



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



ESCUELA TÉCNICA  
SUPERIOR INGENIEROS  
INDUSTRIALES VALENCIA

**TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL**

# **DISEÑO E IMPLEMENTACIÓN DE UN CIRCUITO ELECTRÓNICO EN FPGA PARA LA OPTIMIZACIÓN DE LA DETECCIÓN DE EVENTOS EN EL EXPERIMENTO DE FÍSICA DE PARTÍCULAS NEXT (NEUTRINO EXPERIMENT WITH A XENON TPC)**

AUTOR: GUILLERMO NOE ESTEBAN FERRER

TUTOR: RAÚL ESTEVE BOSCH

**Curso Académico: 2017-18**



# RESUMEN

El experimento NEXT pretende, mediante el estudio de la desintegración doble beta del xenón en una cámara TPC, demostrar que el neutrino es su propia antipartícula. Para ello, se debe estimar la energía de las desintegraciones doble beta que ocurren en la cámara, pero teniendo en cuenta que ocurren otros eventos en la cámara al mismo tiempo, siendo necesario discriminarlos.

El sistema de *trigger* es el encargado de discriminar qué eventos poseen una energía de interés, por lo que la eficiencia de este sistema afecta a la capacidad del sistema para adquirir datos. Este trabajo plantea la implementación de un *shaper* digital que permita mejorar la discriminación de eventos en el sistema de *trigger*, mejorando así su eficacia.

Para implementar el *shaper* se desarrollará un modelo informático del *trigger* que permitirá discriminar, entre los eventos previamente adquiridos, eventos no válidos. Mediante este modelo se podrán cuantificar los efectos del shaping obteniendo los parámetros que maximicen la proporción entre eventos válidos y no válidos. Con estos parámetros definidos se realizará la implementación del *shaper* en el sistema de *trigger* del experimento para, finalmente, evaluar la eficacia y viabilidad del *shaper* mediante el análisis de datos reales del detector NEXT-NEW actualmente instalado en las instalaciones del Laboratorio Subterráneo de Canfranc (LSC).

**Palabras Clave:** NEXT, *shaper*, Python, FPGA, *trigger*, digital.



# RESUM

L'experiment NEXT pretén, mitjançant l'estudi de la desintegració doble beta del xenó en una cambra TCP, demostrar que el neutrí és la seua pròpia antipartícula. Per a això, s'ha d'estimar l'energia de les desintegracions doble beta que ocorren a la cambra, però tenint en compte que ocorren altres esdeveniments a la cambra al mateix temps, per la qual cosa és necessari discriminar-los.

El sistema de *trigger* és l'encarregat de discriminar quins esdeveniments posseïxen una energia d'interés, per això l'eficiència d'aquest sistema afecta la qualitat i quantitat de les dades adquirides. Aquest treball planteja la implementació d'un *shaper* digital que permeta una major discriminació d'esdeveniments en el sistema de *trigger*, millorant així la seua eficàcia.

Per a implementar el *shaper* es desenvoluparà un model informàtic del *trigger* que permetrà discriminar esdeveniments invàlids. Així, mitjançant el processat de dades adquirides per l'experiment es podran quantificar els efectes del *shaping*, obtenint els paràmetres del *shaper* que maximitzen la proporció entre els esdeveniments vàlids i invàlids. Amb aquests paràmetres definits es realitzarà la implementació del *shaper* en el sistema de *trigger* de l'experiment per a, finalment, avaluar l'eficàcia i viabilitat del *shaper* mitjançant l'anàlisi de dades reals del detector NEXT-NEW actualment instal·lat a les instal·lacions del Laboratori Subterrani de Canfranc (LSC).

**Paraules clau:** NEXT, *shaper*, Python, FPGA, *trigger*, digital.



# **ABSTRACT**

The NEXT experiment pretends, through studying the double beta decay of xenon on a TCP camera, to demonstrate that the neutrino is its own antiparticle. To do this, is necessary to measure the energy of the double beta decays occurring in the chamber. However, there are a large number of other events occurring in the chamber, so is necessary to discriminate the events occurring in the chamber.

The trigger system is the responsible for discriminating which events have an energy of interest, therefore, the efficiency of this system affects the capacity of the system to acquire data. This work proposes the implementation of a digital shaper in the trigger system. The shaper will allow a greater discrimination of events, improving the effectiveness of the trigger.

To implement the shaper a computer model of the trigger will be developed, this model will process the data acquired by the experiment and will be able to discriminate invalid events in order to quantify the effects of the shaper, this will allow to obtain the parameters of the shaper that maximize the ratio between valid and invalid events. With these parameters defined the shaper will be implemented in the trigger system of the experiment, to finally evaluate the efficacy and viability of the shaper analysing the real data collected from the NEXT-NEW detector located under the mountains in Canfranc.

**Keywords:** NEXT, shaper, Python, FPGA, trigger, digital.





# Índice de la memoria

<b>1.Introducción .....</b>	<b>1</b>
1.1. Objetivo .....	1
<b>2. Antecedentes .....</b>	<b>1</b>
2.1. Introducción .....	1
2.2. El experimento NEXT.....	1
2.2.1. El neutrino y la desintegración doble $\beta$ .....	2
Desintegración doble $\beta$ del xenón.....	2
2.2.2. EL detector NEXT .....	3
La Cámara.....	3
EL DAQ.....	5
2.3. <i>Shaping</i> .....	6
<b>3. Motivación y objetivos.....</b>	<b>9</b>
3.1. Motivación .....	9
3.2. Objetivos .....	10
<b>4. Determinación de las condiciones del diseño.....</b>	<b>11</b>
4.1. Introducción .....	11
4.2. <i>Trigger</i> en energía .....	12
4.3. Elección del <i>shaper</i> .....	14
<b>5. Estudio para la optimización del <i>trigger</i>.....</b>	<b>17</b>
5.1. Introducción .....	17
5.2. Herramientas.....	18
5.3. Formato del archivo de RUN .....	19
5.3. Módulos de simulación del hardware.....	20
5.4. Simulación y resultados.....	29
<b>6. Implementación.....</b>	<b>34</b>
6.1. Introducción .....	34
6.2. Implementación del <i>shaper</i> .....	34
6.2.1. Forma digital del <i>shaper</i> .....	34
6.2.2. Estudio de cuantificación .....	36
6.2.3. Implementación en verilog .....	40

Verificación.....	48
6.3. Integración en el experimento NEXT .....	53
<b>7. Conclusiones.....</b>	<b>57</b>
7.1. Trabajo futuro .....	57
<b>BIBLIOGRAFIA.....</b>	<b>59</b>

## Índice del Presupuesto

<b>1. Consideraciones previas.....</b>	<b>63</b>
<b>2. Presupuesto. ....</b>	<b>63</b>
2.1 Capítulos y unidades de obra .....	63
2.2 Estado de mediciones. ....	64
2.3 Presupuesto de ejecución.....	64
2.4 Presupuesto de inversión. ....	65
2.5 Presupuesto de licitación. ....	65

# MEMORIA



# 1.Introducción

## 1.1. Objetivo

Si bien el progreso científico es realizado en los laboratorios de físicos, químicos y científicos en general; cabe destacar la importancia de las herramientas necesarias para la demostración experimental de grandes ideas teóricas. La demostración de dichas teorías, en muchos casos, lleva detrás de sí la creación y desarrollo de herramientas y tecnología que, a un nivel directo, son casi o más revolucionarias de lo que supondría la demostración de la teoría para la cual se desarrollaron.

Aun así, no cabe restar importancia al desarrollo e implementación de las herramientas que los físicos experimentales utilizan para formular sus estudios.

De esto mismo versa este proyecto: de la continua mejora a la que se someten estas herramientas con tal de que los científicos puedan obtener la mejor calidad de datos y resultados con los cuales apoyar sus teorías.

Así, englobado en el conjunto de necesidades de una mejor instrumentación en el experimento NEXT, nace este trabajo cuyo título es:

Diseño e implementación de un circuito electrónico en FPGA para la optimización de la detección de eventos en el experimento de física de partículas NEXT (*Neutrino Experiment with a Xenon TPC*).

## 2. Antecedentes

### 2.1. Introducción

Como se ha mencionado antes, este trabajo versa en mejorar una pequeña parte de la instrumentación del experimento NEXT; concretamente de la instrumentación electrónica. No supone un cambio fundamental, es más un análisis con el objetivo de mejorar la eficacia del sistema para determinados casos experimentales.

Para comprender el objetivo y desarrollo de este proyecto es necesario tener en cuenta la necesidad específica que motiva este proyecto.

### 2.2. El experimento NEXT

El experimento NEXT es un esfuerzo común liderado por instituciones de investigación y universidades de distintos países, que busca demostrar si los neutrinos son su propia antimateria mediante la observación de la desintegración doble  $\beta$  del xenón mediante una cámara TPC, *Time Projection Chamber*, capaz de medir la energía del evento, así como de obtener una imagen tridimensional del mismo.



Figura 1 Logo del experimento Next [6]

### 2.2.1. El neutrino y la desintegración doble $\beta$

La ley CP, o simetría de Carga y Paridad, establece que las leyes del universo se aplicarían igual, tanto en el caso de que intercambiáramos cargas positivas por negativas, cumpliendo la simetría de carga, como en el caso de una posible proyección especular del universo. Lo que quiere decir, que las leyes se aplicarían de la misma forma en el reflejo en un espejo de nuestro universo, cumpliendo la simetría de paridad.

Estas leyes están demostradas para los grandes procesos físicos: gravedad, electricidad y magnetismo. Sin embargo, se ha demostrado que no se cumplen para interacciones débiles entre partículas.

En esta violación encontramos una posible explicación a la diferencia entre materia y antimateria presentes en nuestro universo.

Sin embargo, no se ha demostrado aún una violación directa de la ley CP, es decir, una partícula la cual ella misma y su antipartícula no se comporten de manera simétrica.

Los neutrinos, a diferencia de otros fermiones, no tienen carga; esto implica la posibilidad de que su masa sea de tipo majorana y, por lo tanto, que el neutrino sea su propia antipartícula. Si el neutrino fuese su propia antimateria, al tener masa y ser cuantificable en las reacciones entre partículas, rompería de forma directa la ley CP, al no poder explicar las interacciones de los neutrinos entre sí. La clave para la demostración de esto reside en la desintegración doble  $\beta$ .

La desintegración doble  $\beta$  es un tipo de interacción en la cual dos protones de un átomo se convierten en dos neutrones, liberando un electrón y un neutrino cada uno.

Si la suposición anterior es correcta, en esta reacción los dos neutrinos podrían aniquilarse entre sí, produciéndose una desintegración en la cual sólo se liberasen dos electrones.

La energía liberada en esta interacción se reparte entre los electrones y los neutrinos. En el caso buscado de que estos últimos se aniquilasen entre sí, toda la energía liberada en la reacción irá a parar a los electrones. Así, con una medición precisa de la energía liberada y la disipada por los electrones, podríamos demostrar la existencia de interacciones en las cuales los neutrinos se hayan aniquilado entre sí, demostrando la naturaleza majorana de los mismos.

#### Desintegración doble $\beta$ del xenón

La desintegración doble  $\beta$  del xenón consiste en la desintegración de éste en un átomo de bario, dos electrones y dos neutrinos. Esta reacción emite una pequeña cantidad de energía en forma de fotones conocida como centelleo primario. Los electrones liberados reaccionan con el medio liberando aún más fotones y causando el fenómeno conocido como centelleo secundario.

Como se ha explicado antes, en caso de darse desintegración sin neutrinos, la energía portada por los electrones sería mayor. Así pues, se obtendría un centelleo secundario de un mayor nivel energético. Antes de buscar esta sutil diferencia hay que poder localizar y registrar desintegraciones doble  $\beta$  del xenón, así como discriminarlas de otros eventos de naturaleza similar. Para ello hace falta un detector de partículas aislado con gran sensibilidad de medición de energía y capacidad para discriminar el evento buscado de otros similares.

### 2.2.2. EL detector NEXT

La cámara

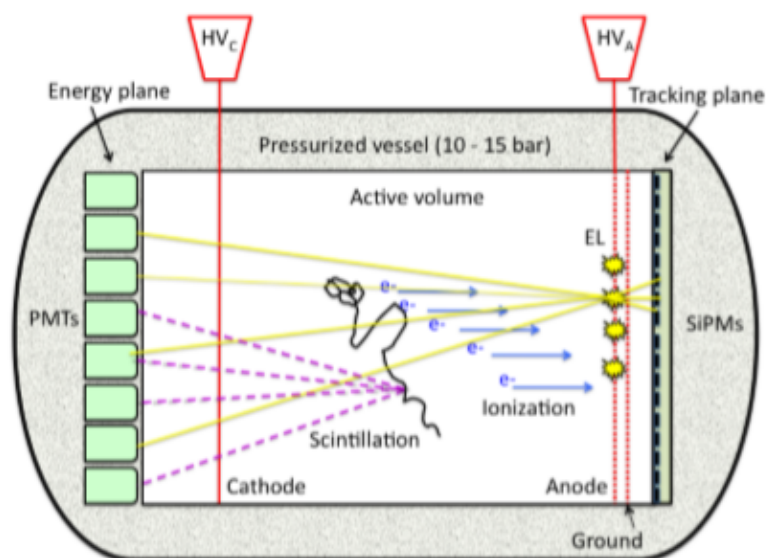


Figura 2: Cámara TPC [3]

El detector de partículas empleado en el experimento es una TPC o cámara de proyección temporal.

Como se ejemplifica en la figura 2, una TPC es una cámara sellada inundada de un gas o fluido que separa un ánodo y un cátodo que someten a la cámara a un campo eléctrico. Este hecho causa que los electrones generados en las interacciones del interior de la cámara se desplacen hacia el ánodo en el cual se encuentra la zona electroluminiscente o EL, con la cual los electrones reaccionan dejando trazas electroluminiscentes al ser acelerados. Los sensores se agrupan en los planos del cátodo y del ánodo. Para la desintegración doble beta del xenón estos sensores buscan captar dos momentos en los cuales se libera energía. El primero de ellos se denomina S1, o centelleo primario, que corresponde a la energía liberada en la propia desintegración doble beta, el otro es el S2, o centelleo secundario, en el cual la liberación de energía es mucho mayor y corresponde a la energía liberada por los electrones al llegar a la zona EL.

En el experimento NEXT la cámara está bajo tierra en Canfranc para evitar la máxima radiación o interferencia exterior.

La cámara está rellena de xenón y existen dos planos de sensores diferenciados: los PMTs en el cátodo y los SiPMs en el ánodo junto a la zona EL.

El plano de SiPM o de fotomultiplicadores de silicio está compuesto por matrices de estos sensores agrupados en módulos de 64 sensores cada uno llamados "DICE". El plano está compuesto por 28 "DICE".

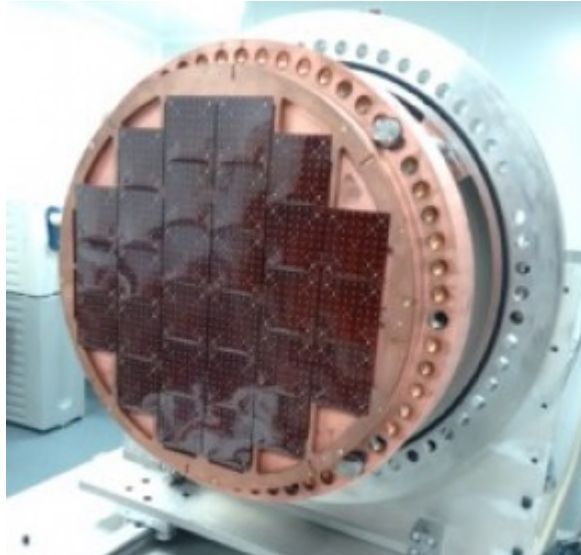


Figura 3 Plano SiPM [3]

El plano TP, *Tracking Plane*, está formado por agrupaciones en DICEs de SiPM, concretamente el modelo S10362-650P de Hamatsu. Se encargan de adquirir la energía liberada en forma de luz por los electrones al interactuar con la zona EL. Al recomponer estos datos el sistema genera la traza de éstos (figura 4), lo que permite a posteriori la identificación del evento, para así descartar los eventos de interés, de otros eventos.

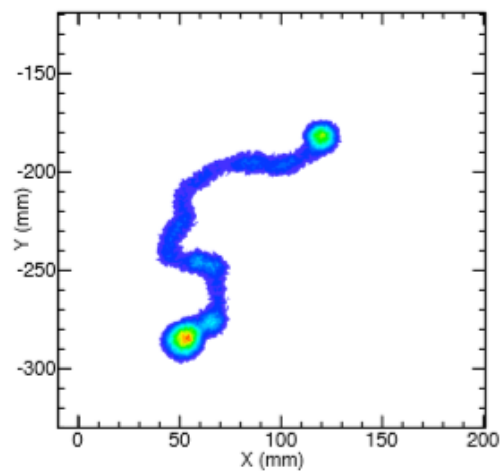


Figura 4 Desintegración captada por el experimento [3]

El plano EP, *Energy Plane*, está formado por los PMTs o fotomultiplicadores. En concreto el plano EP está formado por un array de 60 fotomultiplicadores Hamatsu R11410-10 ocupando el 32.5 % del cátodo de la cámara. Como muestra la figura 5, los fotomultiplicadores van aislados en cápsulas de cobre individuales con una lente en la parte que da a la cámara puesto que no pueden soportar los 10 bares de presión en el interior de ésta.



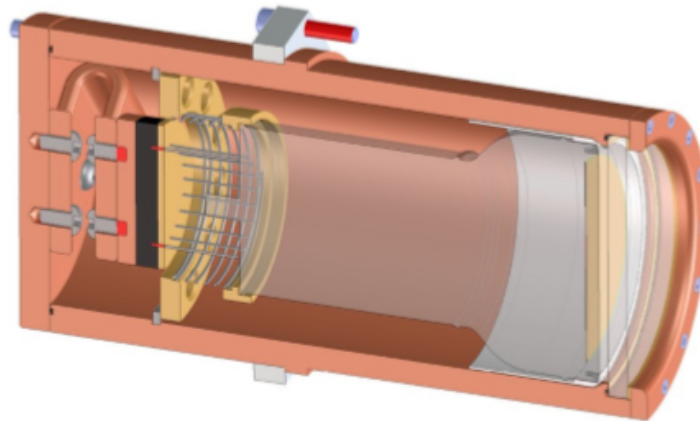


Figura 5: Fotomultiplicador encapsulado [3]

Los fotomultiplicadores detectan de forma muy precisa la energía liberada en forma de fotones de las propias interacciones, que se producen dentro de la cámara. Esta precisión es clave para distinguir S1 y S2 de otros eventos que se producen, de forma simultánea, en la cámara. Además, actúan como fuente de la señal de *trigger* del sistema.

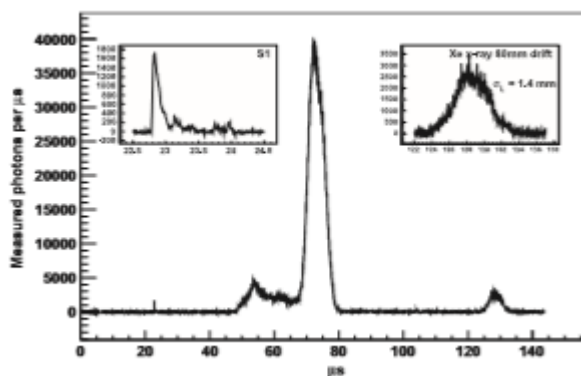


Figura 6: Medida del plano EP [5]

Los PMTs producen pulsos de señal del nivel de incidencia de fotones sobre su fotocátodo. Estos pulsos son de una duración inferior a 5ns. La salida de los fotomultiplicadores va conectada a un filtro RC para eliminar ruidos y, posteriormente, realizar una amplificación simple de una etapa y bajo ruido, del orden de  $2nV/\text{raíz}(\text{Hz})$ . La amplificación realizada es la justa para compensar la atenuación del posterior filtrado mediante un filtro pasivo RC con frecuencia de corte de 800Khz, extendiendo los pulsos lo suficiente para su correcta adquisición a 40MHz. Los datos adquiridos toman la forma de la señal mostrada en la figura 6.

#### El DAQ

El DAQ, *Data Acquisition System*, del experimento es el sistema que se encarga de almacenar, procesar, y enviar los datos generados por los sensores para su posterior análisis.

El sistema está gobernado por un software informático de adquisición DATE. Este software compacta los datos recibidos en sub-eventos que son transmitidos a concentradores de datos que generan y almacenan eventos completos para su posterior análisis. La figura 7 ejemplifica este proceso.

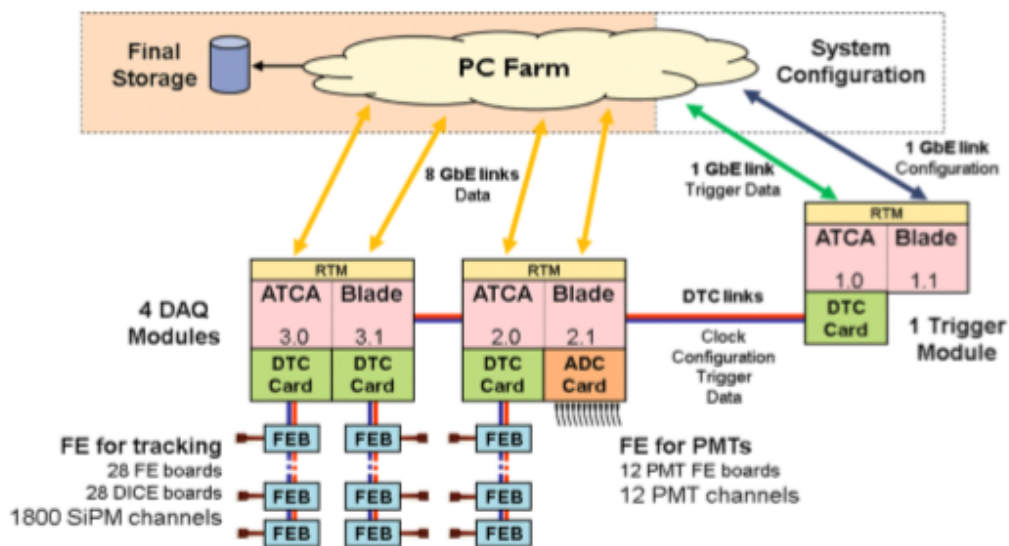


Figura 7 Arquitectura del DAQ [5]

El sistema de adquisición se agrupa mediante tarjetas FEC (medio ATCA Blade en la figura anterior) o concentradoras de *frontend*, que concentran distintos módulos de adquisición de forma independiente y están conectadas al sistema informático mediante enlaces Gigabit Ethernet.

El DAQ dispone de 5 FECs; 4 para la adquisición de datos en sí, 2 de ellas equipadas con una tarjeta digitalizadora de 24 canales 12 bits y 40mHz; y otras 3, equipadas con un módulo de expansión de señales LVDS que actúan como módulo de control y *trigger*, distribuyendo las señales de *trigger* control y reloj, y como tarjetas concentradoras de datos del plano de TP.

Las FECs están equipadas con una FPGA Virtex 6. Una FPGA es un dispositivo lógico programable que dispone de numerosos bloques lógicos configurables mediante programación. Esto permite implementar circuitos y lógica digital personalizada en la tarjeta mientras se mantiene la capacidad de adaptación de un dispositivo programable.

Los módulos de DAQ van almacenando datos en un *buffer*. Los correspondientes al plano de los PMTs van procesando los datos recibidos generando candidatos a *trigger* y envían éstos al módulo de *trigger* principal. Cuando éste genera un *trigger* válido, marcan los datos con el *timestamp* del *trigger* y los vuelcan mediante la conexión Ethernet en los ordenadores del sistema DATE.

### 2.3. Shaping

En base a [8] Y [9] podemos simplificar lo siguiente respecto al shaping.

El *shaping* o *pulse shaping* es un procedimiento de tratamiento de señal por el cual se altera la forma de onda de una señal o pulso. Se realiza principalmente mediante la aplicación de una tipología de filtro concreta.

El objetivo principal de un shaping suele ser adecuar la señal recibida con tal de que sea más sencilla la obtención de una determinada información de la señal procesada; por ejemplo, alterando la constante de tiempo de un determinado pulso con el objetivo reducir el ratio señal

ruido y mejorar la medición de amplitud de la señal o aumentar la duración de los pulsos de una señal para que puedan ser adquiridos con mayor fiabilidad por un DAC. Esto se realiza aplicando acciones integradoras y diferenciadoras sobre la señal alterando la constante de tiempo de ésta.

Hay una gran cantidad de tipologías de filtro que pueden aplicarse para shaping, desde complejos filtros con respuesta temporal activa a combinaciones sencillas de tipo RC. La utilización de una configuración u otra se basará principalmente en las necesidades en la calidad del *shaping*.

Configuraciones complejas permiten mejores ajustes y prestaciones; sin embargo, configuraciones sencillas pueden realizar el mismo trabajo con un coste menor.

Los parámetros principales para el *shaping* son: el tiempo de respuesta de la señal, el índice balístico y la tipología de pulso resultante.

La tipología de pulso representa la forma del pulso de entrada para distintas tipologías de entrada y viene dada principalmente por la tipología y orden del *shaper*, con diferentes tipologías y órdenes podemos obtener una respuesta distinta.

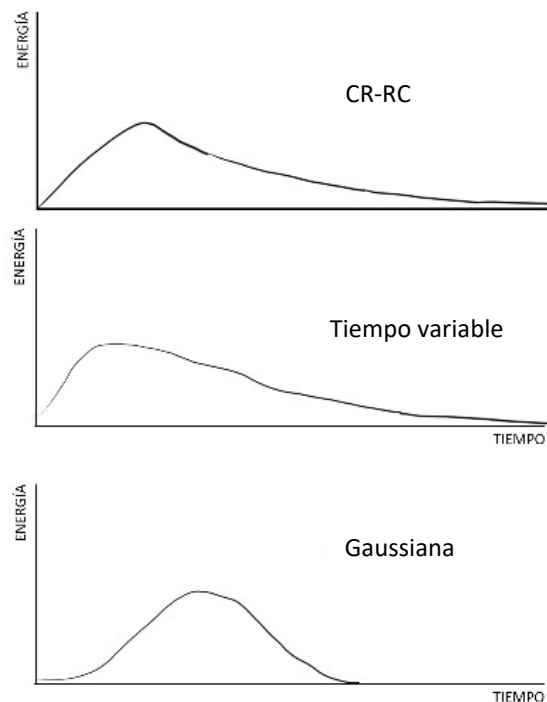


Figura 8 Tipologías de shaping

El tiempo de establecimiento es el tiempo en el cual el pulso resultante alcanza su máximo, depende principalmente del factor de amortiguamiento del filtro.

El déficit balístico es la diferencia de amplitud de pico del pulso por el aumento de la duración en el tiempo de la señal, y va ligado a la simetría de la señal y al factor de amortiguamiento del filtro.

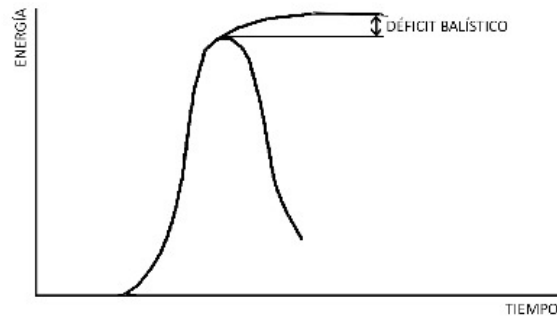


Figura 9 Déficit Balístico

Estos factores representarán la caracterización del filtro a elegir. Un *Shaper* de mejor calidad presenta déficits balísticos bajos o nulos y tiempos de establecimiento rápidos para evitar el solape de pulsos en la señal tratada.

Algunas tipologías típicas de *shaper*, aunque no las únicas existentes son:

- Shaping CR-RC. Consiste en una combinación de un diferenciador y un integrador, el cual puede ser de distintos órdenes. Según aumenta el orden del factor RC, el tiempo de establecimiento es menor. Esta tipología no sigue exclusivamente esta estructura, puede variar los factores haciéndose más simples, conformando un *shaper* CR o un *shaper* RC o variar el orden del factor RC.

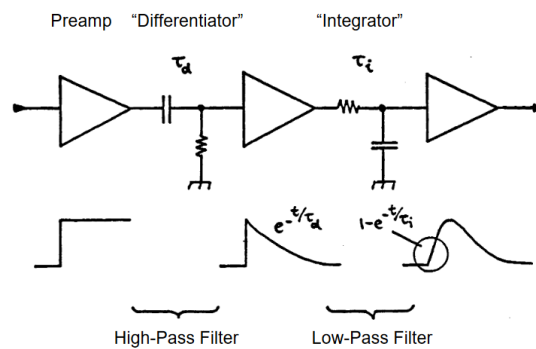


Figura 10 *Shaper* RC-CR [7]

- Shaping Gaussiano. Esta clasificación se da por la respuesta del *shaper*. La salida de un *shaper* gaussiano se aproxima a una distribución gaussiana del pulso.
- *Shaper* de tiempo variante. Esta tipología más compleja cambia los parámetros del filtro durante el tratamiento de pulsos individuales.

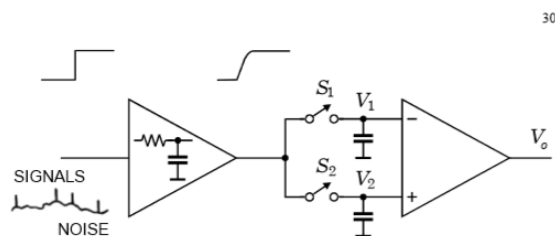


Figura 11 *Shaper* de tiempo variante [7]

Estas tipologías son típicamente las más utilizadas para adquisición de datos, si bien existen numerosas variantes de éstas o distintas combinaciones. Todas estas tipologías analógicas pueden adaptarse fácilmente a su uso digital a partir de sus funciones de transferencia.

### 3. Motivación y objetivos

#### 3.1. Motivación

La problemática que motiva este trabajo es la aparición de *triggers* erróneos o falsos en gran porcentaje durante el funcionamiento del sistema de adquisición de datos; estos *triggers* son provocados por comportamientos en la señal adquirida muy similares en duración y energía total a los de interés, principalmente para el centelleo primario (S1).

Estos falsos *triggers* generan una considerable cantidad de datos erróneos que tienen que ser descartados posteriormente, consumiendo tiempo y recursos.

Además, dada la gran cantidad de datos que se manejan, la eficiencia del sistema de adquisición es un aspecto fundamental para la eficacia del análisis de los datos del experimento. También para la adquisición en sí misma, dado que la disminución de eventos no deseados disminuye el *throughput* del sistema, reduciendo tiempos muertos en la detección de eventos.

Dada la necesidad de reducir el número de *triggers* falsos, se opta por estudiar la implementación de un *shaping* digital de la señal para el accionamiento del *trigger*. Este *shaping* tendría como objetivo suavizar la forma de la señal, reduciendo los falsos *triggers*; es sencillo de implementar, activar y desactivar del sistema, y se ha utilizado anteriormente para paliar problemas similares en experimentos previos a NEXT-100, como el prototipo NEXT-DEMO. La diferencia radica en que, en este último prototipo, el *shaping* era analógico.

Así pues, se estudiará la implementación de una tipología de *shaper* igual a la utilizada en los casos mencionados.

### 3.2. Objetivos

El objetivo principal es la optimización del sistema de *trigger* con el fin de eliminar la mayor cantidad posible de falsos *triggers*. Esto se llevará a cabo en tres fases:

- Estudiar la optimización del *shaping* digital aportado para la eliminación parcial o completa de *triggers* inválidos a partir de datos adquiridos por el sistema.

Esto implica desarrollar una herramienta informática que permita procesar datos de forma que se simule el efecto del módulo de *trigger* y, a su vez, permita distinguir *triggers* válidos de inválidos. Esta simulación utilizará como entrada ficheros de ventanas de datos ya adquiridos por el sistema, para lo que habrá que realizar una función que los lea. A su vez, habrá que implementar un módulo que aplique la función de *shaping* a estos datos para su posterior procesado.

Todo esto se aglutinará en una función que permita contrastar la proporción de *triggers* válidos e inválidos de datos con *shaping* y los originales.

Una vez desarrollada una función que permita evaluar la eficacia del *shaping*, se añadirá a una función que realice un barrido de coeficientes para el *shaper* buscando los que den mejores resultados.

- Estudiar y realizar la implementación del *shaper* en el hardware actual del experimento.

Una vez estimados los mejores coeficientes, se procederá a implementar la tipología del *shaper*. Para ello habrá que:

- Estudiar y minimizar los posibles errores de cuantificación al hacer la implementación en hardware del módulo de *shaping*.
- Diseñar en hardware el módulo de *shaping* mediante un lenguaje de descripción de hardware.
- Verificar el hardware a implementar mediante un *testbench* con tal de descartar errores fundamentales de diseño.

- Analizar el funcionamiento de la implementación en un entorno real.

Una vez implementado el módulo de *shaping* se añadirá al módulo de PMTs del experimento. Se utilizará la herramienta informática anterior para realizar un análisis de los datos adquiridos, compararlos con los simulados y, a partir de los resultados, sacar conclusiones de la utilidad del *shaping*. Esto incluye:

- Llevar a cabo un análisis de los datos experimentales obtenidos.
- Realizar un informe de la viabilidad en base a los datos analizados.

-

## 4. Determinación de las condiciones del diseño

### 4.1. Introducción

Funcionamiento del sistema de detección:

Como se ha explicado anteriormente, el EP genera los *triggers* internos del sistema. El sistema de *trigger* o detección de eventos está basado en un módulo FEC independiente, denominado TRG FEC, que gestiona las señales de reloj, *trigger* y control para el resto de FECs del sistema. Este módulo recibe de los FEC del plano EP, denominados PMT FEC, candidatos de *trigger*, que se generan mediante el análisis preliminar en la señal en los propios módulos PMT FEC.

En una adquisición se establecen un número determinado de coincidencias entre candidatos de *trigger*. Además, se pueden fijar un conjunto de PMTs activos concretos como únicas fuentes de candidatos de *trigger* posibles. En el detector actual, hay 12 PMTs conectados a 2 módulos PMT FEC, 6 PMTs cada uno. La figura 12 muestra este proceso:

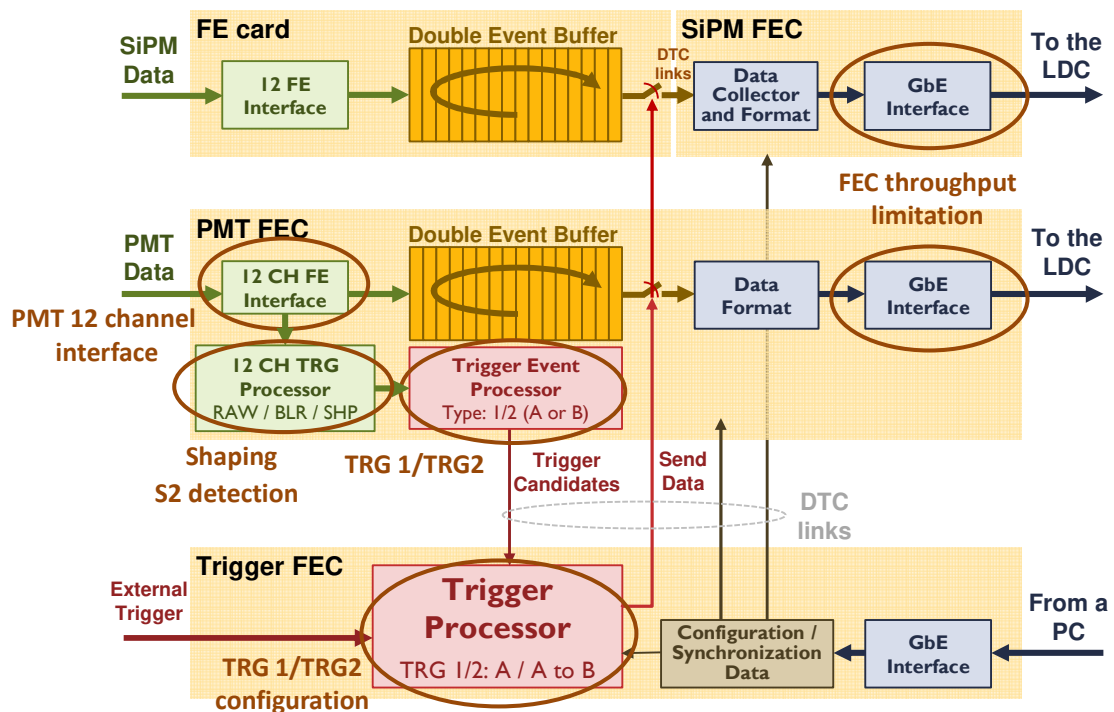


Figura 12: Funcionamiento del algoritmo de *trigger* [4]

## 4.2. Trigger en energía

La generación de los candidatos de *trigger* en los módulos PMT FEC se basa en un análisis de la energía de la señal. Ésta es proporcional al área de los pulsos medidos por el sistema. Por lo tanto, el *trigger* digital evalúa el área y la duración de los pulsos que conforman la señal.

Para la ejecución del *trigger* se definen los siguientes parámetros reflejados en la figura 13:

- Desviación de *baseline*. Es el nivel de energía de fondo presente en el sistema durante la adquisición. La desviación respecto al *baseline* es la variación del nivel de energía respecto a la cual se empieza a contabilizar el nivel de energía del posible *trigger*.
- Mín./Máx. nivel de energía. Es el margen en el cual el valor de energía del pulso medido es tomado como un *trigger* válido.
- Nivel de amplitud máximo. Si la señal pasa en algún momento durante la cuenta este nivel, ya no se considera un *trigger* válido y se deja de contar.
- Duración del pulso. Si la cuenta dura más o menos de un determinado tiempo, el *trigger* deja de ser válido. Este tiempo se traduce a un número de cuentas o ciclos de funcionamiento en función de la velocidad del sistema de adquisición.

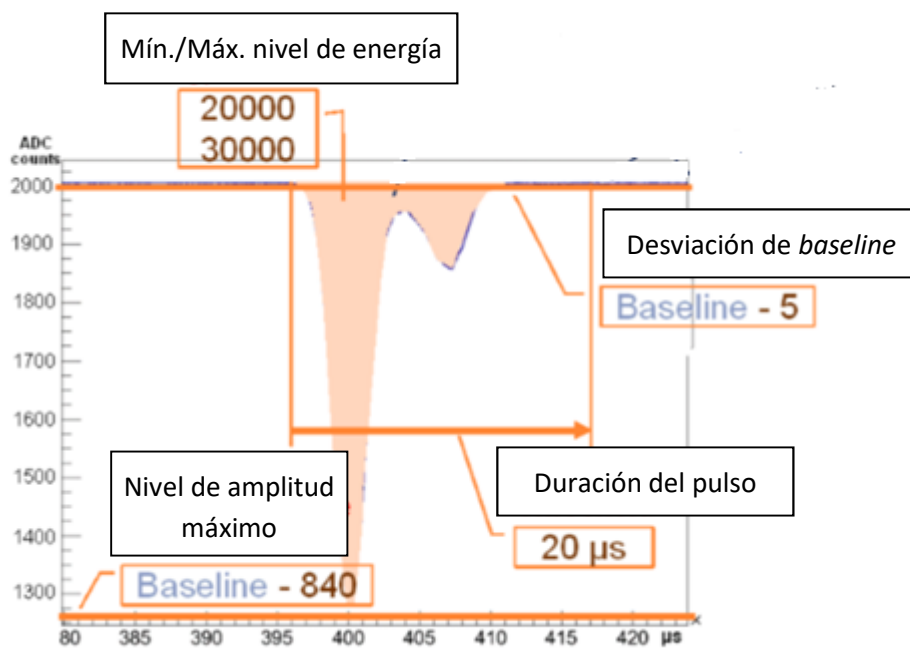
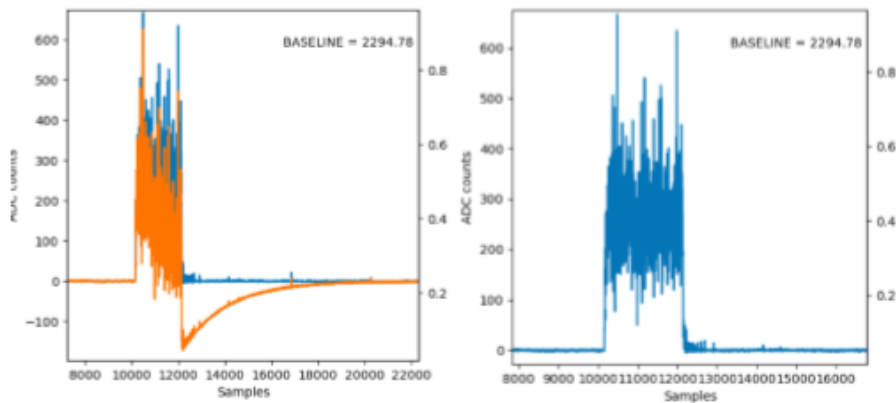


Figura 13: Calculo de candidatos de *trigger* [4]

Todos estos parámetros son definibles en el sistema de adquisición.

La señal de entrada, por razones de diseño, debe ser restaurada mediante un filtro que recupera el *baseline* de la señal, este filtro está basado en un algoritmo de restauración de nivel de base (BLR).





(a) Blue - Real input signal; Orange - FEE output signal (b) Digital baseline reconstruction applied

Figura 14: Ejemplo funcionamiento del BLR [15]

Esto es necesario para realizar una medición correcta de la energía de la señal, la cual es proporcional al área de los pulsos que componen la señal. Para un mayor detalle en el funcionamiento de este módulo consultar “*The electronics of the energy plane of the NEXT-White detector*” [15]

Conocer el *baseline* de la señal es imprescindible a la hora de cuantificar los pulsos que componen los eventos.

Lo primero que se hace es calcular el *baseline* de la señal. A continuación, se compara el valor de los datos de la señal adquirida con un margen sobre el *baseline*. Si la señal supera dicho margen, empieza a sumar el valor de energía de la señal (sumatoria de muestras de la señal con relación al *baseline*) hasta que ésta vuelve a descender durante más de tres cuentas por debajo del margen. La suma se empieza tres cuentas por delante de donde se supera el margen y finaliza tras tres cuentas cuyo valor sea inferior al margen. Las sumas del valor de energía que duran menos o más de un determinado número de cuentas se descartan. Este número de cuentas define la duración para la cual un pulso se considera válido, en base a la frecuencia de muestreo, se adquiere un dato cada 25ns. Una vez finalizada la cuenta se compara ésta con los límites de energía para un candidato de *trigger* válido; si está dentro del margen establecido y no se ha superado, o no se ha alcanzado, el valor por debajo o límite de energía propuesto, se envía un candidato de *trigger* al módulo principal TRG FEC.

El módulo TRG FEC computa los candidatos recibidos y en base a un algoritmo configurable que compara la coincidencia, cantidad o distancia entre los mismos, genera una señal de *trigger*, que envía al resto de FECs del sistema para realizar una adquisición de datos.

El problema que aparece es que, debido a la forma en la cual se generan los candidatos de *trigger*, el sistema puede generar falsos candidatos de *trigger*. Estos falsos *triggers* se localizan principalmente en los flancos inicial o final de un S2, donde, tal y como muestra la figura 15, aparecen pequeños picos de energía que pueden llegar a cumplir las condiciones establecidas para un *trigger* en S1.

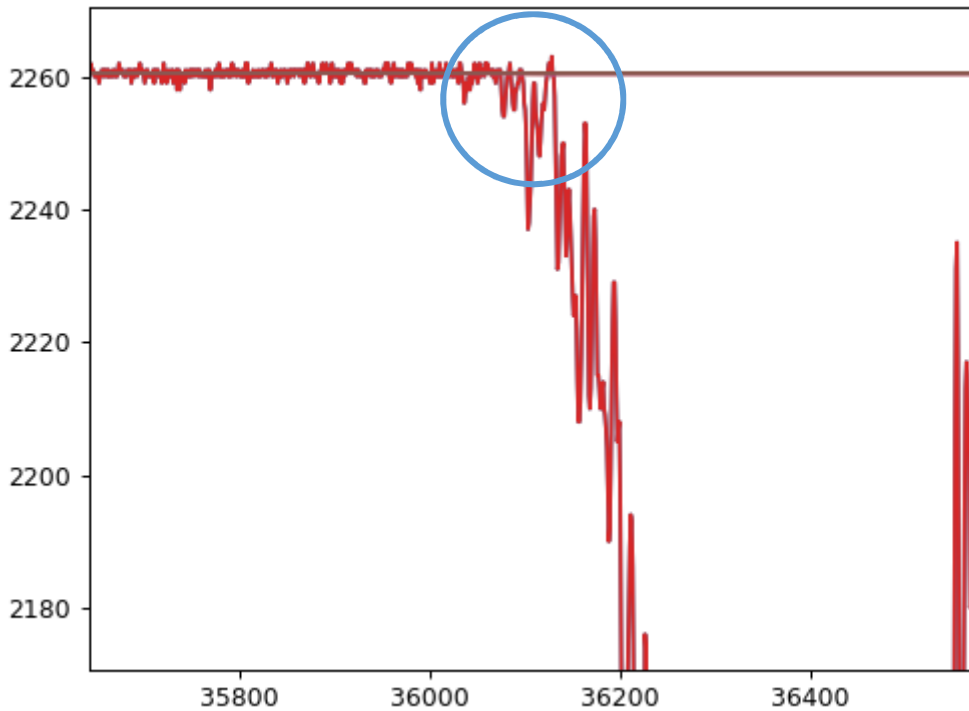


Figura 15: Posible falso *trigger* en flacos de S2

Se estudiará la implementación de un *shaper* digital entre la salida del BLR y la entrada al módulo de *trigger* con el objetivo de suavizar los pulsos y eliminar el máximo posible de estos *triggers*.

#### 4.3. Elección del *shaper*

La tipología *shaper* que se va a implementar es una variante simplificada de un *shaper* CR-RC. En concreto un *shaper* de tipo RC<sup>2</sup>. Este *shaper* prescinde de la constante CR por lo que prácticamente no afecta a la componente de baja frecuencia de la señal.

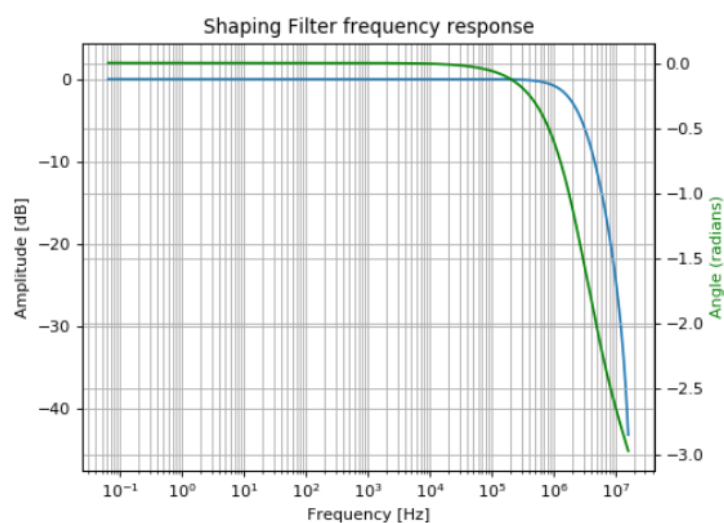


Figura 16: Respuesta de la tipología de filtro

Esto implica que, según factoricemos el *shaper*, esta frecuencia de corte se hará mayor o menor filtrando una determinada proporción del componente de alta frecuencia de la señal.

En la práctica esta tipología introduce un déficit balístico importante puesto que los valores pico de la señal se encuentran en la componente de alta frecuencia de la señal, aunque el déficit varía con la frecuencia de corte establecida. En la figura 17 podemos observar el déficit comparando el valor pico del pulso, en azul, con el de la señal tratada con *shaping*, en rojo.

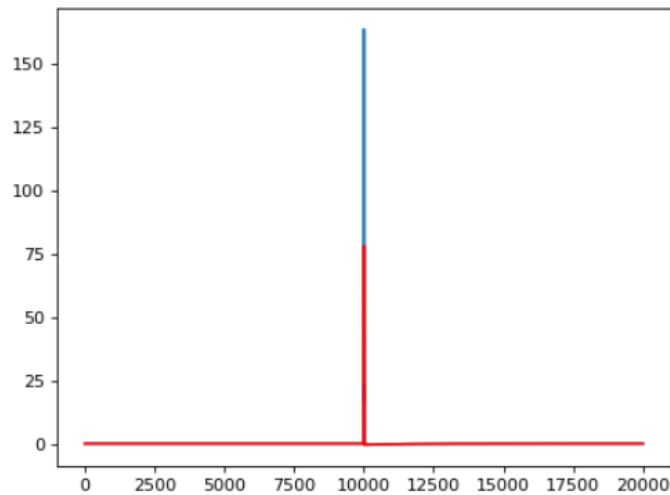


Figura 17: Ejemplo déficit balístico del *shaper*

El *shaper* se ha elegido porque es sencillo de implementar y hay antecedentes de la implementación de esta tipología realizando una función similar en el DAQ de NEXT-DEMO (prototipo del actual detector). Aunque el *shaper* implementado era analógico existía suficiente experiencia con el procesamiento de datos de las señales de NEXT-DEMO como para realizar las primeras pruebas con el mismo tipo de *shaper*. Su función de transferencia es:

$$\frac{Y(z)}{X(z)} = H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{a_0 + a_1z^{-1} + a_2z^{-2}}$$

Ecuación 1: Función de transferencia del *shaper*

Como podemos ver en la ecuación 1 el *shaper* un filtro IIR (de Respuesta Infinita al Impulso) de orden 2. Los coeficientes están por determinar con el objetivo de lograr la frecuencia de corte óptima para el funcionamiento del sistema.

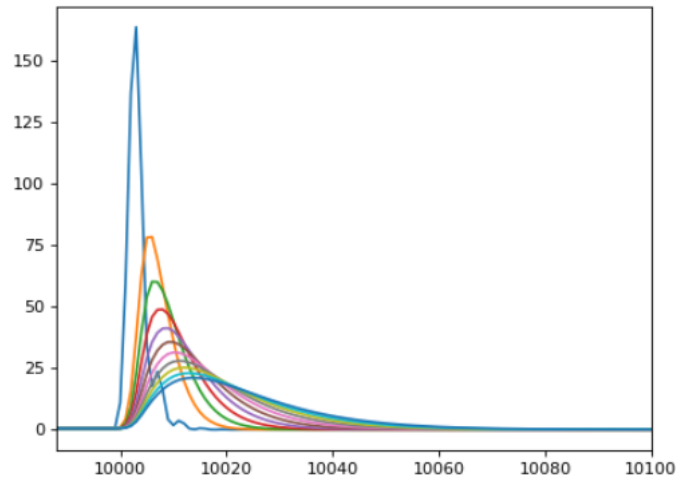


Figura 18: Barrido de coeficientes en base a factor RC

En principio los coeficientes serán función de un factor RC directamente relacionado con la frecuencia de corte del *shaper*. Como podemos observar en la figura 18, según aumenta el factor RC, el déficit balístico y la duración del pulso aumentan. Parte de del objetivo de este proyecto es determinar el conjunto de coeficientes que optimicen la duración y amplitud del pulso, tal y como ya se ha comentado, para descartar eventos en flancos de S2.

## 5. Estudio para la optimización del *trigger*

### 5.1. Introducción

Como hemos descrito antes, el funcionamiento del *trigger* está basado en el sumatorio de los valores discretizados de los pulsos de la señal, proporcional a la energía del evento, y está implementado en hardware en la FPGA del DAQ del plano EP, módulos PMT FEC.

Para estudiar los efectos que tendría la implementación del *shaper* en la generación de candidatos de *trigger*, se ha desarrollado una modelización de la implementación de hardware del algoritmo de *trigger* a nivel de análisis de señal.

A su vez, se ha modelizado el *shaper* digital que se va a implementar en el módulo de *trigger* y un módulo que diferencia los *triggers* en S2 con el objetivo de medir la eficiencia del sistema.

Se utilizarán datos ya adquiridos por el sistema en el laboratorio subterráneo de Canfranc, que es donde está instalado el detector NEXT. Los sets de datos se denominan RUNs de eventos.

Estos datos de señal adquirida se pasarán por la modelización del *shaper*, obteniendo dos juegos de datos: los originales sin *shaping* y los filtrados.

Estos dos juegos de datos se pasarán por la modelización del módulo de *trigger* y el módulo de detección de S2, con tal de obtener una comparativa de la eficiencia del sistema.

Previamente se realizará una comparación a nivel energético entre las dos señales, dado que el *shaping*, al ser un filtrado, atenúa la señal, y las configuraciones de medición de energía deberán ajustarse.

El objetivo es poder realizar este análisis para un RUN completo y, a su vez, hacer un barrido de coeficientes del *shaper* en busca de la máxima proporción de *triggers* en S2 eliminados frente a *triggers* válidos perdidos para su posterior implementación en hardware.

Por cada RUN los datos se dividen en eventos. Cada evento está formado por una ventana con las señales leídas por PMT alrededor de la posición de *trigger*, es decir, son los datos adquiridos cuando el sistema tiene un evento de *trigger* válido. Tienen una duración determinada en función de la cantidad de datos guardados; en el caso de la figura 19 es de 800  $\mu$ s, que se corresponden con 32000 muestras por señal y por evento, y un *pretrigger* de 200  $\mu$ s, 8000 muestras.

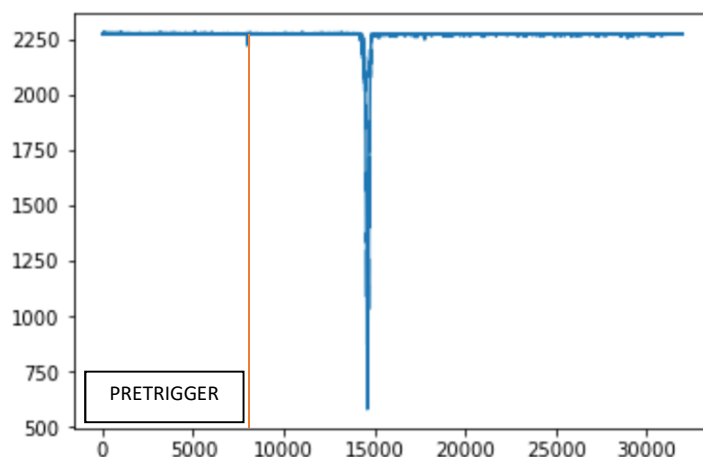


Figura 19: Ventana de adquisición

Esta ventana se corresponde a los datos adquiridos para un evento de *trigger* determinado en uno de los PTM del sistema, es decir por cada evento de *trigger* se generan tantas ventanas de señal como número de PTMS activos.

Un RUN puede contener miles de estas ventanas. Los RUNs se agrupan en ficheros de formato HDF5 que incluyen todos los datos relevantes del RUN, así como la configuración del *trigger*.

El sistema implementado debe ser capaz de identificar:

- *Trigger* válido. *Triggers* que el sistema detecta y no son en S2, es decir que cumplen los parámetros establecidos para ser candidatos de *trigger*.
- *Trigger* no válido. Hace referencia a cuando al analizar una ventana de señal no se encuentra ningún *trigger* válido o éste está fuera del punto de *trigger* de la ventana.
- *Trigger* en S2. Son los *triggers* detectados que el sistema confirma que están en un flanco de S2.

## 5.2. Herramientas

Para la modelización del sistema del *trigger* y esta parte del proyecto se va a utilizar Python 3.7 como lenguaje de programación.

Se elige Python principalmente porque es el lenguaje ya utilizado para programar las aplicaciones para el análisis de datos en el experimento y nos permite utilizar herramientas ya creadas para la gestión de los datos de eventos a utilizar. Aun así, tras el uso dado del mismo, se puede concluir que Python permite una gran flexibilidad y capacidad a la hora de analizar datos e incorpora una serie de herramientas que lo hacen tan capaz o más de realizar este tipo de análisis como otras herramientas más específicas como Matlab. Es un software libre, con una gran variedad de herramientas y librerías comunitarias.

En concreto se va a utilizar el *Kernel* de Anaconda bajo la plataforma Jupyter Notebook. Jupyter Notebooks es una herramienta que se ejecuta en una pestaña de navegador y permite crear documentos con bloques de código embebidos. Esta plataforma, frente al uso de una consola tradicional, permite exponer datos, realizar ajustes y recrear las ejecuciones de los módulos programados de una forma mucho más intuitiva y eficaz, además de ser interactiva.

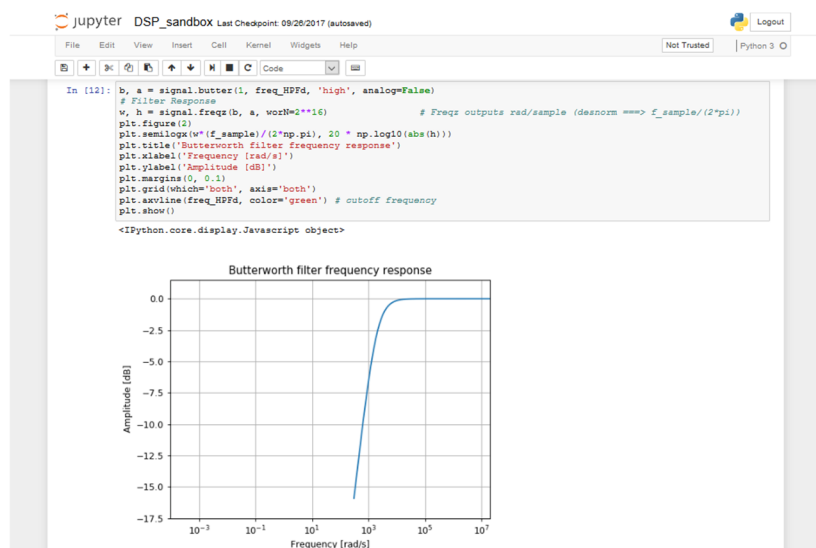


Figura 20: Ventana de Jupyter notebook

El *notebook*, figura 20, se muestra en bloques de código seguidos por los resultados de su ejecución.

Los bloques se ejecutan de forma secuencial y comparten variables, aunque puedan ejecutarse de forma independiente.

### 5.3. Formato del archivo de RUN

Como se ha mencionado anteriormente, para realizar esta primera parte se utilizarán datos ya adquiridos de RUNs en los que se sufre el problema de los falsos *triggers* en S2.

Estos RUNs vienen en un formato de datos llamado HDF5, definido internacionalmente, para un desarrollo de este formato consultar la web oficial [14].

En resumen, HDF5 es un formato de datos diseñado para almacenar y organizar gran cantidad de datos. Se basa en dos tipos de objetos: *datasets*, que son *arrays* multidimensionales de datos, y *groups*, que son agrupaciones de *datasets* y otros grupos. Esto permite crear acumulaciones de datos indexados en estructuras complejas. Sin embargo, la principal ventaja del HDF5 es que permite incorporar los metadatos correspondientes a los datos adquiridos en el mismo fichero.

Para el caso estudiado en el mismo fichero, se encuentran tanto los datos adquiridos como los parámetros y configuración de los sensores que realizaron la adquisición.

Los datos de cada RUN son agrupaciones de ventanas de adquisición. Estas ventanas corresponden a los datos volcados por un PMTs para un determinado *trigger* del sistema.

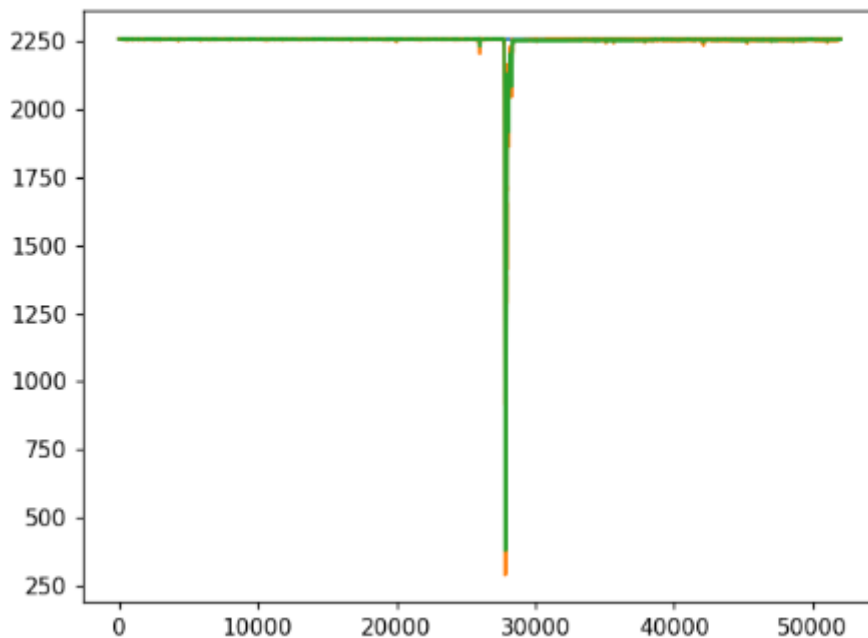


Figura 21: Ejemplo datos en ventana

Una ventana, como la mostrada en la figura 21, está compuesta por un vector de datos individuales que corresponden a los diferentes puntos adquiridos ordenados en el tiempo, cada punto dista 25ns del anterior, debido a que, como ya se ha mencionado en la sección anterior,

la frecuencia de muestreo de la señal de los PMTs es de 40 MHz. Estos datos indican un valor de energía y normalmente se encuentran invertidos para el cálculo. La ventana se genera a partir de los datos del *buffer* del DAQ a partir del tamaño de ventana definido del *trigger* y en función de la configuración de *pre-trigger*. Este *pre-trigger* marca la distancia en cuentas (tiempo) entre el inicio de la ventana y la localización del *trigger*.

Cada fichero HDF5 agrupa un número determinado de ventanas agrupadas por el PTM por el que son agrupadas según el siguiente formato:

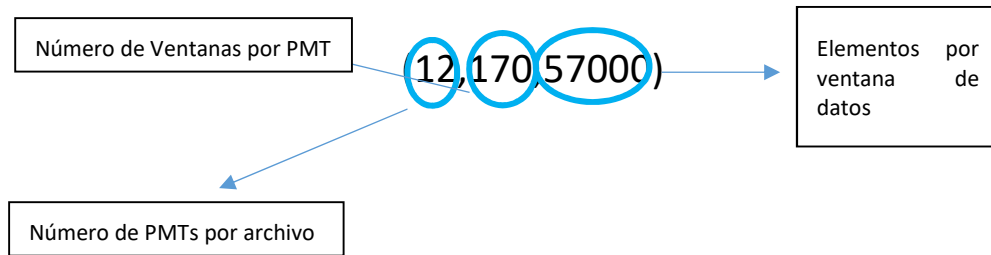


Figura 22: Formato de datos fichero HDF5

Cada RUN está formado por varios de estos archivos con correspondencia de PMTs y un determinado orden temporal.

### 5.3. Módulos de simulación del hardware

#### Bibliotecas

Antes de comenzar a describir los distintos módulos que forman la herramienta de análisis, vamos a introducir las bibliotecas y funciones utilizadas para la programación en Python:

```
In [8]: #librerias necesarias
import matplotlib
matplotlib.use('nbagg')
import cal_IO_lib as io
from DBLR import BLR_ACC3_i as BLRc
from DBLR import find_baseline as base
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image
from IPython.core.display import HTML
from scipy import signal
#%%matplotlib nbagg
#%%matplotlib inline
```

Figura 23: Bibliotecas y funciones Python

- Cal\_IO\_lib. Esta biblioteca incluye las funciones necesarias para leer y escribir ficheros HDF5.
- BLR\_ACC3. Esta función aplica la función de BLR a una ventana de datos leídos.
- Find\_baseline. Esta función calcula el *baseline* de una ventana dada.
- Numpy es la biblioteca de Python para trabajar con funciones matemáticas.
- Matplotlib y el resto de los elementos son bibliotecas para trabajar con gráficos necesarias para el desarrollo de la herramienta, pero no para su funcionamiento.



## Módulo de lectura

Como se ha descrito anteriormente, los datos que se van a utilizar para esta parte del proyecto se encuentran en formato HDF5, por lo que el primer elemento de la herramienta de análisis es una función que permite extraer y copiar los datos de esos ficheros:

```
In [12]: DATA=io.get_DATE_hdf5("C:/users/guillermo/Desktop/TFM/Pyton/dst_waves.gdcnext.000_4829.root.h5")
DATAT = DATA.transpose(0,2,1)
print(DATA.shape)
print(DATAT.shape)

(12, 52000, 452)
(12, 452, 52000)
```

Figura 24: Módulo de lectura

Este módulo adquiere los datos del fichero HDF5 y los copia a una variable interna. Se trasponen los datos con el objetivo de hacerlos más fáciles de gestionar posteriormente por el flujo de trabajo.

## Cálculo del *baseline*

El *baseline* es el valor que representa el nivel de energía de fondo leído por los PMTs en el tiempo y es necesario restárselo a la ventana para trabajar sólo con los picos de energía que provocan los efectos en la cámara.

En principio, este valor se calculaba aplicando la función *find\_baseline* que sencillamente hace una media ponderada de los valores contenidos en la ventana de señal. Esta función es adecuada para el módulo de BLR, sin embargo, para el módulo de *trigger* este método es una mera aproximación ya que el hardware que calcula el *baseline* en el módulo de *PMT* no funciona así.

El hardware que calcula el *baseline* en el DAQ del experimento lo calcula a través de un filtro de media deslizante. Cuando el valor de esta media varía en un determinado nivel del valor actual de *baseline*, éste se actualiza y se adapta a fluctuaciones del ruido fondo del sistema, y no tiene en cuenta los pulsos de señal.

```
In [10]: DATAB_aux = np.copy(DATAB)
for pmt in range(0,3):
    print(pmt)
    if pmt == 1:
        continue
    for k in range(0, a):
        if k==70:
            print('va por la mitad del pmt')
            window = 512
            thr = 5
            top= np.sum(DATAB[pmt,k][0:6000]) / 6000

            tpop=0
            zcount = 0
            topb = 0
            aux = len(DATAB[pmt,k])
            goodtop = np.array([top])
            for n in range(1, aux):
                if (n<window+1):
                    goodtop = np.append(goodtop, goodtop[n-1])
                else:
                    tpop = np.sum(DATAB_aux[pmt,k][n-window:n]) / window
                    if (DATAB[pmt,k,n] <= goodtop[n-1]+thr) and (DATAB[pmt,k,n] >= goodtop[n-1]-thr):
                        goodtop = np.append(goodtop, tpop)
                    else:
                        if (n>6):
                            goodtop = np.append(goodtop, goodtop[n-5])
                            DATAB_aux[pmt,k,n]=goodtop[n];
                        else:
                            goodtop = np.append(goodtop, goodtop[n-1])
            if k== 0:
                base = np.array([goodtop])
            else:
                base = np.append(base, [goodtop],axis=0)

    if pmt== 0:
        baseline = np.array([base])
    else:
        line = np.array([base])
        baseline = np.append(baseline, line, axis=0)
print(baseline.shape)
```

Figura 25: función *baseline*

Esta función se desplaza por las ventanas calculando el *baseline* con una media deslizante, de forma similar al módulo de hardware descrito anteriormente. Con el *baseline* de cada una de las ventanas, se crea la variable vectorial *baseline* del mismo formato que la variable DATA, la cual incluye las ventanas adquiridas por el DAQ; de forma que cada ventana dispone su correspondiente ventana de *baseline* para el cómputo de la función de *trigger*.

Esta función es, con diferencia, la que implica mayores necesidades de cómputo para el sistema teniendo el tiempo de ejecución más alto.

### Módulo de BLR

El módulo BLR, o *Baseline Restoration*, se utiliza para restaurar el *baseline* de la señal de entrada al módulo de *trigger* tras la distorsión originada por el filtrado del *frontend* de la tarjeta DAQ.

```
#extrae el numero de PMTs y Ventanas por PMTs
a=DATAT.shape[1]
b=DATAT.shape[2]
#crea DATAB del mismo formato que DATAT
DATAB = np.copy(DATAT)
# El programa avanza secuencialmente los pmt (i) y las ventanas(k)
for pmt in range (0,3):# Solo recorre los PMTs del 0 al 2 y se salta el PMT 1
    if pmt == 1:
        continue
    for k in range (0 , a): # recorre todas las ventanas en el archivo

        # Conversión de señal a señal deconvolucionada BLR
        #BLR_ACC3_1(signal_daq, coef, thr = 2, acum_FLOOR=1000, coef_clean=1, filter=False, i_factor=1
        # Invierte la señal

        f_sample = (1/25E-9); # Hz

        freq_HPF = 1/(4700*18E-9); # rad/sec
        freq_HPFd = freq_HPF / (f_sample*np.pi); # Normalized by Nyquist Freq (half-cycles/sample)

        # Valores para el CH1
        coef_CH1 = 0.00078783
        coef_cleanCH1 = 0.0000017324
        #Copia la ventana correspondiente a introducir en la funcion BLR
        DATA1 = DATAT[pmt,k]*1
        #llamada a la funcion de BLR
        signal_DBLR,acum,signal_input= BLRc(DATA1,coef_CH1,coef_cleanCH1, filter = False)

        #genera un vector de la longitud e los datos y el valor de la funcion de baseline.
        top=base(DATAT[pmt,k])
        baseline = ( np.ones(len(DATAB[pmt,k]), dtype=np.double))* top
        signal_DBLR = signal_DBLR *1 - baseline
        signal_input = signal_input *1 - baseline

        #invierte la señal
        signal_OUT = - signal_DBLR
        #Guarda la ventana procesada en su posicion correspondiente
        DATAB[pmt,k] = signal_OUT

print(DATAB.shape)
```

Figura 26: Implementación del BLR

Para aplicar el BLR se han utilizado funciones desarrolladas por el equipo técnico del experimento NEXT y se han integrado en un módulo que recorre secuencialmente las ventanas copiadas del archivo HDF5, aplicando dichas funciones con los parámetros estándar del experimento.

### Módulo de *shaper*

La base de la función de *shaper* es un filtro paso bajo IIR RC2, cuyos coeficientes son la convolución de los coeficientes de dos filtros digitales Butterworth paso bajo de orden 1 y de frecuencia crítica normalizada mediante la expresión de la ecuación 2.

$$Fec_{RCD} = \frac{1}{RC * f_{sample} * \pi}$$

Ecuación 2: Factor RC

Siendo RC el parámetro RC de la frecuencia crítica y f\_sample la frecuencia de muestreo del sistema. Para generar dichos coeficientes se utiliza la siguiente función:

```
: RC=0.1E-6
freq_RCd = 1/(RC*(f_sample)*np.pi)
#funcion para obtener los coeficientes de los filtros RC simples
bs11, as11 = signal.butter(1, freq_RCd, 'low', analog=False);
BS = np.convolve(bs11,bs11)
#BS = np.convolve(bs11,BS)
AS = np.convolve(as11,as11)
#AS = np.convolve(as11,AS)

print (BS)
print (AS)

[ 0.01246091  0.02492182  0.01246091]
[ 1.          -1.55348621  0.60332985]
```

Figura 27: Generación de los coeficientes del *shaper*

Esta función genera los coeficientes de un filtro simple para los parámetros dados y realiza su convolución para obtener los coeficientes del *shaper* para esos parámetros.

La idea es variar este parámetro para obtener el mejor conjunto de coeficientes posible.

A partir de los coeficientes generados, se aplica una función que filtra los valores de cada ventana de señal de forma secuencial.

```
for pmt in range (0,3):
    if pmt == 1:
        continue
    for k in range (0 , a):
        y = signal.lfilter(BS,AS,DATAB[pmt,k]) #funcion filtro BS y AS son los coeficientes
```

Figura 28: Función *shaper*

### Módulo de *Trigger*

Este módulo se compone de tres funciones principales:

La primera es la simulación del módulo que calcula la carga de la señal para la realización de candidatos de *trigger*.

La segunda es una simplificación del algoritmo principal de *trigger* para esta aplicación concreta, es decir, *trigger* por coincidencia de dos PMTs.

Finalmente, está el módulo que distingue *triggers* en S2 de *triggers* en S1 implementado junto al módulo de cálculo de candidatos de *trigger*.

## Cálculo de candidatos de *trigger* y detección de eventos en S2:

```
#parametros que se pueden escribir en la funcion:
#Esta espera la ventana de señal a analizar el numero de ventana y la zona de busqueda

def triggerQ(signal, nvent, fromsample, tosample, treshold = 5, pretrgwindow = 6,
            Qmin = 90, Qout = 0, Qmax = 625, thmax=300, tmin = 4, tmax = 14, pretrig=26000, pulse=3,S2I=400):

    # definicion de constantes iniciales
    time = 0
    timestamp = 0
    Q = 0
    Qcount = 0
    timeS2 = 0
    QcountS2 = 0
    trigger = False
    triggerdown = False
    trigg = False
    qtmax = False
    S2=0
    Qout=0
    trigertime2=0
    Qout2=0
    ttime2=0
    val=0
    trump = False
    counter= 0
```

Figura 29: Inicialización de la función

Esta función busca posibles *triggers* dentro de un número de cuentas en una ventana de señal y devuelve un vector con la siguiente tipología:

**([[nvent, trigertime, ttime, val, Qout],])**

- Nvent: Número de ventana de adquisición.
- Trigertime: Número de cuenta en la cual se produce el *trigger*.
- Ttime: Duración del pulso de *trigger*.
- Val: Indica si el *trigger* es válido o no.
- Qout: El total de energía medida en el pulso de *trigger*.

La función es la siguiente:

```
for k in range(fromsample, tosample): #se recorre la funcion entre estos parametros
    if trigg: #genera el array de datos devuelto en caso de trigger
        triggers = np.array([[nvent, trigertime, ttime, val, Qout, S2], ] )
        trigg = False #bajada de flag
        break
    #lectura de datos entrantes. Recorre el vector de datos de la ventana

    if signal[k] >= treshold:#empieza a sumar carga cuando la señal supera el treshold
        counter= 0
        #la suma suma carga en valor absoluto no puede ser menor que 0.
        Qcount = Qcount + abs(signal[k-pulse])#suma carga en 3 valores por delante
        time = time + 1 #conteo de duracion de pulso
        if not trigger:
            timestamp = k
            if signal[k] > thmax: #limite superado
                qtmax = True

            trigger = True

        | if signal[k] < treshold and trigger:
            Qcount = Qcount + abs(signal[k-pulse])
            counter = counter + 1
# Si no pasan tres cuentas por debajo del treshold no acaba el sumatorio
    if counter >= pulse:
        trigger = False
        Q = Qcount
        #print(timestamp,Q)
        timecount = time
        ttime = timecount
        triggerdown = True
        #counter= 0
```

Figura 30: Búsqueda y suma de carga en pulsos

```

if triggerdown: #una vez un pulso ha sido contado se evalua
    Qcount = 0
    time = 0
    if (timestamp <= pretrig+500 and timestamp >= pretrig-10): #detro de la zona de pretrigger
        trigertime = timestamp #+ timecount/2

        if timecount <= tmax and timecount >= tmin: #dentro de la duracion establecida
            if Q <= Qmax and Q >=Qmin: #dentro de la carga establecida
                if trigertime <= pretrig-pretrigwindow or trigertime >= pretrig + pretrigwindow or qtmax:
                    val=0 #se evalua como invalido si incumple por q max o esta fuera de la ventana
                    Qout = Q
                    if trigertime >= pretrig-500: #en una señal puede haber mas de un trigger
                        # Este algoritmo se asegura que el trigger devuelto es el mas cercano al pretrigger
                        if trigertime >= trigertime2:
                            trigertime2=trigertime
                            Qout2=Qout
                            ttime2=ttime

                            if trigertime >= pretrig: #si se supera le pretrigger se finaliza la busqueda
                                trigg = True
                                trump = True
                            else: #en caso de trigger valido lo marca y finaliza la funcion
                                val=1
                                trigg = True
                                trump = True
                                triggerdown = False
                                Qout = Q
                                if trigertime >= pretrig-500:
                                    if trigertime >= trigertime2:
                                        trigertime2=trigertime
                                        Qout2=Qout
                                        ttime2=ttime
                                z = trigertime

```

Figura 31: Evaluación del *trigger*

```

# Busqueda de S2 por detras del timestamp del trigger
if (timestamp + timecount + S2T>tosample):
    limitH = tosample;
else:
    limitH = timestamp + timecount + S2T; # 140*25ns=3.5us

if (timestamp - S2T < fromsample):
    limitL = fromsample;
else:
    limitL = timestamp - S2T;
S2=0
for z in range(limitL, timestamp-1):
    if signal[z] >= threshold:
        timeS2 = timeS2 + 1
        QcountS2 = QcountS2 + signal[z]
    else:
        # Si transcurre la mitad del tiempo y no ha habido pulso, se aborta
        timeS2 = 0
        QcountS2 = 0
        if (z > (limitL + (int(S2T/2))) and QcountS2==0):#+2*tmax):
            break
        # Si se detecta un pulso de mayor carga y duración que lo que se busca para un
        # se considera el inicio de un S2
        if (QcountS2 > Qmax and timeS2 > tmax):
            #y se marca
            S2=1
            break
if (S2==0): #si no se detecta s2 se busca antes del pretrigger
    for z in range(timestamp + timecount, limitH):
        #S2=0
        if signal[z] >= threshold:
            timeS2 = timeS2 + 1
            QcountS2 = QcountS2 + signal[z]
        else:
            # Si transcurre la mitad del tiempo y no ha habido pulso, se aborta
            timeS2 = 0
            QcountS2 = 0
            if (z > (limitH - (int(S2T/2))) and QcountS2==0):#+2*tmax):
                break
            # Si se detecta un pulso de mayor carga y duración que lo que se busca para
            # se considera el inicio de un S2
            if (QcountS2 > Qmax and timeS2 > tmax)
                S2=1
                break

```

Figura 32: Búsqueda de *s2*

Para identificar si un candidato es en S2, se contabiliza la señal por delante y por detrás del mismo dentro de un determinado margen, buscando un pulso cuyo valor y duración supere el límite máximo establecido para el *trigger*. En caso de encontrarlo se marca, si no se encuentra o no cumple estos parámetros se concluye que el *trigger* no es en S2.

```

-----
# en caso de que el trigger sea no valido por tiempo o carga se evalua cuando se deja de buscar
# esto ocurre al superar el punto de pretrigger

    else:

        val=0
        Qout = Q
        if trigertime >= pretrig-500:
            if trigertime >= trigertime2:
                trigertime2=trigertime
                Qout2=Qout
                ttime2=ttime

            if trigertime >= pretrig:
                trigg = True
                trump = True
                trigertime=trigertime2
                Qout=Qout2
                ttime=ttime2

        else:
            val=0
            Qout = Q
            if trigertime >= pretrig-500:
                if trigertime >= trigertime2:
                    trigertime2=trigertime
                    Qout2=Qout
                    ttime2=ttime

            if trigertime >= pretrig:
                trigg = True
                trump = True
                trigertime=trigertime2
                Qout=Qout2
                ttime=ttime2

#reseteo de valores
Q = 0
timecount = 0
qmax = False
trigerdown = False
#fin de la función y configuración del retorno
if not trump:
    trigertime=trigertime2
    Qout=Qout2
    ttime=ttime2
    triggers = np.array([[nvent, trigertime, ttime, val, Qout, S2],] )

return triggers

```

Figura 33: Cierre y retorno del algoritmo de evaluación de *trigger*

### **Trigger por coincidencia y evaluación de resultados:**

Esta función engloba a la anterior y se encarga de acumular los resultados de las ventanas que se van pasando por la función anterior, generando un vector que contiene los datos para todo el RUN.

A partir de dichos resultados evalúa los *triggers* por coincidencia, los *triggers* en S2 y los *triggers* inválidos.

```

def Run_trigguer(DATAB, DATAF,baseline):
    #procesado los datos originales
    for pmt in range (0,3):
        if pmt == 1:
            continue
        #inicializa el vector con los datos de trigguer el primer valor es todo 0
        triguers=np.array([[0,0,0,0,0,0,0,1] ])
        for k in range (0 , a):
            if pmt==3 or pmt ==2: #ajuste para que la coincidencia entre baseline y ventana sea correcta
                STRIFA = -(DATAB[pmt,k]) + baseline[pmt-1,k]
            else:
                STRIFA = -(DATAB[pmt,k]) + baseline[pmt,k]
            tru = triggerQ(STRIFA, k, 1,len(STRIFA))
            triguers=np.append(triguers, tru , axis = 0)
        if pmt ==0:
            trigl=np.array([triguers] )
        else:
            trigall = np.array([triguers] )
            trigl = np.append(trigl, trigall , axis = 0)
    #procesado los datos con shaping
    for pmt in range (0,3):
        if pmt == 1:
            continue
        triguers=np.array([[0,0,0,0,0,0,0,1] ])
        for k in range (0 , a):
            if pmt==3 or pmt ==2:
                STRIFA = -(DATAF[pmt,k]) + baseline[pmt-1,k]
            else:
                STRIFA = -(DATAF[pmt,k]) + baseline[pmt,k]
            tru = triggerQ(STRIFA, k, 1,len(STRIFA), Qmin = 70 ,tmax = 40, S2T=400, pretrgwindow= 25)
            triguers=np.append(triguers, tru , axis = 0)
        if pmt ==0:
            triglF=np.array([triguers] )
        else:
            trigallF = np.array([triguers] )
            triglF = np.append(triglF, trigallF , axis = 0)

```

Figura 34: Generación de vector con datos de *trigger*

```

# Trigger por coincidencia tambien evalua el total de trigguers validos e invalidos

#ajuste de la forma del vector de resultados
TRIGDATA = trigl.transpose(1,0,2)
#inicializacion de valores
shape=TRIGDATA.shape
nv=shape[0]
npmt=shape[1]
Tval = 0 #trigguer validos
Tnval = 0 #trigguer invalidos
Tvals2 = 0 #trigguer validos en S2
coinW=5
#esta funcion compara coincidencias que haya un trigger valido y que el time stamp coincida en un cierto margen
for k in range (1,nv):
    ts1=0
    ts2=0
    for pmt in range (0 , npmt):
        for pmtb in range (pmt+1 , npmt):
            if TRIGDATA[k,pmt,0] == TRIGDATA[k,pmtb,0]:
                #coincidencia valido
                if TRIGDATA[k,pmt,3] == 1 and TRIGDATA[k,pmt+1,3] == 1 :
                    if TRIGDATA[k,pmt,5] == 0 :
                        if TRIGDATA[k,pmt,1] >= TRIGDATA[k,pmtb,1]-coinW and TRIGDATA[k,pmt,1] <= TRIGDATA[k,pmtb,1]+coinW:
                            ts1=ts1+1
                #coincidencia en S2
                else:
                    if TRIGDATA[k,pmt,1] >= TRIGDATA[k,pmtb,1]-coinW and TRIGDATA[k,pmt,1] <= TRIGDATA[k,pmtb,1]+coinW:
                        ts1=ts1+1
                        ts2=ts2+1
    #print (ts1)
    if ts1 >= 1: #si coinciden mas de un numero determinado da trigguer valido
        #print(TRIGDATA[k,pmt,0], 'trigger valido')
        Tval = Tval+1
    if ts2 >= 1: #si coinciden mas de un numero determinado da trigguer en S2
        Tvals2 = Tvals2+1
    else:
        Tnval = Tnval+1
#si no coincide trigguer invalido
Tnval = Tnval+1

```

Figura 35: *Trigger* por coincidencia

Esta función procesa el vector de resultados, de forma que se comparan los resultados de cada ventana con los resultados de esa misma ventana en otros PMTs.

Si el número de coincidencias es mayor al configurado, se determina un *trigger* válido en esa ventana, de forma análoga se determina si es en S2 o no.

Hay una función gemela que evalúa los datos con *shaping*.

La tercera parte del módulo evalúa los resultados obtenidos.

```
# representacion de resultados

print('El numero de triggers validos es',Tval,'el numero de triggers validos en s2 es',Tvals2,'el numero de tri
print('El numero de triggersF validos es',TvalF,'el numero de triggers validos en s2 es',Tvals2F,'el numero de

# estadistica de los resultados obtenidos

pVal = ((Tval+Tvals2)/nv)*100
PS2 = (Tvals2 / (Tval+Tvals2))*100
Pinval = ( Tnval /nv ) *100
pValF = ((TvalF+Tvals2F)/nv)*100
PS2F = (Tvals2F / (TvalF+Tvals2F))*100
PinvalF = ( TnvalF /nvF ) *100
print('El % de triggers validos es',pVal,'% de los cuales son en s2',PS2, '% el % de triggers invalidos es',Pin
print('El % de triggers filtrados validos es',pValF,'% de los cuales son en s2 el',PS2F,'%el % de triggers inva
print('La perdida de triggers validos es del',((1-(TvalF/Tval))*100),'% La eliminacion de triggers invalidos es
dtrigg = np.array([(1-(TvalF/Tval))*100],[(1-(Tvals2F/Tvals2))*100])
return dtrigg
```

Figura 36: Presentación de resultados

Esta función cuenta el número de ventanas con *triggers* válidos, válido en S2 e inválidos tanto en los datos originales como en los sometidos a *shaping*, y genera un texto de resultados, como el de la figura 37, con tal de evaluar el rendimiento del *shaping*.

```
El numero de triggers validos es 121 el numero de triggers validos en s2 es 29 el numero de triggers total es de
150 el numero de triggers invalidos es 20
El numero de triggersF validos es 120 el numero de triggers validos en s2 es 26 el numero de triggers total es d
e 146 el numero de triggers invalidos es 24
El % de triggers validos es 87.71929824561403 % de los cuales son en s2 19.333333333333332 % el % de triggers in
validos es 11.695906432748536
El % de triggers filtrados validos es 85.38011695906432 % de los cuales son en s2 el 17.80821917808219 %el % de
triggers invalidos es 14.035087719298245
La perdida de triggers validos es del 0.8264462809917328 % La eliminacion de triggers invalidos es del 10.3448275
86206895 %
```

Figura 37: Resultados finales de la función de *trigger*



## Módulo principal

Este módulo es el que lleva la iteración principal del barrido de coeficientes y ejecuta los módulos anteriores.

```
DATAF = np.copy(DATAT)
RCstart=0.01E-6 #valores maximo y minimo del barrido de coeficientes
RCend=0.2E-6
Steps=100 # numero de iteraciones
dtriggT = np.array([[0,0],1]) #inicializacion del vector de resultados final
for i in np.linspace(RCstart,RCend,Steps): # Shaper, se ejecuta en funcion del barrido
    RC=i
    f_sample = (1/25E-9); # Hz
    freq_RCd = 1/(RC*f_sample*np.pi)
    bs11, as11 = signal.butter(1, freq_RCd, 'low', analog=False);
    BS = np.convolve(bs11,bs11)
    G = BS[0]
    AS = np.convolve(as11,as11)
    print ('G = ',G)
    print ("A = ",AS)
    print ("Freq = ",1/(RC*2*np.pi))
    print ("RC = ",RC)
    print ("")
    for pmt in range (0,3):
        if pmt == 1:
            continue
        for k in range (0 , a):
            y = signal.lfilter(BS,AS,DATAB[pmt,k]) #funcion filtro BS y AS son los coeficientes
            for z in range(100):
                if pmt==3 or pmt ==2:
                    y[z]= baseline[pmt-1,k,z]
                else:
                    y[z]= baseline[pmt,k,z]
            DATAF[pmt,k] = y
    #llamada a funcion de analisis de trigger
    dtrigg=Run_trigguer(DATAB,DATAF,baseline)
    #genera el vector de resultados con porcentajes de trigger validos perdidos y triggwers en s2 eliminados
    dtriggT=np.append(dtriggT,[dtrigg], axis=0)
```

Figura 38: Módulo principal

## 5.4. Simulación y resultados

### INTRODUCCIÓN

Para elegir el mejor conjunto de coeficientes se ha procesado un RUN completo con las herramientas presentadas anteriormente y se han evaluado los resultados mediante el uso de gráficas de tendencia, con tal de observar cómo la variación de coeficientes afecta a la eliminación de *triggers* en S2 y a la pérdida de *triggers* válidos.

Como ya se ha mencionado, el objetivo de esta parte del proyecto es optimizar al máximo posible el rendimiento de la función de *shaping*.

No se busca tanto eliminar el máximo posible de *triggers* en S2 sino eliminar todos los posibles manteniendo las pérdidas de *triggers* evaluados como válidos lo más bajos posibles en el orden de entre un 10% y un 15%.

La función descrita anteriormente se ejecutará secuencialmente de forma manual para todos los ficheros HDF5 del RUN el cual se compone de entre 6 y 20 de estos archivos. Las razones de hacerlo mediante un proceso manual y no una función que gestione varios archivos es, principalmente, el tiempo de procesado; esta función tarda en ejecutarse aproximadamente 4 horas. Para ejecutar el RUN completo poniendo como ejemplo el RUN analizado con 6 archivos de datos, sumaría aproximadamente 24 horas, un tiempo muy excesivo para los recursos informáticos disponibles.

Por lo tanto, se ejecutó la función para cada uno de los archivos por separado y, posteriormente, se juntaron los resultados obtenidos mediante el uso de la herramienta Excel de Microsoft.

#### RUN 4707

El nombre de los RUNs viene sencillamente del orden de ejecución de éstos; el RUN 4707 es el RUN utilizado para buscar el juego de coeficientes más adecuado para el *shaper*. Este RUN esta realizado a propósito para ejemplificar los problemas de falsos *triggers* en flancos de S2.

La configuración del *trigger* para el RUN es la siguiente:

Configuración de <i>Trigger</i>												
Parámetros globales			<i>Trigger</i> interno									
Máscara	Frecuencia	Auto Trig externo	<i>Trigger</i> doble			Máx. tiempo <i>trigger</i> A/B	Eventos <i>trigger</i> A		Eventos <i>trigger</i> B			
ON	10	OFF	OFF			0	2		1			
Configuración <i>trigger</i> A Canal PMTS												
On/Off	Inversión Polar	<i>Trigger</i> Q	Auto BS	TRG B	<i>Baseline</i> deviation	Máx. A	Q mín.	Q máx.	Máx. time	Mín. time	Pulse valid	
CH18	1	0	1	1	0	5	300	100	625	350(ns)	100(ns)	50(ns)
CH19	1	0	1	1	0	5	300	100	625	350(ns)	100(ns)	50(ns)

Tabla 1 RUN 4707

Estos parámetros indican que el sistema genera un *trigger* cuando detecta dos eventos de *trigger* tipo A coincidentes, concretamente los correspondientes a los canales 18 y 19, donde se definen los parámetros de *trigger*.

Para saber qué PMTs del fichero HDF5 corresponden con los canales 18 y 19, hay que ver los metadatos del fichero HDF5.

	channel	sensorID	position
0	18	0	-23.9414...
1	22	1	-44.9951...
2	19	2	68.9365, ...
3	23	3	-0.0, 185...
4	9	4	-118.916...

Figura 39: Identificación de canales

Como podemos observar en la figura 39 los canales 18 y 19 corresponden con el PMT 0 y el 2 respectivamente.

## ANÁLISIS DE LOS EFECTOS DEL SHAPING Y AJUSTE DE PARÁMETROS DE TRIGGER

El *shaping* implica, como se ha mencionado anteriormente, una atenuación de la señal, por lo que será necesario aplicar un factor de corrección a parámetros del algoritmo de generación de candidatos de *trigger*. Para estimar el porcentaje de energía perdida se ha utilizado una aplicación que comparte muchos elementos con la descrita anteriormente.

```
In [7]: #Esta funcion calcula la carga por encima del baseline para filtrada y sin filtrar y las compara
#luego las agrupa y las cuenta segun varien en menos de 2 , 20 o 40% o > 40%
for pmt in range (0,3):
    if pmt == 1:
        continue
    print('Data del FMT',pmt)
    triggers=np.array([[0,0,0,0,1] ])
    #triggers = np.array([[invent, trigertime, s2, val],])
    dosval= 0
    veinteval= 0
    cuarentaval= 0
    inval= 0

    for k in range (0 , a):
        q=0
        qf=0
        aux = len(DATAB[pmt,k])
        top= np.sum(DATAB[pmt,k][0:6000]) / 6000
        baseline = ( np.ones(len(DATAB[pmt,k])), dtype=np.double))* top
        signal = -(DATAB[pmt,k]) + baseline
        signalf = -(DATAF[pmt,k]) + baseline
        #plt.figure(k)
        #plt.plot(signal)
        #plt.plot(signalf)
        #plt.show()
        for n in range(0,aux):
            if signal[n] >=10:
                q = q + signal[n]

            if signalf[n] >=10:
                qf = qf + signalf[n]
        porc = ((q-qf)/q)*100
        #print('ventana',k, 'q',q, 'qfilt',qf, 'porc',porc)
        if porc >=40 or porc <= -40:
            plt.figure(k)
            plt.plot(DATAB[pmt,k])
            plt.plot(DATAF[pmt,k])
            plt.show()
            inval= inval +1
        if (porc >=0 and porc <=2) or (porc <= 0 and porc >=-2):
            dosval= dosval+1

        if (porc >2 and porc <=20) or (porc < -2 and porc >=-20):
            veinteval= veinteval+1

        if (porc >20 and porc <40) or (porc < -20 and porc >=-40):
            cuarentaval= cuarentaval+1
    print('numero de coincidencias', '2%', dosval, '20%', veinteval, 'cuarentaval', cuarentaval, 'inval', inval )
```

Figura 40: Función de análisis de variación de carga

Hay un porcentaje de ventanas en las que salta el *trigger* real, puesto que tenemos una adquisición, pero el *trigger* modelizado que se utiliza en este análisis no sé activa. El porcentaje, como observaremos cuando se presenten los resultados, es de entre un 5 % y un 25%. En la mayoría de los casos estos pulsos son de una tipología que no coincide con la tipología S1, S2 buscada. Los resultados de estas ventanas se marcan como inválidos. Esto se debe a las diferencias que puede haber entre los algoritmos implementado off-line y los implementados en el hardware de adquisición, cuyas limitaciones suelen estar en la limitación del número de bits que se pueden usar para realizar operaciones aritméticas y que hacen que dichos algoritmos tengan una muy buena precisión, pero limitada si la comparamos con el número de bits utilizado por los formatos de datos usados en el software de procesamiento disponible en un ordenador.

Esta función compara la carga por encima del *baseline* para cada ventana y las clasifica, según si la diferencia es del 2%, el 20%, el 40% o directamente inválida. En definitiva, al pasar datos por la función anterior obtenemos el siguiente ratio: el 70% presenta una diferencia menor al 2%;

en el 14.7% la diferencia es de un máximo de un 20%; en el 5% presenta una variación hasta el 40%; y un 10.3 % se consideran inválidos al diferir en un 40% o más.

En base a estos resultados y observaciones directas sobre las ventanas de señal, este porcentaje de inválidos es en su mayor parte de ventanas que, por su tipología, distan de la configuración S1 + S2 buscada. Son ventanas que el *trigger* modelizado ya da como inválidas, por lo que se obviarán de las estimaciones.

Así pues, y debido al shaping, se tiene que en la mayoría de los casos la distorsión energética es mínima, pero hace falta una corrección de los parámetros de energía que desencadenan un candidato de *trigger* con tal de adecuar esta pérdida de energía.

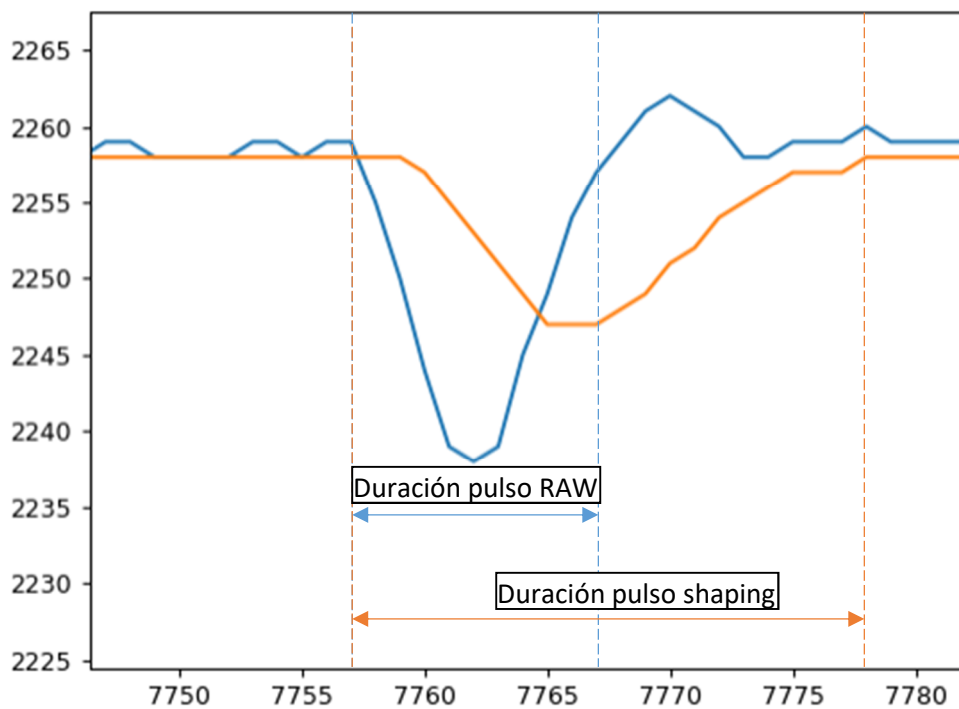


Figura 41: s1 señal filtrada frente a señal raw

Como se observa en figura 41, también se produce un alargamiento en la duración del pulso. Este alargamiento no afecta al comienzo de este, pero sí a su duración, en general la duración del pulso se dobla. Debido a esto, es necesario compensar fuertemente los límites de duración del pulso para evitar pérdidas de *triggers* identificados como válidos.

En definitiva, los cambios aplicados para los parámetros para este RUN son:

Señal	Qmin	Tmax
Sin Shaping	90	14
Con Shaping	70	40

Tabla 2: variación condiciones de *trigger*

## Resultados

Los resultados son los siguientes:

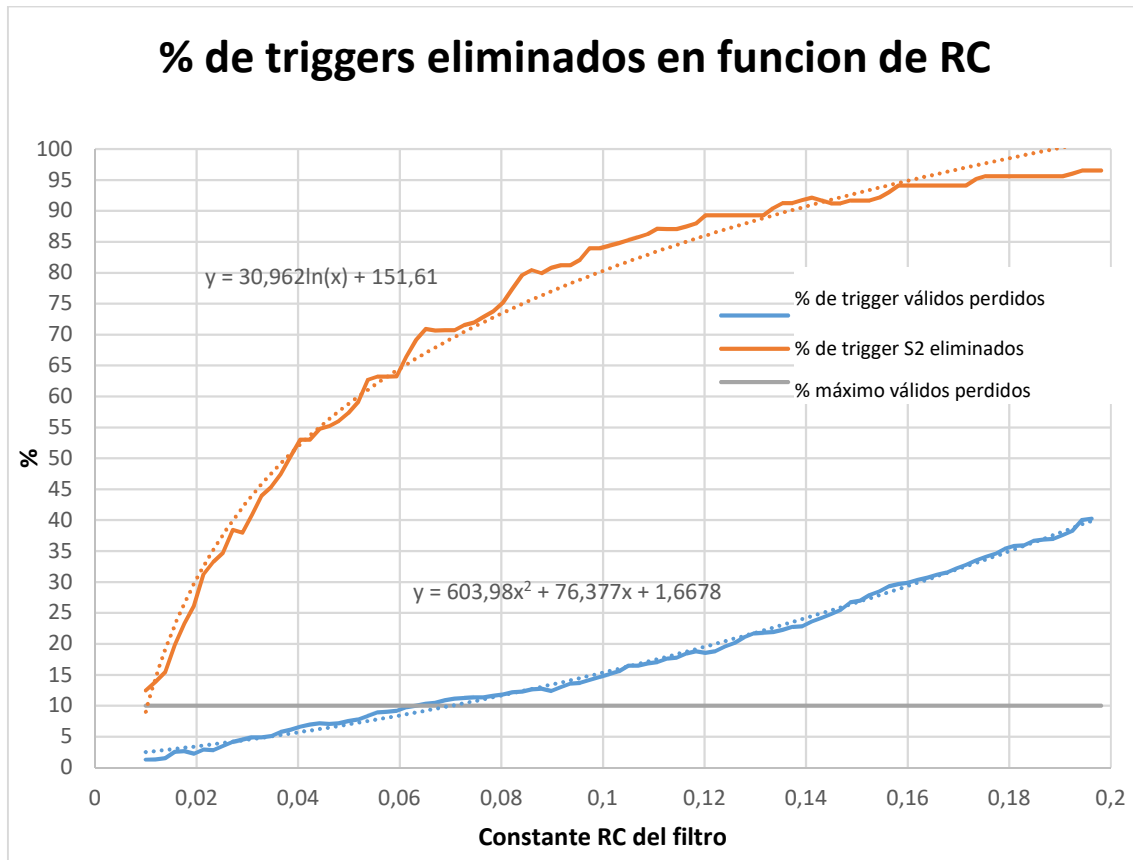


Figura 42: Resultados del barrido de coeficientes

La figura 42 indica el porcentaje de *triggers* en S2 eliminados y *triggers* válidos perdidos frente al valor de la constante RC del *shaper*. Según aumenta la constante, la distorsión de la señal se hace mayor y, por lo tanto, en ambos casos el número de *triggers* se reduce. Sin embargo, observamos que la tendencia para los *triggers* en S2 inválidos eliminados es logarítmica, por lo que se puede buscar optimizar el nivel de RC.

Otro criterio para tener en cuenta es el porcentaje máximo de *triggers* válidos perdidos. Se estima, en las condiciones de diseño, que las pérdidas máximas no han de superar más de un 10%, y este criterio es más limitante que el punto óptimo entre *triggers* válidos perdidos y *triggers* en S2 eliminados.

Por lo tanto, se concluye que los coeficientes óptimos serán aquellos que rondan entre el 0,06 y 0,07 de factor RC, el cual equivale a un 10% de pérdidas y aproximadamente un 70% de *triggers* en S2 eliminados, que ya es un porcentaje realmente elevado.

## 6. Implementación

### 6.1. Introducción

Como hemos visto anteriormente, la base del DAQ del experimento se basa en implementación de circuitos digitales en FPGAs.

La introducción en el sistema del *shaper* se basaría en la incorporación de éste en un módulo independiente con entrada de señal de reloj, *reset* y la señal de BLR del *trigger* como entrada. Este módulo trabajaría con coma fija y se podría deshabilitar o habilitar con una señal de ENABLE si se considera necesario para el RUN o no.

La implementación se divide en tres partes:

- Un estudio de la cuantificación del filtro, indispensable para estimar su comportamiento como circuito digital con una precisión finita. En este apartado se definirá el número de bits necesarios para la implementación.
- El diseño en sí del circuito digital en base a técnicas de implementación digital que consistirá en desarrollar los elementos del circuito y luego agruparlos en un módulo listo para implementarse en el circuito actual del experimento.
- Finalmente, antes de la implementación real se verificará el diseño, no solo en atención al resultado final, sino también comprobado la ausencia de desbordamientos.

La implementación del módulo se realizará en Verilog utilizando el software de Xilinx para la implementación y Modelsim para simulación y verificación de funcionamiento.

### 6.2. Implementación del *shaper*

#### 6.2.1. Forma digital del *shaper*

Para la implementación de la forma digital del *shaper* se han utilizado las metodologías de presentes en “*Understanding Digital Signal Processing*” [11].

La función de transferencia asociada al *shaper* a implementar es la siguiente:

$$\frac{Y(z)}{X(z)} = H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 + a_1z^{-1} + a_2z^{-2}}$$

Ecuación 3: Función de transferencia del *shaper*

Dado que en la sección anterior se ha concluido que los coeficientes óptimos son los que están entre el 0,06 y 0,07 de factor RC, los coeficientes son:

$$\begin{bmatrix} b_0 & b_1 & b_2 \\ a_0 & a_1 & a_2 \end{bmatrix} = \begin{bmatrix} 0.0207362 & 0.0414724 & 0.02073621 \\ 1 & -1.42399144 & 0.50693791 \end{bmatrix}$$

Ecuación 4: coeficientes del *shaper*

También se puede expresar como:

$$Y[n] = b_0X[n] + b_1X[n - 1] + b_2X[n - 2] - a_1X[n - 1] - a_2X[n - 2]$$

Ecuación 5: Expresión discretizada del *shaper*

Aplicando la forma directa II a esta expresión con las correlaciones:

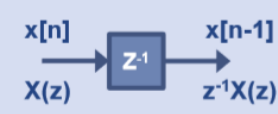
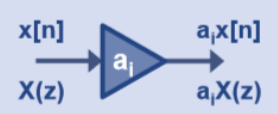
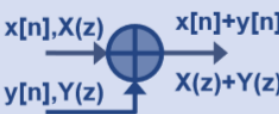
Retardo	Multiplicación	Suma
$x[n]$ $X(z)$ 	$x[n]$ $X(z)$ 	$x[n], X(z)$ $y[n], Y(z)$ 

Figura 43: Conversiones implementación directa [12]

Obtenemos la red digital siguiente:

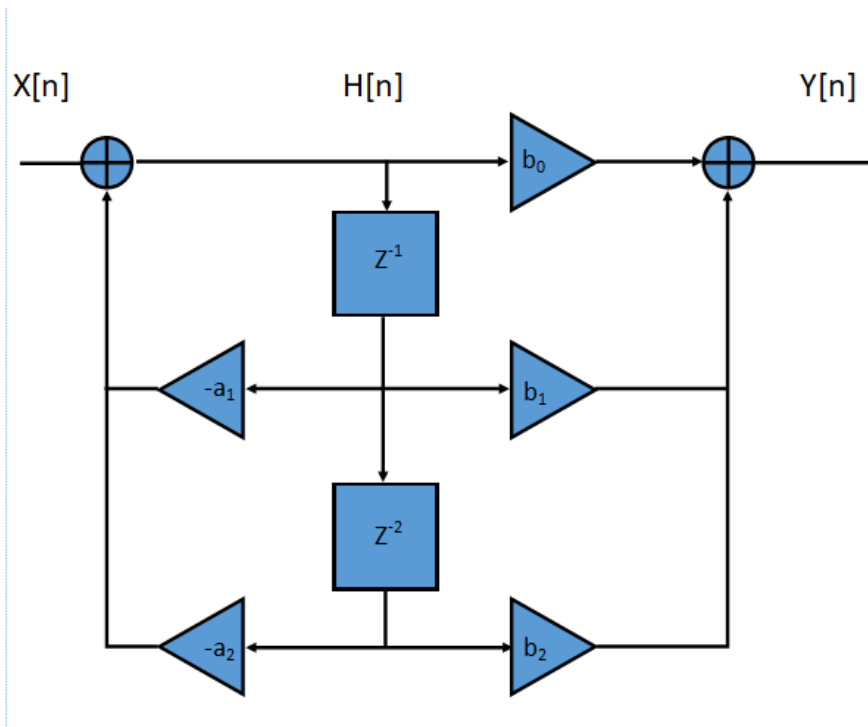


Figura 44: Red digital shaper

Teniendo en cuenta la configuración de coeficientes, podemos observar que los coeficientes  $b$  siguen la forma:

$$g * [1 \ 2 \ 1]$$

Ecuación 5: Simplificación de coeficientes

Con:

$$g = 0.0207362$$

Ecuación 6: valor  $g$  proporcional

Podemos, a consecuencia de esto, reducir a un más la red digital:

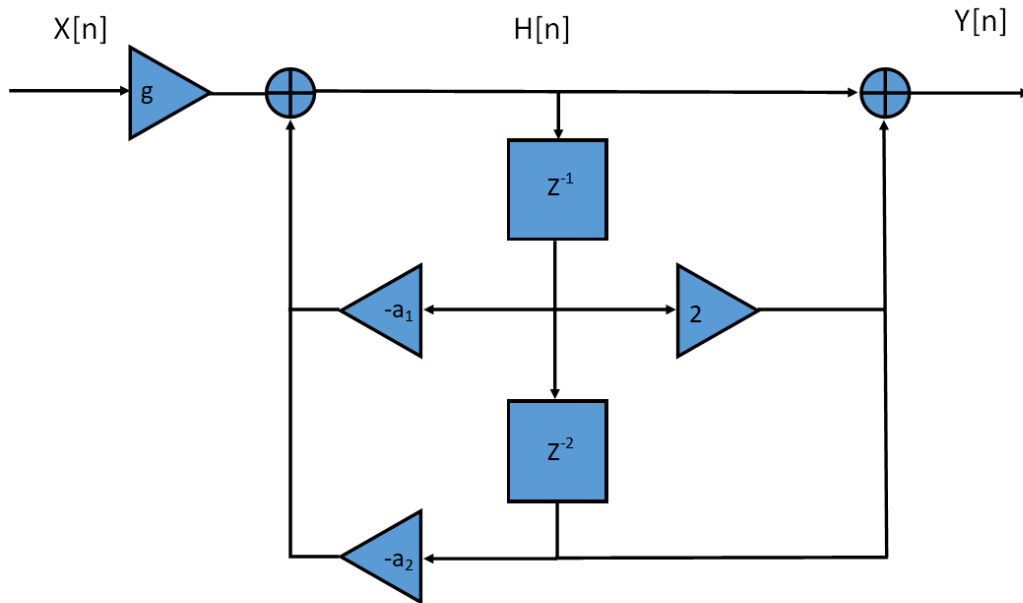


Figura 45: red digital *shaper* simplificada.

Esta red será la que implementemos mediante el lenguaje de descripción de hardware Verilog, qué se describe más adelante.

### 6.2.2. Estudio de cuantificación

Antes de implementar la red se ha de estudiar su comportamiento cuantificado, con el objetivo de dimensionar el número de bits del sistema y, así, alcanzar el nivel de precisión deseada y evitar el desbordamiento de los datos.

Para realizar este análisis se utilizó Matlab, ya que dispone de las herramientas y funciones necesarias para ejecutar este tipo de análisis.

Para aplicar la red digital en Matlab utilizaremos su función de transferencia asociada:

$$\begin{cases} H[n] = gX[n] - a_1H[n - 1] - a_2H[n - 2] \\ Y[n] = H[n] + 2H[n - 1] + H[n - 2] \end{cases}$$

Ecuación 7: función de transferencia del *shaper*

Con los coeficientes:

$$\begin{aligned} g &= 0.0207362 \\ a_1 &= -1.42399144 \\ a_2 &= 0.50693791 \end{aligned}$$

Ecuación 8: coeficientes finales del *shaper*



En primer lugar, hay que tener en cuenta el formato de la señal de entrada al sistema desde el BLR, esta tiene un tamaño de palabra de 42 bits, *signed* y coma fija con una parte decimal de 20 bits.

La salida que ira al módulo de *trigger* tiene que tener un formato de 12 bits sin signo.

Para garantizar la precisión del *shaper* cuantificado hay que garantizar la representación de los coeficientes, los coeficientes han de ser de formato *signed* y el mayor de todos tiene 9 dígitos de precisión se necesita una precisión de 0.00000001. Esto se traduce que en formato *signed* el número de bits decimales necesarios es de 28 como mínimo.

Teniendo esto en cuenta, se ajustará el tamaño de palabra en el *shaper* a 32 bits, tanto para los datos como para los coeficientes, el formato será de coma fija "2,30" bits, 2 bits de parte entera y 30 bits decimales. El usar 32 bits nos garantiza una precisión superior a la mínima indispensable y como el valor absoluto de los coeficientes es mayor que 1 hace falta más de un bit entero.

Ahora se procederá a estudiar si este formato se comporta como toca y no se producen desbordamientos

Primero se evaluará en MATLAB la respuesta ante un impulso y un escalón del *shaper* cuantificado, con tal de compararla con el *shaper* sin cuantificar.

La función de Matlab es la siguiente:

```

%declaracion de coeficientes
Freq = 2122065.90789;
ws=Freq;
wny=ws/2;
RC = 7.5e-08;
b = [0.0273662 0.04147323 0.02073662];
a = [1 -1.42399144 0.50693791];
g=0.02073662;
bl = [1 2 1];
al = [1 -1.42399144 0.50693791];
%Creacion de funcion de transferencia
fsh = filt(b,a);
% Cuantificacion de los coeficientes
bit_c=32;
bit_d=20;
b_c=fi(b,1,bit_c,bit_d,'ProductWordLength',32,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
a_c=fi(a,1,bit_c,bit_d,'ProductWordLength',32,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
bl_c=fi(bl,1,bit_c,bit_d,'ProductWordLength',32,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
g_c=fi(g,1,bit_c,bit_d,'ProductWordLength',32,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
al_c=fi(al,1,bit_c,bit_d,'ProductWordLength',32,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
%Creacion de funcion de transferencia cuantificada
fsh_c = filt(b_c,a_c);
%Filtor con dfilt cuantificado y sin cuantificar y comparacion
fsh1=dfilt.df2(b,a);
fsh2=dfilt.df2(b_c,a_c);
fsh3=dfilt.df2sos(bl_c,al_c,g_c);
fvtool(fsh1,fsh2,'Fs',ws);

```

Figura 46: Función respuesta del *shaper*

Esta función nos devuelve la respuesta ante escalón e impulso de entrada máximos, buscamos comprobar que la respuesta de la función entra dentro de los valores límites para el tamaño de palabra y formato elegido y evitar problemas de desbordamiento.

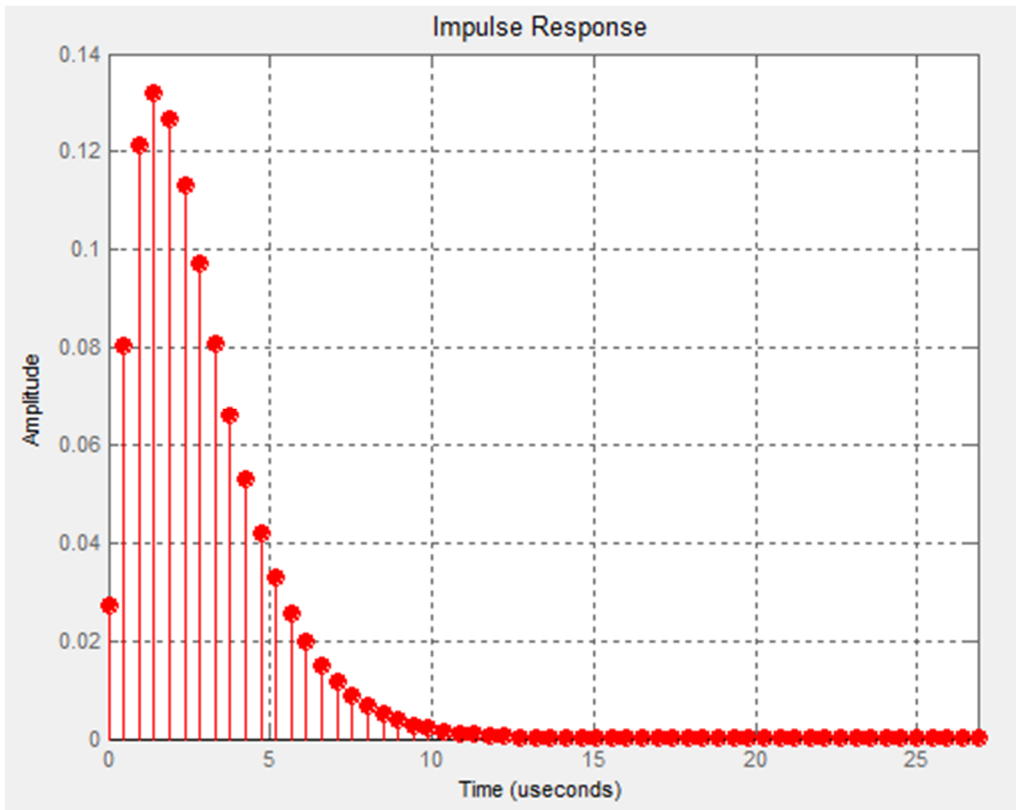


Figura 47: Respuesta ante impulso

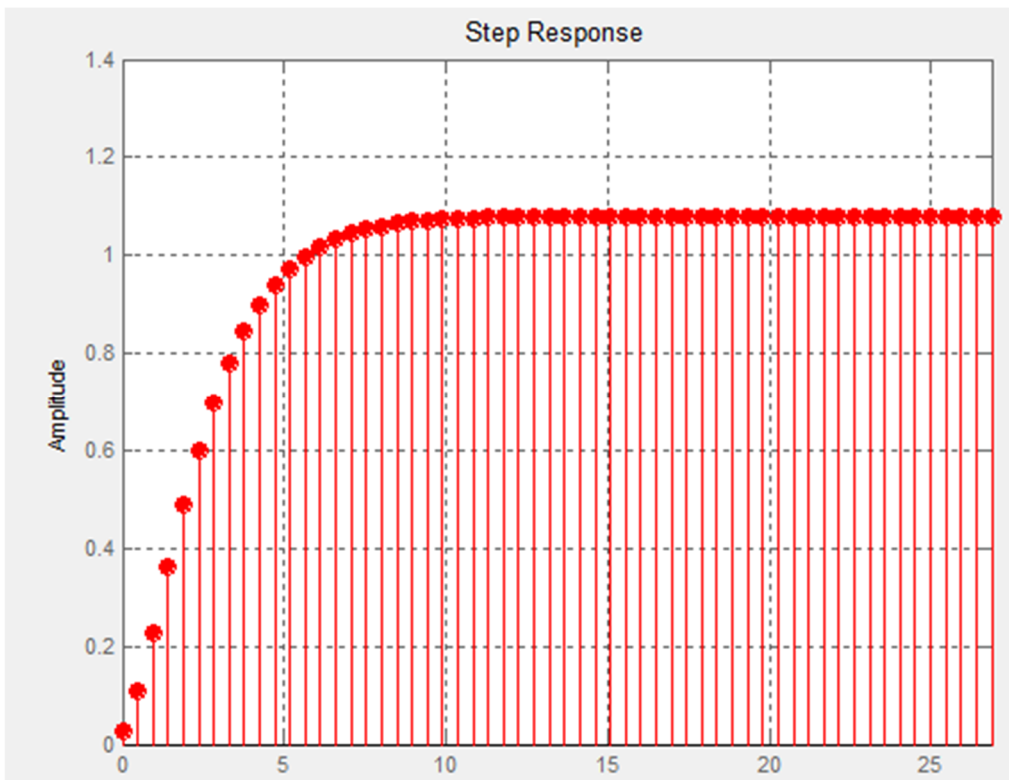


Figura 48: Respuesta ante escalón

La respuesta ante un impulso, figura 47, del 100% es menor a 0.14 y es correcta para el formato de coma fija de 32 bits con 30 bits de parte decimal, sin embargo, la respuesta ante un escalón, figura 48, es mayor que 1 y menor que 2, por lo tanto, habrá que utilizar un formato con 30 bits de parte decimal para evitar problemas de desbordamiento.

Finalmente, se estudiará la precisión comprobando la variación del resultado del *shaping* cuantificado frente a una ventana de señal del experimento.

Los datos disponibles de la ventana de señal son de formato 12 bits por lo tanto, se reconvertirá al formato de salida del BLR 42 bits con 20 de parte decimal, con tal de emular el formato de entrada del sistema.

La función de Matlab es la siguiente:

```

1  %Lectura de ficheros
2  file='./BLRfile0.dat';
3  % Se pasa a formato real de 32 bits
4  input12bit = fi(input_mod,1,42,20,'OverflowAction', 'Saturate','RoundingMethod','Floor');% 'Wrap');%, 'MaxProductWordLength',NBITS_acum*2);
5  LSB=2/2^12;
6  % A un formato de 32 bits, pero con valores inferiores a la unidad y 20 bits de parte decimal
7  data_in=(input12bit.data-(2^11))*LSB;
8  % A un formato de 12 bits, pero con valores inferiores a la unidad
9  data_in=(input12bit.data-(2^12))*LSB;
10 %Implementacion de la red digital del shaper en matlab
11 % Declaración de constantes
12 g=0.0207362;
13 al=-1.142399144;
14 a2=0.50693791;
15 %Cuantificacion de coeficientes
16 bit_c=32;
17 bit_d=30;
18 g_c=fi(g,1,bit_c,bit_d,'ProductWordLength',64,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
19 al_c=fi(al,1,bit_c,bit_d,'ProductWordLength',64,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
20 a2_c=fi(a2,1,bit_c,bit_d,'ProductWordLength',64,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
21 %funcion sin cuantificar
22 x=data_in;
23 % Declaración de vectores para las señales 'y' y 'h'
24 y=zeros(size(x));
25 h=zeros(size(x)); % h[n] y h[n-1]
26 % Inicialización
27 h(1)=g*x(1);
28 y(1)=h(1);
29 h(2)=g*x(2)- al*h(1);
30 y(2)=h(2)+2* h(1);
31 for k=3:length(x)
32 h(k)=g*x(k) - al*h(k-1)- a2 *h(k-2);
33 y(k)=h(k) +2* h(k-1)+ h(k-2);
34 end
35 %funcion cuantificada
36 % Declaración de vectores para las señales 'y' y 'h'
37 y1=fi(y,1,bit_c,bit_d,'ProductWordLength',64,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
38 h1=fi(h,1,bit_c,bit_d,'ProductWordLength',64,'SumWordLength',32,'ProductMode','KeepMSB','SumMode','KeepMSB');
39 % Inicialización
40 h1(1)=g_c*x(1);
41 y1(1)=h1(1);
42 h1(2)=g_c*x(2)- al_c*h1(1);
43 y1(2)=h1(2)+2* h1(1);
44 for k=3:length(x)
45 h1(k)=g_c*x(k) - al_c*h1(k-1)- a2_c*h1(k-2);
46 y1(k)=h1(k) +2* h1(k-1)+ h1(k-2);
47 end
48 plot(y);
49 hold all;
50 plot(y1);
51 hold off;
52 legend('Sin Cuantificar','Coeficientes Cuantificados');
```

Figura 48: Función análisis en MATLAB

Esta función devuelve la siguiente gráfica que compara la respuesta del filtro cuantificado y la respuesta del filtro sin cuantificar de precisión mucho mayor:

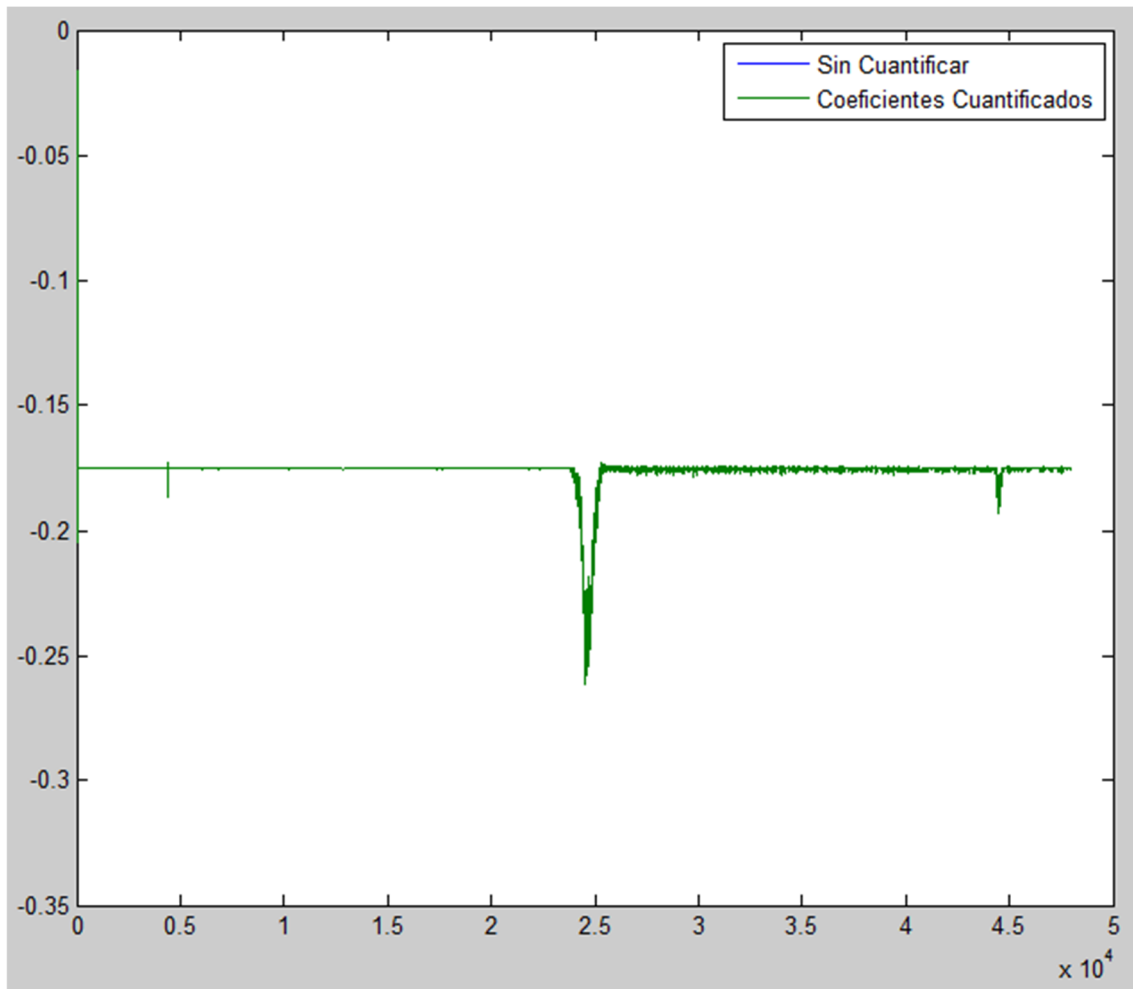


Figura 49: Cuantificado vs No cuantificado

Los resultados de filtrado como se observa en la figura 49, obtenidos son los mismos para ambas funciones por lo que podemos garantizar la precisión con los tamaños de palabra y formatos elegidos se adecua a las necesidades del sistema.

### 6.2.3. Implementación en verilog

Con los tamaños de palabra definidos se implementará la red digital en hardware. Para ello utilizaremos el lenguaje de descripción de hardware Verilog.

Verilog es un lenguaje de descripción de hardware se utiliza para modelar sistemas electrónicos digitales a un nivel de abstracción RTL o de transferencia entre registros este nivel describe las relaciones entre entradas o salidas del sistema estructurando la funcionalidad del sistema y sus partes. Es el nivel de abstracción más alto que garantiza la síntesis del sistema siempre y cuando se mantenga el estándar. La síntesis es la traducción del lenguaje al hardware en sí, de estructuras descritas, a conjuntos de LUTs y otros elementos básicos electrónicos. La FPGA que compone el núcleo de cada FEC es una Virtex -6 de Xilinx.

Para implementar el hardware utilizaremos el software ISE y para la verificación el software MODELSIM. Es importante destacar que el proceso de verificación y diseño van ampliamente ligados puesto que un fallo de verificación supone un retorno a la etapa de diseño.

Las condiciones de diseño iniciales del sistema son las siguientes:

- Sistema síncrono y con *reset* síncrono.
- Con opción de elegir la salida del *shaper* o la del BLR sin tratar.
- Tamaño de palabra de 32 bits.
- Palabra de salida de 12 bits.
- Palabra de entrada de 42 bits.
- Configurable, es decir se pueden cambiar los coeficientes del *shaper*.

A partir de estas condiciones de diseño se pueden empezar a decidir qué elementos incluirá el circuito.

Entradas:

- 3 entradas booleanas para señal de reloj, el reset y el selector de salida.
- 3 entradas de 32 bits para los coeficientes del filtro.
- 1 entrada de 42 bits para la entrada de la señal de 42 bits.
- 1 entrada de 12 bits para la entrada de la señal de 12 bits.

Salidas:

- 1 salida de 12 bits con la señal para entrar al módulo de *trigger*.

Elementos del circuito:

- 2 convertidores de tamaño de palabra.
- Multiplexor de 12 bits.
- Red digital del *shaper*:
  - 3 multiplicadores de 64 bits.
  - 1 multiplicador x 2 (desplazador).
  - 2 biestables de 32 bits.
  - 4 sumadores.

Estos elementos configuraran en principio la red digital que compone el *shaper* y las conexiones necesarias para que funcione dentro del sistema.

Un diagrama de bloques del circuito sería:

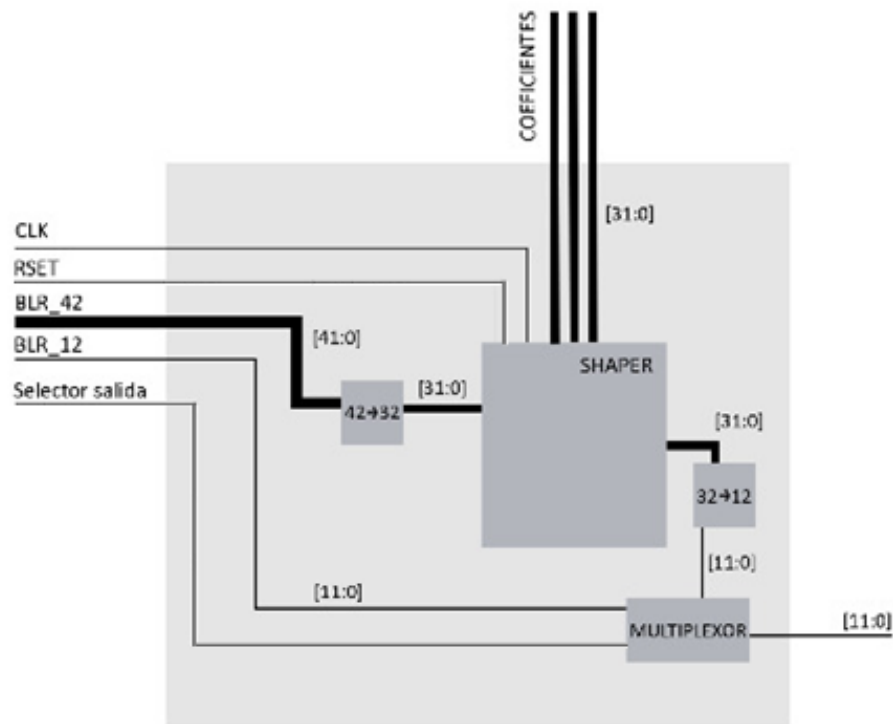


Figura 50: Esquema de módulo de *shaper*

Del BLR entraran al módulo dos señales, una de 42 bits que se utilizará como entrada del *shaper* tras reconvertirla a 32 bits, y otra de 12bits que será la salida del módulo cuando la señal selector-salida está a 0.

Las señales *clock* y *reset* son necesarias para los dos biestables de la red digital del *shaper*.

Al *shaper* entran también los coeficientes codificados en 32 bits.

La salida con la señal tratada se convierte a 12 bits y cuando la señal selector-salida esta activo es la que sale del módulo hacia el *trigger*.

Elementos del circuito:

En Verilog el diseño es mediante la descripción de módulos, y la instanciación de estos en el módulo principal.

Los principales módulos del circuito son:

#### **Convertidor de tamaño de palabra de 42 bits a 32 bits.**

La nomenclatura a utilizar para explicar el formato será la siguiente, A, B bits con A siendo el número de bits total y B el número de bits decimales.

La salida del BLR es de formato coma fija 42,20 bits, la entrada al *shaper* es de formato coma fija 32,2 bits, por lo tanto, se necesitará que desplazar el 0 para la señal de 42 bits.

La adaptación es la siguiente, se traslada el bit de signo y los bits más significativos:

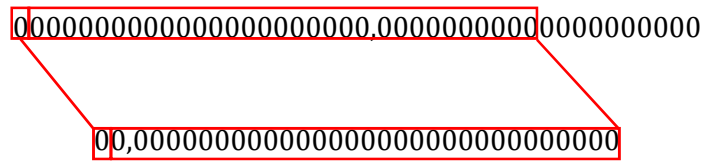


Figura 51: Formato de 42 a 32 bits

El módulo es el siguiente:

```
module fortytwotothirtytwo(
    input [41:0] BLR_OUT42,
    output [31:0] DATA_IN
);
wire [41:0] DATA;

assign DATA = BLR_OUT42;
assign DATA_IN [31] = {DATA[41]};
assign DATA_IN [30:0] = {DATA [40:10]};

endmodule
```

Figura 52: Modulo formato 42 a 32 bits

### Convertidor de tamaño de palabra de 32 bits a 12 bits.

Este elemento adapta la salida del *shaper* de 32,30 bits a la salida del módulo de 12 bits *unsigned*

Para cambiar el tamaño de palabra y formato, se traslada el complementario del bit más significativo y los siguientes 11 bits

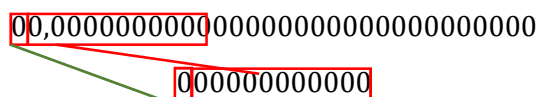


Figura 53: Formato de 32 a 12 bits

La implementación:

```
module thirtytwototwelve(DATA_OUT,FILT_OUT);

input [31:0] DATA_OUT;
output [11:0] FILT_OUT;
wire [31:0] DATA, DTA;
wire [31:0] XOR;
assign DATA = DATA_OUT;
assign FILT_OUT = {{1'b1^DATA [31]},DATA [30:20]};

endmodule
```

Figura 54: módulo Formato de 32 a 12 bits

## Multiplexor

Multiplexor de 12 bits con una entrada de selección en función del *enable*.

La implementación:

```
assign OUT = (filter_out == 1'b0)? BLR_OUT12 : FILT_OUT ;
```

Figura 55: Implementación multiplexor

Esta incorporado en la función principal.

## Red digital del *shaper*

Este módulo está compuesto por una serie de submódulos.

- **Biestables**

Los retardos en la red digital del filtro están implementados mediante biestables síncronos de 32 bits, este par de biestable conforman la rama central de la red.

Su módulo es el siguiente:

```
module Z(SIG_IN, SIG_OUT, CLK, RST );
input CLK, RST;
input [31:0] SIG_IN;
output [31:0] SIG_OUT;
reg [31:0] SIG_OUT;
always @ (posedge CLK or posedge RST)
begin
if(RST)
SIG_OUT = 32'b0;
else
SIG_OUT = SIG_IN;
end
endmodule
```

Figura 56: Módulo biestable

Este módulo actualiza la salida por la entrada a cada ciclo de reloj.

- **Multiplicador**

Para implementar los multiplicadores, al ser elementos concretos muy utilizados para los cuales el hardware de la FPGA dispone de módulos especiales, se ha utilizado un *template*:



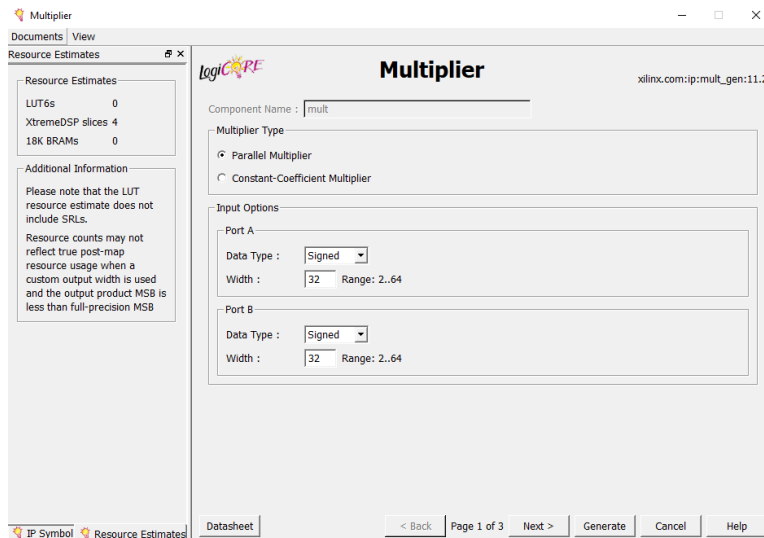


Figura 57: Configurador multiplicador

Tanto es así para los multiplicadores de los coeficientes como para el multiplicador por 2.

Hay diferencias significativas entre ambos. Los multiplicadores de los coeficientes son de 32 bits por 32 bits por lo que su salida es de 64 bits y habrá que adaptarla a 32 bits otra vez.

Respecto al multiplicador por 2 es mucho más sencillo puesto que multiplicar por 2 en binario es desplazar los bits una posición a la izquierda. La salida es de 33 bits.

- **Sumadores**

los sumadores se han implementado mediante un *template*:

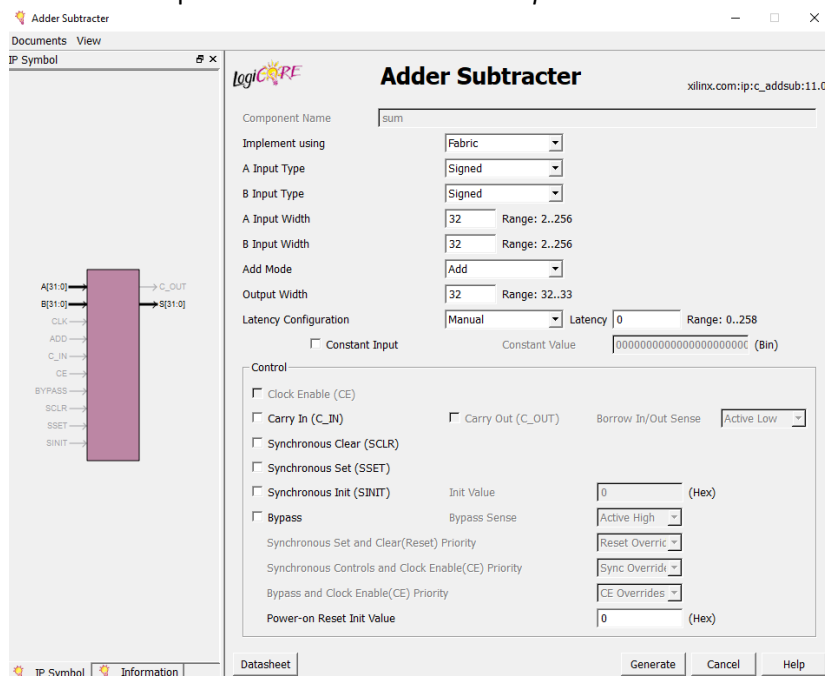


Figura 58: Configurador sumador

Ahora que se han descrito los módulos que lo componen, pasamos a presentar la implementación del *shaper*. Es la conexión de 4 multiplicadores, 2 biestables y 2 sumadores sumados a todos los ajustes de señal necesarios para conectarlos:

```

module filter (RST, CLK, DATA_IN, COEFA1, COEFA2, COEFG, DATA_OUT); //INICIALIZACION DE ENTRADAS Y CONEXIONES
input RST, CLK;
input [31:0] COEFA1, COEFA2, COEFG;
input [31:0] DATA_IN;
output [31:0] DATA_OUT;
wire [31:0] DATA, H_1, H_2, H_IN, H_OUT, Z1, Z2, PA1, PA2, PB1;
wire [63:0] ma, mb, mg;

assign H_2 = H_1; //aSIGNA H1 Y H2 COMO H
//AJUSTE DE LA SALIDA DEL MULTIPLICADOR DE 33 BITS
twomult two (//INSTANCION DEL MULT X2
.a(Z1), // input [31 : 0] a
.p(PB1) // output [31 : 0] p
);
//AJUSTE DE LA SALIDA DE LOS MULTIPLICADORES DE 64 BITS
mult A_1 (//INSTANCION DEL MULT A1
.a(Z1), // input [31 : 0] a
.b(COEFA1), // input [31 : 0] b
.p(ma) // output [32 : 0] p
);
assign PA1 = {ma[63],ma[60:30]};
mult A_2 (//INSTANCION DEL MULT A2
.a(Z2), // input [31 : 0] a
.b(COEFA2), // input [31 : 0] b
.p(mb) // output [32 : 0] p
);
assign PA2 = {mb[63],mb[60:30]};
mult G (//INSTANCION DEL MULT G
.a(DATA_IN), // input [31 : 0] a
.b(COEFG), // input [31 : 0] b
.p(mg) // output [32 : 0] p
);
assign DATA = {mg[63],mg[60:30]};

// SUMADORES 4 SEMISUMAS
ADDER AA1 (
.IN_A(DATA),
.IN_B(PA2),
.OUT(H_IN)
);
// sum AA2 (
//.a(H_IN), // input [31 : 0] a
//.b(PA1), // input [31 : 0] b
//.s(H_1) // output [31 : 0] s
//);
ADDER AA2 (
.IN_A(H_IN),
.IN_B(PA1),
.OUT(H_1)
);
sum AA3 (
.a(H_2), // input [31 : 0] a
.b(PB1), // input [31 : 0] b
.s(H_OUT) // output [31 : 0] s
);
sum AA4 (
.a(H_OUT), // input [31 : 0] a
.b(Z2), // input [31 : 0] b
.s(DATA_OUT) // output [31 : 0] s
);
//INSTANCIACION DE BIESTABLES
Z Z_1 (
.SIG_IN(H_2),
.SIG_OUT(Z1),
.CLK(CLK),
.RST(RST)
);
Z Z_2 (
.SIG_IN(Z1),
.SIG_OUT(Z2),
.CLK(CLK),
.RST(RST)
);
endmodule

```

Figura 59: módulo *shaper*

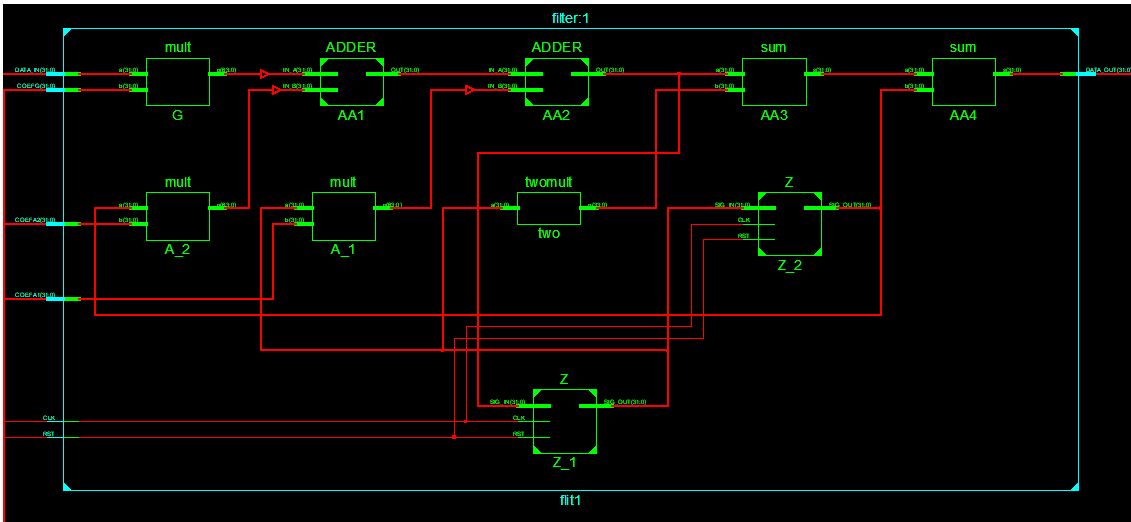


Figura 60: RTL módulo *shaper*

### Módulo principal.

El módulo principal es la instanciación de los módulos descritos anteriormente y su conexión con las entradas y salidas del sistema.

```

module shaper( RST, CLK, BLR_OUT12, BLR_OUT42, COEFA1, COEFA2, COEFG, OUT, filter_out );
input RST, CLK;
input filter_out ;
input [31:0] COEFA1, COEFA2, COEFG;
input [11:0] BLR_OUT12;
input [41:0] BLR_OUT42;
output [11:0] OUT;
wire [31:0] DATA_IN , DATA_OUT;
wire [11:0] FILT_OUT;

//ajuste entrada 42,20 ---->> 32,30
fortytwo to thirtytwo init (
    .BLR_OUT42 (BLR_OUT42),
    .DATA_IN (DATA_IN)
);
// Filtro
filter flit1 (
    .RST (RST),
    .CLK (CLK),
    .DATA_IN (DATA_IN),
    .COEFA1 (COEFA1),
    .COEFA2 (COEFA2),
    .COEFG (COEFG),
    .DATA_OUT (DATA_OUT)
);
//Ajuste salida 32,30 ---->> 32,30
thirtytwo to twelve exit (
    .DATA_OUT (DATA_OUT),
    .FILT_OUT (FILT_OUT)
);
// Multiplexor salida
assign OUT = (filter_out == 1'b0)? BLR_OUT12 : FILT_OUT ;
endmodule

```

Figura 61: módulo top





### Verificación del módulo principal.

Para verificar el módulo principal se comprobará la respuesta de este ante la ventana de señal utilizada en Matlab y se contrastaran los resultados con los de implementación del filtro en Matlab.

El módulo de *testbench* es el siguiente:

```
module shapertest;

    // Inputs
    reg RST;
    reg CLK;
    reg [11:0] BLR_OUT12;
    reg [41:0] BLR_OUT42;
    reg [31:0] COEFA1;
    reg [31:0] COEFA2;
    reg [31:0] COEFG;
    reg filter_out;
    reg [41:0] data42;
    reg [11:0] data;
    integer fd;
    integer f,i;
    integer code, dummy;
    reg [8*10:1] str;
    // Outputs
    wire [11:0] OUT;

    // Instantiate the Unit Under Test (UUT)
    shape uut (
        .RST(RST),
        .CLK(CLK),
        .BLR_OUT12(BLR_OUT12),
        .BLR_OUT42(BLR_OUT42),
        .COEFA1(COEFA1),
        .COEFA2(COEFA2),
        .COEFG(COEFG),
        .OUT(OUT),
        .filter_out(filter_out)
    );
endmodule
```

Figura 68: Parte 1 del módulo de *testbench*

Este fragmento incluye la declaración de los inputs del sistema y la implementación del módulo top.

El segundo fragmento se encuentran los valores iniciales para las entradas, así como su variación temporal incluyendo la carga del fichero de datos que contiene datos de una ventana de señal.

```

initial begin
    // Initialize Inputs
    fd = $fopen("_blr12.dat","r");
    f = $fopen("_output.dat","w");
    data = 0;
    code = 1;
    $monitor("data = %d", data);
    RST = 1;
    CLK = 0;
    BLR_OUT12 = 0;
    BLR_OUT42 = 0;
    COEFA1 = 32'ha4dd532d;
    COEFA2 = 32'h2071ABB4;
    COEFG = 32'h153bded;
    filter_out = 1;

    //rutado de datos

    // Wait 100 ns for global reset to finish
    #100;
    RST=0;

    // Add stimulus here

end

//Lectura
always @(posedge CLK)
begin
    if (code)begin
        code = $fgets(str, fd);
        dummy = $sscanf(str, "%d", data);
        $display("write = %d", OUT);
        $fwrite(f,"%d\n", OUT);
    end
    else
        $stop;
    data42 = {{data[11]^1'b1},data[10:0],30'h0000000};
    BLR_OUT12 = data;
    BLR_OUT42 = data42;

end
// initial begin

always #25 CLK = ~CLK;

```

Figura 69: Parte 2 del módulo de *testbench*

Los coeficientes calculados anteriormente se introducen como entrada en formato hexadecimal. Representan un formato 32,30 *signed*.

$$\begin{aligned}
 g &= 0.0207362 = H153BDED \\
 a_1 &= -1.42399144 = HA4DD532D \\
 a_2 &= 0.50693791 = H153BCDED
 \end{aligned}$$

Ecuación 4: Coeficientes en hexadecimal

Se establece como señal de reloj los 25 ns del sistema y como entrada de datos un fichero cuyos datos se convierten a formato 42,20 antes de introducirlos al módulo de *shaper*.

Los resultados de la simulación son los siguientes:

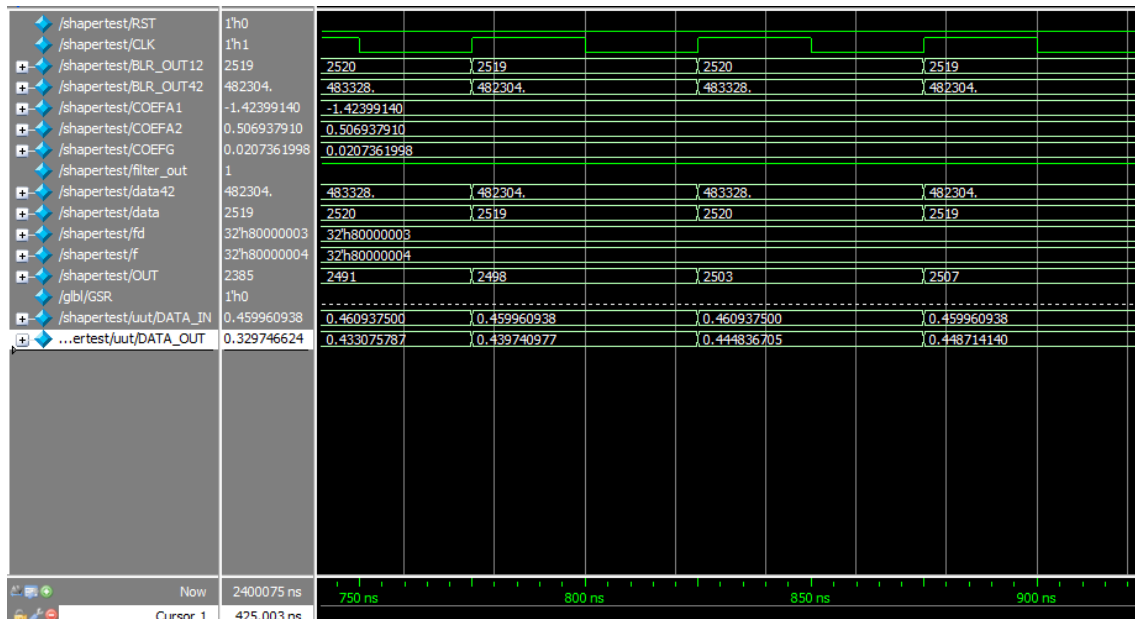


Figura 70: Respuesta temporal del Módulo de *shaper*

Finalmente, si comparamos la salida del módulo con los resultados de Matlab:

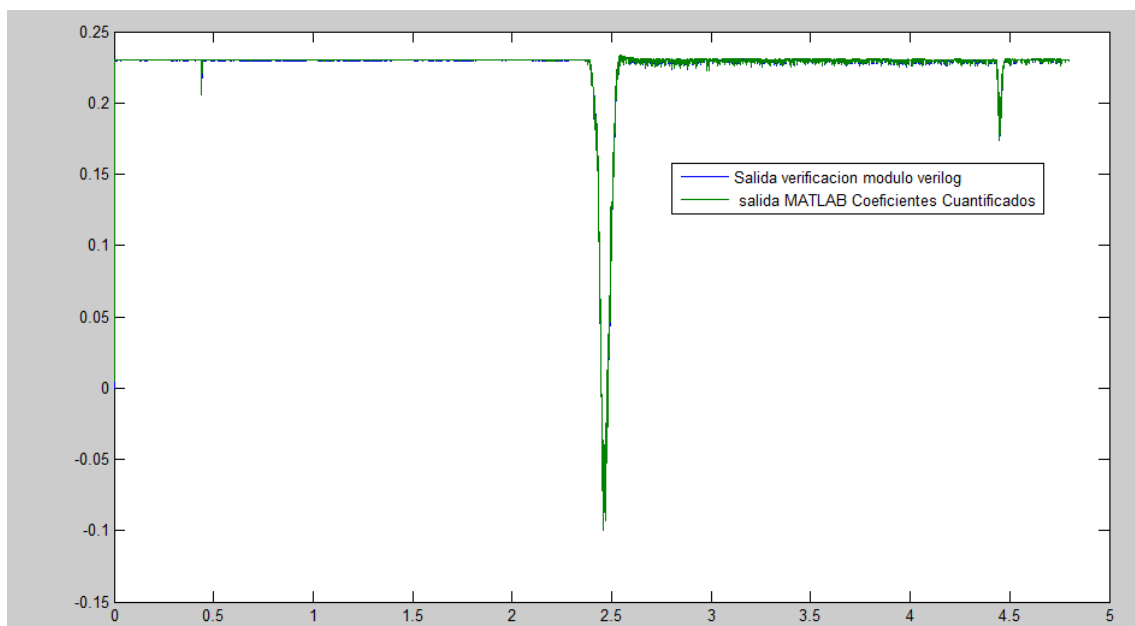


Figura 71: Comparación resultado *shaper* verilog con *shaper* MATLAB

Como se observa en la figura 71, la respuesta del módulo implementado y de la función cuantificada en Matlab es prácticamente la misma.



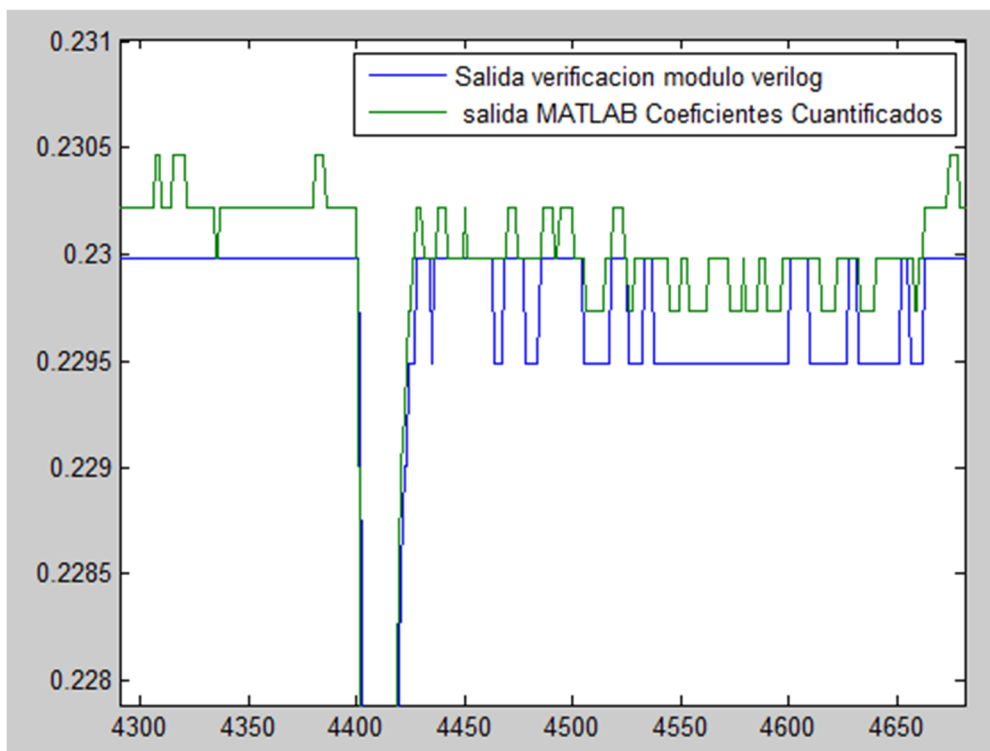


Figura 72: Comparación resultado detalle alrededor de S1

Con más detalle como se muestra en la figura 72, se aprecia que las diferencias son a un nivel de precisión mucho menor que el S1 por lo tanto podemos dar el resultado como correcto.

Se puede afirmar que la respuesta del módulo es la esperada por lo que estaría listo para probar su rendimiento mediante la implementación en la FPGA del sistema DAQ del experimento.

### 6.3. Integración en el experimento NEXT

El módulo se implementó y verificó a nivel funcional, integrando el módulo *shaper* en la cadena de procesamiento del DAQ, mediante tomas de datos del NEXT-NEW instalado en el Laboratorio Subterráneo de Canfranc (LSC). Para ello se realizaron dos pares de RUNs, con las siguientes características:

RUN 4803, 4806

Configuración de <i>Trigger</i>												
Parámetros Globales					<i>Trigger</i> Interno							
Máscara	Frecuencia	Auto Trig externo	<i>Trigger</i> doble		Max. tiempo <i>trigger</i> A/B	Eventos <i>trigger</i> A		Eventos <i>trigger</i> B				
ON	10	OFF	OFF		0	2		1				
Configuración <i>trigger</i> A Canal PMTS												
On/Off	Inversión Polar	<i>Trigger</i> Q	Auto BS	TRG B	<i>Baseline</i> deviation	Max A	Q min	Q max	Max time	Min time	Pulse valid	
CH18	1	0	1	1	0	5	300	100	625	350(ns)	100(ns)	50(ns)
CH19	1	0	1	1	0	5	300	100	625	350(ns)	100(ns)	50(ns)

Tabla 2: Parámetros RUNs 4803, 4806 y 4828

4803 y 4806 son consecutivos con las mismas configuraciones de *trigger*, pero 4806 tiene aplicado el módulo de *shaper* diseñado. Es importante resaltar que las condiciones del *trigger* interno para la búsqueda de pulsos de interés es la misma en ambos.

RUN 4828, 4829

Configuración de <i>Trigger</i>												
Parámetros Globales			<i>Trigger</i> Interno									
Máscara	Frecuencia	Auto Trig externo	<i>Trigger</i> doble		Max. tiempo <i>trigger</i> A/B	Eventos <i>trigger</i> A		Eventos <i>trigger</i> B				
ON	15	OFF	OFF		0	2		1				
Configuración <i>trigger</i> A Canal PMTS												
On/Off	Inversión Polar	<i>Trigger</i> Q	Auto BS	TRG B	Baseline deviation	Max A	Q min	Q max	Max time	Min time	Pulse valid	
CH18	1	0	1	1	0	5	300	<b>75</b>	625	<b>700(ns)</b>	<b>200(ns)</b>	50(ns)
CH19	1	0	1	1	0	5	300	<b>75</b>	625	<b>700(ns)</b>	<b>200(ns)</b>	50(ns)

Tabla 3: parámetros RUN 4829(parámetros adaptados en negrita)

Al igual que el par anterior el RUN 4829 lleva aplicado el *shaper*; pero a diferencia del RUN 4806 se han modificado los parámetros Qmin, Max time y Min Time. teniendo en cuenta el efecto del *shaper* sobre la energía y duración de la señal.

Para analizar los resultados obtenidos se utilizarán las funciones para evaluar la validez de *triggers* en el estudio de optimización de coeficientes.

Se examinarán las diferencias de resultados entre el filtrado de procesado de datos y el filtro implementado con el objetivo de evaluar la implementación y los resultados de la aplicación del *shaper* en sí.

### RUN 4803 y 4806

Este par de RUNS se centra en estudiar la importancia de compensar la distorsión temporal provocada por el *shaping* también se compararán los resultados con el resultado de aplicar esta corrección al procesado del RUN 4803.

Los resultados del procesado con las condiciones del RUN se muestran en la tabla 5:

RUN	Datos	<i>Trigger</i>	% de <i>trigger</i>	% válido	% S2	% inválidos	válido /s
4803	RAW	Parámetros RUN	71,50%	62,41%	12,70%	28,50%	4,86
	<i>Shaped</i> (Off-line)	Parámetros RUN	20,60%	20,39%	1%	79,40%	1,59
	<i>Shaped</i> (Off-line)	Parámetros RUN adaptado	62,55%	60,11%	3,90%	37,45%	4,68

Tabla 5: Resultados RUN 4803

En esta figura se puede ver que los datos RAW corresponden a los datos extraídos del fichero HDF5 sin tratar y los *shaped* son los tratados mediante el módulo de *shaper* de la herramienta informática.

“% de *trigger*” son los *triggers* que la modelización del *trigger* del sistema detecta y “% inválidos” los que no detecta. “% válido” es el porcentaje total absoluto de *triggers* válidos y “% S2” el porcentaje de *triggers* detectados que son en S2.

“Válidos/s” es el número aproximado de eventos válido por segundo calculado en base al ratio de adquisición del RUN y el % de eventos válidos detectados.

Como se ha mencionado, de los resultados del RUN 4803 y 4806 podemos observar la importancia de la corrección de los parámetros del *trigger* tras el filtrado.

Se destaca la gran pérdida de *triggers* al procesar los datos y no compensar este efecto, de un 71,50% de *triggers* en RAW a un 20,60%, una reducción muy significativa.

Los resultados del RUN 4806 son los siguientes:

RUN	Datos	<i>Trigger</i>	% de <i>trigger</i>	% válido	% S2	% inválidos	válido /s
4806	RAW	Parámetros RUN	85,75	81,1	5,4	14,25	1,622

Tabla 6: Resultados RUN 4806

A primera vista los resultados de este RUN son muy positivos y no se observa una diferencia tan extrema entre los factores de *trigger* adaptado y los no adaptados, esto es debido, a que el ratio de adquisición de eventos es solo de 2 eventos por segundo frente a los 7,8 del RUN 4803, el cual no lleva aplicado el filtro. Es decir, el no corregir el filtrado es muy restrictivo y la pérdida de eventos válidos es muy significativa.

Aun así, podemos constatar una equivalencia entre el *shaping* real y el simulado en el número de eventos válidos por segundo siendo de 1,62 % para la implementación real del *shaper* en el RUN 4806 y de 1,59 % para el procesado informático del RUN 4803, sin compensar los parámetros del módulo de *trigger*. Esto implica la pérdida de muchos datos a priori válido (en torno al 50 %) por la no adaptación de los parámetros cuando el *shaper* está activo.

En los RUNs 4828 y 4829, si se tiene en cuenta la distorsión y se modifican los parámetros de energía mínima y duración del pulso para compensar dicha alteración, los resultados son los siguientes:

RUN	Datos	Trigger	% de trigger	% válido	%S2	% inválidos	válido /s
4828	raw	Parámetros RUN	53,75	46,01	14,4	46,25	3,91085
	Shaped (Off-line)	Parámetros RUN adaptado	45	43,25	3,9	55	3,67625

Tabla 7: Resultados RUN 4828

RUN	Datos	Trigger	% de trigger	% válido	%S2	% inválidos	válido /s
4829	shaped	Parámetros RUN adaptado	48,65	44,9	7,72	51,35	4,2655

Tabla 8: Resultados RUN 4829

En primer lugar, cabe destacar la diferencia de ratios de adquisición entre ambos RUNs, el RUN 4828 tiene un ratio de 8,5 eventos por segundo y el RUN 4829 de 9,5. Este aumento de ratio es significativo, aunque esperable en el sentido de que unas condiciones de *trigger* más amplias son menos restrictivas. Esto implica un mayor número de datos recogidos por el sistema.

Este par de RUNs tienen unos niveles de eficacia bastante bajos respecto a otros RUNs analizados, de un 70% de *triggers* válido a aproximadamente un 50%. El hecho de que esto afecte a ambos RUNs y no sólo al que tiene el módulo de *shaping* activo, sugiere al hecho de que en la cámara se produjeron un mayor número de interferencias durante este par de RUNs.

En base a comparar los resultados del *shaping* implementado y los del modelo informático podemos observar que el modelo informático es más eficaz eliminando *triggers* en S2, pero pierde un mayor número de *triggers* válidos. Aun así, el comportamiento es bueno, aunque se pierde una cantidad de eficacia significativa en la eliminación de *triggers* en S2

resultados	Sin shaping	Con shaping
<i>trigger</i> válidos	46,01%	44,9%
<i>trigger</i> en s2	14,4%	7,72%

Tabla 9: Resultados modelo e implementación

Aun así, como se observa en la Tabla 9 podemos constatar que se elimina un gran porcentaje de S2 al aplicar el *shaping*.

En definitiva, la efectividad del *shaping* para eliminar *triggers* en S2 implementado es menor de la esperada, aunque aun así está dentro de parámetros aceptables, puesto que se eliminan gran cantidad de *triggers* en S2.

## 7. Conclusiones

Tal y como se especificó en los objetivos se ha desarrollado un método válido para la eliminación de falsos *triggers* en flancos de S2, mediante la aplicación de un *shaper* a la señal de entrada del módulo FEC del plano EP. Así como las herramientas necesarias para evaluar el funcionamiento de este.

Como se puede observar en los resultados de la implementación, tablas 7 y 8, la cantidad de *triggers* en S2 de los RUNS con el *shaper* implementado es significativamente menor, también se observa una ligera reducción de los *triggers* válidos, los cuales, se ven afectados también por el shaping, aunque en mucha menor medida, hecho ya predicho y mostrado en la sección 5.4 (figura 42).

Ya existe de base una diferencia significativa en el funcionamiento del algoritmo que hay implementado en la FEC del plano EP y el del modelo informático creado, que sobre todo radica en el estudio continuo del nivel de base en el primer caso, frente a la obtención del mismo valor cuando los datos están disponible off-line.

Por otro lado, está el ajuste de los parámetros de búsqueda de pulso con se aplica el shaping. Si bien estas diferencias se han tenido en cuenta a la hora de hacer el estudio, son junto a los resultados de los RUNS 4803 y 4806 un claro indicador de la importancia del ajuste de dichos parámetros para garantizar el óptimo funcionamiento del *trigger*. Este hecho debería ser objeto de un mejor estudio, el *shaper* incorpora una serie de efectos adversos sobre el comportamiento del *trigger* que merman la eficacia de éste. Ahondar en este hecho queda fuera de los objetivos inicialmente marcados para este trabajo.

En base a los datos obtenidos se puede concluir que el shaping es capaz de eliminar un porcentaje significativo de *triggers* en S2. Esto deriva en un mejor rendimiento en el ratio entre datos válidos e inválidos por adquisición, lo que radica en un uso más óptimo de los recursos de almacenamiento y de post-procesado del experimento.

En conclusión, la implementación de un *shaper* en la cadena de procesado de la FEC del plano EP es un método que pese a presentar distintas desventajas es capaz de eliminar significativamente la cantidad de falsos *triggers* en S2 mejorando la eficacia. Ahora bien, es necesario con tal de mejorar su utilidad el análisis de las interacciones entre los efectos del shaping y el algoritmo de generación de candidatos de *trigger*.

### 7.1. Trabajo futuro

Queda claro durante la realización de este trabajo que para un uso más eficaz del *shaper*, es necesario realizar un estudio en profundidad sobre la interacción entre el shaping y el algoritmo de generación de candidatos de *trigger*. Así como profundizar en otras tipologías de *shaper*.

Durante la realización de este proyecto, especialmente en la fase de estudio y modelización del módulo de *trigger*, se llegó a la conclusión que para una optimización del módulo que calcula los candidatos de *trigger*, era necesario conocer qué *triggers* detectados lo eran realmente en S2. Para ello, tal y como se ha descrito en el apartado 5.3, se implementó un módulo off-line para realizar esta tarea. Este módulo ha resultado ser muy útil y eficaz en la detección de *triggers* en S2, por lo que se podría estudiar la implementación de un módulo

forma que el sistema de *trigger* implementado fuese capaz de eliminar candidatos de *trigger* si se detecta que estos son en S2, sin necesidad de uso de un *shapper*.

# Bibliografía





- [1] Ferrer Soria, A. (2015). *Física Nuclear Y De Partículas*.
- [2] R. Esteve, J. Toledo, J. Rodríguez, M. Querolb and V. Álvarezb. (2016). “*Readout and data acquisition in the NEXT-NEW Detector based on SRS-ATCA*”, SISSA MEDIALAB.
- [3] V. Álvarez, F.I.G.M. Borges, S. C´arcel, J.M. Carmona, J. Castel, J.M. Catal´ , S. Cebri´an, Cervera, D. Chan, C.A.N. Conde, T. Dafni, T.H.V.T,...(2012). “*NEXT-100 Technical Design Report (TDR). Executive summary*”, SISSA MEDIALAB.
- [4] R. Esteve. (2012). “*The Trigger System in the NEXT-DEMO detector*”, SISSA MEDIALAB.
- [5] V. Álvarez, F.I.G.M. Borges, S. C´arcel, J.M. Carmona, J. Castel, J.M. Catal´ , S. Cebri´an, Cervera, D. Chan, C.A.N. Conde, T. Dafni, T.H.V.T,...(2013). “*Near-intrinsic energy resolution for 30-662 keV gamma rays in a high pressure xenón electroluminescent TPC*”, ELSEVIER.
- [6] <http://next.ific.uv.es/next/>
- [7] Helmuth Spieler. (2012). “*Solid State Detectors – IV Signal Processing*”, USPAS-MSUCourse.
- [8] Helmuth Spieler. (2008).” *Front-end electronics and trigger systems - status and challenges*”.
- [9] Knoll, G. (1999). “*Radiation Detection and Measurement*.”
- [10] Padgett, W., & Anderson, D. (2009). “*Fixed-point Signal Processing*”.
- [11] Lyons, R. (2004). “*Understanding Digital Signal Processing*.”
- [12] R. Esteve. (2014). “*Sistemas Digitales de Control de Potencia*”, Temas 1 y 2.
- [14] <https://portal.hdfgroup.org>
- [15] V. Álvarez, V. Herrero, R. Esteve, A. Laing, J. Rodríguez, M. Querol, F. Monrabal, J. F. Toledo, J.J. Gómez-Cadenas. (2018).” *The electronics of the energy plane of the NEXT-White detector*”.



# Presupuesto



## 1. Consideraciones previas

Este trabajo es un estudio sobre la posibilidad de la aplicación de un módulo de software, por lo tanto, no hay ningún tipo de ejecución material que presupuestar.

Sin embargo, sí se aplica un trabajo y uso de equipamientos y software para la realización del proyecto que se incluirán en la partida presupuestaria.

Para los periodos de amortización de equipos y software se estima unas 2000 horas de uso.

## 2. Presupuesto

- Capítulo 01

Este capítulo corresponde con el proceso de búsqueda de información sobre el proyecto a realizar y la ejecución de la primera parte de este, es decir, el análisis informático de los parámetros del *shaper* a implementar.

Definimos la unidad de obra UO01 correspondiente al coste de una hora de este proceso.

- Capítulo 02

Este capítulo abarca el proceso de diseño e implementación del *shaper* para el cual se necesita software especializado.

Definimos la unidad de Obra UO02 correspondiente al coste horario del proceso de diseño incluyendo el coste de los programas especializados.

- Capítulo 03

Este capítulo corresponde con el proceso de análisis de resultados y la redacción de la memoria del proyecto.

Definimos la unidad de Obra UO03 correspondiente al coste horario del análisis de resultado incluyendo el coste de programas de ofimática profesionales.

### 2.1 Capítulos y unidades de obra

- CAP01

UO01

Descripción	Clase	Rendimiento	Precio €/h	Importe
Ingeniero industrial	Mano de obra	1	50	50
Ordenador	Equipamiento	1	0,5	0,5
Costes directos complementarios		2%		1,01
			Precio total €/h	51,51

Tabla 1: Coste unidad 1

- CAP02

UO02

Descripción	Clase	Rendimiento	Precio €/h	Importe
Ingeniero industrial	Mano de obra	1	50	50
Ordenador	Equipamiento	1	0,5	0,5
ISE xilincs + modelsim	Equipamiento	0,5	1,86	0,93
MATLAB	Equipamiento	0,2	0,125	0,025
Costes directos complementarios		2%		1,0291
			Precio total €/h	52,4841

Tabla 2: Coste unidad 2

- CAP03

UO03

Descripción	Clase	Rendimiento	precio €/h	Importe
Ingeniero industrial	Mano de obra	1	50	50
Ordenador	Equipamiento	1	0,5	0,5
Office Pack	Equipamiento	0,9	0,025	0,0225
Costes directos complementarios		2%		1,01045
			Precio total €/h	51,53295

Tabla 3: Coste unidad 3

## 2.2 Estado de mediciones

Unidad	Medición
UO01	150 horas
UO02	50 horas
UO03	100 horas

Tabla 4: Medición

## 2.3 Presupuesto de ejecución

Capítulo	Importe
CAP01	7726,5€
CAP02	2624,205€
CAP03	5153,295€
total	15504€

Tabla 5: Presupuesto de ejecución

#### 2.4 Presupuesto de inversión

Ejecución	15504€
Beneficio industrial (6%)	930,24€
Total	16434,24€

Tabla 5: Presupuesto de inversión

#### 2.5 Presupuesto de licitación

Gasto inversión	16434,24€
IVA (21%)	3451,1904€
TOTAL	19885,4304€

Tabla 6: Presupuesto de licitación