & Generate y el Hold & Fire. A partir de estos elementos hemos construido diversos escenarios de simulación combinándolos con el modulador PWM y el Spikes Expansor (PFM), para de esta manera poder analizar comparativamente las cualidades del uso de uno u otro mecanismo. A continuación se han desarrollado un integrador y un derivador, basados ambos en el Integrate & Generate, de spikes. Con estos elementos más el Hold & Fire se han obtenidos controladores PID, que posteriormente se han simulado. A partir de las simulaciones hemos podido analizar las respuestas en cada caso y compararlas entre ellas. Consiguiendo respuestas similares a los sistemas tradicionales de control PID.

Una vez simulados todos los elementos necesarios para implementar controladores PID basados en spikes, hemos procedido a llevarlos a la realidad. Como primer paso hemos diseñado y construido la plataforma AER-Robot, la cual da soporte físico a los controles. A

continuación hemos procedido a adaptar las implementaciones de los controles para llevarlos a la realidad, estableciendo mecanismos de comunicación desde el exterior hasta los controles, e implementando un monitor basado en la representación AER para el monitorizado y posterior análisis de los controles.

A continuación hemos construido un pequeño robot móvil, Eddie, como plataforma de demostración. Eddie es un robot diferencial, contiene controles más complejos que simples controles PID, permitiéndole así navegar por el mundo con controles neuro-inspirados en su interior. Para comprobar el correcto funcionamiento de Eddie hemos ampliado el monitor AER y analizado sus respuestas ante diversas señales de excitación.

Finalmente, hemos realizado un análisis de los elementos diseñados para el control PID desde el punto de vista del procesamiento de señales, implementando filtros paso baja, de banda y de alta, basados en spikes y equivalentes a los filtros analógicos. Caracterizando los parámetros y ajustes necesarios de dichos filtros, para posteriormente simular y probar sus respuestas. Como aplicación práctica se ha realizado una propuesta de una nueva cóclea artificial utilizando bancos de filtros pulsantes, proponiendo y usando algoritmos genéticos para ajustar adecuadamente los diversos parámetros de los filtros, dado su complicación a nivel paramétrico.

9.3. Aportaciones más importantes y conclusiones

- Se ha diseñado, implementado y probado, creemos que por primera vez y de forma satisfactoria, controladores PID basados exclusivamente en señales pulsantes.
- Se ha construido una plataforma robótica real en la que probar dichos controladores pulsantes.
- Se ha diseñado, implementado y probado, creemos que por primera vez y de forma satisfactoria, diversos filtros frecuenciales basados exclusivamente en sistemas pulsantes. El procedimiento utilizado es general y puede servir para implementar una gran diversidad de filtros.
- Se ha diseñado una cóclea artificial basada exclusivamente en banco de filtros pulsantes utilizando para su ajuste mecanismos genéticos.
- De forma general se ha mostrado, creemos que por primera vez, procedimientos simples para la obtención de sistemas de procesado pulsante a partir de sistemas analógicos tradicionales, basándose en la transformada de Laplace de su función de transferencia.
- A partir de la aportación anterior se han propuesto estructuras de procesado pulsante equivalentes a las de los computadores analógicos. Lo que permite realizar un computador analógico pulsante basado en FPGAs de relativo bajo coste, dada su simplicidad.
- Se ha podido comprobar por equivalencia de acciones con los sistemas digitales que en general los sistemas de procesado pulsante tienen un coste hardware bastante más reducido que estos.

Es evidente que para cierto tipo de sistemas empotrados los sistemas de procesado pulsante mostrados en este trabajo resultan más ventajosos que sus equivalentes analógicos o digitales. Por otra parte una de las ventajas de la representación pulsante radica en que al

estar a medio camino entre la analógica y la digital es fácil transformarla en un sentido u otro, lo que hace que sea bastante versátil.

9.4. Trabajos futuros.

Es necesario desarrollar elementos que implementen operaciones logarítmicas y exponenciales para tener la total equivalencia con los computadores analógicos. Por otra parte, es necesario desarrollar y probar los multiplicadores esbozados en este trabajo para comprobar cuales son más convenientes. Con respecto a los elementos implementados debería estudiarse e identificar aspectos como la saturación, ancho de banda equivalente, etc. para mostrar las limitaciones que estos sistemas tienen.

Una vez puestas las bases para el computador pulsante se debería probar el mismo con la resolución de problemas complejos para ver si efectivamente se consigue soluciones con menor coste.

Con respecto a la cóclea sintética diseñada, no ha sido probada aún en profundidad ni se ha realizado procesado alguno con ella, este interesante elemento debería servir para poder realizar una gran diversidad de aplicaciones de audio, englobando aplicaciones tan diversas como son la eco-localización y el reconocimiento del habla.

En lo que se refiere al control de motores, el desarrollo realizado no debe quedarse sólo en controles de bajo nivel, sino implementar controles neuro-inspirados de más alto nivel, como son controles para la navegación, actualmente con algunas propuestas [Gomez07][Jimenez09c]. Así como desarrollar mecanismos para la fusión de sistemas, combinando estos controles con otros sensores y sistemas neuromórficos. Gracias a la combinación de diversos elementos neuromórficos, deberíamos ser capaces de construir robots más complejos y eficientes, aplicando estos controles a otros tipos de robots, como son los manipulares, e incluso combinarlos con robots móviles.

Profundizando en las aplicaciones relacionadas con el control, y teniendo en cuenta la analogía presentada con los computadores analógicos, se podría plantear la posibilidad de diseñar controladores adaptativos y predictivos. Ya que el sistema a controlar podría ser modelado precisamente por un análogo sistema pulsante, de manera que nos permitiera el acceso a los observadores ocultos de la planta. Modificando, en base a esta información, de alguna manera la forma de actuar de un controlador pulsante.

Resulta también muy atractiva la idea del desarrollo de mecanismos que nos permitan a partir del modelo de un sistema obtener un circuito equivalente pulsante compuesto por los elementos expuestos. De la misma manera que existen mecanismos para diseñar circuitos analógicos y polinomios discretos en Z a partir la función de transferencia de un sistema.

En los sistemas que se han planteado se opera con señales pulsantes en la que la información reside en la frecuencia de pulsos, en los sistemas AER la información va además en la dirección del evento, la cual suele representar su posición espacial en los sistemas de vídeo. En estos momentos se están construyendo numerosos filtros espaciales, tanto analógicos como digitales, basados fundamentalmente en las convoluciones. Siendo estas operaciones meramente algebraicas, en la que un kernel se desplaza a lo largo de los elementos que componen la matriz de vídeo ¿no sería automático obtener dichas expresiones

y calcularlas para cada caso usando la operativa de pulsos mostrada en este trabajo? ¿Qué coste hardware tendría dichas operaciones? ¿Sería más eficiente que los convolucionadores, tanto analógicos, como digitales, actuales? Para lograr este avance sería muy interesante la multiplexación temporal del hardware de los componentes para el procesamiento de spikes expuestos, es decir, en lugar de usar un componente dedicado para procesar cada señal pulsante como hemos hecho en este trabajo, usar el hardware de dicho componente para procesar un elevado número de señales. Ya que aunque podemos alojar un elevado número de componentes dentro de una FPGA, no son suficientes para competir con los convolucionadores actuales basados en FPGA, los cuales también hacen una multiplexación del hardware. Sin embargo, dado que un sistema continuo está caracterizado por sus condiciones iniciales, lo cual se traduce a los valores de los contadores y registros de cada componentes para el procesamiento de spikes, si almacenamos en una memoria dichos valores para cada señal a procesar, cada vez que se recibiera un evento AER (señal de spikes), podríamos obtener de la memoria las condiciones iniciales en la que se debería encontrar el componente dedicado a procesar esta dicha señal, estableciendo las condiciones iniciales en el hardware del componente, procesando el evento AER y almacenando el valor de las condiciones en las que se ha quedado el filtro para el próximo evento.

10. Referencias

[Amax26] Graphite Brushed 11W DC Motor Datasheet:
http://shop.maxonmotor.com/maxon/assets_external/Katalog_neu/eshop/Downloads/Katalog_PDF/maxon_dc_motor/A-max-programm/new/newpdf_09/A-max-26-110958_09_EN_112.pdf

[Alexan90] Alexander and J. H. Maddocks. "On the Kinematics of Wheeled Mobile Robots". International Journal of Robotics Research in Autonomous Robot Vehicles, Springer Verlag, 1990.

[Aström09] K. J. Aström, "Control PID Avanzados". Pearson 2009.

[Barlow61] Barlow, H., "Possible principles underlying the transformation of sensory messages in Sensory Communication". MIT Press 1961

[Barbaro02] M. Barbaro, P.Y. Burgi, A. Mortara, P. Nussbaum, F. Heitger, "A 100x100 pixel silicon retina for gradient extraction with steering filter capabilities and temporal output coding". IEEE Journal of Solid-State Circuits, Vol. 37. Pags. 160-172. 2002.

[Berge07] Berge, H.K.O., Hafliger, P., "High-Speed Serial AER on FPGA". IEEE International Symposium on Circuits and Systems, ISCAS 2007. Pags.: 857 - 860.

[Berner07] R. Berner, Tobi Delbrück, Antón Civit Balcells, Alejandro Linares-Barranco. "A 5 Meps $100 USB2.0 Address-Event Monitor-Sequencer Interface". IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007.

[Boahen98] K.A.Boahen, "Communicating Neuronal Ensembles between Neuromorphic Chips". Neuromorphic Systems Engineering: Neural Networks in Silicon, capítulo 11. Noruega 1998.

[Boahen99] K. Boahen, "Retinomorphic Chips that see Quadruple Images". Proc. Int. Conf. Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (Microneuro99), pp. 12-20, Granada, España, 1999.

[Boahen00] K. Boahen, "Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events," IEEE Trans. on Circuits and Systems Part-II, vol. 47, No. 5, pp. 416-434, Mayo 2000.

[C8051F320] Datasheet C8051F320:
https://www.silabs.com/products/mcu/usb/Pages/C8051F32021.aspx

[Camuñas08] L. Camuñas-Mesa, A. Acosta-Jiménez, T. Serrano-Gotarredona, and B. Linares-Barranco, "Fully digital AER convolution chip for vision processing". Proc. of the 2008 IEEE Int. Symp. Circuits and Systems, ISCAS08, pags. 652-655, Mayo 2008.

[Capocaccia] Capocaccia Cognitive Neuromorphic Engineering Workshop:
http://capocaccia.ethz.ch/capo/

[Catsoulis05] John Catsoulis, "Designing Embedded Hardware". O'Reilly Media 2005.

[Chan06] Chan, V.; van Schaik, A.; Shih-Chii Liu, "Spike response properties of an AER EAR". IEEE International Symposium on Circuits and Systems. ISCAS 2006. Pags. 862 - 866

[Chan07] Chan, V.; Shih-Chii Liu; van Schaik, A., "AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface". IEEE Transactions on Circuits and Systems I: Regular Papers, Volume 54, Issue 1, Enero 2007 Pags.:48 – 59

[Chakrabartty2010] S. Chakrabartty y S.-C. Liu, "Exploiting spike-based dynamics in a silicon cochlea for speaker identification". IEEE International Symposium on Circuits and Systems, ISCAS2010.

[CHIPSCOPE] "ChipScope ILATools Tutorial":
http://www.xilinx.com/products/software/chipscope/chipscope_ila_tut.pdf

[Clemente01] M. Carmen Clemente Medina, Alberto Peinado Dominguez, "Caracterización de Secuencias Binarias Pseudoaleatorias generadas mediante LFSR". XVIII Simposium Nacional de la URSI (Union Radio-Scientific Internationale), 2003

[CoolRunner2] "CoolRunner2 CPLD Family Datasheet":
http://www.xilinx.com/support/documentation/data_sheets/ds090.pdf

[Costas07] J. Costas-Santos, T. Serrano-Gotarredona, R. Serrano-Gotarredona, and B. Linares-Barranco, "An AER Contrast Retina with On-Chip Calibration". Proc. of the 2007 IEEE Int. Symp. Circuits and Systems, ISCAS07, pags. 3075-3078.

[Culurciello03] E. Culurciello, R. Etienne-Cummings, and K. Boahen, "An Address Event Digital Imager". IEEE J. Solid-State Circuits, Vol. 38, No. 2, Febrero 2003.

[CypressFX2] "Cypress EZ-USB FX2 Data Sheet":
http://www.keil.com/dd/docs/datashts/cypress/cy7c68xxx_ds.pdf

[Delbrück05] T. Delbrück and A. van Schaik, "Bias Current Generators with Wide Dynamic Range". Analog integrated Circuits and Signal Processing, Vol. 43, pags. 247-268, 2005.

[Delorme00] A. Delorme, G. Richard, and M. Fabre-Thorpe, "Ultra-rapid categorisation of natural images does not rely on colour: A study in monkeys and humans". Vis. Res., 40:2187-2200, 2000.

[Delorme01] A. Delorme and S. J. Thorpe, "Face identification using one spike per neron: resistance to image degradations". Neural Networks., 14:795-803, 2001.

[Diaz97] Fernando Diaz del Rio, "Análisis y evaluación del control de un robot móvil: Aplicación a sillas de ruedas eléctricas". Tesis doctoral, Sevilla 1997.

[Ertan04] Ertan, H.B.; Simsir, N.B.; "Comparison of PWM and PFM induction drives regarding audible noise and vibration for household applications". IEEE Transactions on Industry Applications, Volume 40, Issue 6, 2004 Pags.:1621 – 1628.

[Eterman60] I.I. Etterman, "Analog Computers". 1960- Pergamon Press – London

[Fasnacht08] D. B. Fasnacht, A. M. Whatley, *and* G. Indiveri, "A Serial Communication Infrastructure for Multi-Chip Address Event Systems". Proc. of the 2008 IEEE Int. Symp. Circuits and Systems, (ISCAS08), pp. 648-651, May 2008.

[Fujii96] H. Fujii, H. Ito, K. Aihara, N. Ichinose, M. Tsukada, "Dynamical Cell Assembly Hipótesis – Theoretical Possibility of Spatio-Temporal Codign in the Cortex". Neural

Networks, vol. 9, pp. 1303-1350, 1996.

[GP26] Planetary Gearhead Datasheet:
http://shop.maxonmotor.com/maxon/assets_external/Katalog_neu/eshop/Downloads/Katalog_PDF/maxon_gear/Planetengetriebe/new/newpdf_09/GP-26-B-144026_09_EN_230.pdf

[Goldberg01] David H. Goldberg, Gert C. Cauwenberghs, Andreas G. Andreou, "Analog VLSI spiking neural network with address domain probabilistic synapses". The 2001 IEEE International Symposium on Circuits and Systems, 2001. ISCAS 2001, vol. 2, pp. 241-244, 2001.

[Goldberg03] David H. Goldberg, Gert C. Cauwenberghs, Andreas G. Andreou, "Probabilistic Synaptic Weighting in a Reconfigurable Network of VLSI Integrate-and-Fire Neurons". Neural Networks. NN01, Elsevier Science ,Vol. 14, No. 6-7.Pag.781-793.

[Gomez01] Francisco Gomez Rodriguez, Alejandro Linares Barranco, Rafael Paz Vicente, Lourdes Miro Amarante, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "Aer Image Filtering". Proceedings of SPIE, the International Society for Optical Engineering.

[Gomez05] F. Gomez-Rodriguez, R. Paz, L. Miro, A. Linares-Barranco, G. Jimenez, A. Civit. "Two Hardware Implementation of the Exhaustive Synthetic Aer Generation Method". LNCS. Vol. 3512. 2005. Pag. 534-540.

[Gomez06] F. Gomez-Rodriguez, R. Paz, A. Linares-Barranco, M. Rivas, L. Miro, S. Vicente, G. Jimenez, A. Civit. "AER tools for communications and debugging". IEEE International Symposium on Circuits and Systems, ISCAS 2006.

[Gomez07] Gomez-Rodriguez, F.; Linares-Barranco, A.; Miro, L.; Shih-Chii Liu; van Schaik, A.; Etienne-Cummings, R.; Lewis, M.A., "AER Auditory Filtering and CPG for Robot Control". IEEE International Symposium on Circuits and Systems, 2007, ISCAS 2007. Pags.: 1201 – 1204.

[Häfliger04] P. Häfliger. Especificaciones protocolo asíncrono AER para el proyecto CAVIAR: http://heim.ifi.uio.no/~hafliger/CAVIAR/Consortiumstandards.pdf

[Häfliger07] P. Hafliger. "Adaptive WTA with an Analog VLSI Neuromorphic Learning Chip". IEEE Transactions on Neural Networks, Vol. 18, No 2, pags. 551-572. Marzo de 2007.

[Hamilton08] Hamilton, T.J.; Jin, C.; van Schaik, A.; Tapson, J., "An Active 2-D Silicon Cochlea". IEEE Transactions on Biomedical Circuits and Systems, Vol. 2, Issue 1, Marzo de 2008 Pags:30 – 43.

[Hamilton09] Hamilton, T.J.; Jin, C.; van Schaik, A.; Tapson, J., "A 2-D silicon cochlea with an improved automatic quality factor control-loop". IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008. Pags.:1772 – 1775.

[HEDS550] HEDS 550: Quick Assembly Two and Three Channel Optical Encoders.
http://test.maxonmotor.com/docsx/Download/Product/Pdf/HEDS550-E.pdf

[Hodgkin52] A. L. y Huxley, A. F., "Action potentials recorded from inside a nerve fibre". Nature 144, pags. 710-711, 1952.

[Houpis89] D'azzo Houpis, "Sistemas Realimentados de Control". Parainfo 1989.

[Howe05a] Robert M. Howe, "Fundamentals of the Analog Computer". IEEE Control Systems Magazine. Junio 2005

[Howe05b] Robert M. Howe, "Analog Computers in Academia and Industry". IEEE Control Systems Magazine. Junio 2005

[Hynna01] Kai Hynna, Kwabena A. Boahen, "Space-Rate Coding in an Adaptive Silicon Neuron". Neural Networks, Special Issue on Spiking Neurons in Neuroscience and Technology, vol. 14, no. 6-7, pp. 645-656, 2001.

[I2S] I2S Bus Specification: http://www.nxp.com/acrobat_download2/various/I2SBUS.pdf

[Ifeachor02] Emmanuel Ifeachor, Barrie Jervis, "Digital Signal Processing: A practical Approach". Prentice Hall 1992.

[Inidvieri99] G. Indiveri, A.M. Whatley, and J. Kramer, "A Reconfigurable Neuromorphic VLSI Multi-Chip System Applied to VisualMotion Computation". Proc. Int. Conf. Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (Microneuro99). Pags.37-44. España 1999.

[Indiveri00] Giacomo Indiveri, "A 2D neuromorphic VLSI architecture for modeling selective attention". In Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000.

[Indiveri06] Indiveri, G. Chicca, E. Douglas, R. "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity". IEEE Trans. Neural Networks, vol. 17, No. 1, pp. 211-221, 2006.

[jAER] jAER open-source software project. http://jaer.wiki.sourceforge.net/

[Jimenez07]A. Jimenez-Fernandez. "Actuación y sensado bio-inspirado sobre motores de DC basado en Address Event. La plataforma AER-ROBOT". Proyecto fin de carrera, Sevilla 2007

[Jimenez08a] A. Jiménez-Fernández, A. Linares-Barranco, R. Paz-Vicente,C.D. Luján-Martínez, G. Jiménez, A. Civit. "AER and dynamic systems co-simulation over Simulink with Xilinx System Generator". IEEE International Conference on Electronics, Circuits and Systems, 2008. ICECS 2008.

[Jimenez08b] A. Jiménez-Fernández, R. Paz-Vicente, M. Rivas, A. Linares-Barranco, G. Jiménez, A. Civit. "AER-based robotic closed-loop control system". IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008.

[Jimenez09a] A. Jimenez-Fernandez, A. Linares-Barranco, R. Paz-Vicente, G. Jimenez-Moreno, A. Civit, "Neuro-inspired Spike-based closed-loop controller for Robotics". International Journal of Factory Automation, Robotics and Soft Computing, Volumen:3/2009, Pag. 91-100

[Jimenez09b] A. Jimenez-Fernandez, R. Berner, A. Linares-Barranco, R. Paz-Vicente, G. Jimenez-Moreno. "Spike-based control monitoring and analysis with Address Event Representation". International Conference on Computer Systems and Applications, AICCSA 2009.

[Jimenez09c] A. Jimenez-Fernandez, C. Lujan-Martinez, R. Paz-Vicente, A. Linares-Barranco, G. Jimenez, A. Civit. "From Vision Sensor to Actuators, Spike Based Robot Control through Address-Event-Representation". International Work-Conference on Artificial Neural Networks, Lecture Notes in Computer Science, Vol 5517/2009, Pags. 797-804.

[Jimenez10a] A. Jimenez-Fernandez, A. Linares-Barranco, R. Paz-Vicente, G. Jimenez, A. Civit, "Building Blocks for Spike-based Signal Processing". Enviado a International Joint Conference in Neural Networks 2010.

[Jimenez10b] A. Jimenez-Fernandez, J.L. Fuentes-del-Bosh, R. Paz-Vicente, A. Linares-Barranco, G. Jiménez, "Neuro-inspired system for real-time vision tilt correction". IEEE International Symposium on Circuits and Systems, ISCAS2010. París 2010.

[Johnston95] D. Johnston, S. Wu, "Foundations of Cellular Neurophysiology". MIT Press, Cambridge MA, 1995.

[Lamberti97] Pedro W. Lamberti, Víctor Rodríguez, "Desarrollo del modelo matemático de Hodgkin y Huxley en neurociencias". Electroneurobiología vol. 15 (4), pp. 31-60, 2007

[Lazzaro93] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Silvilotti, and D. Gillespie, "Silicon Auditory Processors as Computer Peripherals". IEEE Trans. on Neural Networks, vol. 4, pp. 523-528, May 1993.

[Leñero09] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A Mismatch Calibrated Bipolar Spatial Contrast AER Retina with Contrast Threshold". Proc. of the 2009 IEEE Int. Symp. Circuits and Systems (ISCAS09), accepted for publication.

[Lewis01] M. A. Lewis, M. Hartmann, R. Etienne-Cummings, and A. Cohen, "Biomorphic Control of a Running Robot Leg using a Custom aVLSI CPG Chip". Neurocomputing, Vol. 38-40, pags. 1409-1421, Junio 2001.

[Lewis03] M. Anthony Lewis, R. Etienne-Cummings, M. H. Hartmann, A. H. Cohen, and Z. R. Xu, "An In Silico Central Pattern Generator: Silicon Oscillator, Coupling, Entrainment, Physical Computation & Biped Mechanism Control". Biological Cybernetics, Vol. 88, No. 2, pp 137-151, Febrero 2003 .

[LFSR] "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators". http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf

[Lichtsteiner05] P. Lichtsteiner and T. Delbrück, "64x64 Event-Driven Logarithmic Temporal Derivative Silicon Retina". Proc. IEEE Workshop on Charge-Coupled Devices and Advanced Image Sensors, pp. 157-160, Nagano, Japan, 2005.

[Lichtsteiner06] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change". 2006 IEEE ISSCC Digest of Technical papers, pp. 508-509, San Francisco, 2006.

[Lichtsteiner08] P. Lichtsteiner, C. Posch and T. Delbrück, "A 128x128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor". IEEE Journal of Solid State Circuits, vol. 43, No. 2, pp. 566-576, Feb. 2008.

[Linares02] A. Linares-Barranco, G. Jimenez-Moreno, A. Civit-Balcells, J. L. Sevillano-Ramos, R. Paz-Vicente, "Software Generation of Address-Event-Representation for Interchip Images

Communications". 28th Annual Conference of IEEE Industrial Electronics Society.Pag.1915-1919. España 2002.

[Linares03] Alejandro Linares Barranco, "Estudio y Evaluación de Interfaces para conexión de Sistemas Neuromórficos mediante Address-Event-Representation". Tesis doctoral, Sevilla 2003

[Linares06a] CGI-95 Linares-Barranco A, Paz-Vicente R, Jimenez G, López Coronado J , "AER neuroinspired interface to anthropomorphic robotic hand". 3th IEEE International Joint Conference on Neural Network. Vols 1-10 Pages: 1497-1504, Vancouver, CANADA, July, 2006.

[Linares06b] A. Linares-Barranco, G. Jimenez-Moreno, A. Civit, B. Linares-Barranco, "On Algorithmic Rate-coded AER Generation". IEEE Transaction on Neural Network,Vol 17, Nº 3.Pag.771-788. 2006.

[Linares07a] Alejandro Linares Barranco, Gabriel Jimenez Moreno, Rafael Paz Vicente, Sergio Varona Moya, Angel Francisco Jiménez Fernández "An Aer-Based Actuator Interface for Controlling an Anthropomorphic Robotic Hand". Lecture Notes in Computer Science. Vol. 4528. Núm. 2. 2007. Pag. 479- 489

[Linares07b] A. Linares-Barranco, M. Oster, D. Cascado-Caballero, G. Jimenez, A. Civit, B. Linares Barranco. "Inter-Spike-Intervals Analysis of Aer Poisson-Like Generator Hardware". Neurocomputing . Vol. 70. Núm. 16-18. 2007. Pag. 2692-2700

[Linares07c] A. Linares Barranco, F. Gomez Rodriguez, A. Jimenez-Fernandez, T. Delbruck, P. Lichtsteiner. "Using FPGA for Visuo-Motor Control With a Silicon retina and a Humanoid Robot". IEEE International Symposium on Circuits and Systems, ISCAS 2007.

[Linares08] A. Linares-Barranco, J. L. Sevillano, M. S. Obaidat, N. Ferrando, J. Cerda, D. Cascado, G. Jimenez, A. Civit. "AER filtering using GLIDER: VHDL cellular automata description" IEEE International Conference on Electronics, Circuits and Systems, 2008. ICECS 2008.

[Linares09] A. Linares-Barranco, R. Paz, F. Gómez-Rodríguez, A. Jiménez, M. Rivas, G.Jiménez, A. Civit. "FPGA Implementations comparison of Neuro-Cortical Inspired Convolution Processors for Spiking Systems". International Work-Conference on Artificial Neural Networks, Lecture Notes in Computer Science, Volumen 5517/2009, Pag. 97-105.

[Liu02] Shih-Chii Liu, Tobias Delbruck, Jorgene Kramer, Giacomo Indiveri, Rodney Douglas, "Analog VLSI: Circuits and Principles". MIT press 2002.

[Liu04] S. C. Liu, R. Douglas, "Temporal coding in a network of silicon integrate-and-fire neurons". IEEE Transactions on Neural Networks, Vol. 15, pags 1305-1314, 2004

[Lujan07a] C. D. Luján Martínez, A. Linares Barranco, M. Rivas Pérez, A. Jiménez Fernández, G. Jimenez Moreno, A. Civit Balcells, "Multi-Task Implementation for Image Reconstruction of an Aer Communication". Lecture Notes in Computer Science. Vol. 4507. Núm. 1. 2007. Pag. 717-724.

[Lujan07b] C. D. Luján Martínez, A. Linares Barranco, M. Rivas Pérez, A. Jiménez Fernández, G. Jimenez Moreno, A. Civit Balcells, "Spike Processing on an Embedded Multi-Task Computer: Image Reconstruction. Proceedings of the 5th International Workshop on Intelligent

Solutions in Embedded Systems". 5th International Workshop on Intelligent Solutions in Embedded Systems, Wises07 . Leganés, Madrid 2007.

[Lyon88] Lyon, R.F.; Mead, C., "An analog electronic cochlea". Acoustics, Speech and Signal Processing, IEEE Transactions onVolume 36, Issue 7, July 1988 Page(s):1119 - 1134

[Maass99] Wolfgang Maass, Christopher M. Bishop: "Pulsed Neural Networks". The MIT press, 1999

[Mahowald92] M. Mahowald, "VLSI Analogs of Neural Visual Processing: A Synthesis of Form and Function". Ph.D. Thesis, California Institute of Technology, Pasadena CA, 1992.

[Mahowald94] Misha A. Mahowald, "An Analog VLSI System for Stereoscopic Vision", Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston, MA, pp. 215, 1994.

[Mandal09] Mandal, S.; Zhak, S.M.; Sarpeshkar, R., "A Bio-Inspired Active Radio-Frequency Silicon Cochlea " Solid-State Circuits, IEEE Journal of Volume 44, Issue 6, June 2009 Pags. 1814 - 1828

[MATLAB] The MathWorks - *MATLAB* and Simulink for Technical Computing: http://www.mathworks.com/

[Maxfield04] Clive Maxfield, "The Desing Warrior's Guide To FPGAs: Devices, Tools and Flows". Newnes 2004.

[Maxon] Maxon Motors: http://www.maxonmotor.es/

[Mead89] Carver Mead, **"**Analog VLSI and neural systems". Addison-Wesley Publications 1989.

[Michalewicz99] Zbigniew Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs". Springer-Verlag 1999.

[Miro06] L. Miró-Amarante, A. Jiménez-Fernández, A. Linares-Barranco, F. Gomez-Rodriguez, R. Paz-Vicente, G. Jimenez-Moreno, A. Civit-Balcells, "An LVDS Serial AER LINK". IEEE International Conference on Electronics, Circuits and Systems. ICECS 2006. Pags. 938-941. Francia

[Miro07] Lourdes Miró Amarante, A. Jimenez, Alejandro Linares Barranco, Francisco Gomez Rodriguez, RafaelPaz Vicente, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, Rafael Serrano Gotarredona, "LVDS Serial AER LINK Performance". IEEE International Symposium on Circuits and Systems, ISCAS2007, Pags. 1537-1540. Estados Unidos

[Mitra93] Sanjit Mitra, James Kaiser, "Handbook for Digital Signal Processing". Wiley-interscience publication, 1993.

[Molina08] Molina Vilaplana J., Lopez Coronado J. "A neural network architecture for progressive learning of robotic grasps". 11th International Conference on Climbing and Walking Robots and Supporting Tecnologies for mobile Machines. Advances in Climbing and Walking Robots (Clawar-2008), 2008, Pags. 120-126, Septiembre 2008.

[Moore65] Gordon Moore, "Cramming more components onto integrated circuits". Electronics Magazine 38, Vol. 8, Pags 114-117, 1965.

[Morgado04] Arturo Morgado Estevez, "Analisis y modelado de sistemas pulsantes bioinspirados basados en buses de altas prestaciones: Bus AER". Tesis Doctoral, Sevilla 2004.

[Ogata96] Katsuhiko Ogata, "Sistemas De Control Tiempo Discreto". Pearson Educación 2003

[Ogata04] Karsuhiko Ogata, "Ingenieria De Control Moderna". Pearson Educación 2003.

[Ollero01] Aníbal Ollero, "Robótica; manipuladores y robots móviles". Marcombo 2001

[Oppenheim98] Alan V. Oppenheim,Alan S. Willsky,S. Hamid Nawab, "Señales y sistemas". Prentice-Hall International 1998.

[Oster07] M. Oster, R. Douglas, L. Shih-Chii Liu, "Quantifying Input and Output Spike Statistics of a Winner-Take-All Network in a Vision System" IEEE International Symposium on Circuits and Systems. ISCAS 2007. Pags. 853 – 856.

[Pallás99] Ramón Pallás-Areny, John Webster, "Analog Signal Processing". Wiley-Interscience Publication 1999.

[Patterson09] David Patterson, Jhon Hennessy, "Computer Organization and Design: The hardware / software interface, 4th Edition". Morgan Kaufmann Publisher 2009.

[Paz05] R. Paz , F. Gomez-Rodriguez , M.A. Rodriguez , A. Linares-Barranco , G. Jimenez, A. Civit,"Test Infrastructure for Address-Event-Representation Communications". International Work-Conference on Artificial Neural Networks, Lecture Notes in Computer Science, Vol. 3512/2005. Pags. 518-526

[Paz06] R. Paz-Vicente, A. Linares-Barranco, D. Cascado, S. Vicente, G. Jimenez, A. Civit. "PCI-AER interface for Neuro-inspired Spiking Systems". IEEE International Symposium on Circuits and Systems 2006. ISCAS 2006.

[Paz08] R. Paz-Vicente, A. Jiménez-Fernández, A. Linares-Barranco, G. Jimenez-Moreno, F-Gomez-Rodriguez, L- Miró-Amarante, A. Civit. "Image Convolution Using a Probabilistic Mapper on Usb-Aer Board". IEEE International Symposium on Circuits and Systems. ISCAS 2008

[Paz09a] Rafael Paz Vicente, "Una aportación al procesamiento de la información visual mediante técnicas bioinspiradas". Tesis doctoral, Sevilla 2009

[Paz09b] R. Paz-Vicente, A. Linares-Barranco, A. Jimenez-Fernandez, G. Jimenez-Moreno, A. Civit-Balcells."Synthetic retina for AER systems development". International Conference on Computer Systems and Applications, AICCSA 2009

[PCM1804] PCM1804: Full Differential Analog Input 24-Bit, 192-kHz Stereo A/D Converter. http://focus.ti.com/lit/ds/symlink/pcm1804.pdf

[Penrose89] Roger Penrose, "The Emperor's New Mind: Concerning Computers, Minds and The Laws of Physics". Oxford University Press 1989

[Perez08] Perez-Carrasco, J.A.; Serrano-Gotarredona, T.; Serrano-Gotarredona, C.; Acha, B.; Linares-Barranco, B.; "High-speed character recognition system based on a complex

hierarchical AER architecture" IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008. Mayo 2008. Pags, :2150 – 2153.

[Philipp04] R. Philipp and R. Etienne-Cummings, "A Single Chip Stereo Vision System". Analog Integrated Circuits and Signal Processing Journal, Vol. 7, pags. 703-712, Julio 2004.

[Rakic88] P. Rakic, "Specification of cerebral cortical areas". Science 241 Vol. 4862, Pags.: 170–176, 1988.

[Rajago97] R. Rajagopalan. "A Generic kinematic formulation for wheeled mobil robots". Journal of Robotics Systems, 14(2), Pags. 77-91, 1997.

[Rivas07] M. Rivas-Pérez, A. Jiménez-Fernández, C. D. Luján-Martínez, A. Linares-Barranco, R. Paz-Vicente, G. Jimenez-Moreno. "An Aer Based Actuators Controller". Actas del II Simposio de Inteligencia Computacional. Simposio de Inteligencia Computacional. Zaragoza 2007.

[Rousselet02] G. A. Rousselet, M. Fabre-Thorpe, and S. J. Thorpe, "Parallel processing in high level categorisation of natural images" Natural Neuroscience, Vol. 5, pags.: 629-641, 2002.

[Rousselet03] G. A. Rousselet, M. J. Mace, and M. Fabre-Thorpe, "Is it an animal? Is it a human face? Fastprocessing in upright and inverted natural scenes". Journal of Neural Vision Modeling, Vol. 3, pags.: 440-455, 2003.

[Rummer69] Dale I. Rummer, " Introduction to analog computer programming". 1969- Holt, Rinehart and Winston, Inc – USA.

[Schaik97]André Van Schaik, "Analogue VLSI building blocks for an electronic auditory pathway". Tesis Doctoral, Swiss Federal Institute of Technology, Lausanne 1997.

[Schaik05] van Schaik, A.; Shih-Chii Liu, "AER EAR: a matched silicon cochlea pair with address event representation interface". Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on 23-26 May 2005 Pags:4213 - 4216 Vol. 5

[Serrano-Gotarredona99] T. Serrano-Gotarredona, A. G. Andreou, B. Linares-Barranco. "AER Image Filtering Architecture for Vision-Processing Systems". IEEE Transactions on Circuits and Systems. Fundamental Theory and Applications, Vol. 46, N0. 9, September 1999.

[Serrano05] R. Serrano-Gotarredona , M. Oster, P.. Lichtsteiner, A. Linares-Barranco, R. Paz, F. Gomez-Rodriguez, H. Kolle Riis, T. Delbrück, S.C. Liu, S. Zahnd, A.M. Whatley, R. Douglas, P. Häfliger, G. Jimenez, A. Civit, T. Serrano-Gotarredona, A. Acosta, B. Linares-Barrancoet "AER Building Blocks for Multi-Layer Multi-Chip Neuromorphic Vision Systems". Neural Information Processing Systems 2005, NIPS 2005.

[Serano07]R. Serrano-Gotarredona, L. Camuñas-Mesa, T. Serrano-Gotarredona, J. A. Leñero-Bardallo, and B. Linares-Barranco, "The Stochastic I-Pot: A Circuit Block for Programming Bias Currents" IEEE Trans. Circuits and Systems, Part-II: Brief Papers, vol. 54, No. 9, Pags. 760-764, Septiembre 2007.

[Serrano08] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, C. Serrano-Gotarredona, J. A.Pérez- Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jiménez-Moreno, and A. Civit-Ballcels,"On Real- Time AER 2D Convolutions Hardware for Neuromorphic Spike Based Cortical Processing". IEEE Trans. Neural Networks, June 2008.

[Serrano09] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas, T. Delbrück, S.C. Liu, R. Douglas, P. Häfliger, G. Jiménez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jiménez, B. Linares-Barranco, "CAVIAR: A 45k-neuron, 5M-synapse AER Hardware Sensory-Processing-Learning-Actuating System for High-Speed Visual Object Recognition and Tracking". IEEE Trans. on Neural Networks, Volume 20, Issue 9, Sept. 2009 Pags.: 1417 - 1438

[Shadlen94] M. Shadlen, W. T. Newsome, "Noise, Nerual Codes and Cortical Organization", Current Opinion in Neurobiology, vol. 4, pp. 569-579, 1994.

[Shepherd90] G. M. Shepherd, "The Synaptic Organization of the Brain". Oxford University Press, 3rd Edition, 1990.

[Silviotti91] M. Silviotti, Wiring Considerations in analog VLSI Systems with Application to Field-Programmable Networks, Ph.D. Thesis, California Institute of Technology, Pasadena CA, 1991.

[SIMULINK] Simulink - Simulación y diseño basado en modelos:
http://www.mathworks.es/products/simulink/

[Soren04] Poulsen Soren, Michael A. E. Andersen, "Single conversion audio amplifier and DC-AC converters with high performance and lowcomplexity control scheme". 35th Annual IEEE Power Electronics Specialists Conference. PECS 2004.

[SYSGEN] "System Generator for DSP".
http://www.xilinx.com/support/sw_manuals/sysgen_gs.pdf

[SPARTAN3] "Spartan-3E FPGA Family: Data Sheet".
http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[Telluride] Neuromorphic Engineering Workshop:
http://www.ine-web.org/telluride-conference-2009/telluride-2009/index.html

[Thorpe01] S. J. Thorpe, A. Delorme, and R. VanRullen, "Spike-based strategies for rapid processing". Neural Networks, 14:715-725, 2001.

[TMS320C672x] TMS320C672x Floating-Point Digital Signal Processors Series Datasheet:
http://www.ti.com/lit/gpn/tms320c6727

[Varona07] Varona Moya S., Molina Vilaplana J., Linares Barranco A., Feliu Batlle J., Lopez Coronado J. "Biologically inspired architecture for control of grasping movement of an anthropomorphic gripper". 10rd International Conference on Climbing and Walking Robots and Supporting Tecnologies for mobile Machines. Advances in Climbing and Walking Robots, Clawar-2007, 2008, pp. 533-540.

[VIRTEX2] "Virtex-II Platform FPGAs: Complete Data Sheet".
http://www.xilinx.com/support/documentation/data_sheets/ds031.pdf

[Vogelstein06] R.J. Vogelstein, R. Etienne-Cummings, N. Thakor and A. Cohen, "Phase-Dependent Effects of Stimulation of the Spinal Central Pattern Generator for Locomotion". IEEE Trans. Rehabilitation Engineering, Vol. 14, No. 3, pp.257-265, Septiembre 2006.

[Vogelstein08] R. Jacob Vogelstein, Francesco Tenore, Lisa Guevremont, Ralph Etienne-Cummings, and Vivian K. Mushahwar, "A Silicon Central Pattern Generator Controls Locomotion *in vivo*". IEEE Transactions on Biomedical Circuits and Systems, vol. 2, no. 3, pp. 212-222, 2008

[Watts92] Watts, L.; Kerns, D.A.; Lyon, R.F.; Mead, C.A. "Improved implementation of the silicon cochlea". IEEE Journal of Solid-State Circuits, Volume 27, Issue 5, May 1992 Pags.: 692 - 700

[Watts93] L.Watts, "Cochlear mechanics: Analysis and analog VLSI". Ph.D. dissertation, California Inst. Technol., Pasadena, 1993.

[Wen09] Bo Wen; Boahen, K. "A Silicon Cochlea With Active Coupling". Biomedical Circuits and Systems, IEEE Transactions on Volume 3, Issue 6, Dec. 2009 Pags. 444 – 455

[Westerman97] Wayne C. Westerman, David P. M. Northmore, John. G. Elias, "Neuromorphic Synapses for Artificial Dendrites". Analog Integrated Circuits and Signal Processing, Kluwer Academic Publishers, Vol. 13, no. 1, Pags. 167-184, Mayo 1997.

[Williams81]Williams, Arthur "Electronic filter design handbook". McGraw-Hill, 1981.

[Wong06] Wong, C.K.; Leong, P.H.W. "An FPGA-Based Electronic Cochlea with Dual Fixed-Point Arithmetic**".** International Conference on Field Programmable Logic and Applications, FPL 2006.

[Yang06] Yang Z., Murray, A. Worgotter, F. Cameron, K. Boonsobhak, V. **"**A neuromorphic depth-frommotion vision model with STDP adaptation". IEEE Trans. Neural Networks, vol. 17, No. 2, pp. 482-495, 2006.

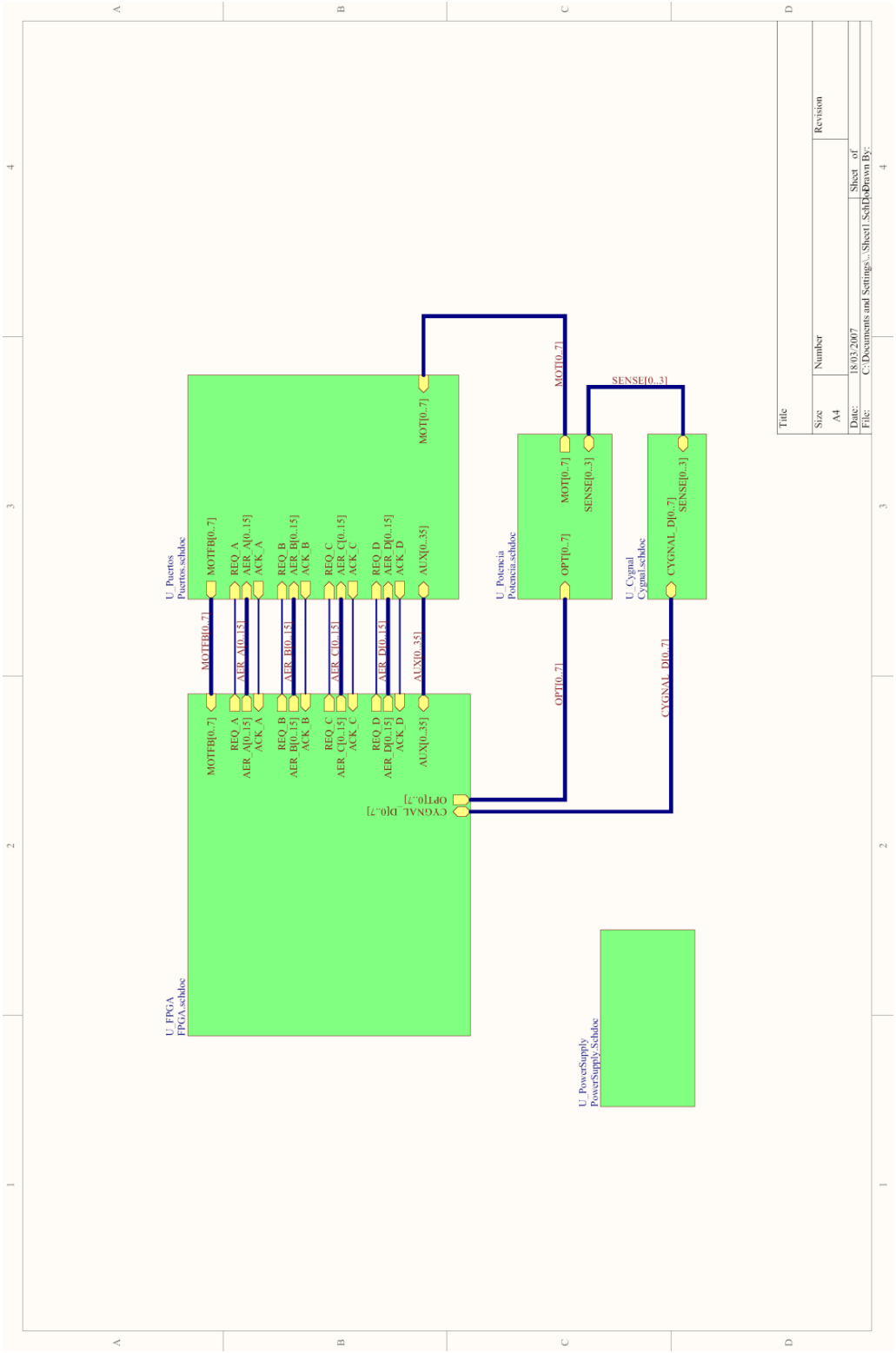
[Yu09] Yu, T.; Schwartz, A.; Harris, J.; Slaney, M.; Shih-Chii Liu, "Periodicity detection and localization using spike timing from the AER EAR". IEEE International Symposium on Circuits and Systems, ISCAS 2009. Pags.: 109 - 112

[Zaghloul04] K. A. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip: Part I and II". IEEETrans. Biomed. Eng., vol. 51, no. 4, pp. 657-675, Apr. 2004.

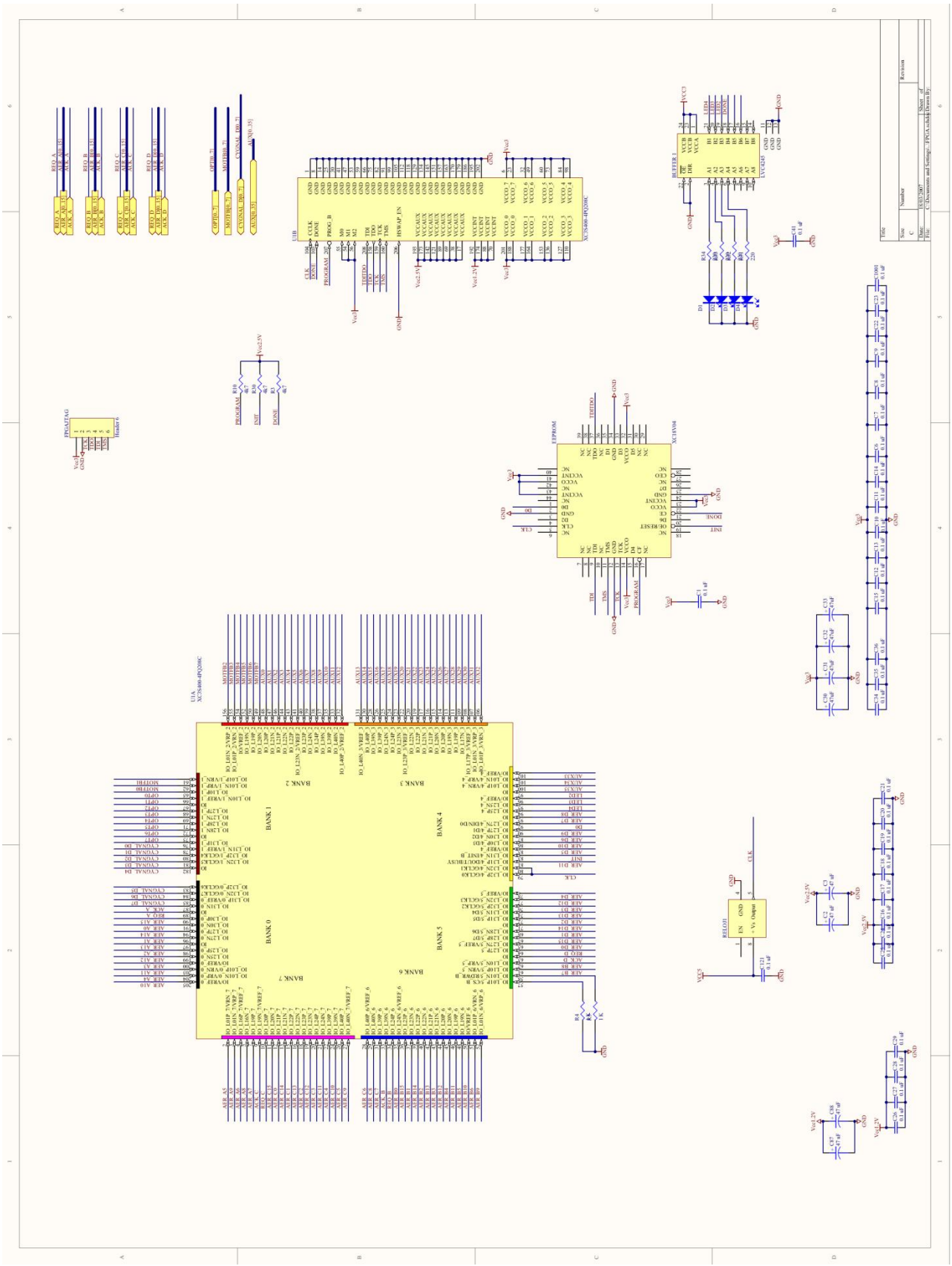
[Zamarreño08] C. Zamarreño-Ramos, R. Serrano-Gotarredona, T. Serrano-Gotarredona, and B. Linares-Barranco, "LVDS interface for AER links with burst mode operation capability". IEEE International Symposium on Circuits and Systems, ISCAS 2008, Pags. 644-648.

11. Apéndice: Esquemáticos de la AER-Robot

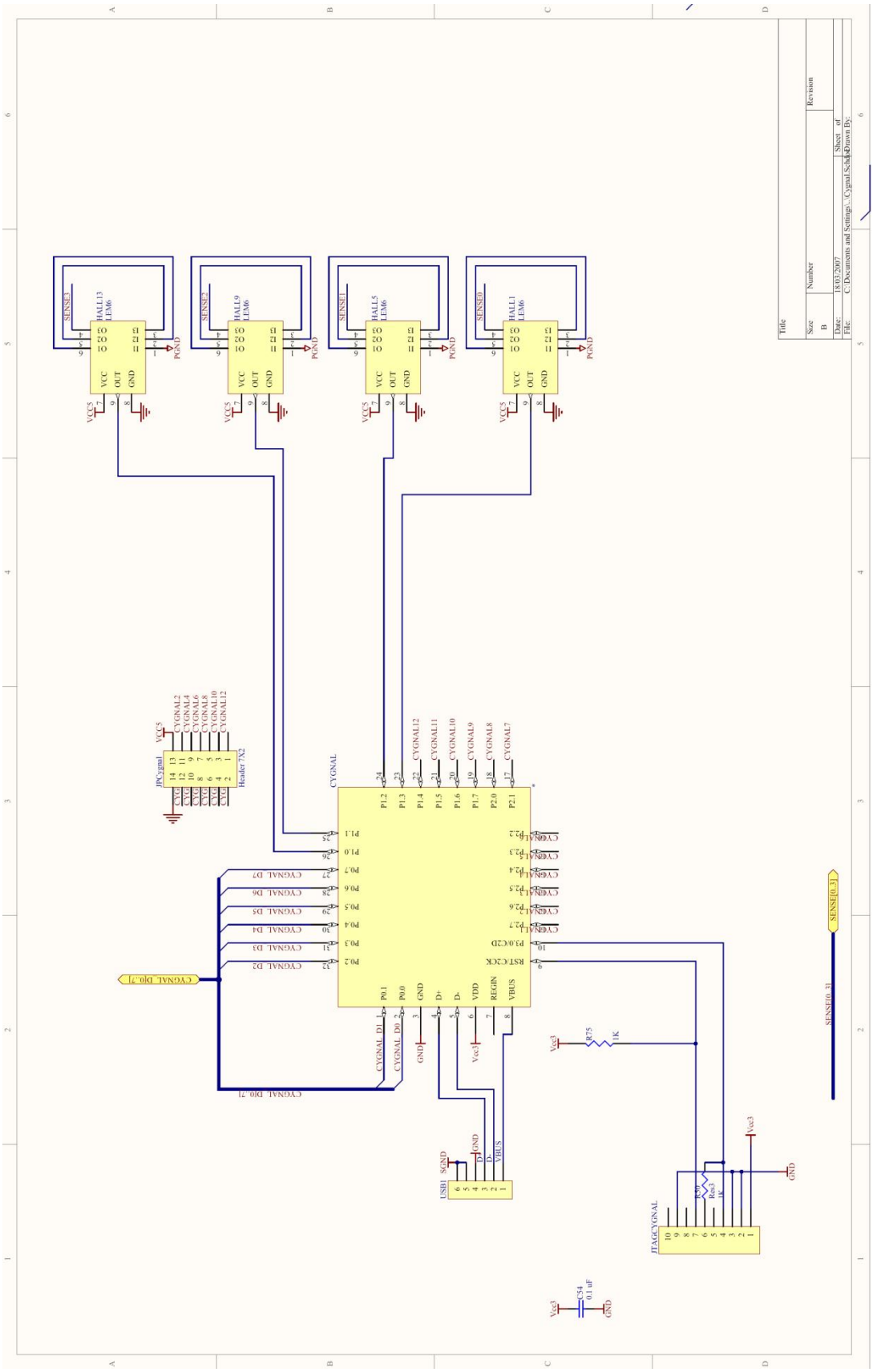
11.1. Esquemático: *AER-RobotTopLevel.SchDoc*



11.2. Esquemático: *FPGA.SchDoc*

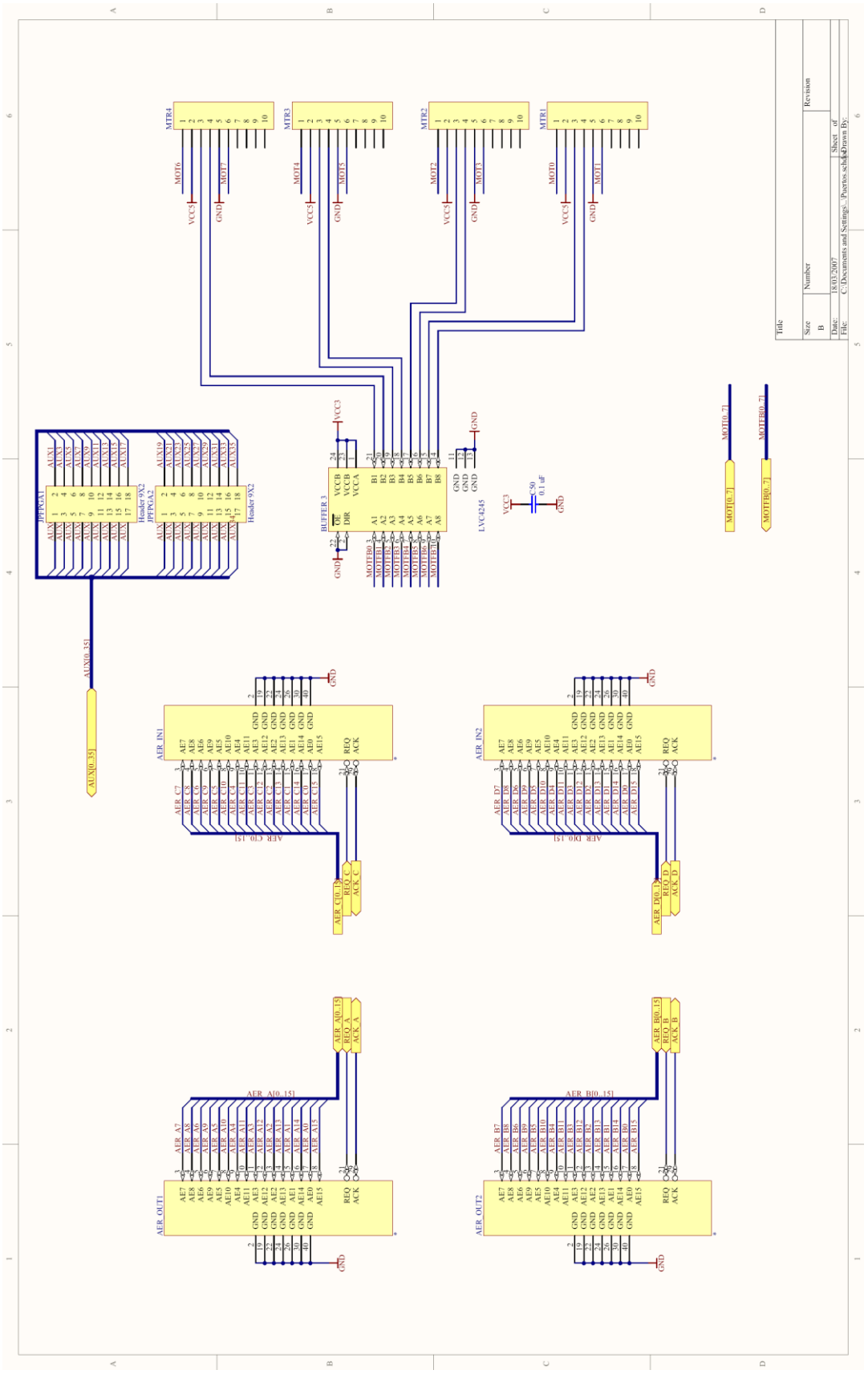


11.3. Esquemático: *8051.SchDoc*

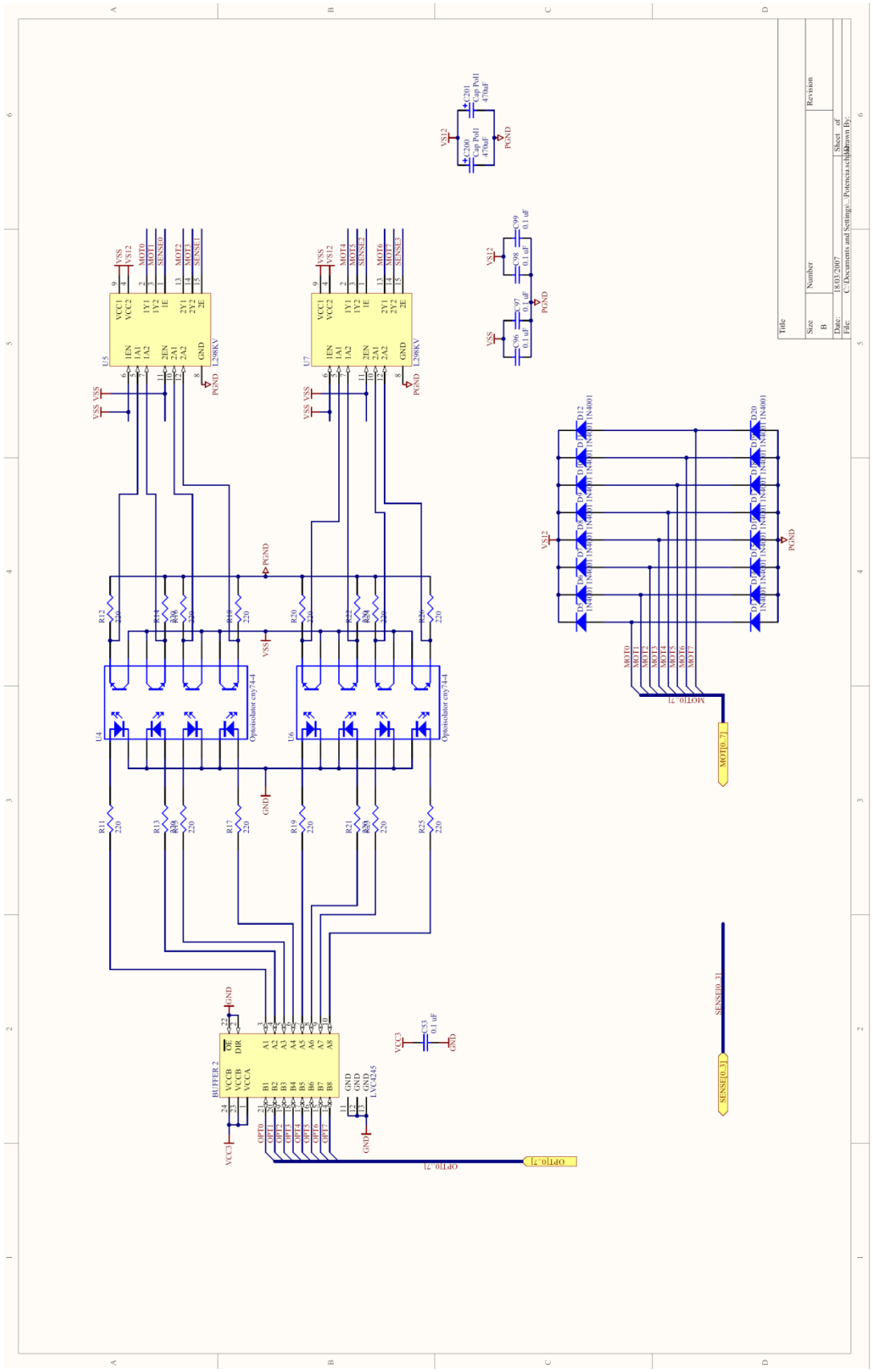


Departamento de Arquitectura y Tecnología de Computadores
Universidad de Sevilla

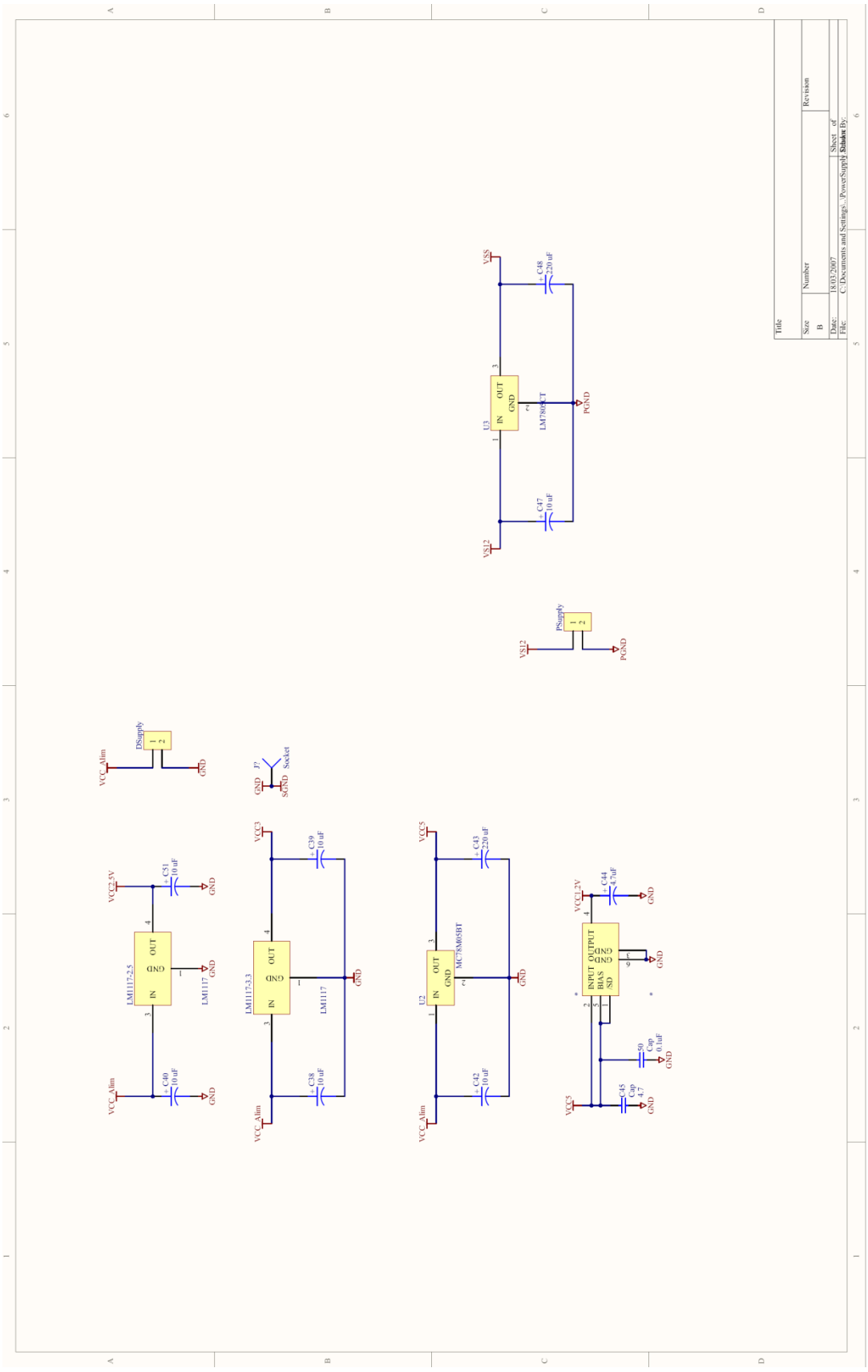
11.4. Esquemático: *Puertos E/S.SchDoc*



11.5. Esquemático: *Potencia.SchDoc*



11.6. Esquemático: *PowerSupply.SchDoc*



11.7. PCB: *AER-ROBOT.BrdDoc*