TRABAJO DE FIN DE GRADO
GRADO EN FÍSICA

Algoritmos para la conformación de pulsos en tiempo real de detectores de radiación

Autor:
Juan Pellejero Solans

Directores:
Santiago Celma Pueyo
Guillermo Díez Señorans

Facultad de Ciencias
Zaragoza, 26 de junio de 2022

Resumen

En este Trabajo Fin de Grado se ha realizado tanto un análisis teórico como un estudio experimental de sistemas digitales de procesado de señales con aplicación a sistemas de detección de radiación para espectroscopía de rayos gamma.

La parte teórica se ha centrado en las bases de dicho procesado aplicado a sistemas de detección de radiación. Se han estudiado también las señales en cada una de sus fases. De cara al estudio experimental, se ha entrado en un estudio de los conformadores analógicos de pulsos del tipo CR-RC^m, y en una implementación de la cancelación de polo cero en estos conformadores. Además de esto, se ha estudiado la metodología y el flujo de diseño necesario para implementar uno de estos conformadores en una matriz de puertas programables FPGA.

La parte experimental ha consistido en la implementación en FPGA de conformadores digitales de pulsos CR-RC^m con cancelación de polo cero, basados en sus equivalentes analógicos. Tras la implementación, se han expuesto los resultados obtenidos, y extraído conclusiones de los mismos.

Listado de acrónimos

Acrónimo	Significado
AA	Anti-Aliasing
ADC	Analog-to-Digital Converter
BD	Ballistic Deficit
DSP	Digital Signal Processor
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HV	High Voltage
IIR	Infinite Impulse Response
LTI	Linear time-invariant
LUT	Look Up Table
PCB	Printed Circuit Boards
PZC	Pole Zero Cancellation
RAM	Random Access Memory
RTL	Register Transfer Level
SDK	System Development Kit
SNR	Signal to Noise Ratio
SoC	System on Chip
UART	Universal Asynchronous Receiver-Transmitter
VHDL	Very high speed Hardware Description Language

Índice

1. Introducción

En el mundo moderno hay una casi interminable variedad de disciplinas científicas y tecnológicas que precisan del conocimiento de sistemas analógicos y digitales. La física nuclear no es una excepción. Hoy en día usamos esta física para aplicaciones médicas, energéticas, medioambientales y un largo etcétera. En el Grado en Física hemos estudiado parte del comportamiento físico de estos sucesos y su detección. Los detectores de radiación son, por supuesto, algo esencial en el estudio de la física nuclear, y en este trabajo se profundizará en el estudio de las señales que generan estos dispositivos y su posterior procesado. Las señales de salida de los detectores deben ser tratadas para extraer de ellas información optimizada de la radiación incidente. Esto requiere de un procesado adecuado para obtener de las señales información útil: energía de radiación, tiempo de ocurrencia, tasa de incidencia, etc.

Junto al avance de la tecnología digital y la consecuente mejora de la instrumentación, el procesamiento digital de las señales ha ganado popularidad frente al analógico. El desarrollo de convertidores analógico-digital (ADC), es cada vez más rápido y económico, y ha llevado a una preeminencia de la aproximación digital sobre la analógica [6]. Los conocimientos aprendidos en el Grado nos permitirán aproximarnos al procesado de señales desde esta perspectiva, comprendiendo al mismo tiempo el tratamiento analógico en el que se basan muchos de los conformadores de pulsos digitales. Con esto en mente, se ha decidido trabajar con conceptos electrónicos ya aprendidos, tales como los integradores y diferenciadores, para estudiar una de sus aplicaciones en este campo. El procesado digital de señales de alta velocidad ha sido tradicionalmente implementado usando coprocesadores matemáticos, o los denominados procesadores digitales de señales (DSPs). Sin embargo, en este trabajo, se empleará una FPGA (*Field Programmable Gate Array*) para el procesado de señales.

Una FPGA puede ser usada en una gran variedad de aplicaciones, desde dispositivos de comunicación hasta sistemas en chip (SoC) completos que se asemejan a un pequeño computador, pero con rendimientos de cómputo mucho más elevados. Esta versatilidad viene acompañada de una sencilla reprogramabilidad, que permite al diseñador definir y cambiar las funciones del circuito mediante programación. Por esto, no es de extrañar que los sistemas basados en FPGA se hayan convertido en una línea preferente para muchas aplicaciones científicas de instrumentación, o como es nuestro caso, para los sistemas de detección de radiación en tiempo real [6].

Con esto en mente, se desarrollará un conformador de pulsos digital en uno de estos sistemas FPGA a partir de conformadores de pulsos empleados en equipos analógicos tales como los filtros semi-gaussianos CR-RC$^{\mathrm{m}}$ y sus variantes.

1.1. Objetivos del trabajo

La idea inicial de este trabajo es implementar en una FPGA un conformador de pulsos digital que pueda operar en tiempo real simulando un filtro CR-RC$^{\mathrm{m}}$ analógico para sistemas de detección de radiación. Para hacer esto deberemos desarrollar una serie de algoritmos a partir del filtro analógico e implementarlos en un diseño a nivel de hardware en la FPGA. En el proceso de aprendizaje se contempla un estudio de los fundamentos del procesado digital de señales y la adquisición de conocimientos de las herramientas específicas para la conformación de pulsos en FPGA.

1.2. Herramientas utilizadas

- **Vivado:** Software de AMD Xilinx para el sintetizar e implementar diseños en HDL (*Hardware Description Language*). Incluye aplicaciones para el diseño de sistemas en chip y síntesis de alto nivel.
- **Verilog:** Lenguaje de descripción de hardware usado por Vivado para describir la estructura y el comportamiento de circuitos lógicos en FPGA. El HDL consiste de expresiones y declaraciones que se parecen a lenguajes que ya hemos estudiado como C.
- **HLS:** Síntesis de alto nivel. Proceso de diseño automatizado que traslada unas instrucciones en un lenguaje de alto nivel (C/C++) a un lenguaje de descripción de hardware. A grandes rasgos traslada algoritmos digitales a una estructura de hardware que realiza lo mismo que el código digital.
- **Arduino UNO CH340:** Micro-controlador Arduino que junto a electrónica analógica periférica nos ayudará a replicar un sistema de detección de eventos a la entrada del conformador.
- **C++:** Lenguaje de programación usado para programar los micro-controladores Arduino.
- **Placa PYNQ-Z2:** Placa de desarrollo PCB fabricada por TUL®. En esta placa se montan diferentes dispositivos entre los que se encuentra un chip Zynq 7000.
- **Chip Zynq 7000:** SoC desarrollado por Xilinx. Contiene dos elementos independientes: Un microprocesador ARM Cortex-9 y una FPGA modelo Artix. Esta FPGA recibe instrucciones desde el ordenador mediante un puerto serie UART (*Universal Asynchronous Receiver-Transmitter*).

2. Procesado de señales

2.1. Formación y adquisición de pulsos

Este trabajo se va a centrar más en el procesado de las señales que en cómo se obtienen. Sin embargo, conviene tener clara la base de los detectores de radiación para un tratamiento adecuado de la señal. Nos limitaremos al procesado de señales para la espectroscopía de rayos gamma, ya que este es un campo donde el procesado digital de pulsos es ampliamente utilizado [3, 4], aunque muchas de las conclusiones son fácilmente generalizables a otros tipos de radiación ionizante.

En la actualidad, los detectores de radiación más usados para este tipo de espectroscopía son los detectores de semiconductores, especialmente los de germanio [1]. En vez de pares de iones que generan las cámaras de ionización de gas, el principal mecanismo de generación de pulso de corriente eléctrica en este tipo de detector se basa en la generación de pares electrón-hueco producidos por la interacción de la radiación. Esto se produce en energías tan bajas como 3 eV por lo que supone una ventaja frente al mecanismo de ionización. Los detectores de germanio están hechos en diferentes geometrías según la aplicación, siendo estas la planar y diferentes tipos de coaxiales. Su modelo equivalente es fundamentalmente de tipo capacitivo. La decisión de cuál elegir suele ser una cuestión relacionada con la sensibilidad y el tiempo de captura. El pulso que se obtiene con cada geometría tiene tan solo ligeras diferencias. Para el conformado de pulsos no nos interesan tanto los principios físicos de los detectores si no la forma de los pulsos de corriente generados.

En sistemas de espectroscopía de rayos gamma y otros detectores de radiación, los pulsos de corriente tienen una forma generalmente abrupta y de muy corta duración. Estos valores tienen relación con los tiempos de desplazamiento para los portadores de carga en el semiconductor y se pueden calcular aplicando el formalismo de la física de los semiconductores. La información que nos interesa extraer es la energía de la radiación incidente, la cual es proporcional a la carga generada. Para esta sección nos basta con saber que la altura de los pulsos de corriente está por debajo del μA, y su anchura total ronda los 10 ns. Con la finalidad de aumentar la relación señal ruido de la señal y la duración del pulso, a la salida del detector, y antes del sistema conformador suele incluirse un pre-amplificador, que en estos casos es un amplificador sensible a la carga o un integrador. La entrada del conformador por lo tanto será la salida de este pre-amplificador.

El pre-amplificador es comúnmente usado para extraer la señal de salida del detector. De esta forma convertiremos unos impulsos de corriente estrechos y débiles en una señal lo suficientemente fuerte como para evitar problemas de ruido. También resultará en una salida más limpia. El perfil de los pulsos de salida de un pre-amplificador suele tratarse de voltajes escalón con amplitud proporcional a la carga generada, y con un decaimiento constante.

En este acople detector pre-amplificador, R será la resistencia de polarización, C el condensador de desacoplo y HV el voltaje de polarización. Este último es del orden de los 100 V.

Asumiendo la constante de tiempo de polarización RC muy elevada y la capacidad intrínseca del detector mucho más elevada que el condensador de *feedback* C_f, la tensión de salida en el

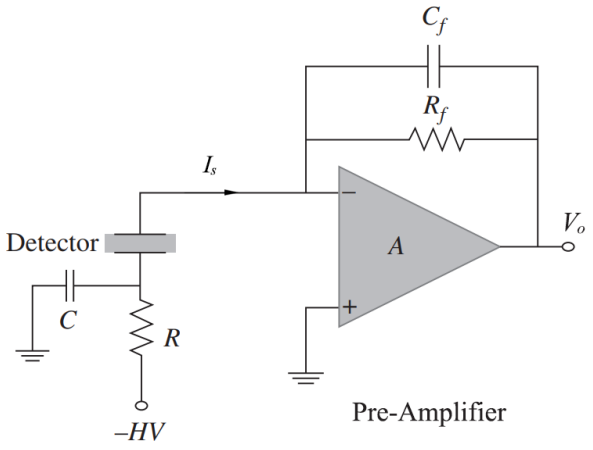

Figura 1: Montaje de detector y amplificador conectados en DC [2].

campo transformado puede expresarse como:

$$\frac{V_o}{I_s} = \frac{-R_f}{sR_fC_f + 1} = \frac{-R_f}{s\tau_o + 1} \tag{1}$$

Establecemos τ_o como la constante de tiempo del pre-amplificador, que marcará el tiempo de decaimiento de los pulsos. Los artículos consultados que tratan sobre la espectroscopía de rayos gamma [3, 4] establecen este tiempo de decaimiento en el orden de los microsegundos. El tiempo de subida t_s de estos pulsos escalón idealmente sería nulo. Sin embargo, tanto el detector como el pre-amplificador contribuyen a que exista, y será del orden de los nanosegundos. Este tipo de pulsos con un rápido tiempo de subida y un largo decaimiento se suelen llamar pulsos de cola [1].

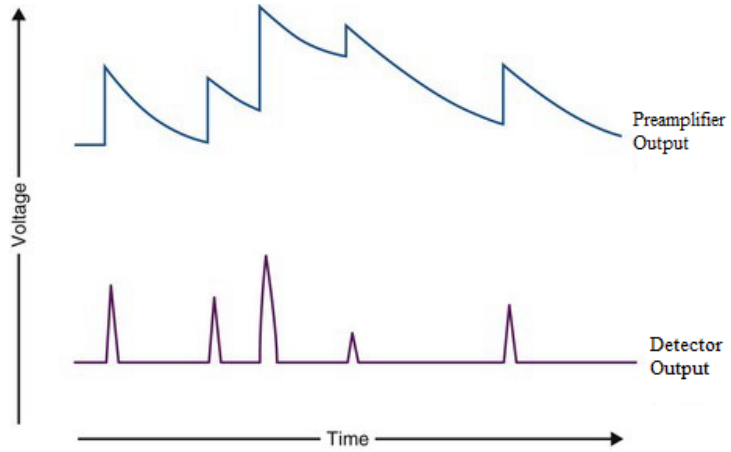

Figura 2: Perfil aproximado de la señal del pre-amplificador y del detector [5].

Habrá que tener en cuenta tanto t_s como τ_0 para procesar los pulsos y obtener un espectro de energía preciso.

2.2. Conceptos generales

Generalmente al operar con un detector de radiación buscaremos Q, la carga generada en el detector, la cual es proporcional a la energía de la radiación incidente. El procesado de señales

para detectores de radiación tiene como objetivo medir con precisión dicha carga. Para obtener un valor proporcional a Q, debemos integrar los impulsos de corriente con un integrador (pre-amplificador). Los pulsos del detector tienen un decaimiento de unas pocas decenas de nanosegundos [1]. Si estamos trabajando con detectores con una tasa de eventos alta, la consecuencia directa de los decaimientos del pre-amplificador, es un apilamiento de los pulsos sobre la cola del anterior. Es inmediato ver que al ser la amplitud un factor clave relacionado con la energía depositada en el detector, este apilamiento es algo que tiene que ser corregido. Este proceso tiene soluciones variadas y diferentes metodologías.

En este campo podemos distinguir entre un procesado analógico y uno digital. En este contexto, cuando hablamos de una señal analógica nos referimos a una función en el tiempo que puede tomar cualquier valor. Sucede lo mismo al referirnos a su procesado, que se refiere a un tratamiento analógico de las señales mediante dispositivos electrónicos. Todos los detectores de radiación generan pulsos analógicos. Debido a esto, en los sistemas digitales, antes del conformador, hay que digitalizar la señal. Como cabría esperar, no hay un sistema inherentemente mejor que otro, y usar el uno o el otro depende idealmente del caso particular.

En la figura (3) se puede ver un esquema del procesado analógico de señales. El conformador recibiría la salida del amplificador y obtendría un perfil optimizado usando electrónica analógica. Un conformador analógico puede ser recomendado para algunos detectores con respuestas extremadamente rápidas, o para consumos de potencia muy bajos

Figura 3: Esquema de un sistema de procesado analógico de pulsos.

2.3. Procesado digital de señales (DSP)

Un sistema DSP (*Digital Signal Processing*) opera con señales mediante un conjunto de reglas y operaciones determinadas. En la figura (4) se muestra un esquema del procesado digital de señales aplicado a la conformación de pulsos en sistemas de detección de radiación. El sistema tendrá que digitalizar con un ADC la señal analógica procedente del detector, que será un *array* *x[n]* de longitud N replicando la señal analógica muestreada y cuantificada. Una vez tiene la señal digitalizada *x[n]* opera sobre esta para conformar una nueva señal de salida *y[n]* con propiedades más adecuadas para la aplicación que se desee. Estas operaciones pueden realizarse en ordenadores programados para estas funciones o en hardware digital dedicado. Este último caso se refiere a algoritmos implementados directamente en silicio, donde las propias operaciones están fijadas por la estructura del hardware. En nuestro caso, utilizaremos una FPGA, necesitando programar el hardware interno y el proceso de intercambio de la información extraída con un ordenador.

El procesado digital es en general preferible al analógico en el caso de la espectroscopía, por

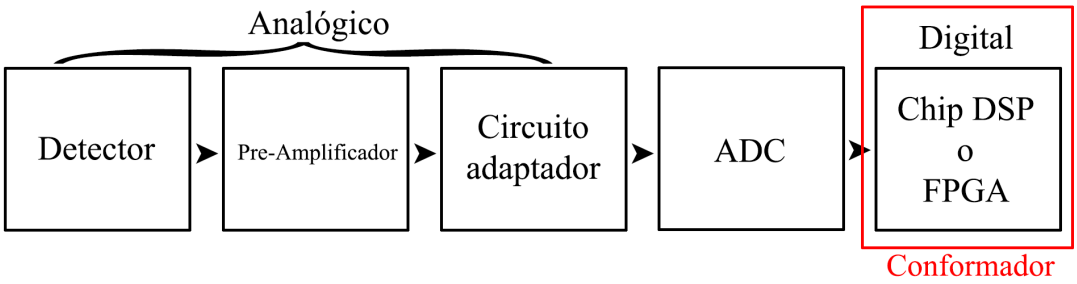

Figura 4: Esquema de un sistema de DSP clásico de espectroscopía.

las siguientes razones:

- **Precisión del conformador:** En un procesado analógico, los valores de los componentes pueden no estar disponibles, o pueden tener cierta tolerancia que habría que tener en cuenta. Si nos centramos en el digital, podemos establecer exactamente los parámetros del procesado.
- **Flexibilidad:** Uno puede cambiar el algoritmo o los parámetros para probar otra versión del sistema, mientras que en un conformador analógico habría que re-diseñarlo por completo cada vez que lo quisiéramos modificar.
- **Potencia:** A las anteriores ventajas se les añade también la sencillez de aplicación de operaciones complejas, que son difíciles de lograr con un procesado analógico, pero digitalmente sí que se pueden implementar. Por ejemplo, el filtrado no causal.
- **Análisis *offline*:** Tras discretizar la señal, el procesamiento digital nos permite elegir entre un procesado a tiempo real de la señal (*Online Processing*) y un procesado en diferido (*Offline Processing*) en el que almacenamos la señal para procesarla más tarde.

Aun con estas importantes ventajas, hay que tener en cuenta las complicaciones que tienen los sistemas DSP. Entre los más importantes están: los errores de cuantificación, los efectos del muestreo, la complejidad y potencial inestabilidad de algunos algoritmos y la limitada capacidad de almacenamiento y tiempo de cómputo. Por suerte muchos de estos problemas se pueden solventar con un ajuste adecuado del sistema.

El error de cuantificación puede llegar al punto de ser despreciable con una buena elección de digitalizador. En aplicaciones de espectroscopía los ADC empleados tienen un orden de 12-14 bits, y esto es más que suficiente para poder ignorar errores de cuantificación en la mayoría de los casos.

La frecuencia de muestreo puede ser escogida para que optimice la respuesta del sistema. Una de las bases de la digitalización de una señal es la frecuencia de muestreo del ADC. La elección de esta frecuencia viene dada por el teorema de muestreo de Nyquist-Shannon, que dice que una señal con componentes de máxima frecuencia f_{max} puede ser representada al completo por muestras equidistantes si la velocidad de muestreo es al menos $2f_{max}$. Cuando este criterio no se cumple, ocurre un fenómeno al que llamamos *aliasing*, que imposibilita reconstruir la señal original [2]. Debido a que no es siempre posible asegurar el criterio de Nyquist, se opta por poner un filtro pasa-baja previo al ADC que elimine las frecuencias altas que den problemas para la

condición (filtro *anti-aliasing*). En el caso de los sistemas de detección de radiación, debemos tener en cuenta que la forma de los pulsos será modificada (ensanchada) por este filtro previo. A cambio, las frecuencias por encima de la frecuencia de Nyquist se eliminarían hasta un nivel menor al ruido de cuantificación. También conviene comentar que se puede elegir una velocidad de muestreo mucho mayor a la frecuencia de Nyquist (*oversampling*). Esta solución ayuda a simplificar el filtro *anti-aliasing* y a minimizar el ruido de cuantificación, a costa de un mayor coste y consumo de potencia.

2.4. Filtros digitales

Los filtros digitales se separan en dos principales categorías, respuesta de pulso infinita (*Infinite Impulse Response* IIR) y respuesta de pulso finita (*Finite Impulse Response* FIR). Para elegir entre ellos se pueden seguir criterios como el número de operaciones aritméticas, requerimientos de memoria, y el efecto de precisión numérica.

Un filtro de respuesta de impulso finita FIR produce una salida de longitud finita para una excitación. Se trata de un filtro no recursivo. La salida dependerá únicamente de la entrada. No es de extrañar por lo tanto que estos filtros sean inherentemente estables. También cuentan con la ventaja de ser de fase lineal. Un ejemplo claro de un FIR es un diferenciador. Podremos expresar el diferenciador digital más simple como la ecuación en diferencias: `y[n]=x[n]-x[n-1]`. Otro filtro FIR ampliamente utilizado es el filtro de media móvil [2]. Pese a las ventajas de simplicidad y estabilidad, los filtros FIR tienen inconvenientes. La computación será más costosa en tiempo y en bloques de operación, compuestos principalmente por multiplicadores e integradores. Tampoco existen eccuaciones de forma cerrada para este tipo de filtros por lo que se tiene que recurrir a la iteración para cumplir con especificaciones fijadas.

La salida de los filtros IIR depende de valores previos de la señal de entrada así como de la señal de salida. Esta propiedad de recursividad permite una buena economía computacional, pero tiene algún inconveniente. Un filtro IIR puede convertirse en inestable y puede ser más sensible a errores de cuantificación dado su comportamiento recursivo. Una de las posibles soluciones para este problema es separar el diseño de los filtros con funciones de transferencia de orden elevado en combinaciones de filtros de menor orden. Esto es posible gracias a la propiedad asociativa y distributiva de la convolución en sistemas LTI (*Linear Time-Invariant*). Otra desventaja de los filtros IIR es que generalmente no pueden obtener una respuesta en fase lineal. La consecuencia de esto es un retraso de la señal dependiente de la frecuencia de la señal de entrada.

Un ejemplo de IIR es un filtro integrador digital. La integración es una operación de procesado de señales tan común como la diferenciación, y es algo que también hemos visto en el Grado. Podemos expresar la integración digital más simple como: `y[n]=x[n]+x[n-1]+x[n-2]+...` Por supuesto esto es más sencillo utilizando un algoritmo recursivo en la forma: `y[n]=y[n-1]+x[n]` en el que auto-referenciamos al anterior elemento de la función de salida

El diseño de muchos IIR tiene su fundamento en filtros analógicos que se han mostrado útiles para el procesado de señales [1, 2]. En el caso de este trabajo, utilizaremos un filtro digital basado en un conformador CR-RC^{m}[1].

2.5. Imperfecciones en el procesado de pulsos

Como se ha mencionado anteriormente, el principal motivo para procesar señales es eliminar los defectos de las mismas. Es por tanto indispensable conocer cada tipo de imperfección a la hora de intentar corregirla y de analizar los resultados. En el procesado digital podemos corregir y minimizar la deriva causada por el deterioro de los componentes, más propia de un procesado analógico, pero existen todavía imperfecciones que comparten ambos métodos. Explicaremos aquí tres tipos de defectos a evitar.

- **Apilamiento:** Eliminar el problema de apilamiento, o *pileup* es el desafío principal del procesado de pulsos con tasa de ocurrencia elevada. Un apilamiento de cola sucede cuando un pulso comienza en el decaimiento de uno o varios pulsos anteriores. De ocurrir esto, el pulso que se apila tendría una amplitud modificada ya que habría empezado desde un nivel diferente que el de la línea base. También puede suceder un apilamiento de dos pulsos más próximos en el tiempo, cuando la diferencia entre estos dos pulsos es menor que el tiempo de resolución del conformador. A este caso se le llama apilamiento de cabeza. En este caso particular el sistema verá a los pulsos como uno solo y no podrá interpretar correctamente la amplitud de ninguno. Este último caso es más problemático porque también afecta al número de eventos, aunque es mucho menos común, en especial en bajas velocidades de producción de pulsos. Para resolver problemas de apilamiento debemos de buscar conformadores que minimicen el tiempo de decaimiento del pulso de salida. De esta forma la mayoría de los pulsos podrán empezar en la línea base.

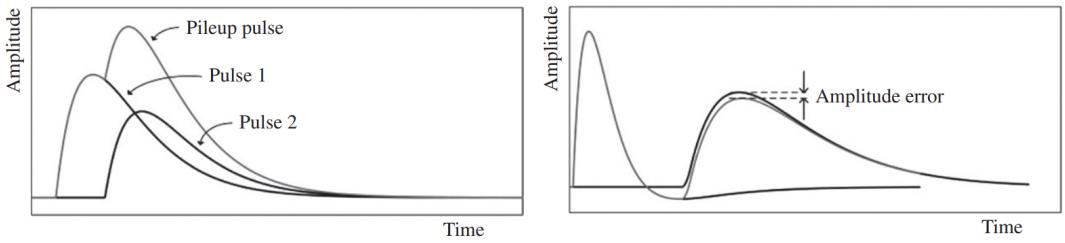

Figura 5: Apilamiento de cabeza (izquierda) y de cola (derecha) [2].

- **Déficit balístico:** Independientemente del perfil de tiempo original de la señal de origen, un conformador de pulsos debería proporcionar una altura de pulso directamente proporcional a la carga eléctrica generada. El conformado de pulsos, sin embargo, puede tener una dependencia con el tiempo de subida del pulso. De ocurrir esto existirían diferentes amplitudes para pulsos de misma amplitud original pero distinto tiempo de subida. Podemos considerar dos pulsos $U_o(t)$ y $U(t)$, considerando que el primero tiene tiempo de subida 0 y el segundo T finito. Las respuestas de estos pulsos en un conformador con déficit balístico serían $V_o(t_o)$ y $V(t_m)$ con $V_o(t_o) \neq V(t_m)$. Esto se puede observar en la figura (6). Definiríamos en este caso el déficit balístico como

$$\Delta V = V_o(t_o) - V(t_m) \tag{2}$$

Este defecto es un problema en detectores que produzcan pulsos de diferentes duraciones, ya que la pérdida de amplitud será variable con diferentes perfiles de tiempo de captura.

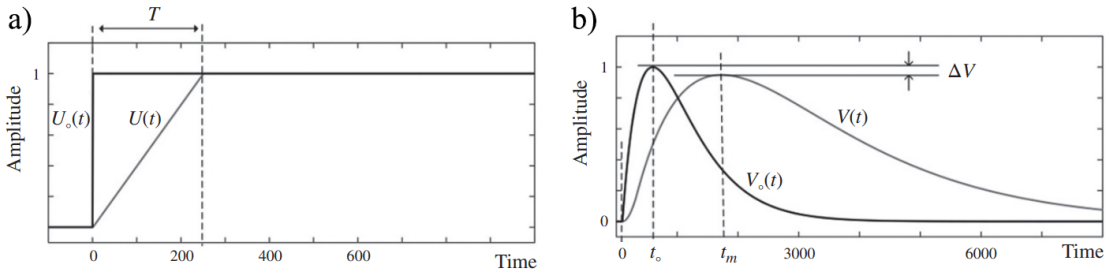

Figura 6: Déficit balístico frente a un tiempo de subida finito en el pulso [2]. La figura a es el pulso de entrada y la b, la respuesta de un filtro CR-RC a dichas entradas.

Podemos por último comentar que una posible solución para minimizar el déficit balístico es aumentar la escala temporal del filtro hasta que supere el tiempo de carga del detector. Esto desgraciadamente puede crear otros problemas de *pileup* y de menor SNR (*Signal to Noise Ratio*).

- **Fluctuaciones de línea base:** Al medir la amplitud de los pulsos sobre una línea base, debemos de asegurar que tras registrar un pulso, el nivel vuelve a la línea base original. Cuando el nivel de la línea base no es estable tendremos un problema similar al error de apilamiento. Al empezar los pulsos desde fuera de la línea base, la amplitud de los pulsos fluctúa y la resolución en energía se ve afectada. Otras causas de fluctuaciones de la línea base son debidas a variaciones de voltaje de continua y a derivas térmicas.

2.6. Conformador de pulsos CR-RC

Un conformador de pulsos es en esencia un sistema electrónico entrada-salida. En el caso de sistemas lineales e invariante temporales (LTI), queda caracterizado por una función de transferencia $H(s) = V_o(s)/V_s(s)$ donde $V_s(s)$ y $V_o(s)$ son respectivamente las transformadas de Laplace de las señales de entrada y salida.

En espectroscopía analógica es habitual construir conformadores de pulsos utilizando filtros simples RC de primer orden, como los mostrados en la figura (7).

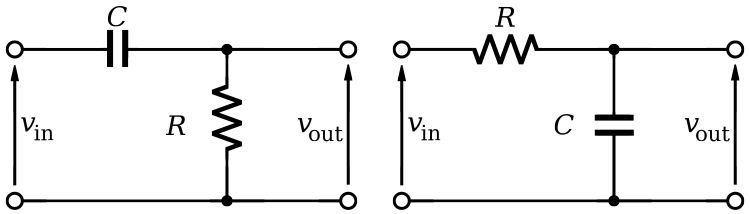

Figura 7: Filtros de primer orden CR (izquierda) y RC (derecha).

El filtro CR es un pasa alta y el RC es un pasa baja. Sus funciones de transferencia son:

$$H_{CR}(s) = \frac{sRC}{1+sRC} = \frac{s\tau}{1+s\tau} \qquad H_{RC}(s) = \frac{1}{1+sRC} = \frac{1}{1+s\tau} \tag{3}$$

Para las que se ha usado la misma constante de tiempo $\tau = RC$. A principios de los años 40 [2], el procesado de señales constaba esencialmente de un único filtro analógico CR, que es un (pseudo) derivador. Pronto se aplicó un RC (pseudo) integrador a la salida de este para

eliminar drásticamente el ruido de alta frecuencia. Nace aquí el concepto de un filtro CR-RC, que constituye el conformador más simple del procesado de señales en sistemas de detección de radiación. Entre cada elemento se debe incluir un amplificador de tensión, que en el caso mas simple tiene ganancia unitaria. La función de transferencia del filtro CR-RC se puede calcular de forma simple con la función de transferencia de ambos.

$$H_{\mathrm{CR-RC}}(s) = \frac{s\tau}{(1+s\tau)^2} \tag{4}$$

Llamaremos aquí a la constante de tiempo $\tau = RC$ el *shaping time*, o *peaking time*, ya que el pulso a la salida alcanza su máximo en este tiempo. Una elección de la misma τ para CR y RC en un filtro CR-RC muestra la mejor respuesta señal-ruido (SNR) [1]. Por esta razón consideraremos que tienen la misma τ de aquí en adelante si hablamos de un filtro CR-RC.

La respuesta de pulso de un filtro CR-RC queda representada en la figura (8). El pulso toma su máximo en el *peaking time* τ, y tras ello muestra una larga cola de decaimiento.

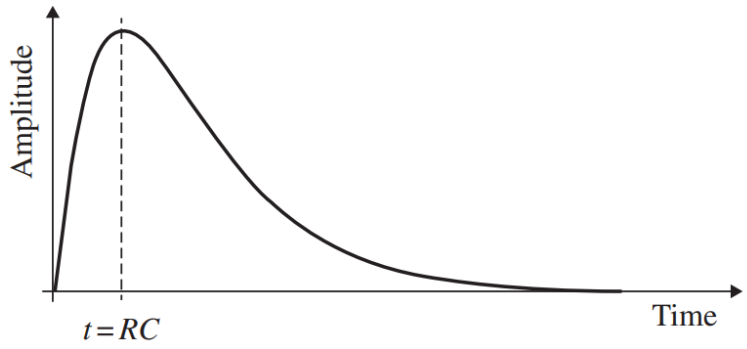

Figura 8: Respuesta del pulso en un filtro CR-RC. [2]

Como hemos visto en esta sección, a altas velocidades este largo decaimiento puede crear errores de apilamiento. Esta es la principal razón por la que este filtro no se usa en sistemas modernos, y por la que debemos de buscar una forma de reducir el ancho del pulso.

2.7. Conformador de pulsos semi-gaussiano CR-RC$^{\mathrm{m}}$

El ancho del pulso de salida de un CR-RC se puede reducir con una serie de integradores RC en su salida usando una disposición en cascada con acoplo con amplificadores de tensión. Esta disposición resulta en un filtro diferenciador CR seguido de m filtros integradores RC. Llamaremos a esto un filtro CR-RC$^{\mathrm{m}}$. En el caso de este filtro, el *peaking time* será $\tau = m\tau_o$ donde τ_o es la constante de tiempo de los componentes y n es la cantidad de elementos RC en el filtro, a lo que llamaremos el orden. Su función de transferencia es de nuevo fácilmente calculable.

$$H_{CR-RC^m}(s) = \frac{s\tau}{(1+s\tau)^{m+1}} \tag{5}$$

El perfil de una señal que pasa por un filtro CR-RC$^{\mathrm{m}}$ de orden 4 ($m = 4$) o más se aproxima a una forma gaussiana, por lo que lo llamaremos a estos filtros semi-gaussianos. Lo normal es no superar este orden, ya que no cambia significativamente el resultado mientras que aumenta su complejidad. Al incluir más etapas integradoras RC hasta este punto, la forma del pulso se

va aproximando a un perfil simétrico y el ancho de los pulsos es reducido. Hay que tener en cuenta que el ancho del pulso solo será reducido si ajustamos las constantes de tiempo para que los filtros tengan el mismo *peaking time* $\tau = m\tau_o$; esta solución es precisamente lo que buscamos para minimizar los errores de apilamiento. Al mismo tiempo se consigue una buena relación señal-ruido. La forma de los pulsos tendrá un perfil similar pero con un menor tiempo de decaimiento y una restauración de línea base más rápida (figura 9).

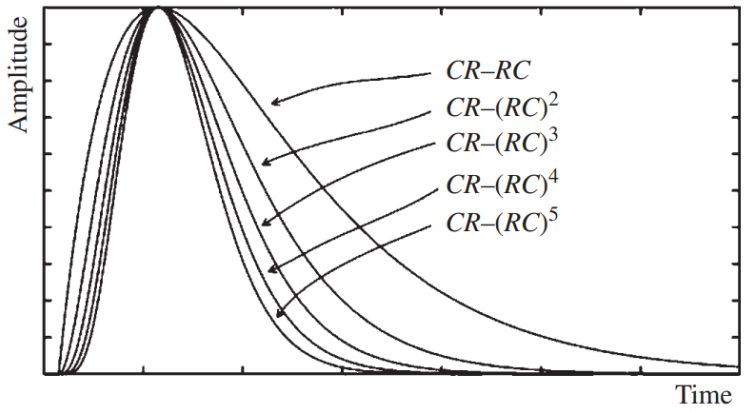

Figura 9: Respuesta de los pulsos con diferentes órdenes de filtros CR-RC^m. Se puede observar como la diferencia entre orden 4 y orden 5 no es notable. [2]

Fuera de su simplicidad, una importante ventaja tanto en los filtros CR-RC como en los CR-RC^m es su buena tolerancia al déficit balístico. Esto se debe a que la cúspide de los pulsos presenta un alto radio de curvatura, en especial para órdenes bajos [3].

2.8. Cancelación de polo cero

Para las figuras (8) y (9) representadas, estamos asumiendo que el pulso de entrada es una función escalón ideal. Lo cierto es que aunque el decaimiento de la señal de salida de un pre-amplificador es largo, no es infinito. Este decaimiento tendrá un efecto directo en la respuesta de los filtros CR-RC^m. El pulso de salida ya no presentará un perfil unipolar, y cruzará la línea base con un *undershoot*. Este subimpulso regresará en un tiempo a la línea base. El decaimiento de los pre-amplificadores suele ser de unos 50 μs, así que el subimpulso durará aproximadamente lo mismo. Al llegar otro pulso en este tiempo, quedará superpuesto a este subimpulso y creará una imperfección por fluctuación de línea base. La solución más común a este problema es la cancelación de polo cero (PZC, *Pole-Zero Cancellation*).

Al fijarnos en la función de transferencia del circuito CR-RC en (4), puede demostrarse que si la excitación presenta un decaimiento, la salida del conformador presentará un subimpulso que sobrepasará la línea base. Un PZC eliminará el subimpulso utilizando una resistencia ajustable kR añadida en paralelo al condensador del diferenciador CR. Se muestra a continuación un esquema de esta solución.

La función de transferencia sería:

$$H_{PZC}(s) = \frac{\tau_2}{\tau_1}\frac{(s\tau_1 + 1)}{(1 + s\tau_2)} \tag{6}$$

En la función de transferencia, $\tau_1 = kRC$ y $\tau_2 = (R \parallel kR)C$. La resistencia kR se escoge

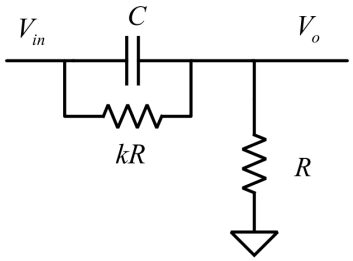

Figura 10: Aplicación de una resistencia kR para la cancelación del polo cero [3].

de tal forma que $\tau_1 = \tau_o$ donde τ_0 es la constante de tiempo del decaimiento, procedente del pre-amplificador anteriormente descrito.

Si hacemos esto el polo del circuito es cancelado por el cero y resulta en un pulso que decae exponencialmente con constante de tiempo τ_2. Si aplicamos esta solución en el elemento diferenciador CR de inicio, la respuesta de salida no tendrá subimpulso, y eliminaremos así un problema de fluctuación de la línea base. Prácticamente todos los conformadores de pulsos en espectroscopía llevan incorporado algún tipo de función PZC, ya que su aplicación es crítica para una buena precisión a altas velocidades [2].

Por último, y pensando ya en el procesado digital de señales, estos filtros tienen todavía una función de transferencia (5) complicada de trasladar a chips DSP o a FPGA para aplicaciones en tiempo real. Esta complejidad solo aumenta si aplicamos un circuito PZC en el derivador CR. Es importante notar que las expresiones obtenidas en esta sección son funciones de transferencia en dominio de Laplace. Para trabajar con una señal digitalizada *x[n]* debemos expresar estas funciones de transferencia en un dominio temporal discreto. Para obtener las funciones de esta forma habrá que tener en cuenta la digitalización de una señal continua, por lo que influirá el periodo de muestreo T en las funciones que obtengamos. Solventar esto y el problema de complejidad necesitará una buena estrategia a la hora de trasladar los circuitos a algoritmos digitales.

3. Diseño del filtro digital

Como ya hemos visto, el procesado digital de señales se basa muchas veces en filtros analógicos ampliamente estudiados. El objetivo en esta sección va a ser trasladar un filtro en tiempo continuo CR-$\mathrm{RC}^{\mathrm{m}}_{\mathrm{PZC}}$ a un diseño en un dominio temporal discreto.

3.1. Cambio de dominio analógico a digital

A continuación, vamos a exponer esquemáticamente qué procedimientos disponemos para convertir los filtros analógicos descritos en el apartado anterior a sus equivalentes filtros digitales.

En primer lugar surge el problema de la conversión de las expresiones de las funciones de transferencia analógicas (Transformada de Laplace) a sus equivalentes funciones de transferencia digitales (Transformada Z). No existe ningún método de aproximación perfecto, pero tres de las técnicas más usadas son las siguientes:

- **Derivada discreta:** Una forma directa de obtener funciones en el dominio temporal discreto es trasladarlas desde el dominio temporal. Para esto habrá que trasladar derivadas que aparezcan en las ecuaciones a un dominio discreto.

  Pasaremos una derivada a una operación discreta como:

  $$\frac{\mathrm{d}x(t)}{\mathrm{d}t} \quad \rightarrow \quad \frac{x[n]-x[n-1]}{T} \tag{7}$$

  Donde T es el intervalo de muestreo, y por lo tanto el intervalo por cada paso digital. En los casos en los que la ecuación en dominio temporal continuo sea posible y simple, este es un método rápido e intuitivo.

  Las funciones de transferencia en tiempo continuo se pueden convertir a $H(s)$, y de aquí a funciones de transferencia en tiempo discreto $H(z)$ mediante un cambio de variable $s \rightarrow (1-z^{-1})/T$, siendo z^{-1} el operador de retardo unitario en tiempo discreto.

- **Invariancia impulsional:** Es una técnica para diseñar filtros IIR en tiempo discreto. Parte de la respuesta al impulso del sistema en tiempo continuo, que se muestrea con una frecuencia de muestreo T para producir una respuesta en tiempo discreto. Para ello el proceso consiste en expresar la función de transferencia $H(s)$ del filtro analógico como una suma de filtros de primer orden.

  $$H(s) = \sum_{k=1}^{N} \frac{c_k}{s-p_k} \tag{8}$$

  Una vez simplificados de esta forma podremos usar la relación $z = e^{sT}$ para hallar la transformada Z del filtro digital.

  $$H(z) = T \sum_{k=1}^{N} \frac{c_k}{1-e^{p_k T} \cdot z^{-1}} \tag{9}$$

  De esta forma, los polos del filtro analógico $s = p_k$ se traducen en polos en $z = e^{p_k T}$.

- **Transformada bilineal:** En esta ocasión partiremos de la relación $z = e^{sT}$. La transformación bilineal aproxima esto a un cociente de polinomios.

$$z = \frac{e^{sT/2}}{e^{-sT/2}} = \frac{1 + sT/2}{1 - sT/2} \tag{10}$$

  Para convertir un diseño analógico en uno digital, se requiere sustituir s por una función de z, y esto se puede aproximar como $s = (1/T) \ln z$, que a su vez se puede aproximar tomando el primer orden de la aproximación del logaritmo por serie hiperbólica. Quedando así la expresión de la transformada.

$$H_d(z) = H_a(s) = H_a\left(\frac{2}{T} \cdot \frac{z-1}{z+1}\right) \tag{11}$$

En nuestro caso se ha optado por el método de derivada discreta por cuestiones de simplicidad computacional. Este método resulta en funciones con expresiones cortas en comparación con las que obtendríamos con otros, como la invariancia impulsional o la transformada bilineal. Para una aplicación en tiempo real esta es una importante ventaja. El inconveniente de esta elección es la potencial inestabilidad de las funciones resultantes, que será algo a tener en cuenta más adelante.

3.2. Diseño del conformador en dominio temporal discreto

Antes se ha comentado que la aplicación de la función de transferencia (5) de forma directa es complicada de trasladar al dominio digital. Yinyu Liu y col. [3] describen la transformada bilineal de (5), tomando T como el periodo de muestreo.

$$H_{CR-RC^m}(z) = \frac{\frac{2RC}{T} \cdot (1 - z^{-1}) \cdot (1 + z^{-1})^m}{\left(\left(1 + \frac{2RC}{T}\right) + \left(1 - \frac{2RC}{T}\right) \cdot z^{-1}\right)^{m+1}} \tag{12}$$

Nos referimos a esta función cuando hablamos de la dificultad de implementar estos filtros en FPGA y chips DSP de forma directa. No solo esto, sino que la función (12) no cuenta con una cancelación de polo cero, así que la función final será todavía más compleja. Aunque es posible trasladar directamente la función a algoritmos, existe una solución más sencilla y conveniente a este problema.

Como estos sistemas son una sucesión de filtros CR y RC más simples, podemos desarrollar algoritmos para estos y aplicarlos en sucesión, del mismo modo que construiríamos un CR-RC$^{\mathrm{m}}$ analógico. Vamos a digitalizar un filtro CR con PZC y un RC por separado y después concatenar la primera etapa CR con tantos elementos RC integradores como queramos.

Para el filtro CR con PZC podemos expresar las relaciones del circuito de la figura (10) como:

$$\frac{V_o(t)}{R} = \frac{V_s(t)}{kR} - \frac{V_o(t)}{kR} + C\frac{\mathrm{d}V_s(t)}{\mathrm{d}t} - C\frac{\mathrm{d}V_o(t)}{\mathrm{d}t} \tag{13}$$

Podemos trasladarlo al dominio digital aplicando la anterior definición de la derivada discreta.

$$\frac{V_o[n]}{V_s[n]} = \frac{V_s[n]}{kR} + \frac{V_o[n]}{kR} + C\frac{V_s[n] - Vs[n-1]}{T} - C\frac{V_o[n] - V_o[n-1]}{T} \tag{14}$$

Desde aquí desarrollamos la ecuación anterior hasta despejar la señal de salida $V_o[n]$.

$$V_o[n]_{CR_{PZC}} = A \cdot V_s[n] - B \cdot (Vs[n-1] - V_o[n-1]) \tag{15}$$

Donde $A = \frac{kd+1-d}{k+1-d}$ y $B = \frac{kd}{k+1-d}$ con $d = RC/(RC+T)$. Al implementarlo en hardware podremos fijar los parámetros de la función. A la hora de determinar k, hay que tener en cuenta que $kRC = \tau_0$ para conseguir la cancelación de polo cero deseada. El integrador RC es la etapa que repetiremos tras el diferenciador $\mathrm{CR_{PZC}}$. Podemos obtener su ecuación de un modo análogo a (15). Desde el circuito RC en la figura (7) podemos escribir:

$$RC\frac{\mathrm{d}V_o(t)}{\mathrm{d}t} + V_o(t) = V_s(t) \tag{16}$$

Lo trasladamos al dominio digital:

$$RC\frac{V_o[n] - V_o[n-1]}{T} + V_o[n] = V_s[n] \tag{17}$$

Finalmente, la función de salida de un filtro RC integrador puede ser obtenida como:

$$V_o[n]_{RC} = (1-d) \cdot V_s[n] + d \cdot V_o[n-1] \tag{18}$$

Igual que en (15), $d = RC/(RC+T)$.

La primera aproximación al conformador ha sido un programa con una primera fase que aplica (15) y le da su salida a la entrada de la función (18). Eso implementa un filtro CR-RC simple, que podemos volver a pasar por (18) para simular un CR-$\mathrm{RC^m}$ con m el número de veces que se ha pasado la señal por el filtro RC. Para los filtros de orden m, el *peaking time* será $m\tau_o$, y por lo tanto habrá que ajustar la constante de tiempo de los filtros. Esto por suerte en dominio digital es tan sencillo como cambiar una constante multiplicativa.

Experimentalmente se ha comprobado que en diseños HLS utilizando la herramienta Vivado, hacer varios bucles en un algoritmo dificulta la síntesis de alto nivel. Esto se debe a la forma de trasladar bucles que tiene la síntesis de alto nivel, que les resta fiabilidad. Sabiendo esto, lo mejor es hacer una única función de salida con un solo bucle para recorrer toda la señal. Para esto deberemos de obtener una función de transferencia única $V_o[n]_{CR-RC^m}$. Aquí el método a seguir pasa por el análisis en el dominio z.

A partir de la ecuación (15) en dominio discreto podemos llegar a una función de transferencia $H(z)$ mediante el operador de retardo unitario z^{-1}.

$$V_o(z)_{CR_{PZC}} = V_s(z)(A - Bz^{-1}) + BV_o(z)z^{-1} \tag{19}$$

$$H(z)_{CR_{PZC}} = \frac{V_o(z)}{V_s(z)} = \frac{A - B \cdot z^{-1}}{1 - B \cdot z^{-}1} \tag{20}$$

De una manera análoga podemos obtener $H(z)$ para el filtro RC desde la ecuación (18) como:

$$V_o(z)_{RC} = V_s(z)(1-d) + d \cdot V_o(z)z^{-1} \tag{21}$$

$$H_{RC}(z) = \frac{V_o(z)}{V_s(z)} = \frac{1-d}{1-d \cdot z^{-1}} \tag{22}$$

Al combinar la ecuación (20) con la (22) podremos obtener la función de transferencia de todo el filtro.

$$H(z)_{CR-RC^m_{PZC}} = \frac{A - B \cdot z^{-1}}{1 - B \cdot z^{-1}} \cdot \left(\frac{1-d}{1-d \cdot z^{-1}}\right)^m \tag{23}$$

Con esto, solo hay que tener en cuenta que aplicar el operador general z^{-m}, es equivalente a retrasar m periodos de muestreo en lugar de uno. Se obtendrán cuatro funciones, una para cada orden. Los algoritmos que se extraen de la ecuación (23) quedan descritos en la tabla (1).

Orden	**Algoritmo en dominio temporal discreto**
1	$V_o[n] = A(1-d)V_s[n] - B(1-d)V_s[n-1] + (d+B)V_o[n-1] - dBV_o[n-2]$
2	$V_o[n] = A(1-d)^2V_s[n] - B(1-d)^2V_s[n-1] + (2d-B)V_o[n-1]$ $-(d^2-2dB)V_o[n-2] + Bd^2V_o[n-3]$
3	$V_o[n] = A(1-d)^3V_s[n] - B(1-d)^3V_s[n-1] + (3d+B)V_o[n-1]$ $-3d(B+d)V_o[n-2] + (3B+d)d^2V_o[n-3] - Bd^3V_o[n-4]$
4	$V_o[n] = A(1-d)^4V_s[n] - B(1-d)^4V_s[n-1] + (4d+B)V_o[n-1]$ $-2d(B+3d)V_o[n-2] + 2d^2(3B+2d)V_o[n-3] - d^3(4B+d)V_o[n-4]$ $+Bd^4V_o[n-5]$

Tabla 1: Algoritmos del filtro CR-RC^m.

Dado el carácter recursivo de los algoritmos guardaremos los últimos valores de la entrada de la señal anterior y los utilizaremos para calcular la siguiente señal de salida. En caso contrario, los primeros $m+1$ valores de la señal de salida se calcularían haciendo referencia a valores que no existen. En el caso de la primera señal leída se fijarán estos valores como 0. Esto también requiere crear un *buffer* en el hardware que guarde la señal de entrada mientras el sistema calcula la señal previa.

4. Implementación en FPGA

Una FPGA es un circuito digital integrado que contiene bloques lógicos e interconexiones programables. La programación se realiza mediante los lenguajes denominados lenguajes de descripción de hardware (HDL). Dos de estos lenguajes son Verilog y VHDL. La ventaja de las FPGA en comparación con los chips DSP es clara, gracias a tener mayores cantidades de RAM en chip, multiplicadores integrados y módulos aritméticos dedicado. Las FPGA se suelen usar como prototipos para chips finales, y en los últimos tiempos también como sistemas en el producto final. Esto viene impulsado por las ventajas económicas y de reprogramabilidad de estos dispositivos. En nuestro caso tener un diseño en hardware permitirá el procesamiento de señales en tiempo real.

4.1. Flujo de diseño digital

El flujo de diseño en dispositivos digitales se refiere a implementar un circuito integrado. Es similar tanto en FPGA como en diseños *full custom*. Las fases del diseño son:

1. **Elaboración:** Se pasa de un código en HDL a una *netlist*, que es una descripción del diseño utilizando puertas lógicas básicas e ideales.
2. **Síntesis:** Se sustituyen las puertas lógicas básicas e ideales con partes reales de la FPGA como LUTs, flip-flops y otros elementos. Esto es una descripción en VHDL.
3. **Implementación:** Sitúa cada uno de estos elementos en el espacio físico disponible y realiza un enrutado para la placa.
4. **Bitstream:** Este paso sólo está en diseños en FPGA. Se crea un archivo compatible con la placa que contiene la información de la implementación. Nos referiremos a este archivo como *Bitstream*.

4.2. Metodología en Vivado

Debemos empezar desde una síntesis de alto nivel HLS, ya que la función que queremos hacer aquí es trasladar un programa a hardware en RTL. Se llama síntesis de alto nivel por que utiliza una descripción en C/C++, lenguajes de programación de alto nivel, para producir un módulo en Verilog, que es un HDL. Deberemos hacer lo que se conoce como un módulo en FPGA con Vivado HLS. Cuando escribamos un código para un módulo en HLS no hay necesidad de llamar a las librerías `stdlib.h` ni `stdio.h`, ya que estas llaman al sistema operativo y al estar trabajando en FPGA, no existe tal sistema. Tampoco existirá una función `main`, al no estar ejecutando ningún programa. Simplemente se creará una función `void`. Una vez escrito el código, Vivado HLS lo convertirá en una diseño de hardware RTL para la placa que le indiquemos.

Ya en Vivado, podremos obtener el módulo RTL, implementarlo en un diseño de bloques con la placa que estemos utilizando. Tras conectar adecuadamente el diseño de bloques se genera el *bitstream*, es decir, se codifica el diseño RTL para que sea compatible con la placa y no haya errores en su implementación.

Una vez tengamos el *bitstream* lanzaremos desde Vivado, SDK (*Software Development Kit*). Éste es un entorno de diseño para crear aplicaciones embebidas en micro-procesadores Xilinx.

Desde Vivado SDK estableceremos una conexión serie entre nuestro ordenador y la FPGA que estemos utilizando. El SDK permite programar la FPGA con el *bitsream* generado. Una vez cargado el bitstream podremos transmitir datos de un modo relativamente sencillo gracias a las librerías creadas al implementar el diseño de bloques. Estas librerías son los controladores del módulo HLS. Vivado SDK dispone de una terminal mediante la que nos vamos a comunicar con la FPGA desde el ordenador. Podremos transferir datos desde aquí, y con programas que realicemos gracias a la ayuda de las librerías generadas. Estos programas compilan automáticamente al guardar, y utilizan el diseño en hardware de la FPGA.

4.3. Arquitectura del diseño en FPGA

En la figura (11) se puede ver el diseño de bloques implementado en Vivado para el conformador de orden 3. Éste será similar para otros órdenes ya que solo cambiará el módulo HLS en cada diseño.

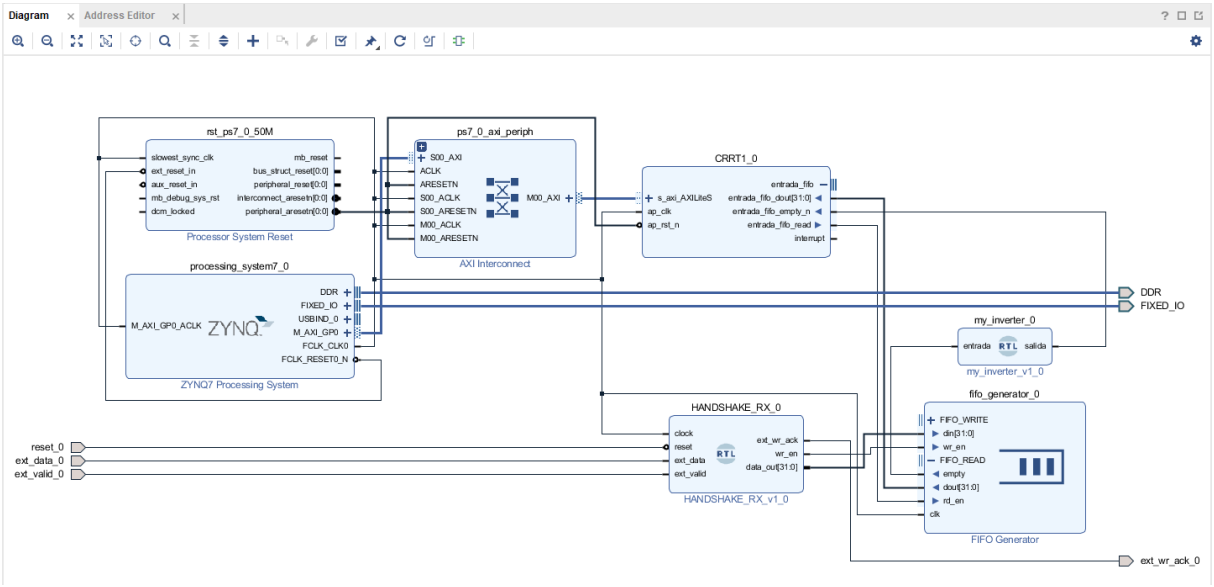

Figura 11: Sistema de bloques en la interfaz de Vivado para el conformador.

En la entrada del conformador se ha incluido un microcontrolador externo tipo Arduino UNO CH340. Éste simula la salida de un pre-amplificador en un sistema de espectrografía con pulsos de diferentes alturas. Su señal tiene una constante de decaimiento fijada en el programa del Arduino, simulando el decaimiento que presentan los pre-amplificadores reales. El Arduino se comunicará con la FPGA para la entrada de datos a través de un puerto serie con un protocolo tipo *handshake*. Esta conexión asegura una transferencia fiable de los datos, evitando problemas de diferentes velocidades de transferencia de bits. La señal recibida pasará a una memoria FIFO (*First In First Out*). Esta transmitirá bloques de datos al módulo HLS, que se encargará del procesado de la señal. Este sistema de entradas se puede ver en el diseño de bloques de la figura (11). Aquí también se muestran las diferentes conexiones entre la entrada, el módulo HLS y el procesador que incluye la placa.

La salida del sistema se puede obtener en una terminal que tiene la propia interfaz de Vivado SDK. Alternativamente, con un programa como PuTTY podremos guardar estos datos de salida para representar el espectro de radiación, aunque ello cae fuera del alcance de este trabajo.

5. Resultados

Los algoritmos de la tabla (1) han sido implementados en diferentes módulos HLS. Estos módulos quedan recogidos en el anexo 2 y 3. Cada uno de los algoritmos ha sido integrado en un diseño de bloques con la estructura expuesta en la figura (11). Una imagen del montaje experimental ha sido incluida en el anexo 5. Al poner en funcionamiento el sistema completo, el Arduino comenzará a enviar la señal de entrada al conformador, y éste la procesará. De esta forma podemos visualizar la señal de cada uno de los conformadores en tiempo real con la metodología descrita en la anterior sección.

Para los órdenes 2 y 4 de la tabla (1) se ha detectado un problema de estabilidad numérica. Ambos algoritmos divergen en condiciones normales de funcionamiento. Para solucionar el problema se ha optado por una aproximación por bloques de operación menores. Así, el filtro no opera en una sola línea sino que pasa a través de una etapa CR_{PZC} y después por m etapas RC. Los módulos HLS para estos órdenes en particular quedan recogidos en el anexo 3.

En las figuras (12) y (13) se muestran dos ejemplos de señales obtenidas utilizando estos conformadores con los algoritmos de la tabla (1).

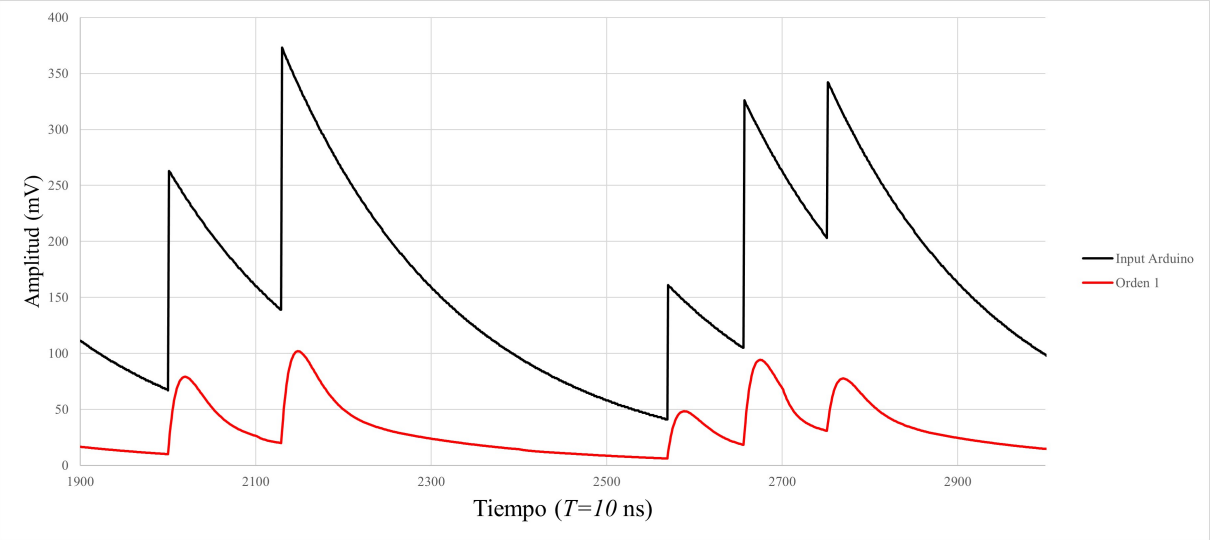

Figura 12: Tramo de una respuesta del filtro CR-RC$^1_{PZC}$ frente a la entrada del arduino. $RC = 0,5\ \mu s$.

En las señales que hemos obtenido se puede ver como el perfil de los pulsos obtenidos coincide con el esperado de un filtro tipo CR-RC.

Podemos observar en las figuras (12) y (13) que el conformador consigue mitigar el efecto del tiempo de decaimiento. Esto era algo que buscábamos, y de importancia crítica, ya que evitará potenciales errores de apilamiento en la señal. Si aumentamos el orden del filtro conseguiremos una señal con pulsos más estrechos, y por lo tanto con menos errores de apilamiento. En las mismas figuras se puede ver que la cancelación de polo cero aplicada elimina el subimpulso en los pulsos obtenidos, independientemente del orden del filtro.

Para mostrar la importancia de la cancelación de polo cero, se ha optado por representar la respuesta de cada filtro a un solo pulso de pre-amplificador en dos casos. Un caso con cancelación

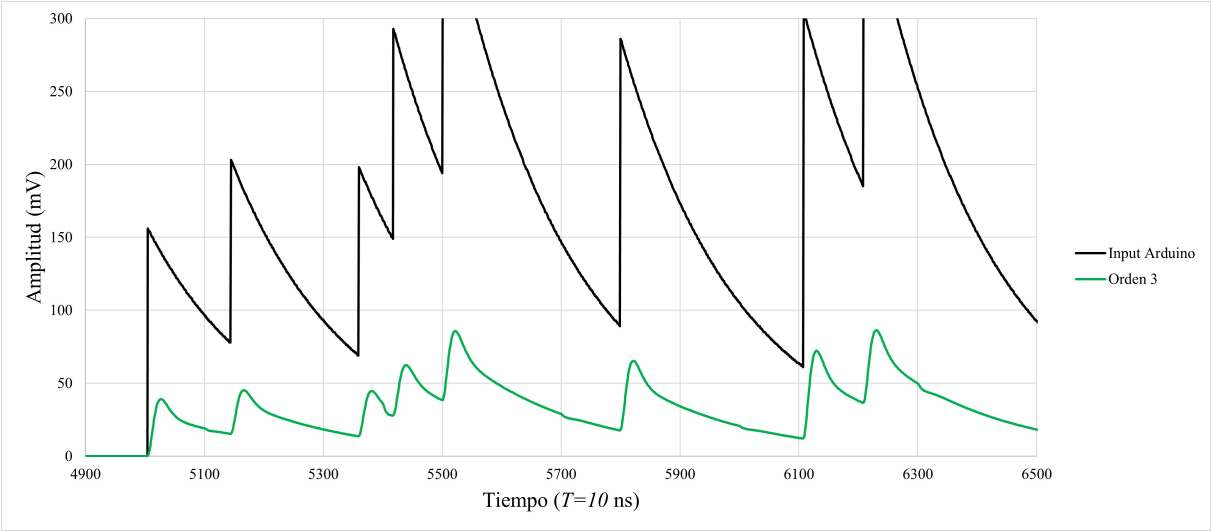

Figura 13: Tramo de una respuesta del filtro CR-RC^3_{PZC} frente a la entrada del arduino. $RC = 0,166\ \mu s$

de polo cero implementada en el diseño y otro sin ella. Los resultados de este experimento se muestran en la figura (14).

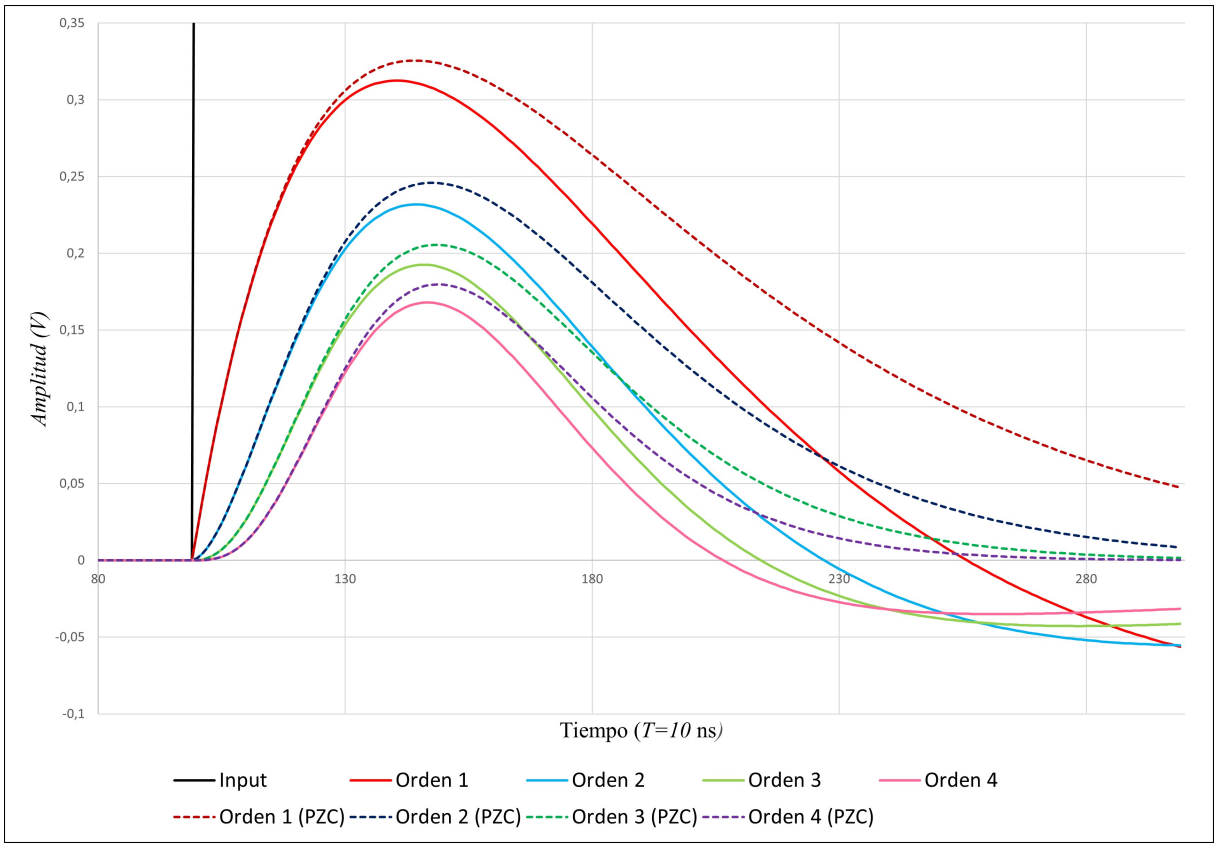

Figura 14: Comparativa entre filtros con PZC y sin PZC para un pulso de pre-amplificador. El decaimiento del pulso de pre-amplificador es $\tau_o = 2\ \mu s$. El valor de las constantes de tiempo es $RC_{Ord=1} = 0,5\ \mu s$, $RC_{Ord=2} = 0,25\ \mu s$, $RC_{Ord=3} = 0,166\ \mu s$, $RC_{Ord=4} = 0,125\ \mu s$.

El sistema también es sensible a la amplitud del pulso de entrada, pero la amplitud de la

respuesta estará sujeta al efecto del déficit balístico. Para medir el impacto de dicho efecto se ha optado por experimentar con dos diferentes pulsos de entrada. Las llamaremos señal 1 y 2. La primera de estas señales será un pulso de pre-amplificador con un tiempo de subida ideal, es decir, nulo. La segunda señal será un pulso de pre-amplificador con un tiempo de subida de 200 ns. Los resultados de esta simulación se muestran en la figura (15) y en la tabla (2).

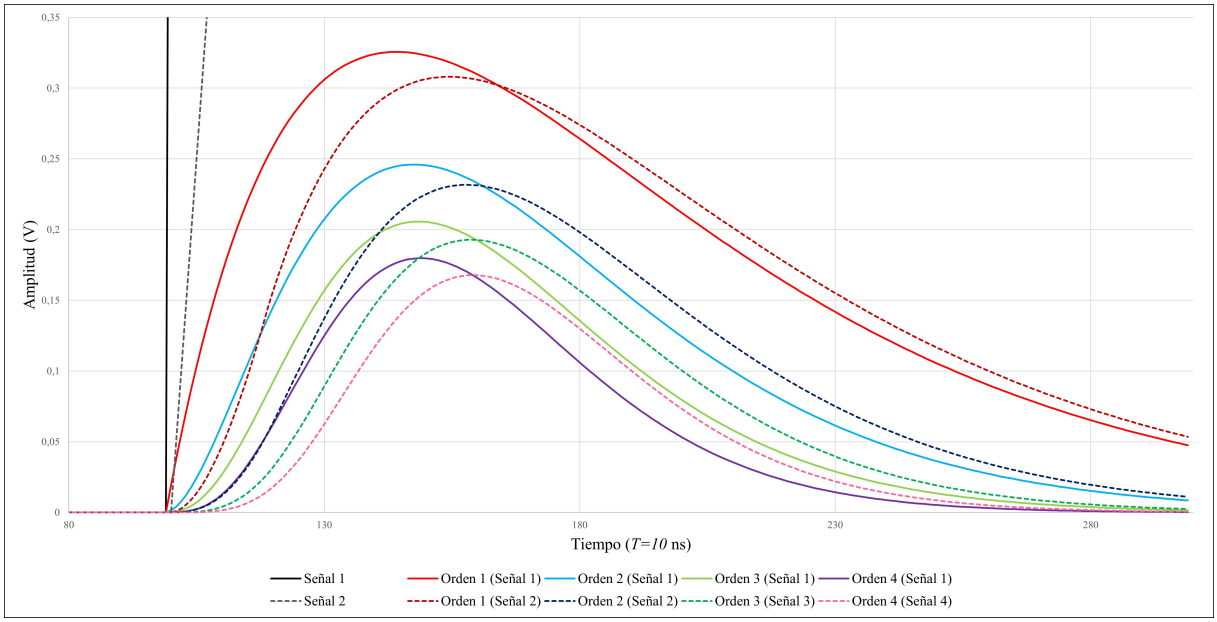

Figura 15: Respuesta de los filtros CR-RC$^{m}_{PZC}$ de órdenes 1 al 4 a diferentes tiempos de subida. Los valores de RC y de τ_o son los mismos que en la figura (14)

A continuación se exponen los resultados en amplitud y déficit balístico BD (2), así como el porcentaje de pérdida.

	Máximo con señal 1	Máximo con señal 2	BD	Pérdida
Pre-Amplificador	1000,00 mV	909,37 mV	90,63 mV	9,06 %
CR-RC$^{m}_{PZC}$ Orden 1	325,63 mV	307,90 mV	17,73 mV	5,45 %
CR-RC$^{m}_{PZC}$ Orden 2	245,94 mV	231,63 mV	14,31 mV	5,82 %
CR-RC$^{m}_{PZC}$ Orden 3	206,08 mV	192,99 mV	13,08 mV	6,35 %
CR-RC$^{m}_{PZC}$ Orden 4	179,80 mV	167,79 mV	12,00 mV	6,68 %

Tabla 2: Resultados de la simulación con filtros CR-RC$^{m}_{PZC}$.

Con los resultados recogidos en la tabla (2) podemos llegar a la conclusión de que la pérdida de amplitud es reducida por el filtro CR-RC$^{m}_{PZC}$. La pérdida de amplitud aumenta conforme aumenta el orden del filtro mientras ajustemos RC para mantener un mismo tiempo de subida. Esto es por que el perfil del pulso con el filtro de mayor orden para el mismo tiempo de subida tendrá menos radio de curvatura en la cúspide. La cancelación de polo cero también contribuye a reducir el déficit balístico, ya que aumentará este radio de curvatura [3]. Esto se puede apreciar en la figura (14).

6. Conclusiones

Este trabajo se ha centrado en el procesado digital de pulsos en detectores de radiación para aplicación en espectrometría. Se han diseñado conformadores digitales de pulsos basados en conformadores analógicos del tipo CR-RC^m con cancelación de polo cero. Se han implementado a nivel de hardware con los conocimientos adquiridos. Para esto se ha trabajado con placas de desarrollo y software propio de este campo.

La utilidad y competitividad de este tipo de conformadores CR-RC^m ha sido discutida por otros autores en artículos como [3, 4], concluyendo que este tipo de conformadores tienen rendimientos algo superiores a los sistemas convencionales.

En cuanto a nuestra propuesta de implementación de conformador, podemos extraer las siguientes conclusiones:

- Los conformadores muestran una alta resistencia al déficit balístico.
- La forma de los pulsos se corresponde con la forma de los pulsos obtenidos por filtros CR-RC^m analógicos.
- Los pulsos obtenidos siguen teniendo un largo decaimiento, por lo que el efecto de apilamiento sigue presente. Esto se puede ver en las figuras (12) y (13).
- La cancelación de polo cero aplicada anula efectivamente el subimpulso de los pulsos resultantes. Esto se puede ver en cada gráfica obtenida, pero se resalta en la figura (14).
- Los algoritmos obtenidos en la tabla (1) resultan inestables para órdenes 2 y 4, pero una aplicación en cadena puede ser una posible solución a este problema, aunque resulten menos eficientes computacionalmente.

Este trabajo podría continuar sus vías de estudio. Una opción sería estudiar la estabilidad de estos conformadores más en profundidad. Más en concreto, se podrían explorar las causas de la inestabilidad de los algoritmos de la tabla (1) con orden par. Podríamos también experimentar con diferentes algoritmos, obtenidos con otros métodos de transformación de dominio. Con todos los algoritmos podríamos tratar de optimizar la disposición en la FPGA y el tamaño del diseño implementado, aunque esto no ha sido un problema en el caso de este trabajo. Por último, también se podría utilizar uno de los conformadores realizados con un detector de radiación real en lugar de una simulación por Arduino.

Referencias

[1] Glenn F Knoll. *Radiation detection and measurement*. John Wiley & Sons, 2010.

[2] Mohammad Nakhostin. *Signal processing for radiation detectors*. John Wiley & Sons, 2017.

[3] Yinyu Liu y col. «Implementation of real-time digital CR–RCm shaping filter on FPGA for gamma-ray spectroscopy». En: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 906 (2018), págs. 1-9.

[4] M Nakhostin. «Recursive algorithms for real-time digital CR-(RC)^n pulse shaping». En: *IEEE Transactions on Nuclear Science* 58.5 (2011), págs. 2378-2381.

[5] Simon R Cherry, James A Sorenson y Michael E Phelps. *Physics in nuclear medicine e-Book*. Elsevier Health Sciences, 2012.

[6] Clive Maxfield. *The design warrior's guide to FPGAs: devices, tools and flows*. Elsevier, 2004.

[7] Soo Hyun Byun. «Radioisotopes and radiation methodology». En: *Med. Phys. 4R06/6R03, version* 15 (2014).

TRABAJO DE FIN DE GRADO

GRADO EN FÍSICA:

Algoritmos para la conformación de pulsos en tiempo real de detectores de radiación

ANEXOS

Facultad de Ciencias
Zaragoza, 26 de junio de 2022

Anexos

Anexo 1: Código del Arduino UNO CH340

```
#define BUFFER_WIDTH 32

#define pulse_density  200 // cuanto mayor sea esta cantidad, menor densidad
    de puntos
#define pulse_height_min 100
#define pulse_height_max 250
#define decay_time     0.005

#define data  2
#define valid  3
#define wr_ack 4

int genPulso()
{
static float pulso=0;

if(random(pulse_density)==0)
pulso+=random(pulse_height_min, pulse_height_max);
else
pulso/=exp(decay_time);

return (int)pulso;
}

void int_to_bin(int entrada, char salida[BUFFER_WIDTH])
{
for(int i=0; i<BUFFER_WIDTH; i++)
{
if(entrada%2==1)
salida[i]=1;
else
salida[i]=0;

entrada = entrada/2;
}
}

void send_buffer(char buffer_out[BUFFER_WIDTH])
{
for(int j=0; j<BUFFER_WIDTH; j++)
{
```

```
digitalWrite(data, buffer_out[j]);

digitalWrite(valid, HIGH);
while(digitalRead(wr_ack)==0);
digitalWrite(valid, LOW);
while(digitalRead(wr_ack)==1);
}
}

int pulso;
char buffer_out[BUFFER_WIDTH];

void setup() {
// put your setup code here, to run once:
pinMode(data, OUTPUT);
digitalWrite(data, LOW);

pinMode(valid, OUTPUT);
digitalWrite(valid, LOW);

pinMode(wr_ack, INPUT);

randomSeed(analogRead(A5)+analogRead(A4)+analogRead(A3)+analogRead(A2)+analogRead(A1)+analog
    // inicializamos el PRNG usando el ruido de los puertos analogicos.

Serial.begin(9600);
}

void loop() {
// put your main code here, to run repeatedly:
while(Serial.read()!='\n'); // enviar un salto de carro para que comience el
    programa.

while(1){
pulso = genPulso();
Serial.println(pulso);
int_to_bin(pulso, buffer_out);
send_buffer(buffer_out);

delay(10); // Esto garantiza que el buffer de la FPGA no se llena.
}
}
```

Anexo 2: Módulos HLS de orden 1 y 3

```
#include <math.h>
#define d 0.948767
#define dm 0.051233
#define A 0.949414
#define B 0.936768
#define N 300

void CRRC_PZC_1(int entrada_fifo[N], float Vo[N])
{

#pragma HLS INTERFACE s_axilite port=Vo
#pragma HLS INTERFACE ap_fifo port=entrada_fifo
#pragma HLS INTERFACE s_axilite port=return

int n;
float Vin[N];
static float Vin_aux=0, Vo_aux_0=0, Vo_aux_1=0;


for(i=0; i<N; i++)
Vin[i] = (float)entrada_fifo[i];

for(n=0;n<N;n++)
{
if(n==0)
Vo[n]=(A*dm*Vin[n])-(B*dm*Vin_aux)+((d+B)*Vo_aux_1)-(B*d*Vo_aux_0);
else if (n==1)
Vo[n]=(A*dm*Vin[n])-(B*dm*Vin[n-1])+((d+B)*Vo[n-1])-(B*d*Vo_aux_0);
else
Vo[n]=(A*dm*Vin[n])-(B*dm*Vin[n-1])+((d+B)*Vo[n-1])-(B*d*Vo[n-2]);
}
Vo_aux_0 = Vo[N-2];
Vo_aux_1 = Vo[N-1];
Vin_aux = Vin[N-1];
}
```

```
#include <math.h>
#define d 0.860585
#define dm 0.139415
#define A 0.865281
#define B 0.831601
#define N 300

void CRRC_PZC_3(int entrada_fifo[N], float Vo[N])
{
#pragma HLS INTERFACE s_axilite port=Vo
#pragma HLS INTERFACE ap_fifo port=entrada_fifo
#pragma HLS INTERFACE s_axilite port=return

int n;
float Vin[N];
static float Vin_aux=0, Vo_aux_0=0, Vo_aux_1=0, Vo_aux_2=0, Vo_aux_3=0;

for(n=0; n<N; n++)
Vin[n] = (float)entrada_fifo[n];

for(n=0;n<N;n++)
{
if(n==0)
Vo[n]=A*dm*dm*dm*Vin[n]-B*dm*dm*dm*Vin_aux+(3*d+B)*Vo_aux_3-3*d*(B+d)*Vo_aux_2
+(3*B+d)*d*d*Vo_aux_1-B*d*d*d*Vo_aux_0;
else if (n==1)
Vo[n]=A*dm*dm*dm*Vin[n]-B*dm*dm*dm*Vin[n-1]+(3*d+B)*Vo[n-1]-3*d*(B+d)*Vo_aux_2
+(3*B+d)*d*d*Vo_aux_1-B*d*d*d*Vo_aux_0;
else if (n==2)
Vo[n]=A*dm*dm*dm*Vin[n]-B*dm*dm*dm*Vin[n-1]+(3*d+B)*Vo[n-1]-3*d*(B+d)*Vo[n-2]
+(3*B+d)*d*d*Vo_aux_1-B*d*d*d*Vo_aux_0;
else if (n==3)
Vo[n]=A*dm*dm*dm*Vin[n]-B*dm*dm*dm*Vin[n-1]+(3*d+B)*Vo[n-1]-3*d*(B+d)*Vo[n-2]
+(3*B+d)*d*d*Vo[n-3]-B*d*d*d*Vo_aux_0;
else
Vo[n]=A*dm*dm*dm*Vin[n]-B*dm*dm*dm*Vin[n-1]+(3*d+B)*Vo[n-1]-3*d*(B+d)*Vo[n-2]
+(3*B+d)*d*d*Vo[n-3]-B*d*d*d*Vo[n-4];
}
Vo_aux_0 = Vo[N-4];
Vo_aux_1 = Vo[N-3];
Vo_aux_2= Vo[N-2];
Vo_aux_3= Vo[N-1];
Vin_aux = Vin[N-1];
}
```

Anexo 3: Módulos HLS de orden 2 y 4

```
#include <math.h>
#define Ord 2
#define d 0.961538
#define A 0.961722
#define B 0.956938
#define N 300

void CRRC_PZC_2(int entrada_fifo[N], float Vo[N], float Voo[N])
{
#pragma HLS INTERFACE s_axilite port=Voo
#pragma HLS INTERFACE ap_fifo port=entrada_fifo
#pragma HLS INTERFACE s_axilite port=return

int n;
float Vin[N];
static float Vin_aux=0, Voo_aux=0;

for (n=0; n<N; n++)
Vin[n]=(float)entrada_fifo[n];

Vo[0]=0;
Voo[0]=0;
for(inord=0; inord<Ord; inord++){
if (inord==0){ //Fase CR con PZC
for(n=0;n<N;n++){
Vo[n]=(A*Vin[n]-B*Vin_aux)+B*Vo_aux;
}
Vo_aux=Vo[n-1];
}
for(n=0;n<N;n++){ //Fase RC que repite Ord veces
Voo[n]=((1-d)*Vo[n])+(d*Voo_aux);
Vo[n]=Voo[n];
}
}
Voo_aux=Voo[n-1];
Vin_aux=Vin[n-1];
}
```

```
#include <math.h>
#define Ord 4
#define d 0.925926
#define A 0.926267
#define B 0.921659
#define N 300

void CRRC_PZC_4(int entrada_fifo[N], float Vo[N], float Voo[N])
{
#pragma HLS INTERFACE s_axilite port=Voo
#pragma HLS INTERFACE ap_fifo port=entrada_fifo
#pragma HLS INTERFACE s_axilite port=return

int n;
float Vin[N];
static float Vin_aux=0, Voo_aux=0;

for (n=0; n<N; n++)
Vin[n]=(float)entrada_fifo[n];

Vo[0]=0;
Voo[0]=0;
for(inord=0; inord<Ord; inord++){
if (inord==0){ //Fase CR con PZC
for(n=0;n<N;n++){
Vo[n]=(A*Vin[n]-B*Vin_aux)+B*Vo_aux;
}
Vo_aux=Vo[n-1];
}
for(n=0;n<N;n++){ //Fase RC que repite Ord veces
Voo[n]=((1-d)*Vo[n])+(d*Voo_aux);
Vo[n]=Voo[n];
}
}
Voo_aux=Voo[n-1];
Vin_aux=Vin[n-1];
}
```

Anexo 4: Programa de Vivado SDK

```
#include "stdio.h"
#include "xparameters.h"
#include "xcrrc_pzc_1.h"
#include "xuartps.h"

#define N 300

typedef union SWAPDATA_UNION
{
int as_int;
float as_float;

} SWAPDATA;

XUartPs Uart;
XCrrc_pzc_1 modulo;
int main()
{
XUartPs_Config *Config;
SWAPDATA data[N];
int i;

Config = XUartPs_LookupConfig(XPAR_PS7_UART_0_DEVICE_ID);
XUartPs_CfgInitialize(&Uart, Config, Config->BaseAddress);
XUartPs_SetBaudRate(&Uart, 9600);

XCrrc_pzc_1_Initialize(&modulo, 0);
XCrrc_pzc_1_DisableAutoRestart(&modulo);

while(1)
{
XCrrc_pzc_1_Start(&modulo);

while(!XCrrc_pzc_1_IsDone(&modulo));

XCrrc_pzc_1_Read_Vo_Words(&modulo, 0, (int*)data, N);

for(i=0; i<N; i++)
printf("%f\n", data[i].as_float);
}

return 0;
}
```

Anexo 5: Montaje experimental

En este anexo se muestra una imagen del sistema utilizado para el conformador, que consta de un Arduino UNO CH340 y una placa PYNQ-Z2, que contiene una FPGA. Ambos dispositivos están conectados a un ordenador mediante puertos serie. En la imagen se muestra la conexión entre Arduino y placa mediante una conexión tipo *handshake*, que consta de 3 cables (*Request*, *Data* y *Acknowledge*, especificados en el programa del Arduino en el Anexo 1).