Electrocardiogram analysis for arrhythmia detection on an FPGA

Análisis de electrocardiogramas sobre FPGA para la detección de arritmias.

TRABAJO DE FIN DE GRADO

Juan Jerez Poblaciones

Directores:
Daniel Báscones García
Carlos González Calvo

Facultad de Informática
Universidad Complutense de Madrid

23 de mayo del 2024

Índice general

Índice de figuras

Resumen

La caracterización automática de la señal del electrocardiograma (ECG) es de importancia crítica en la monitorización y el diagnóstico del paciente. Por ello, en este trabajo lo que se pretende es detectar arritmias partiendo de una señal analizable.

Esto se realizará con un algoritmo previamente prototipado y probado en software, cuyo propósito, es detectar los picos QRS de un paciente y con ello detectar arritmias. Para ello, el algoritmo compara las distancias que tienen los picos actuales con la distancia que formaron los picos anteriores.

Para el diseño *hardware* el programa se organizará en distintos módulos. Este diseño cuenta con unos módulos principales que desempeñaran sus funciones para seguir los pasos del prototipo en software y así poder realizar las tareas de filtrado de señal, detección de picos y detección de arritmias. Estos módulos están contenidos en un módulo principal y a su vez se incluye en un *testbench* para realizar pruebas. Las pruebas se realizarán introduciendo señales del electrocardiograma de un paciente en un módulo de memoria para que el programa las procese. A su vez, en otra memoria se introducirán las anotaciones realizadas por expertos para comprobar si coinciden con las arritmias detectadas por el algoritmo y así poder comprobar la eficacia de la replicación del programa a partir del prototipado en software. Para ello, la FPGA que se utiliza para probar el algoritmo en *hardware* es la Artix 7 de la placa Basys3.

Para concluir, en los resultados experimentales, se especifica la frecuencia ideal, el reporte de *timing* y el consumo en vatios necesarios para el programa. Según los datos obtenidos, este proyecto utiliza muy pocos recursos para el desempeño que tiene y por ello, cumple con los objetivos de ser un algoritmo que puede llegar a ser personalizable, para un dispositivo portable que detecte arritmias a tiempo real.

Palabras clave: electrocardiograma, picos, arritmias, FPGA, frecuencia.

Abstract

The automatic characterization of the electrocardiogram (ECG) signal is of critical importance in patient monitoring and diagnosis. Therefore, the aim of this work is to detect arrhythmias based on an analyzable signal.

This will be done with an algorithm previously prototyped and tested in software, whose purpose is to detect the QRS peaks of a patient and thus detect arrhythmias. To do this, the algorithm compares the distances that the current spikes have with the distance that the previous spikes formed.

For the hardware design the program will be organized in different modules. This design has some main modules that will perform their functions to follow the steps of the software prototype and thus be able to perform the tasks of signal filtering, peak detection and arrhythmia detection. These modules are contained in a main module and in turn are included in a testbench for testing. The tests will be performed by introducing electrocardiogram signals from a patient's electrocardiogram into a memory module for processing by the program. In turn, the annotations made by experts will be introduced into another memory to check whether they coincide with the arrhythmias detected by the algorithm and thus be able to check the effectiveness of the program replication from the software prototyping. For this purpose, the FPGA used to test the algorithm in hardware is the Artix 7 on the Basys3 board.

To conclude, in the experimental results, the ideal frequency, the timing report and the watt consumption needed for the program are specified. According to the data obtained, this project uses very few resources for the performance it has and therefore meets the objectives of being an algorithm that can become customizable, for a portable device that detects arrhythmias in real time.

Keywords: electrocardiogram, peaks, arrhythmias, FPGA, frequency.

Capítulo 1

Introducción

1.1. Concepto de arritmia

Las enfermedades cardiovasculares son la primera causa de muerte en el mundo [10] y una de las causas más comunes de estas enfermedades son las arritmias.

Una arritmia cardiaca es un resultado de una alteración en la iniciación o propagación del ritmo del corazón [12]. Si se produce una arritmia, el corazón puede tener un ritmo demasiado rápido, demasiado lento o irregular. Esto puede provocar síntomas como palpitaciones, mareos, falta de aire e incluso desmayos, y estas pueden llegar a ser mortales.

Los cardiólogos utilizan dispositivos como un *Holter*[1] para generar electrocardiogramas (ECG), que son diagramas que representan los latidos del corazón y con ellos son capaces de analizarlos y encontrar anomalías cardiacas.

Entre esas anomalías están las contracciones prematuras del corazón, un tipo de arritmia que este proyecto se encargará de detectar.

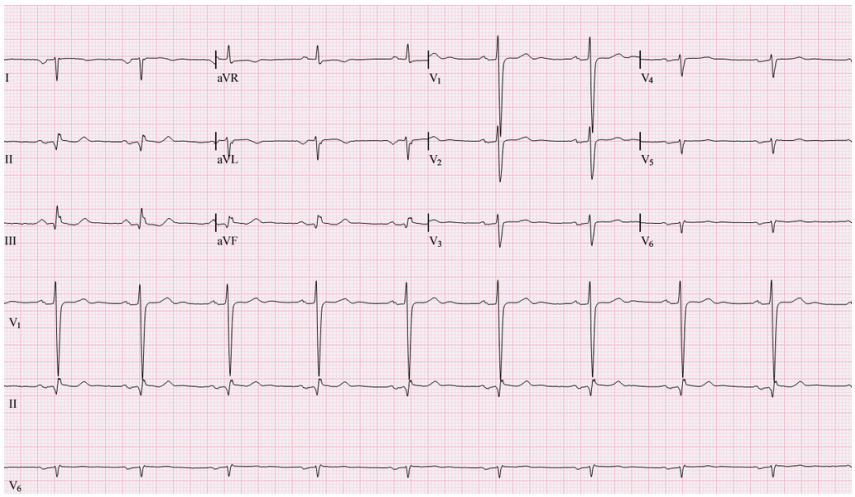


Figura 1.1: Imagen de varios electrocardiogramas recopilados por un *Holter* [9]

.

1.2. Algoritmo de detección

El algoritmo de detección de arritmias que sigue este proyecto se basa en la detección de los picos QRS producidos en el electrocardiograma.

Un pico QRS, como se muestra en la Figura 6.2, en un electrocardiograma es causado por la contracción del ventrículo al bombear la sangre por las arterias. Este es el impulso eléctrico más fuerte que el corazón produce en cada latido [3]. En este proyecto utilizaremos estos picos para comparar la distancia entre ellos y poder determinar si se ha producido una arritmia.

[1] **Holter**: Dispositivo médico portátil que se utiliza para monitorear y registrar la actividad eléctrica del corazón durante un período prolongado

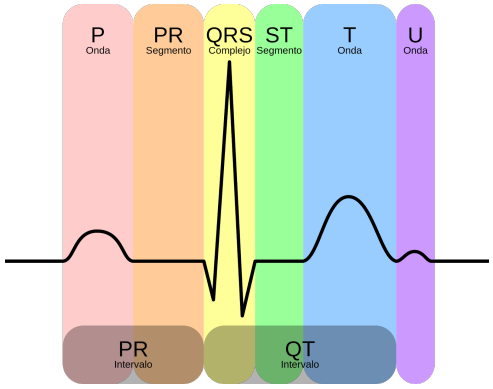


Figura 1.2: Complejo QRS [4]

Este algoritmo es de especial interés[5], ya que el desarrollo de técnicas eficientes para automatizar el análisis de ECG es fundamental para ayudar a los cardiólogos en sus diagnósticos, especialmente en la detección de arritmias. El cardiólogo tendría que dedicar varios minutos para hallar arritmias en un electrocardiograma de, por ejemplo, 30 minutos. Este algoritmo no pretende ser una solución definitiva para la detección de arritmias, ya que se requieren varios años de estudio para entenderlas completamente. Por ello, se enfoca en detectar contracciones ventriculares prematuras.

1.2.1. Filtrado

Como se puede observar en la Figura 6.3 es conveniente hacer un filtrado de las tiras de ritmo para poder detectar mejor los picos QRS, ya que el filtrado centra la onda de la señal en el valor 0 y evita fallos en el algoritmo de detección de picos del que se hablará más adelante.

En la creación del proyecto se ha intentado no filtrar la onda para comprobar si se obtienen mejores resultados que sin dicho filtrado, pero no se ha dado el caso por las irregularidades de la misma.

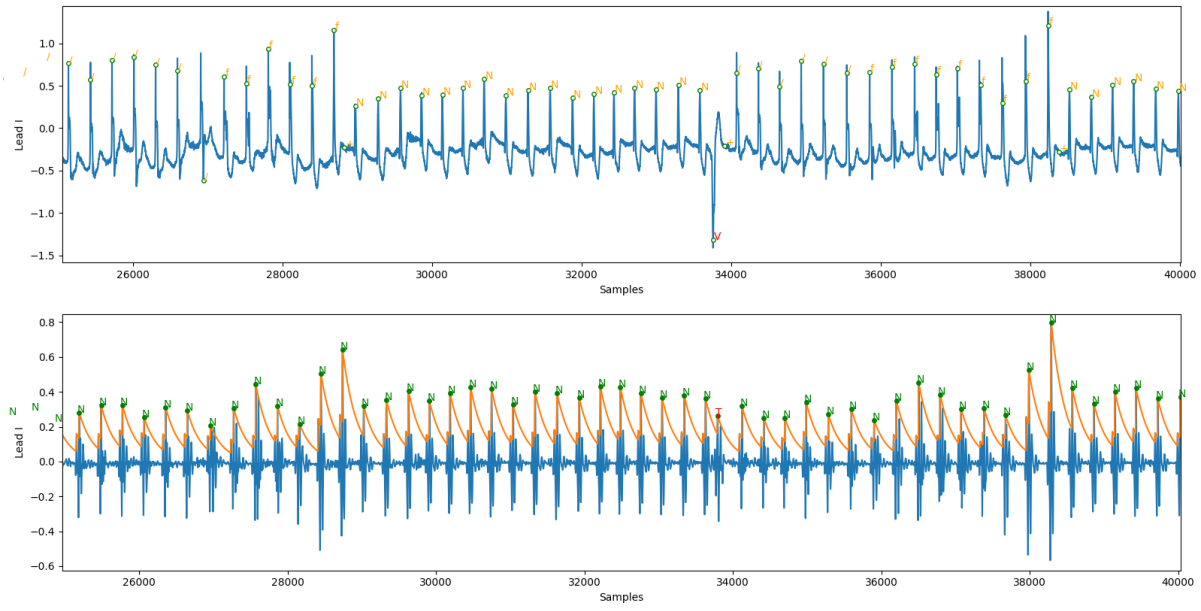


Figura 1.3: Ejemplo de electrocardiograma original y filtrado de paciente 102

1.3. Datos de referencia

Los datos de referencia para probar el algoritmo han sido obtenidos de un estudio realizado en el Instituto de Tecnología de Massachusetts (MIT) [6] en el que se han realizado pruebas de media hora a varios pacientes con edades diversas y algunos de ellos llevaban implantado un marcapasos.

Los datos consisten en electrocardiogramas que previamente han sido analizados por cardiólogos y cuyas anotaciones indican cuando un paciente ha padecido una arritmia, cuando el ritmo es normal y cuando se ha producido un fallo en la lectura de la señal, como se puede ver en la Figura 6.4. También se muestra información menos relevante para nuestro estudio como cuando se ha producido un error en la lectura de la señal y la activación del marcapasos[2].

Record 102 (V5, V2; female, age 84)

Medications: Digoxin

Beats	Before 5:00	After 5:00	Total
Normal	98	1	99
PVC	1	3	4
Paced	243	1785	2028
Pacemaker fusion	24	32	56
Total	366	1821	2187

Ventricular ectopy

- 4 isolated beats

Rhythm	Rate	Episodes	Duration
Normal sinus rhythm	72-78	2	1:22
Paced rhythm	68-78	3	28:44

Signal quality	Episodes	Duration
Both clean	1	30:06

Notes:
The rhythm is paced with a demand pacemaker. The PVCs are multiform.

Points of interest:

0:55 Paced rhythm
1:12 Transition from paced to normal sinus rhythm
1:28 PVC
2:30 Normal sinus rhythm
4:51 Pacemaker fusion beats
9:35 PVC
16:12 Paced rhythm

Figura 1.4: Ejemplo con paciente 102 [8]

1.4. Utilización de las FPGAs

Este algoritmo ha sido pensado para ejecutarse en un dispositivo pequeño, poco pesado y portable, es por ello que es conveniente ejecutarlo en una FPGA [1]. Esto de debe a que hay FPGAs pequeñas que tienen el *hardware* suficiente para poder llegar a ejecutar este algoritmo, además el consumo producido por una FPGA al ejecutar este algoritmo es bastante bajo como se comprobará posteriormente.

[2]**Marcapasos**: Dispositivo que se implanta en el corazón que actúa cuando el corazón no bombea la sangre lo suficientemente fuerte, es decir que el pico QRS no es tan prominente y se necesita la ayuda de dicho dispositivo para proporcionar el impulso eléctrico necesario

Para este proyecto se usó la FPGA Artix-7 [14] en la placa Basys3 para probar el funcionamiento del algoritmo. Sin embargo, para las pruebas, debido a la cantidad de valores de prueba sacados de la base de datos por cada paciente, se ha considerado utilizar otra FPGA cuyo *hardware* pueda soportar dicha cantidad de datos.

1.5. Objetivos del proyecto

El objetivo principal del proyecto es crear un algoritmo basado en el estudio de caracterización de señales usando polinomios de Hermite [4] que detecte arritmias en los pacientes y que dicho algoritmo sea capaz de ejecutarse en un dispositivo portátil, que tenga un bajo consumo y que funcione en tiempo real. Para lograr el objetivo principal es necesario alcanzar una serie de objetivos que son los siguientes:

1. Averiguar qué son las arritmias, cómo se producen, qué arritmias podría detectar el programa y qué patrón siguen la mayoría de ellas.
2. La creación de un algoritmo en software que sea capaz de leer y filtrar la señal original de los pacientes de la base de datos, la realización de un algoritmo que se encargue de detectar los picos donde se produce la contracción ventricular según aparecen en la señal filtrada y, por último, la creación de un algoritmo encargado de detectar las arritmias según el patrón hallado en el objetivo anterior.
3. La replicación del programa en *hardware* según el prototipo creado anteriormente en software. Utilizando el *hardware* necesario para que el algoritmo funcione en una FPGA y el *hardware* necesario para poder hacer pruebas en simulación y sobre la placa.
4. La observación de los resultados experimentales de la cantidad de recursos *hardware* empleados, el consumo utilizado por el algoritmo y el tiempo que tarda en llevarse a cabo adecuándose a la llegada de la señal del paciente. Dichos resultados experimentales deberían tener coherencia con lo que se buscaba en el objetivo principal.

1.6. Plan de trabajo

1. Investigación y fundamentación teórica:
   a) Revisión del libro [12] y consulta con un médico especialista en primeros auxilios para comprender qué son las arritmias y cómo se producen.
   b) Utilizar la información obtenida para establecer la base médica del proyecto.
2. Obtención y preparación de datos:
   a) Localizar y descargar la base de datos del proyecto utilizada en el estudio de caracterización de señales usando polinomios de Hermite [4].
   b) Representar las señales en Python y realizar el filtrado según lo descrito en el estudio mencionado.
3. Desarrollo de Algoritmos:
   a) Implementar el algoritmo de detección de picos QRS y el algoritmo de detección de arritmias.
   b) Realizar pruebas y ajustes en los algoritmos para obtener resultados satisfactorios.
4. Implementación en *hardware* :

   a) Diseñar los módulos principales para cada parte del algoritmo: detección de picos, detección de arritmias, filtrado y un módulo principal.

   b) Crear módulos adicionales para realizar simulaciones en Vivado y verificar el funcionamiento del programa.

   c) Desarrollar un *testbench* para generar la frecuencia de reloj y las señales necesarias para simular el sistema.

5. Análisis de resultados experimentales:

   a) Evaluar los resultados obtenidos de los análisis realizados en Vivado.

   b) Extraer conclusiones relevantes en función de los objetivos del proyecto.

1.7. Análisis y optimización del algoritmo

El algoritmo se centra en tres funciones principales:

1. Filtrado de la señal original: Se utiliza el filtrado FIR[2], lo que hace que la señal sea más fácil de procesar para encontrar los picos QRS. Esto se realiza multiplicando los valores de la señal original por los coeficientes de filtrado almacenados en una memoria.
2. Detección de picos sobre la señal filtrada: Se analiza cada valor de la señal filtrada y si dicho valor es mayor que sus anteriores, se considera un posible pico. Si después de 72 valores se mantiene como el valor más alto, entonces ese valor se considera un pico QRS. Adicionalmente, se establece un *cutoff* dinámico en el que, si los picos de la señal no superan ese valor, no se tienen en cuenta.
3. Detección de arritmias comparando la posición de los picos: Una vez obtenidos los picos QRS, se calcula la distancia del pico actual con el pico anterior y, dependiendo de las distancias anteriores, se determina si hay una arritmia.

1.8. Implementación en la FPGA

Para implementar el código en la FPGA se implementarán varios módulos que por lo general, tratan de replicar las funcionalidades que realiza el algoritmo de software y se convertirán en la parte más importante de dicho programa.

Los módulos principales son.

1. Módulo de filtrado: Se guardan los coeficientes del filtrado en un módulo de memoria ROM y los valores de la señal original en una memoria RAM posteriormente, se multiplica cada muestra por su correspondiente coeficiente y se acumula el resultado. Una vez terminado el proceso de multiplicación y acumulación, el valor calculado se transmite al siguiente módulo principal.
2. Módulo de detección de picos sobre la señal filtrada: Se implementa una máquina de estados que analiza cada valor de la señal filtrada y calcula si es un posible pico, o un pico QRS. Como los valores de la señal son números reales, se implementan varios módulos para poder operar con los valores de la señal en punto flotante.
3. Módulo de detección de arritmias: Se implementa una máquina de estados con el que recibiendo un pico QRS como entrada, sea capaz de almacenar las distancias más recientes, determinar según la diferencia de estas distancias, si se ha producido una arritmia y mostrar como salida un flag indicándolo.

Además de estos módulos se debe crear un módulo que los encapsule y un *testbench* para probar el funcionamiento del programa con las señales de la base de datos [4].

1.9. Organización de la memoria

En el capítulo 2 se verá cómo se ha realizado el prototipado del algoritmo en software. En la sección 2.1 se muestra el transcurso del algoritmo. En la sección 2.2, se indicarán las librerías usadas para la recopilación de datos. En la sección 2.3, se explicará el tipo de filtrado que se ha usado para la señal original. En la sección 2.4, se expondrá el algoritmo de detección de picos QRS y las metodologías seguidas. En la sección 2.5, se explicará el algoritmo de detección de arritmias y en la sección 2.6, el algoritmo para probar el prototipo y obtener estadísticas de las pruebas realizadas.

En el capítulo 3, se hablará de los distintos módulos creados para la implementación *hardware*. En la sección 1, se especificarán los módulos para las operaciones en punto flotante y las memorias utilizadas. De la sección 2 a la 4, se explicará el funcionamiento del módulo de filtrado de la señal, el módulo de detección de picos y el módulo de detección de arritmias. En la sección 5, se tratará la creación del módulo principal y el *testbench* utilizado.

En el capítulo 4, se mostrarán los resultados experimentales obtenidos, como la FPGA utilizada, el análisis de utilización de *hardware*, el análisis de *timing* y el consumo.

Finalmente, en el capítulo 5, se dará una conclusión sobre la realización de este proyecto y se sugerirán trabajos futuros por si se desea ampliar el proyecto.

Capítulo 2

Prototipado del algoritmo en software

2.1. Flujo del algoritmo

Como se ha explicado anteriormente, el algoritmo realizará una recopilación de datos para obtener la señal original del electrocardiograma de cada paciente y así hacer un filtrado de dicha señal para eliminar el ruido y centrarla, seguidamente se realizará la detección de picos con algunos métodos como establecer el *cutoff* dinámico y por último detectar las arritmias y comparar las anotaciones de la señal original con las generadas. Véase Figura 2.1.

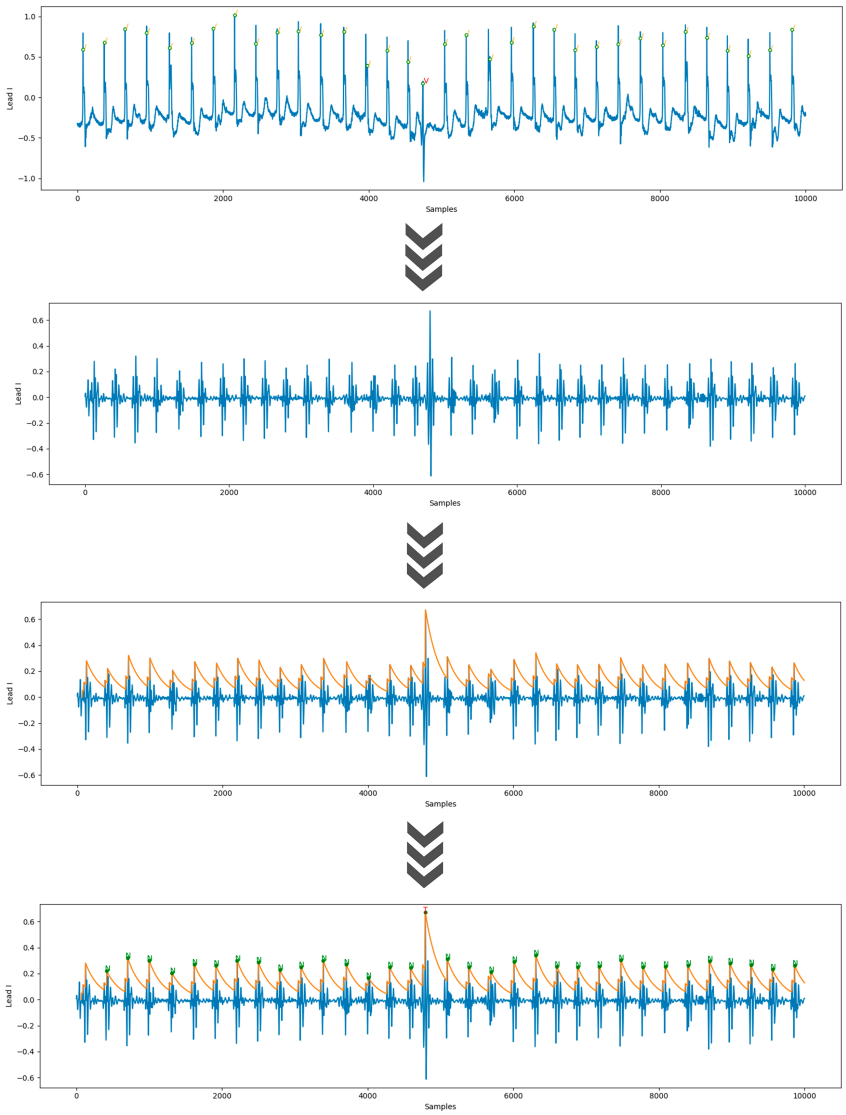


Figura 2.1: Representación del trascurso del algoritmo durante las distintas etapas

2.2. Recopilación de los datos

Para la recopilación de los datos se utilizará la librería *wfdb* [13] que se encarga de proporcionar funciones para leer y escribir archivos de diferentes formatos que contienen señales biomédicas, como archivos de registro de señales (por ejemplo, formato .dat), archivos de anotaciones (por ejemplo, formato .atr) y archivos de cabecera (por ejemplo, formato .hea).

Los pacientes vienen identificados por un identificador (por ejemplo, 101) y hay 3 ficheros por paciente, con extensiones .dat, .atr y .hea

Se descarga la base de datos [8] con la función de la librería de *wfdb*, *dldatabase* que recoge la señal del paciente y las anotaciones de los cardiólogos sobre cada pico QRS.

```
#download the database if not available
if os.path.isdir("mitdb"):
        print('You already have the data.')
else:
        wfdb.dl_database('mitdb', 'mitdb')
```

Los pacientes de la base de datos se han hecho una prueba de 30 minutos con una frecuencia de muestreo de 360 sps (*samples per second*), lo que en la señal equivale a 650000 *samples*.

```
sampfrom = 0
sampto = 650000
record = wfdb.rdsamp('mitdb/102', sampfrom=sampfrom, sampto=sampto)
annotation = wfdb.rdann('mitdb/102', 'atr', sampfrom=sampfrom, sampto=sampto)
```

Por último, para visualizar esta señal con las anotaciones de los cardiólogos y poder comparar con las anotaciones que realiza el algoritmo se usará la librería `matplotlib.pyplot` [7]. Con esto se mostrará la señal original con las anotaciones y la señal filtrada con las anotaciones del algoritmo como en la Figura 6.3

2.3. Filtrado de la señal original

Este filtrado es llevado a cabo por el filtrado FIR [2]. El filtrado FIR, que significa *Finite Impulse Response* (respuesta finita al impulso), es un tipo de filtro utilizado en el procesamiento de señales digitales y analógicas. La fórmula que se utilizará para el filtrado es la siguiente:

$$Y[i] = \sum_{k=0}^{N_x-1} b_k \cdot x[i-k]$$

Siendo b los coeficientes del filtrado y x los 99 valores de la señal original a filtrar.

Los coeficientes se almacenan en una lista de 99 valores en punto flotante simétricos que se iteran de forma circular, con lo que después de ejecutar el último valor vuelve de nuevo al primero.

Para el filtrado se usa la función `lfilter` de la librería `scipy.signal` [11].

```
filtered_signal = lfilter(filter_taps_99_6_28, 1.0, original_signal)
```

2.4. Detección de picos QRS

El algoritmo de detección de picos está representado en esta función que recibe la señal filtrada y trata de detectar los picos QRS.

Este algoritmo está basado en el que se usa en el documento [4] donde en el apartado 4.1.2 muestran una máquina de estados del proceso que realizan.

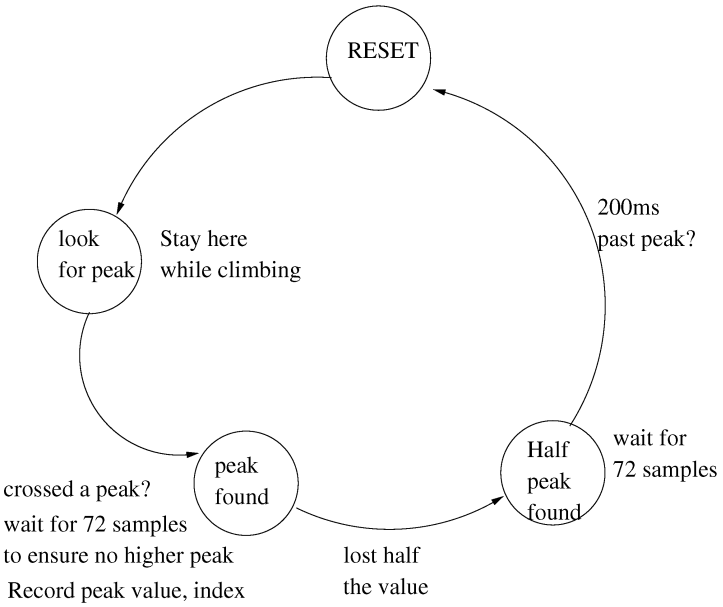


Figura 2.2: Máquina de estados de algoritmo de detección de picos de estudio de caracterización de señales usando polinomios de Hermite [4]

El algoritmo implementado difiere del que se describe en el documento, ya que el del documento es una versión simplificada de un algoritmo de detección de picos y no es tan funcional debido a la falta de un *cutoff* dinámico, el cual sí está presente en este proyecto. Esta solución es necesaria porque a veces hay una distancia entre los picos QRS superior a 72 muestras, lo que ocasiona que en algunos casos se detecte el ruido como un pico QRS, ya que después de las 72 muestras no se ha encontrado una señal. Por esta razón, en un proyecto en el que se sabe que los picos QRS estarán a distancias distintas entre sí, este algoritmo no es adecuado y debe modernizarse.

Aun así, si bien el algoritmo creado es distinto a ese, se replica el tener que esperar a 72 muestras para asegurarse de que no se encuentra un pico superior y así poder considerarlo como un pico QRS. Es por ello que definimos la variable `samples_around_peak` como 72 para comparar dicha condición. Para hallar el pico más alto se necesita definir un pico en `last_peak` y si se encuentra otro pico se realiza el máximo entre el pico encontrado y el de `last_peak`.

```
last_peak = max(last_peak, signal[i])
```

2.4.1. Uso del *cutoff*

El *cutoff* es un umbral bajo con el cual se ignora la señal. Solo cuando los valores de señal son mayores que el *cutoff*, entra en juego la detección de pico QRS. Un *cutoff* es importante pues, de lo contrario, puede que detecte falsos positivos, ya que se buscan máximos en un rango de 72 muestras y puede que se busquen máximos en un intervalo de la señal donde no hay picos QRS.

Debido a la variación de las señales entre pacientes y la variación producida por la activación del marcapasos, el uso de un *cutoff* estático, como el que se representa en la Figura 2.3, se hizo inconsistente para la detección de picos, es por ello que finalmente se utilizó un *cutoff* dinámico, como se puede ver en la Figura 2.4. Este se adapta mejor a la amplitud de ondas en la señal y así no captar picos QRS de más o de menos.

El *cutoff* dinámico recoge ese nombre, ya que su valor depende de si se ha encontrado un pico o no, ya que cuando se encuentra un pico por encima del *cutoff* anterior, el valor del *cutoff* pasa a ser el del pico, por otro lado, cada vez que no se encuentre ningún pico La función del *cutoff* pasa a ser la siguiente:

$$cutoff = cutoff - cutoff/192$$

Se ha dado el valor de 192 a la fórmula debido al buen desempeño en el programa.

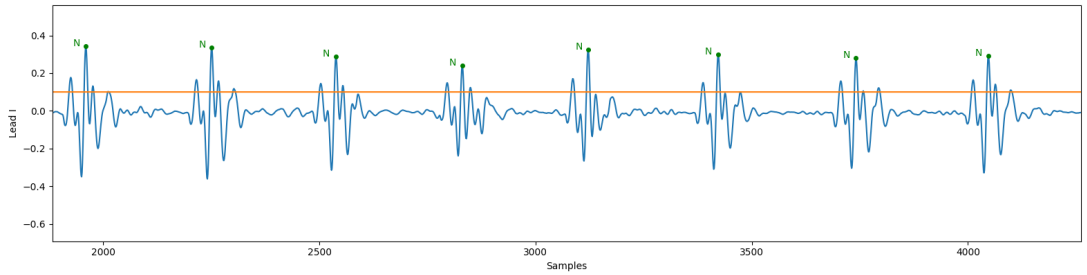


Figura 2.3: Imagen del desempeño del algoritmo con un *cutoff* estático sobre el paciente 102

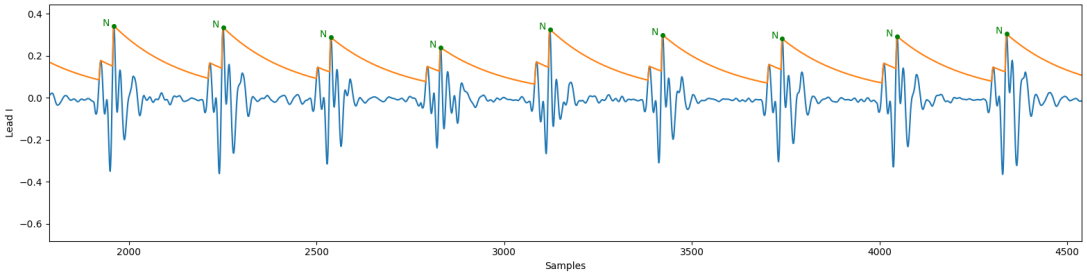


Figura 2.4: Imagen del desempeño del algoritmo con un *cutoff* dinámico sobre el paciente 102

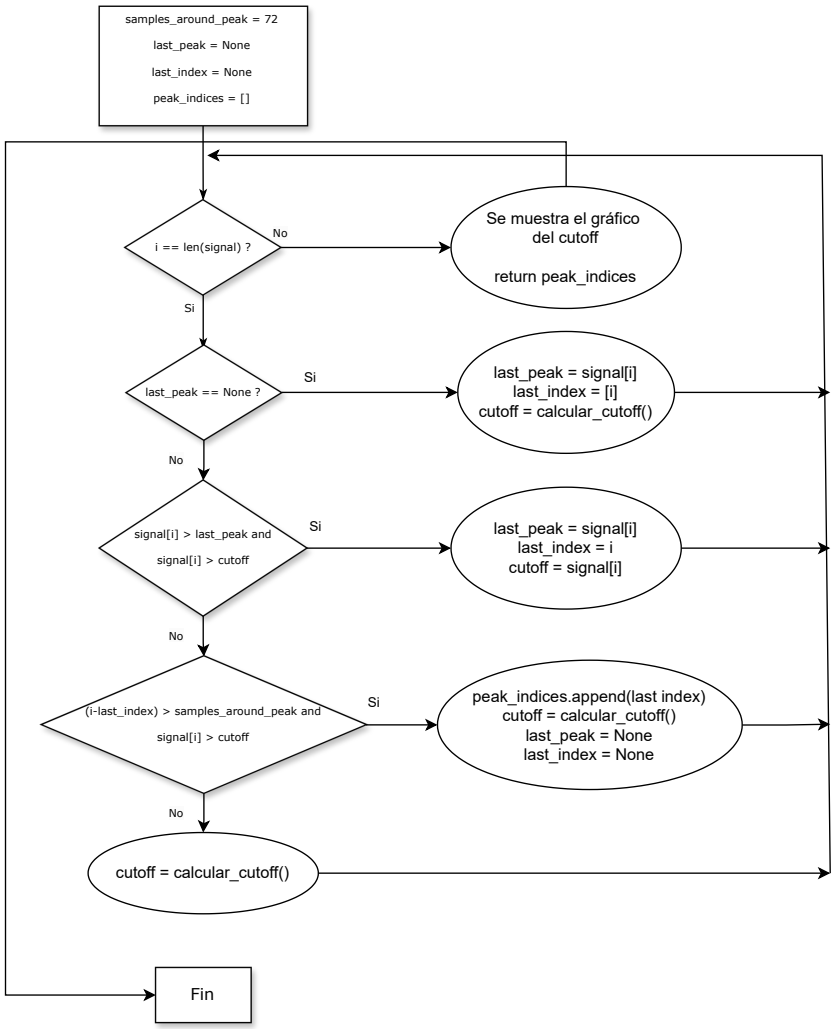


Figura 2.5: Diagrama ASM del algoritmo de detección de picos QRS

La salida de dicha función es un buffer de *samples* que sirven como índices para indicar donde se han encontrado los picos QRS y así poder pasar al módulo de detección de arritmias.

2.5. Detección de arritmias

El algoritmo de detección de arritmias se encarga de comprobar si se ha producido una arritmia según la distancia entre los picos.

En la detección de arritmias es de vital importancia establecer un límite en la distancia entre los picos para poder considerar que ha habido una arritmia o no. Esta tarea solo se pudo hacer probando con diferentes rangos y viendo el índice de aciertos producidos en las pruebas a cada paciente de las que se hablará más adelante.

El algoritmo va almacenando distancias entre los picos QRS (es por ello que en la primera iteración no se almacena nada) y se declaran varias variables:

- `last_distance`: se utiliza para almacenar la última distancia recogida y así poder compararla con la distancia actual en cálculos posteriores.
- `counter_buffer`: utilizado para tener el valor de la posición del buffer donde se escribe.
- `counter_arrythmia`: utilizado para indicar si la distancia anterior fue una arritmia.
- `TNRange`: Se utiliza para indicar si hay una distancia más grande de lo normal entre 2 picos QRS producido por una arritmia. Es importante tener esta distancia en cuenta, ya que si el ritmo del paciente vuelve a la normalidad se compararía la distancia entre el ritmo normal del paciente con el ritmo extendido por la arritmia, ya que de no tenerlo en cuenta el algoritmo lo clasificaría como arritmia como se puede ver en la Figura 2.6, por ello se compara con un valor anterior que sea el ritmo normal del paciente.

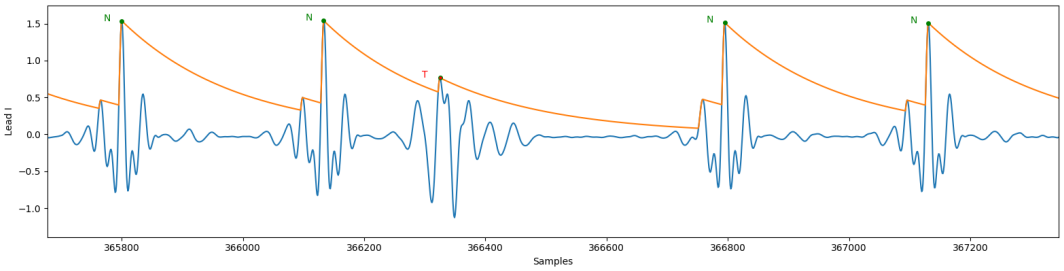


Figura 2.6: Cuando se detecta una arritmia, a veces, la siguiente distancia es considerablemente más grande de lo normal. Para no detectar falsos positivos, se omite esa distancia.

Por ello si se ha detectado una arritmia, la siguiente distancia se compara con la tercera última distancia escrita en el buffer que posiblemente sea una distancia causada por un ritmo normal. Si no se da el caso, se compara con `last_distance`.

La función que compara las distancias devuelve un carácter que va a ser el que se vaya a mostrar en la gráfica. Si el carácter es "N" significa que se ha detectado un ritmo normal y por tanto solo se muestra. Sin embargo, si el resultado es "T" significa que la distancia es más corta de lo normal, se detecta la arritmia y se ponen counter_arritmia a 1 para saber que la distancia es más corta y TNRange a true para que el algoritmo sepa que la distancia que venga después puede ser una ampliada.

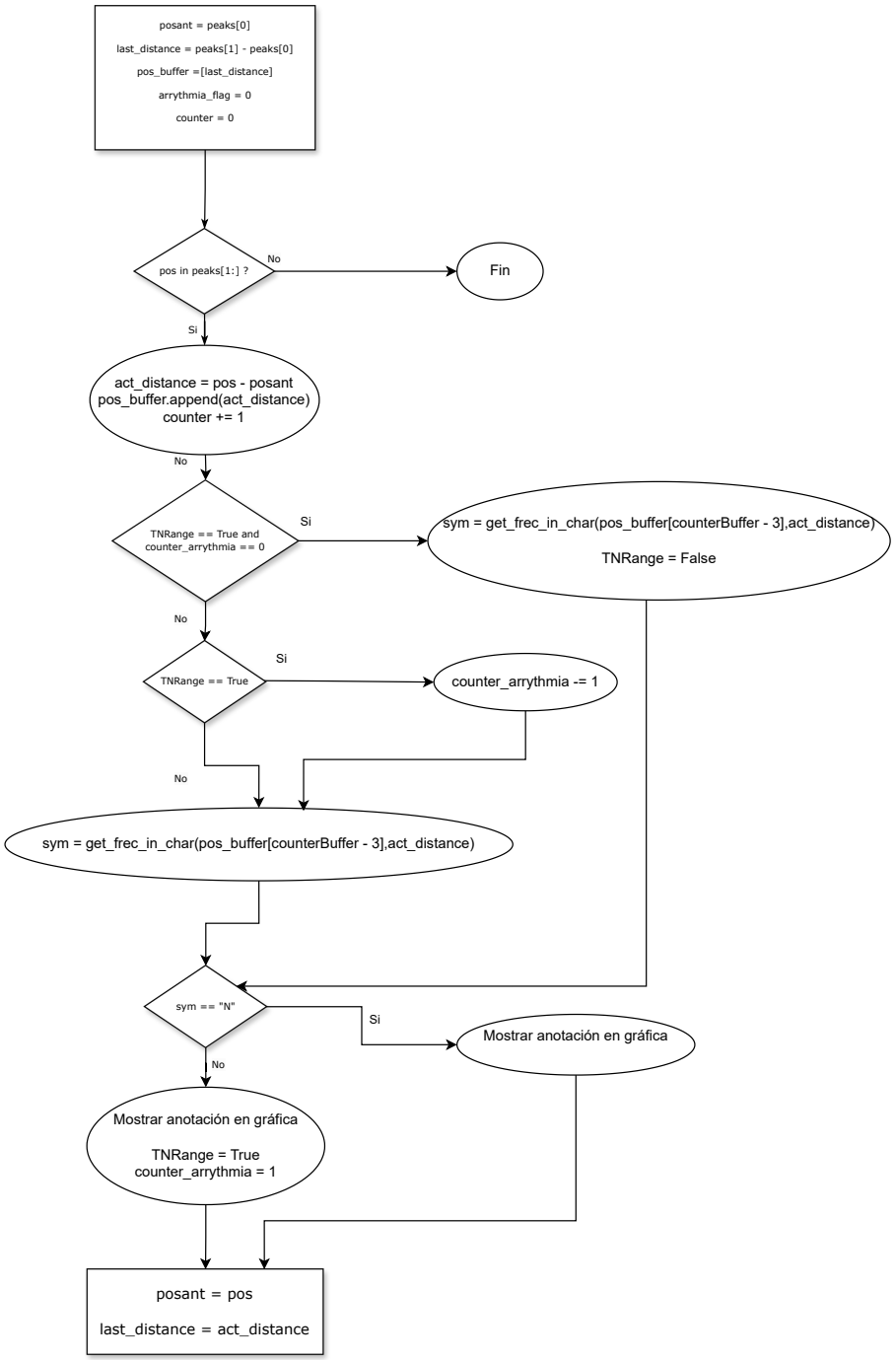


Figura 2.7: Diagrama ASM del algoritmo de detección de arritmias

La función `get_frecuency_in_char()` se encarga de calcular las distancias entre el ritmo actual y un ritmo normal. Para ello recibe como entrada ambas distancias.

Para empezar se calcula el *gap* que es simplemente la diferencia que tiene la distancia anterior con la actual.

$$gap = last\_distance - actual\_distance$$

Acto seguido, se calcula el porcentaje de la diferencia de distancia con la distancia anterior que se sabe que va a ser un ritmo normal.

$$percentaje = (gap/last\_distance) * 100$$

Si ese porcentaje es mayor que el 15 % entonces se considera que la distancia normal es mucho mayor que la actual y por tanto como la distancia actual entre 2 picos es pequeña, se da por hecho que hay una arritmia.

Nótese que no se le da importancia si el *gap* da como resultado un número negativo de cualquier tamaño, esto se debe a que este proyecto solo está pensado para detectar contracciones prematuras del corazón, por ende solo necesitamos saber si la distancia actual es menor que la anterior. Además, ningún paciente parece padecer ninguna arritmia de otro tipo.

2.6. Pruebas con el algoritmo

Se han realizado una serie de pruebas para probar el algoritmo para comprobar si las posiciones donde se ha detectado un pico QRS coinciden con las posiciones de los picos detectados por los cardiólogos, y además se encargan de comparar las anotaciones de los cardiólogos con las generadas por el algoritmo.

Con dichas estadísticas es posible comparar el porcentaje de aciertos, en los que se comprende el número de falsos positivos (referido a los ritmos normales que el algoritmo considera arritmias) y falsos negativos (referido a las arritmias que el algoritmo considera un ritmo normal).

Para desarrollar estas pruebas, se ha creado una clase Pair que contenga, por cada iteración del algoritmo de detección de arritmias, el carácter que indica si el algoritmo ha detectado una arritmia o no y la posición del pico QRS analizado. Dicho objeto se inserta en una lista de Pair para luego poder comparar con las anotaciones de la señal original.

Una vez se rellena todo el buffer de Pair, se comprueban 2 cosas.

1. Si se ha detectado un pico QRS en la señal filtrada y se corresponde con el pico de la señal original situado en un *sample* de una posición aproximada.
2. Si en el caso de que se haya detectado el pico, las anotaciones de los cardiólogos coinciden con las generadas por el algoritmo.

Para este proyecto, solo se valora si el paciente tiene un ritmo normal o una arritmia, pero las anotaciones que contiene la señal original pueden simbolizar otros problemas como la entrada del marcapasos u otros problemas con la onda T. En la clase *Annotation* de la librería *wfdb* [13], vienen explicadas todas las posibles anotaciones que puede haber como se muestra debajo.

```
    ann_labels = [
        AnnotationLabel(0, "-", 'NOTANN', 'Not-an-actual-annotation'),
        AnnotationLabel(1, "N", 'NORMAL', 'Normal-beat'),
        AnnotationLabel(2, "L", 'LBBB', 'Left-bundle-branch-block-beat'),
        AnnotationLabel(3, "R", 'RBBB', 'Right-bundle-branch-block-beat'),
        AnnotationLabel(4, "a", 'ABERR', 'Aberrated-atrial-premature-beat'),
        AnnotationLabel(5, "V", 'PVC', 'Premature-ventricular-contraction'),
        AnnotationLabel(6, "F", 'FUSION', 'Fusion-of-ventricular-and-normal-beat')
            ,
        AnnotationLabel(7, "J", 'NPC', 'Nodal-(junctional)-premature-beat'),
        AnnotationLabel(8, "A", 'APC', 'Atrial-premature-contraction'),
        ...
        AnnotationLabel(12, "/", 'PACE', 'Paced-beat'),
        AnnotationLabel(13, "Q", 'UNKNOWN', 'Unclassifiable-beat'),
        AnnotationLabel(14, "~", 'NOISE', 'Signal-quality-change'),
        AnnotationLabel(16, "|", 'ARFCT',  'Isolated-QRS-like-artifact'),
        ...
        AnnotationLabel(38, "f", 'PFUS',  'Fusion-of-paced-and-normal-beat'),
        ...
    ]
```

Por ello en este proyecto solo se prestará atención a la anotación A y a la anotación V que simbolizan las contracciones prematuras de la aurícula y el ventrículo, las demás anotaciones sobre el pico QRS serán consideradas como ritmos normales.

Para comparar los picos de ambas listas, se examina el sample en el que se encuentra dicho pico en la señal original. Como los picos de la señal filtrada presentan un desfase de 50 samples respecto a la señal original, simplemente hay que comprobar si hay un pico QRS 50 samples más adelante en la señal filtrada. Adicionalmente, para evitar problemas por el filtrado de la señal, se ha establecido un margen de error de 20 samples para incluir los picos que puedan haberse desfasado debido al filtrado. Si por otro lado, el pico no se ha detectado donde tendría que haber un pico QRS puesto en la señal original, se pone doble guión para simbolizarlo.

Se realiza un conteo de las anotaciones correctas en total, las anotaciones incorrectas en total, las anotaciones correctas solo de los picos detectados como arritmia, las incorrectas de ese mismo tipo, y los picos no registrados, como se ve en la Figura 2.8. Todas las pruebas posteriores han sido realizadas sobre la señal del paciente 102 de principio a fin.

Descripción	Valor
Pico detectado	2141
Pico no detectado	46
Anotaciones generales correctas	2140
Anotaciones generales incorrectas	1
Anotaciones de arritmias correctas	4
Anotaciones de arritmias incorrectas	1

Figura 2.8: Métricas de detección de arritmias

Con las estadísticas anteriores se pueden hallar los picos totales que tiene la señal original, el porcentaje de picos detectados, el porcentaje de picos no detectados, el porcentaje de arritmias detectadas correctamente. Esto está representado en la Figura 2.9.

Descripción	Valor
Valores totales	2187
Valores totales detectados	97.90%
Valores totales no detectados	2.10%
Valores totales correctos	99.95%
Valores totales incorrectos	0.05%
Arritmias totales correctas	80.00%

Figura 2.9: Porcentajes de detección de arritmias

Adicionalmente se puede sacar una matriz de confusión como en la Figura 2.10, donde se establece como verdadero la aparición de arritmias en la señal original y como positivo a la detección de arritmias sobre la señal procesada por el algoritmo. En el caso del paciente 102, no ha habido arritmias en 2136 picos, ha habido 4 arritmias detectadas correctamente y un falso positivo que indica que el algoritmo ha detectado una arritmia, pero no era.

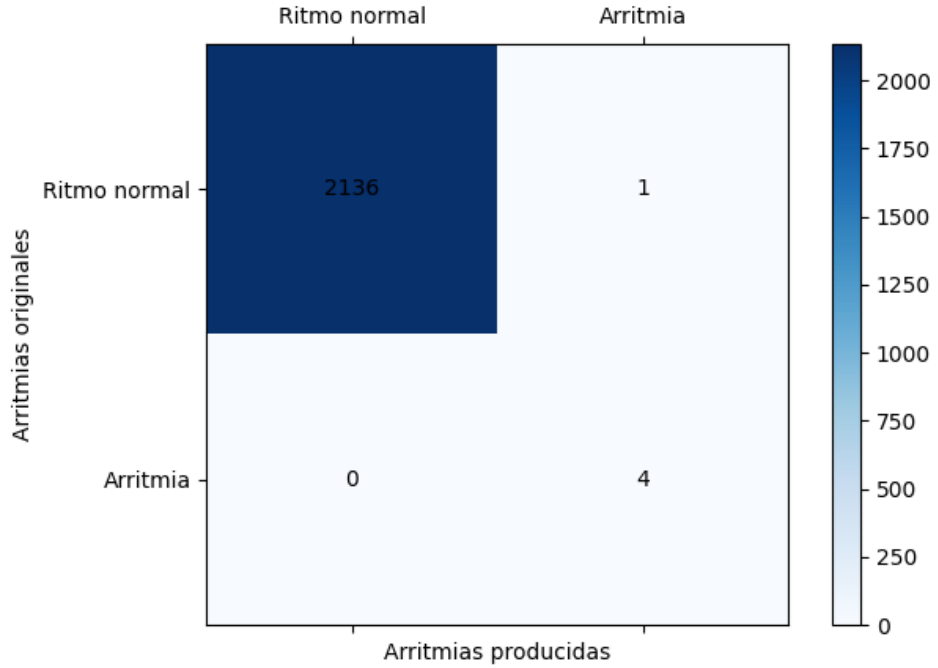


Figura 2.10: Matriz de confusión del funcionamiento del algoritmo sobre la señal del paciente 102

Dichas pruebas mostradas se aplican solo para un paciente, pero es posible aplicar estas pruebas a todos los pacientes. Para ello se ha creado un nuevo programa de *python* que se encarga de realizar la misma prueba para los pacientes cuyo id está almacenado en un buffer.

Este programa tiene 2 modos, uno procesa un paciente individualmente y el otro itera sobre una lista con los identificadores de los pacientes para procesarlos a todos. La lógica del algoritmo visto anteriormente está contenida en una nueva función llamada `calculations()`.

Las pruebas que se realizan para este algoritmo son iguales que en el programa anterior, pero

también se han realizado las siguientes estadísticas.

1. La media de los picos detectados de cada paciente.
2. La media de las arritmias correctas detectadas en cada paciente.

Al realizar pruebas con los demás pacientes de la base de datos, los resultados no son del todo favorables. Esto se debe a la variabilidad entre los pacientes: algunos presentan muchas más arritmias que otros, algunos tienen un marcapasos que cambia de ritmo y otros presentan muchos fallos causados por el sensor. Al final, la manera más efectiva de evaluar y mejorar el algoritmo fue observando su comportamiento en cada caso específico y analizando los motivos de su comportamiento frente a la señal original.

Capítulo 3

Implementación *hardware*

Para implementar el algoritmo en *hardware* dividimos en módulos el algoritmo de filtrado, el algoritmo de detección de picos y el algoritmo de detección de arritmias, estos los unificamos en un módulo principal y probamos la simulación con un *testbench*. Véase Figura 3.1

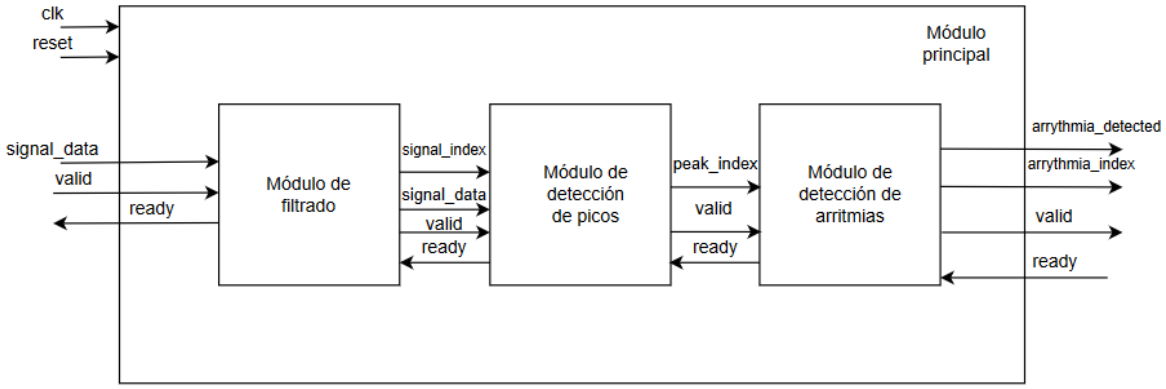


Figura 3.1: Diagrama principal de todos los módulos a evaluar

3.1. Módulos adicionales

Como los valores de las señales están en punto flotante y como se necesitan almacenar valores en una memoria, para poder operar con ellos, es necesario utilizar módulos *hardware* que permitan hacer dichas operaciones. Para ello, se utilizarán los *IP cores* (*Intelectual Property cores*) de Vivado [15]. Estos *IP cores* son bloques que pueden realizar funciones específicas de *hardware*, optimizando así el diseño y facilitando la implementación de operaciones complejas. En este proyecto utilizaremos módulos de multiplicación y suma, resta, división y comparación de números en punto flotante, aprovechando las capacidades de los *IP cores* de Vivado para manejar estos cálculos de manera eficiente y precisa.

3.1.1. Módulos ROM y RAM

Se necesitan módulos de ROM y RAM para las pruebas y para el módulo de filtrado.

Para poder hacer una simulación, en la parte de las pruebas, se necesitan tres memorias ROM: Una para almacenar los valores de la señal original, otra para almacenar un bit que indique si se ha producido una arritmia o no, y otra para almacenar el índice de cada pico. Las configuraciones de estos módulos se muestran respectivamente en la Figura 3.2, en la Figura 3.3 y en la Figura 3.4

Es importante desactivar la opción de *primitive output* para que no se añada un registro extra al principio y la simulación se ejecute en cada tiempo correspondiente.

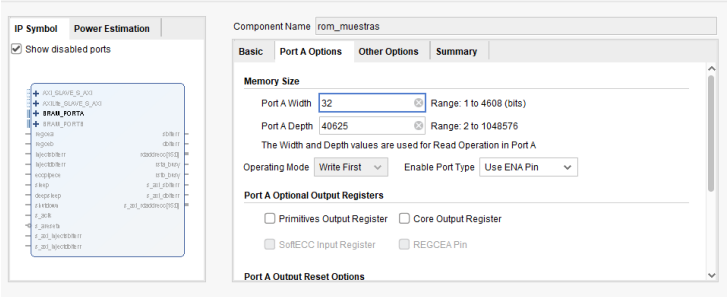


Figura 3.2: Se establece la profundidad y la anchura de palabra de ROM encargada de almacenar los valores de la señal original

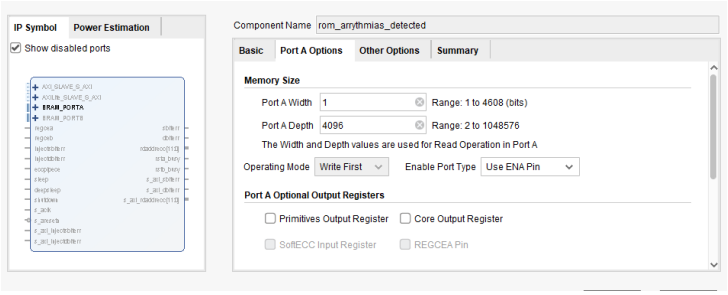


Figura 3.3: Se establece la profundidad y la anchura de palabra de ROM encargada de almacenar el flag indicador de arritmia

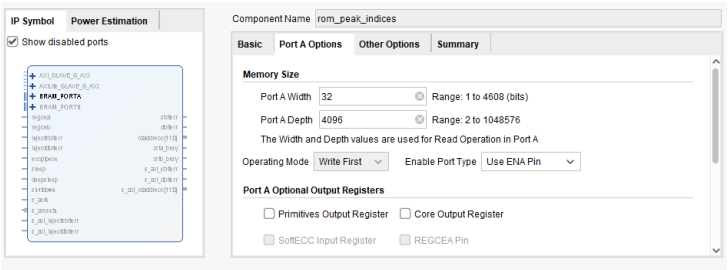


Figura 3.4: Se establece la profundidad y la anchura de palabra de ROM encargada de almacenar el índice de cada pico

Para el módulo de filtrado, es necesario el uso de una ROM que contenga los coeficientes de filtrado y una RAM que contenga los valores de la señal original para ser procesados.

- La ROM se configura como *single port ROM*, este tiene 99 filas de profundidad y de anchura de palabra tiene 32 bits. Se muestra la configuración de este módulo en la Figura 3.5
- La RAM se configura como *single port RAM* y se mantiene desactivado el valor de `primitive output`. Se muestra la configuración de este módulo en la Figura 3.6

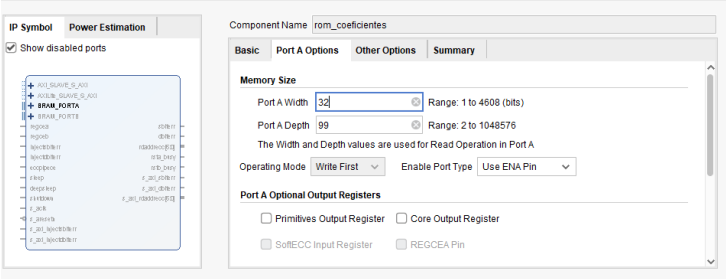


Figura 3.5: Se establece la profundidad y la anchura de palabra de ROM encargada de almacenar los coeficientes

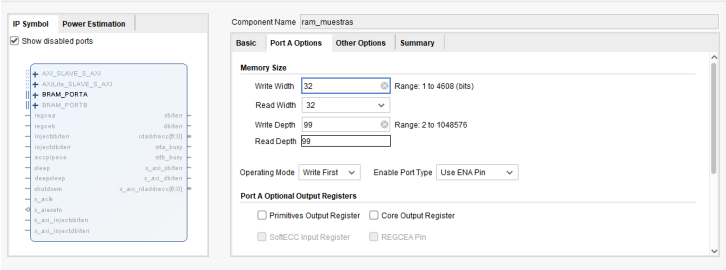


Figura 3.6: Se establece la profundidad y la anchura de palabra de RAM encargada de almacenar los valores de la señal original

3.1.2. Módulos punto flotante

Se han definido varios módulos para hacer las distintas operaciones en punto flotante, ya que en VHDL no se pueden hacer estas operaciones directamente, se necesitan usar otros módulos especializados. Como se operan con valores en punto flotante simple, las señales tienen que ser de 32 bits.

En este programa se necesitan 5 tipos de módulos de operaciones.

- Módulo comparador mayor que: se utiliza para comparar varias señales en el módulo de detección de picos como son:
  - `signal_data gt last_peak`
  - `signal_data gt cutoff`
  - `last_peak gt cutoff`
- Módulo divisor y resta: se utilizan en conjunto para calcular el *cutoff* que tiene la operación:

$$cutoff = cutoff - cutoff/192$$

- Módulo multiplicación y suma: Se usa para poder multiplicar los valores de las muestras con los valores de los coeficientes en el módulo de filtrado.

3.2. Módulo de filtrado

3.2.1. Señales de entrada y salida

Las señales de entrada son:

- `clk` y `reset`.

- `input_signal_data`: Señal que recibe las muestras de la señal original.
- `input_valid` e `input_ready`: Flags que sirven para sincronizar el módulo con la llegada de muestras.

Las señales de salida son:

- `output_filter_data`: Saca los valores de la señal filtrada.
- `output_filter_index`: Saca los índices de cada valor de la señal filtrada.
- `output_valid` y `output_ready`: Sincronizan el módulo del filtrado con el módulo de detección de picos.

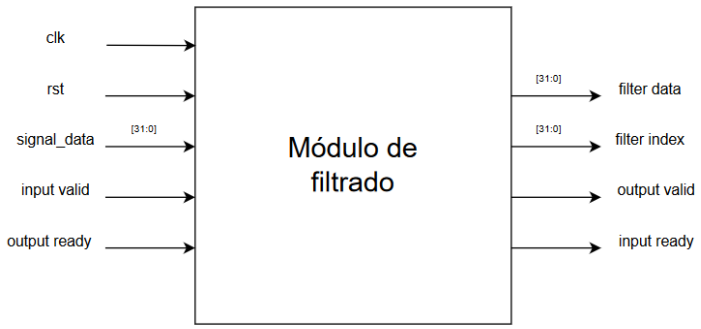


Figura 3.7: Entradas y salidas del módulo de filtrado

3.2.2. Máquina de estados

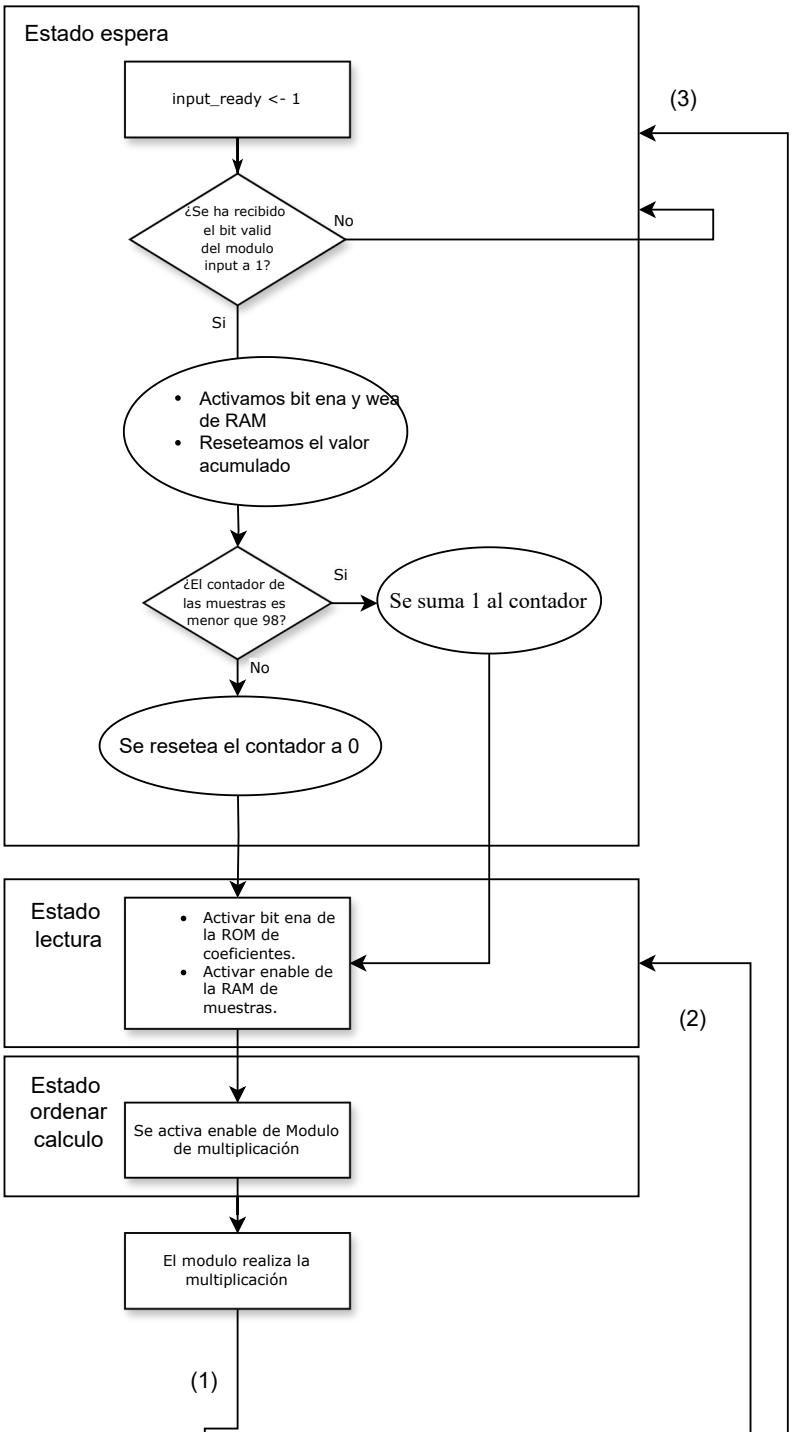


Figura 3.8: Diagrama ASM de Módulo de filtrado de señal

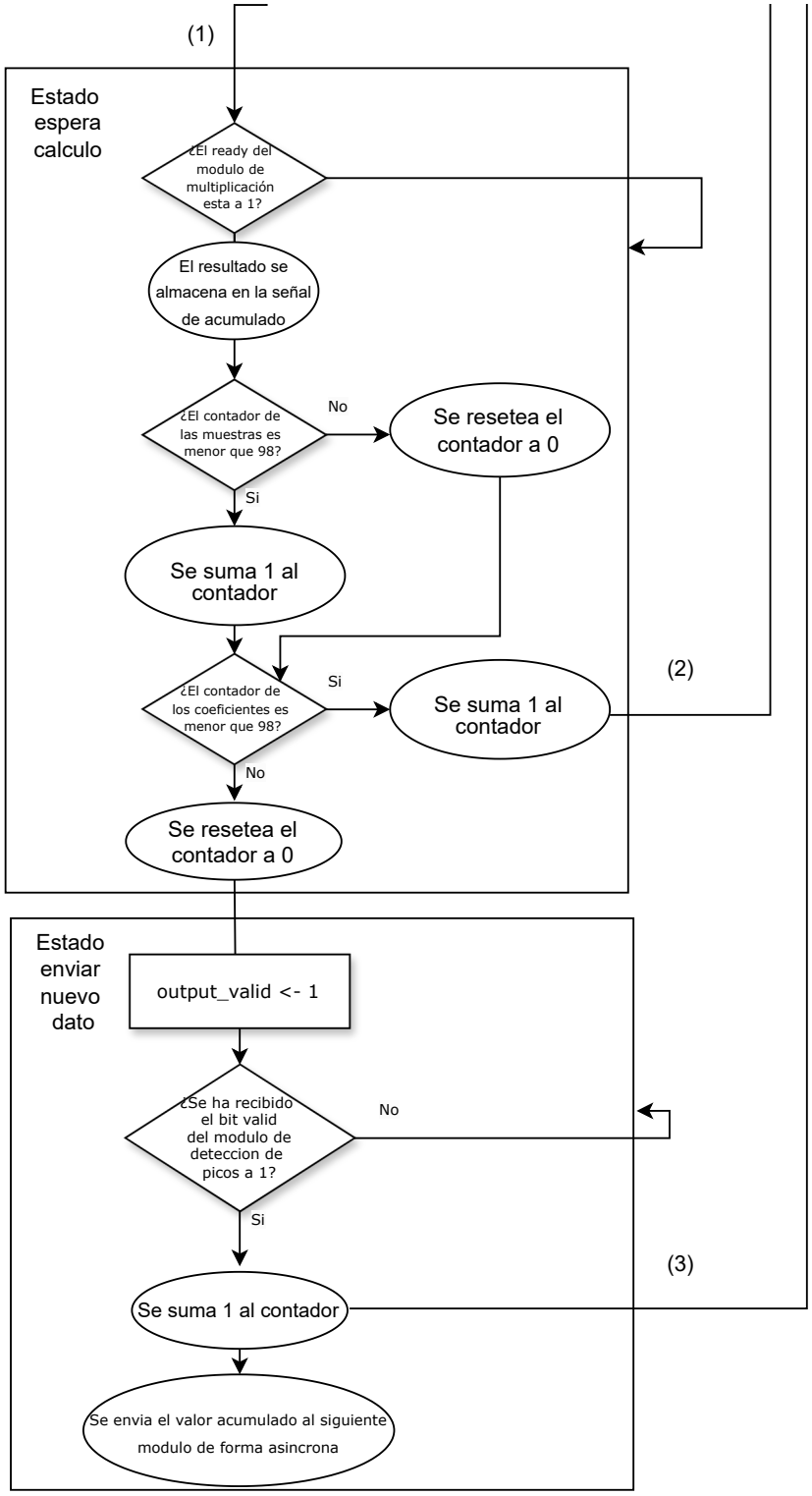


Figura 3.9: Diagrama ASM de Módulo de filtrado de señal

El envío de datos al output se realiza de forma combinacional, se pasa a `output_data` el valor del acumulado y el índice a su respectivo output.

- Estado de espera: En el estado de espera se activa la señal de `ready` y se espera a que se envíe un valor de la señal sin filtrar, después se borra el valor de la solución de la multiplicación anterior, se activan las señales de escritura de la RAM y se establece el índice donde se va a escribir la muestra.
- Estado de lectura: Se activa el bit de lectura de los coeficientes y de las muestras.
- Estado para ordenar el cálculo: Se activa el flag del módulo de multiplicación y suma.
- Estado de espera del cálculo: Espera a que termine el módulo de multiplicación y suma esperando la señal de `ready_muladd` y se almacena el resultado, también se actualiza el contador de los coeficientes, de las muestras y dependiendo de si el índice de coeficientes es menor de 98, se va al estado de lectura o el estado de enviar un nuevo dato al siguiente módulo.
- Estado de envío de nuevo dato: Este estado sincroniza el siguiente módulo, activando así el bit de `valid` a 1 y esperando el bit de `ready` del siguiente módulo para poder enviar el valor filtrado.

3.2.3. Módulos utilizados

El módulo de memoria ROM que almacena los coeficientes devuelve el valor almacenado en la posición de memoria correspondiente al índice proporcionado. Además, tiene un funcionamiento circular: al llegar al índice 99, el siguiente índice será 0. Su diagrama está representado en la Figura 3.10.

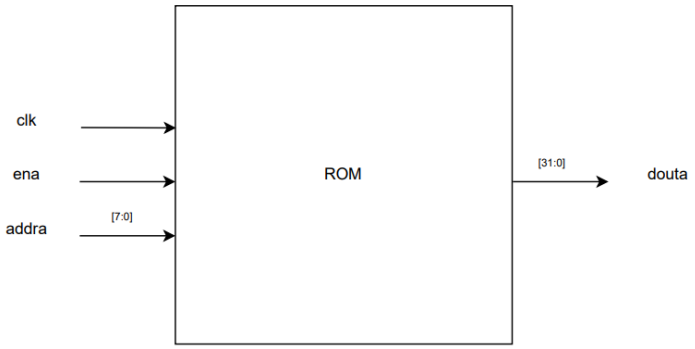


Figura 3.10: Diagrama de la ROM que se usa en el filtrado de la señal

El módulo de la memoria RAM, se encarga de almacenar los valores de la señal original y posteriormente de escribir los resultados del módulo de multiplicación y suma. Este también tiene un funcionamiento circular y su diagrama se representa en la Figura 3.11.

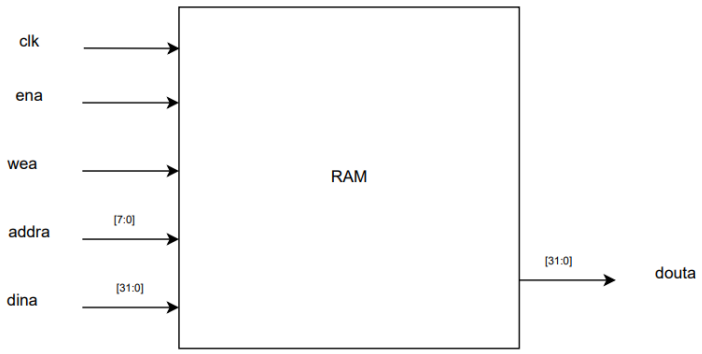


Figura 3.11: Diagrama de la RAM que se usa en el filtrado de la señal

El módulo de multiplicación y suma de números en punto flotante se encarga de multiplicar el valor del coeficiente con el valor de la señal correspondiente y al terminar, lo suma a los valores acumulados. Su diagrama se puede ver en la Figura 3.12.

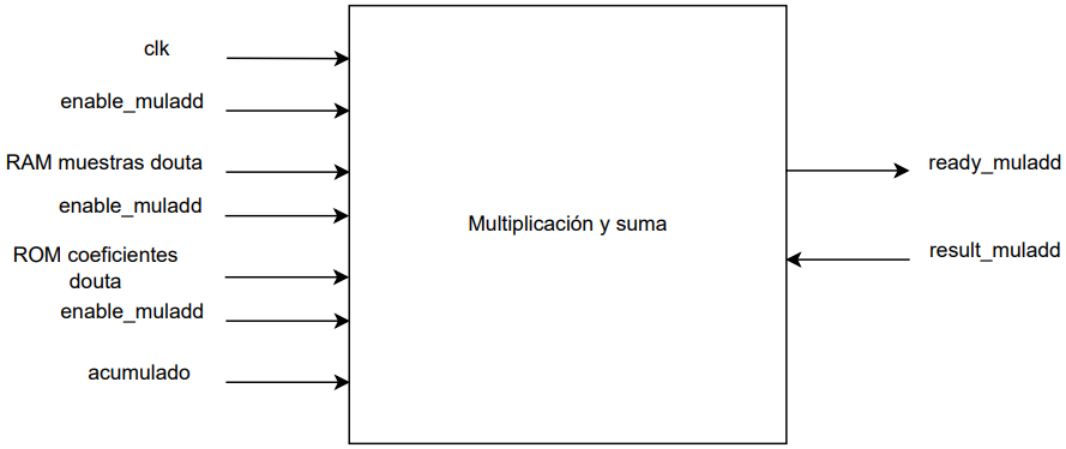


Figura 3.12: Diagrama de Módulo de multiplicación y suma.

3.3. Módulo de detección de picos

Este módulo se encarga de detectar los picos QRS de la señal filtrada.

3.3.1. Señales de entrada y salida

- `clk` y `reset`.
- `input_signal_data`: señal que recibe las muestras de la señal filtrada.
- `input_signal_index`: señal que recibe los índices de las muestras de la señal filtrada.
- `input_valid` e `input_ready`: son flags que sirven para sincronizar el módulo con la llegada de muestras.

Las señales de salida son:

- `output_peak_index`: saca los índices de los picos detectados.

Figura 3.13: Entradas y salidas del módulo de detección de picos

- `output_valid` y `output_ready`: Se encargan de sincronizar el módulo de detección de picos con el módulo de detección de arritmias.

3.3.2. Máquina de estados

- Estado de espera: En este estado, se activa la señal de `ready` y se espera a que se envíe un valor desde el módulo de filtrado.
- Estado de comprobar índice:
  - Si no hay un pico (es decir, si la señal `last_peak` está en 0), se asigna el valor a la señal y se registra el índice. Luego, se pasa al estado de actualizar el *cutoff*, activando la señal de división para que los módulos de división y resta de valores en punto flotante comiencen a calcular el nuevo valor del *cutoff*.
  - Si hay un pico, se ordena la comparación `signal_data >last_peak` pasando las señales correspondientes al módulo de comparación en punto flotante. Además, se anticipa y se realiza la comparación `last_peak >cutoff` para, en caso de que la condición anterior no se cumpla, tener esta comparación lista y poder pasar directamente al siguiente estado. También se activan las señales del módulo de comparación correspondiente. El siguiente estado es el de espera a la condición en la que la señal es mayor que el pico máximo.
- Estado de actualizar *cutoff*:
  - En este estado se espera la señal `ready` del módulo de resta, ya que es la última operación necesaria para calcular el *cutoff*. Primero se ejecuta el módulo de división para calcular `cutoff/192` y luego la resta `cutoff - cutoff/192`. Cuando la señal `ready_sub` esté en '1', se actualiza el *cutoff* y se pasa al estado de espera, terminando así la iteración.
- Estado de espera a la condición en la que la señal es mayor que el pico máximo:
  - Cuando las señales `ready` de los comparadores estén en '1', se podrán ejecutar las funcionalidades de este estado, el cual tiene tres condiciones:
    - Si se ha encontrado un valor mayor que `last_peak`, este valor se convierte en el nuevo `last_peak` y el nuevo *cutoff*; además, se actualiza el índice.
    - La señal que indica la condición de si han pasado 72 picos sin haber encontrado un pico superior se calcula de forma combinacional. Si se cumplen las condiciones de que han pasado 72 muestras sin encontrar un valor mayor que `last_peak` y que `last_peak` es mayor que el *cutoff*, se pasa directamente al estado de envío de nuevo pico para enviar el pico QRS.

    - Si ninguna de las condiciones anteriores se cumple, simplemente se ordena la actualización del *cutoff* activando el `enable` del módulo de división y se pasa al estado correspondiente.

- Estado de envío de nuevo pico:
    - En este estado, se activa la señal de `valid` a '1' y se espera a que el módulo de detección de arritmias envíe la señal de `ready`. Luego, se reinician las señales de `last_peak` y `last_index` a '0' y se actualiza el *cutoff*.

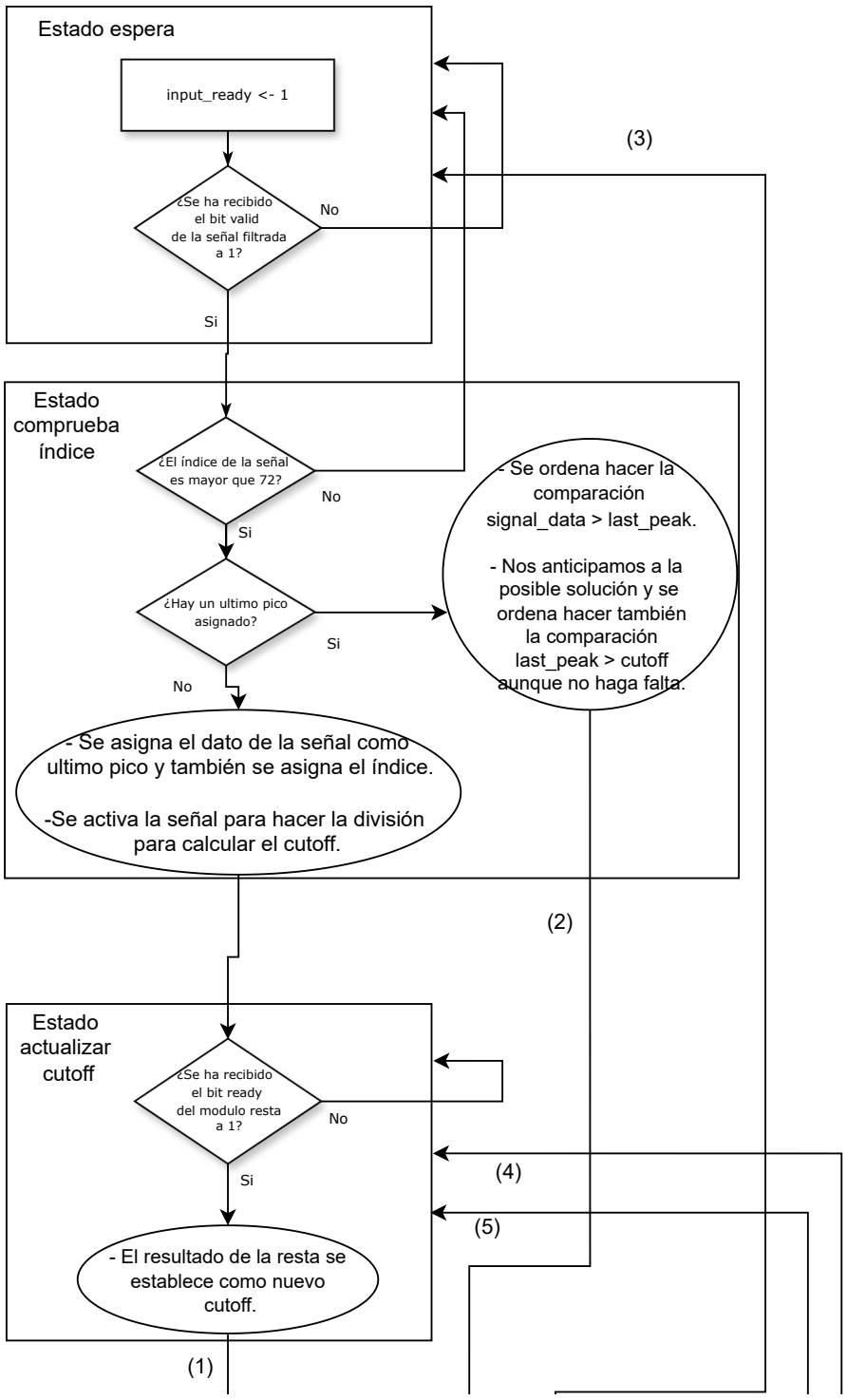


Figura 3.14: Diagrama ASM de Módulo de detección de picos

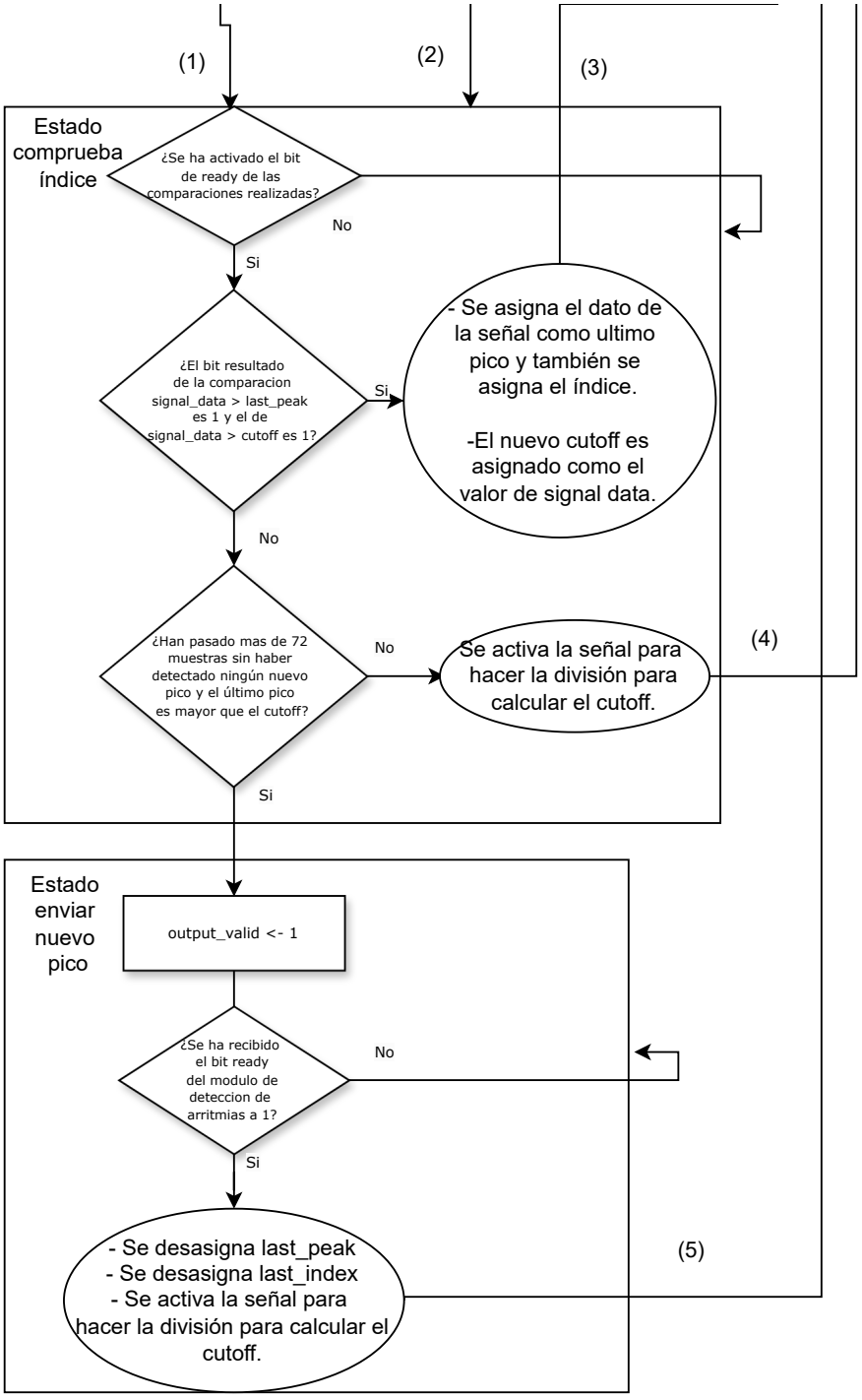


Figura 3.15: Diagrama ASM de Módulo de detección de picos

De manera combinacional se pasa como output `last_index` pero el módulo de detección de arritmias se activa cuando `input_valid` se activa usando así el `last_index` correspondiente.

3.3.3. Módulos utilizados

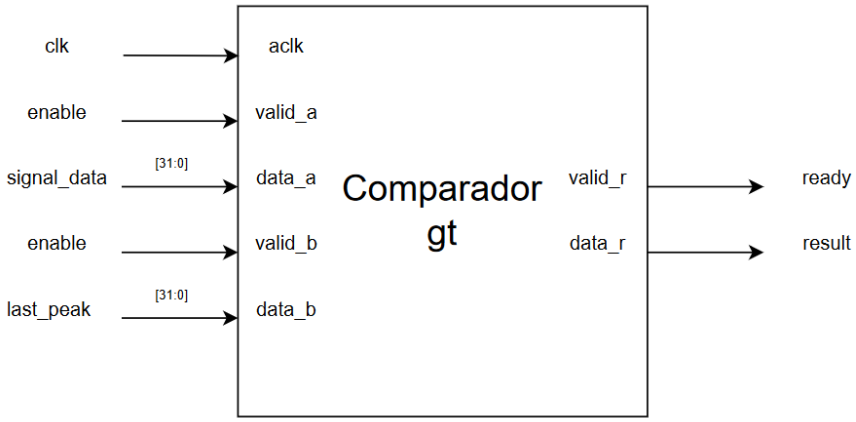


Figura 3.16: Entrada y salida del módulo de comparador

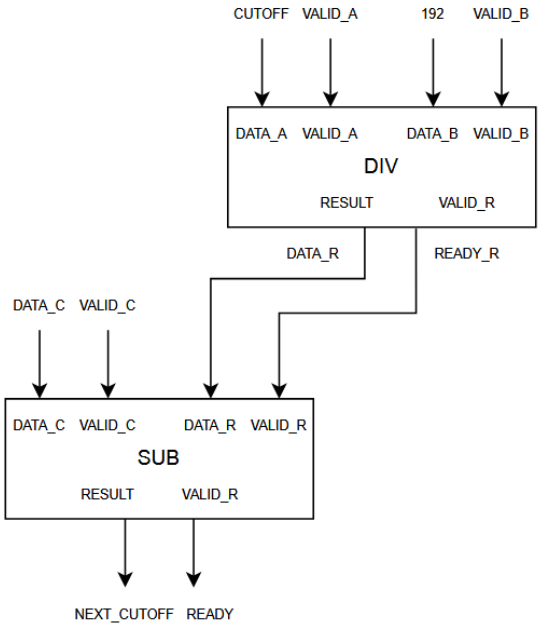


Figura 3.17: Funcionamiento de la conexión de los módulos de divisor y restador

3.4. Módulo de detección de arritmias

El módulo de detección de arritmias se encarga de detectar si la distancia entre 2 picos QRS es considerada una arritmia o no.

3.4.1. Señales de entrada y salida

Las señales de entrada de este módulo son:

- `clk` y `reset`.
- `input_peak_index`: señal que recibe las muestras de los picos QRS.
- `input_valid` e `input_ready`: son *flags* que sirven para sincronizar este módulo con el módulo de detección de picos.

Las señales de salida son:

- `output_arrythmia_detected`: flag que saca 0 si el ritmo es normal y 1 si se ha detectado una arritmia.
- `output_arrythmia_index`: valor que indica en que *sample* se ha producido la arritmia.
- `output_valid` y `output_ready`: para la sincronización con el módulo *output*.

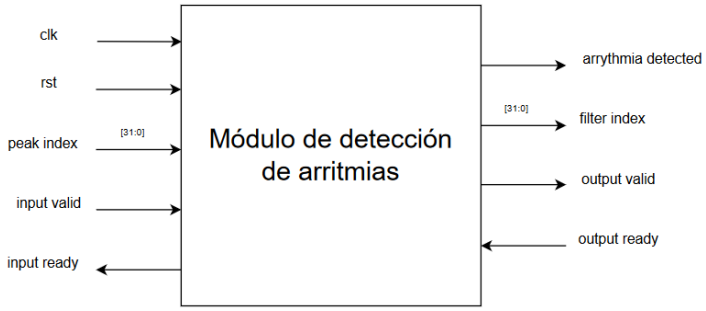


Figura 3.18: Entradas y salidas del módulo de detección de arritmias

3.4.2. Máquina de estados

- Estado de espera: En este estado se activa la señal de *ready* y se espera a que se envíe un pico QRS. Después, se pasa al estado de hallar primera distancia.
- Estado hallar primera distancia:
  - Si el contador es 0 significa que se recibe el primer pico registrado por lo que se guarda para más tarde y se pasa al estado de espera.
  - Si el contador es 1 significa que se recibe el segundo pico y por tanto se compara con el anterior hallando la primera distancia después pasa al estado de calcular distancia actual.
  - Si no se cumple ninguna condición se pasa al estado de calcular distancia actual.
- Estado de calcular distancia actual: Se incrementa el contador en 1 y se calcula la distancia actual.
- Estado actualización de buffer: Se actualizan las variables que crean un buffer ficticio, moviendo los valores una posición cuando se añade la distancia actual, similar al funcionamiento de una cola.
- Estado compara distancia actual:
  - Según lo explicado en la implementación del algoritmo, la señal `TNRange` simboliza la distancia entre el pico detectado como arritmia y el pico normal actual. Este flag se

activa cuando se ha detectado una arritmia, ya que `last_distance` puede ser mayor de lo normal.

- Para calcular el *gap* , cuando el flag de `TNRange` está activo y el de arritmia detectada no lo está, se compara la distancia actual con la distancia de hace tres ciclos (`last_distance - 3`).
- Si esta condición no se cumple, el gap se calcula comparando con la `last_distance`.
- Además, si la distancia anterior correspondía a una arritmia, se desactiva el flag de arritmia detectada.

- Estado procesa porcentaje:
  - Se calcula el porcentaje de forma combinacional. Si el flag de porcentaje es igual a 1, significa que el porcentaje calculado es mayor de lo esperado, por lo que se activan las flags de `TNRange` y `counter_arrhythmias`.
  - Independientemente de las flags, el índice y la distancia actual se actualizan a `last_distance` y `last_index`.
- Estado envío bit: Se activa la señal de *valid* y se espera a la señal de *ready* para enviar el dato al módulo de salida.

Además, se calcula el porcentaje del *gap* entre las 2 distancias con una distancia normal de forma combinacional.

- Estas señales están en complemento a 2, por lo que cuando los números son negativos, el bit más significativo se cambia a 1. Dado que solo se consideran los números positivos, solo se toman en cuenta los números cuyo bit más significativo sea 0.
- Al inicio, la señal de la última distancia es x"00000000". Al comparar esta distancia con la actual, el resultado será una distancia muy grande que activará el bit del porcentaje calculado de forma combinacional. Este caso específico se ignora en las comparaciones.
- En lugar de dividir el *gap* entre `distance_for_calc`, siendo g = *gap* y d = `distance_for_calc`, se utiliza la siguiente fórmula:

$$g/d > 0{,}15$$

$$g > 0{,}15 \cdot d$$

$$g > d \gg 5 + d \gg 3 + d$$

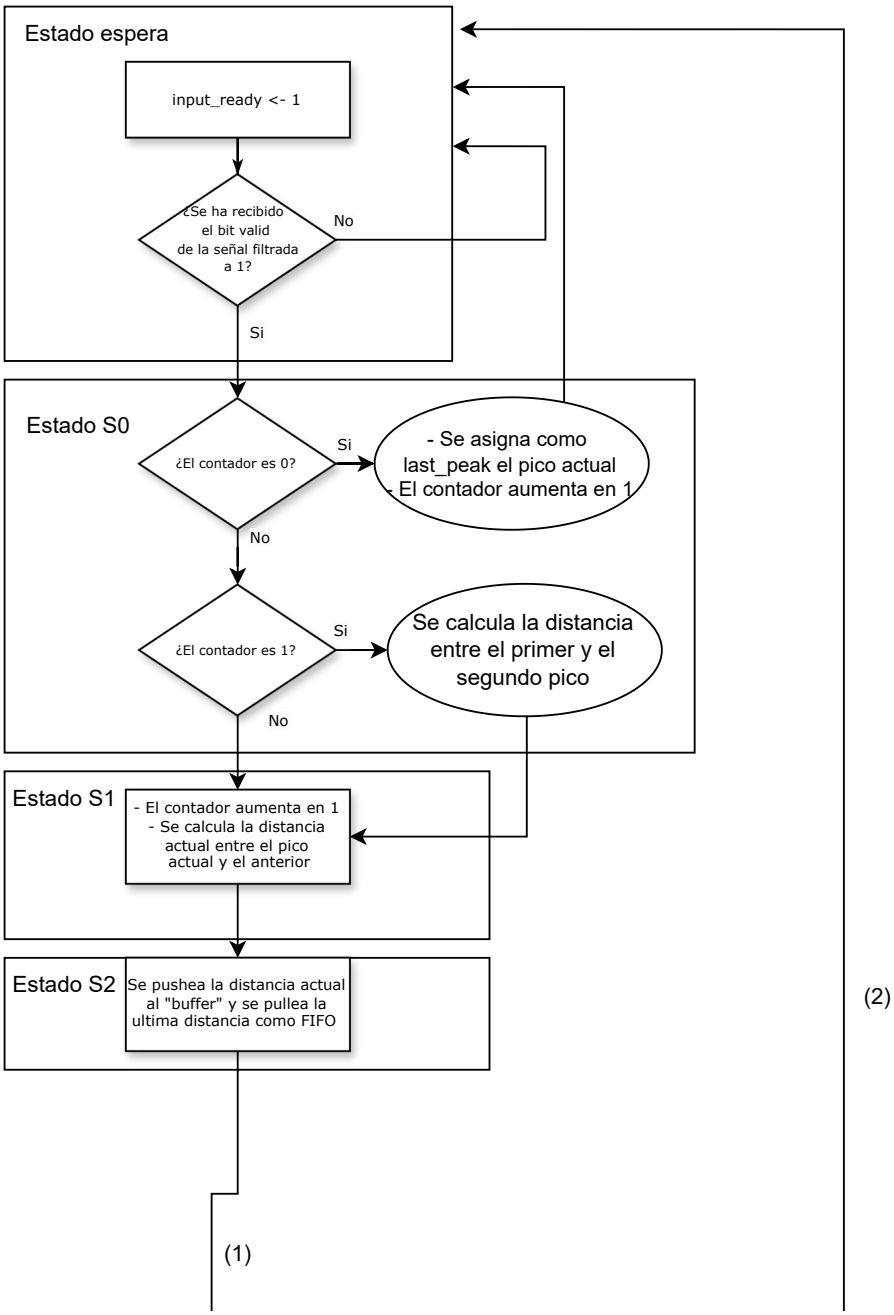


Figura 3.19: Diagrama ASM de Módulo de filtrado de señal

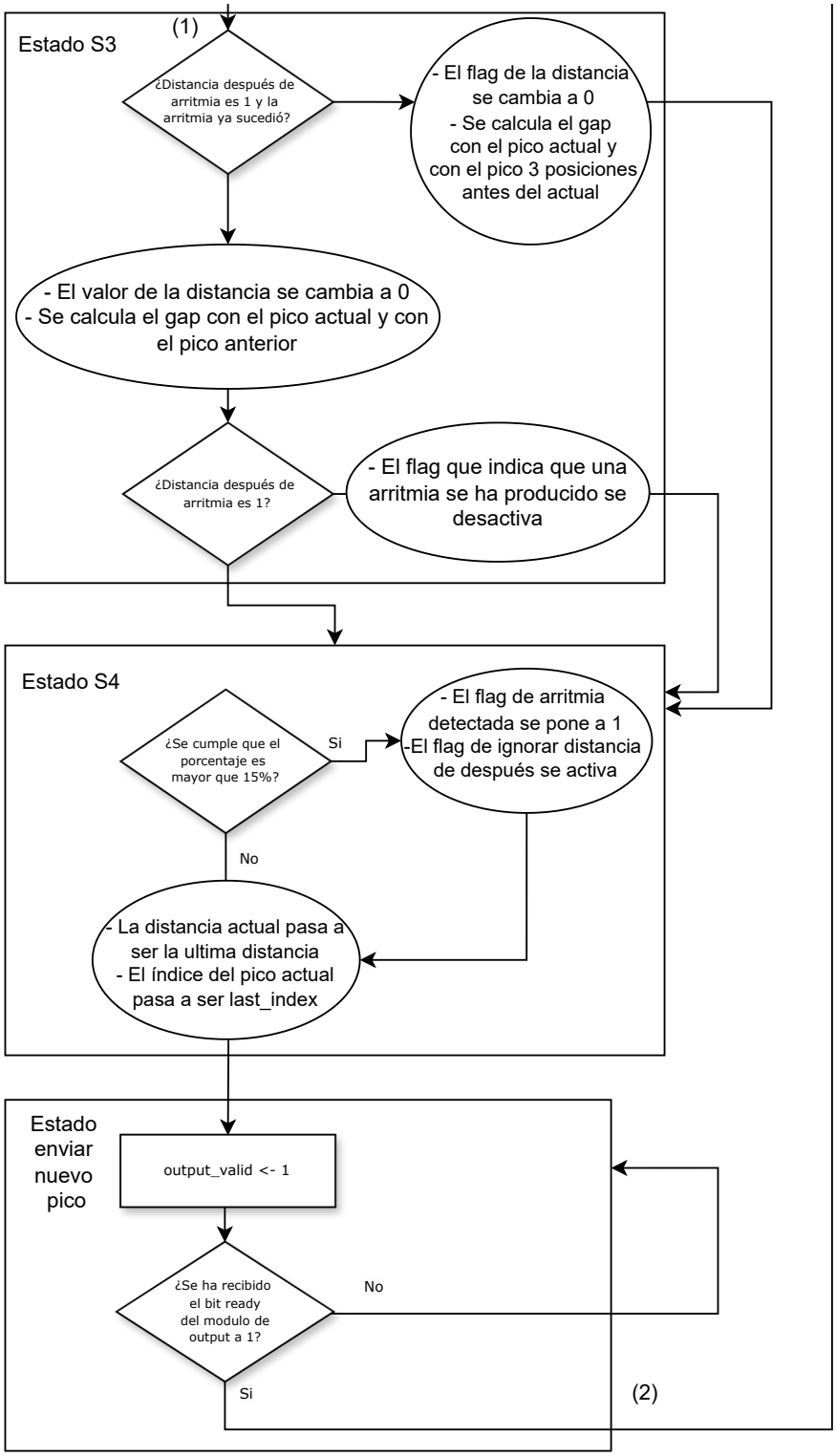


Figura 3.20: Diagrama ASM de Módulo de filtrado de señal

3.5. Módulos input y output

Estos módulos se componen de un estado de lectura y uno de escritura. Uno lee el dato y el otro se encarga de esperar a que se lea el dato y actualizar el contador para que se pueda leer de la siguiente posición de la ROM.

3.5.1. Módulo de entrada

- Estado de lectura: Primero, se asegura de que el contador no ha llegado a la cantidad máxima de muestras, que en este caso es 40,625. Luego, se actualiza el bit de *enable* a 1 y se pasa al estado de espera.
- Estado de espera: Pone el bit de *valid* a 1 y espera al bit de *ready* para que el siguiente módulo lea el dato, actualiza el contador y pasa al estado de lectura.

Este módulo cuenta con una ROM con los valores de la señal original, valores que se van leyendo cuando la señal de *ready* se activa.

3.5.2. Módulo de salida

- Estado de lectura: Primero, se asegura de que el contador no ha llegado a la cantidad máxima de muestras, que en este caso es 144. Luego, se pone el bit de `enable` a 1 y se pasa al estado de espera. Si se han leído todas las muestras, pasa al estado correcto.
- Estado de espera: Pone el bit de *valid* a 1 y espera al bit de *ready* para que el siguiente módulo lea el dato, actualiza el contador y se comprueba si la anotación del pico coincide con la anotación de la ROM, que es la anotación original. Además, se asegura que la anotación pertenece al índice correcto. Si esta condición se cumple, sigue con la ejecución; de lo contrario, pasa al estado de error.
- Estado de error: Pone la señal de `error` a 1 y detiene la ejecución, ya que un resultado no coincide.
- Estado correcto: Pone la señal de `correcto` a 1, indicando que el programa ha sido replicado con éxito.

Este módulo cuenta con una ROM que contiene los valores de la señal original, los cuales se van leyendo cuando la señal de *ready* se activa.

3.6. Módulo principal y *testbench*

El módulo principal se encarga de sincronizar los módulos pasando los datos de un módulo al siguiente así como la señal de *valid* y transferir de vuelta la señal de *ready*.

Las señales de entrada del módulo principal (main) son las señales de reloj (`clk`) y de reinicio (`reset`) y las señales de salida indican si el resultado es correcto o si hay un error.

Módulo de Filtrado:

- Toma `sample_data` y `sample_valid` como entradas.
- Produce `filter_data`, `filter_index`, `filter_valid` y `filter_ready` como salidas.

Módulo de Detección de Picos:

- Toma `filter_data`, `filter_index` y `filter_valid` como entradas.
- Produce `peak_detection_index`, `peak_detection_valid` y `peak_detection_ready` como salidas.

Módulo de Detección de Arritmias:

- Toma `peak_detection_index` y `peak_detection_valid` como entradas.
- Produce `arrythmia_detected`, `arrythmia_index`, `arrythmia_valid` y `arrythmia_ready` como salidas.

También se ha definido un *testbench* para las simulaciones, donde se definen los ciclos de reloj, además del `reset` al principio de la ejecución. Como salida tiene los estados de correcto y error para ver los resultados de la ejecución.

Capítulo 4

Resultados Experimentales

4.1. Entorno de pruebas

Para hacer las pruebas en la placa se ha utilizado la FPGA Artix-7 con placa Basys3, ya que se utiliza la misma en el estudio en el que se basa el proyecto[4].

El problema que se encontró con el uso de la placa es que la ROM no podría almacenar 650000 filas de valores de punto flotante por lo que se probo un dieciseisavo de las pruebas totales que equivale a 40625.

Por lo que, para hacer la prueba con todos los samples, se requiere utilizar una FPGA con más recursos como la Virtex-7 VC709 Evaluation Platform.

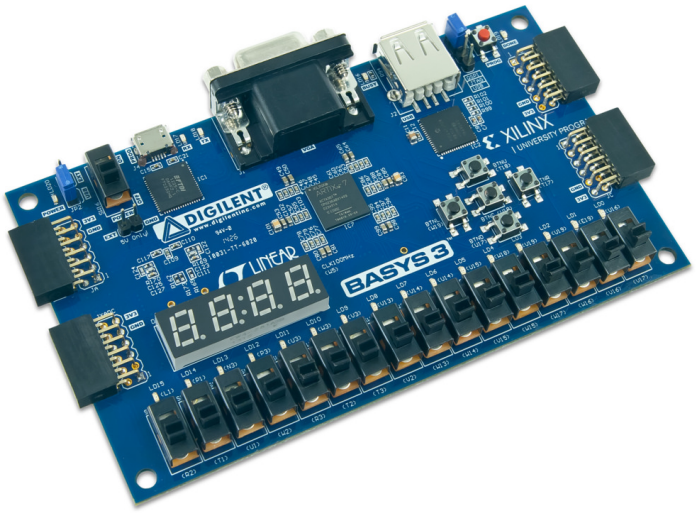


Figura 4.1: Basys3 Artix-7 FPGA

4.2. Recursos necesarios

Para evaluar los recursos necesarios, se tendrán en cuenta los resultados sacados del análisis de síntesis, del reporte de *timing* y del reporte de potencia del módulo principal que contiene el módulo de filtrado de señal, el módulo de detección de picos y el módulo de detección de arritmias.

4.2.1. Análisis de utilización

En este reporte nos encontramos con los siguientes datos:

- *Slice LUTs*: Se utilizan 2016 de un total de 20800 disponibles. La mayoría de los LUTs se asignan a la parte de detección de picos, específicamente al módulo de división seguido del módulo de multiplicación en la parte del filtrado. El módulo de detección de arritmias apenas necesita LUTs en comparación con los otros módulos.

- *Slice registers*: Se utilizan 3086 de un total de 41600 disponibles, y al igual que los LUTs, la mayoría se asignan a los módulos de división y multiplicación.
- Bloque *RAM tile*: Se utiliza 1 tile de los 50 disponibles. La mitad se utiliza para la ROM de coeficientes y la otra mitad para la RAM de los valores de la señal original.
- Se utilizan 4 DSPs de los 90 disponibles. Dos de ellos se utilizan en el módulo de multiplicación y dos en el módulo restador.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Block RAM Tile (50)	DSPs (90)	Bonded IOB (106)	BUFGCTRL (32)
N main	2016	3086	9	1	1	4	4	1
arrythmia_detection_i (arrythmia_detection)	22	35	0	0	0	0	0	0
filter_i (filter)	723	1162	9	1	1	2	0	0
mul_add_i (muladdpf)	701	1081	9	1	0	2	0	0
RAM_muestras_i (ram_muestras)	0	0	0	0	0.5	0	0	0
ROM_coeficientes_i (rom_coeficientes)	0	0	0	0	0.5	0	0	0
peak_detection_i (peak_detection)	1271	1889	0	0	0	2	0	0
divisor (divfp)	785	1381	0	0	0	0	0	0
last_peak_gt_cutoff (gtfp)	46	10	0	0	0	0	0	0
restador (subfp)	197	315	0	0	0	2	0	0
signal_data_gt_cutoff (gtfp)	46	10	0	0	0	0	0	0
signal_data_gt_last_peak (gtfp)	46	10	0	0	0	0	0	0

Figura 4.2: Reporte total de utilización

Se han utilizado menos de un 10 % del *hardware* disponible para cada recurso.

Resource	Utilization	Available	Utilization %
LUT	2016	20800	9.69
LUTRAM	124	9600	1.29
FF	3086	41600	7.42
BRAM	1	50	2.00
DSP	4	90	4.44
IO	4	106	3.77

Figura 4.3: Porcentaje del reporte total de utilización

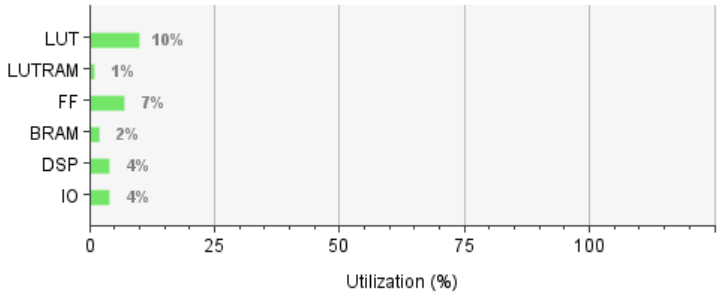


Figura 4.4: Reporte total de utilización

4.2.2. Análisis de *timing*

La frecuencia de funcionamiento se ha calculado según la referencia del artículo [4], que indica que las muestras se toman a 360 sps (*samples per second*), lo que equivale a 360 Hz. Con esta información, sabemos que una muestra llega cada $\frac{1}{360Hz} = 2{,}777ms$.

También se calcula el número de ciclos que tarda en ejecutarse el módulo de filtrado, que resulta ser el más crítico de todos. Este módulo requiere un total de 1780 ciclos, por lo que $\frac{2{,}777ms}{1780ciclos} = 1{,}56microsegundos$ por ciclo. Por lo tanto, en el *waveform* se establecerá un periodo de 1500 ns con una oscilación desde 0 a 750 ns para que sea simétrico.

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]

        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 1500.00 -waveform {0 750} [
            get_ports clk]
```

En el análisis de *timing* se comprobará cuál es el *worst negative slack* y se calculará el periodo mínimo necesario. Este reporte de *timing* muestra lo siguiente. La frecuencia alcanzada con ese periodo es de 0,540 MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1493,225 ns	Worst Hold Slack (WHS):	0,073 ns	Worst Pulse Width Slack (WPWS):	749,020 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5212	Total Number of Endpoints:	5212	Total Number of Endpoints:	3661

All user specified timing constraints are met.

Figura 4.5: Imagen que muestra el reporte de *timing* generado

Para calcular el periodo mínimo necesario se resta el periodo actual menos el *worst negative slack* dando como resultado 6,775 ns de periodo mínimo de funcionamiento.

$$f - wns = fmin$$

$$1500 - 1493,225 = 6,775$$

4.2.3. Análisis de potencia

En el análisis de potencia se evalúa la potencia que necesita la FPGA para poder llevar a cabo la ejecución.

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.069 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25,3°C**
Thermal Margin:	59,7°C (11,9 W)
Effective ϑJA:	5,0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

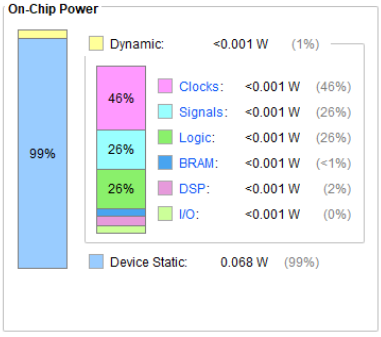


Figura 4.6: Imagen que muestra el reporte de potencia generado

En este análisis la potencia alcanzada es de 0,069 W, pero casi todo el consumo es causado por la placa en sí (Device static), este algoritmo no gasta más de un 0,001 Vatios como se puede ver en la Figura 4.8 del *hardware* utilizado a temperatura ambiente como se ve en la Figura 4.7

Environment

Output Load:	0 pF
Ambient temperature:	25.0 °C
Airflow:	250 LFM
Heat sink:	medium (Medium Profile)
ϑSA:	4.6 °C/W
Board selection:	medium (10"x10")
Number of board layers:	12to15 (12 to 15 Layers)
ϑJB:	7.5 °C/W
Board temperature:	25.0 °C

Figura 4.7: Imagen que muestra la temperatura con la que desempeña el consumo generado

Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	DSP (W)	I/O (W)
<0.001 W (1% of total)	main									
<0.001 W (1% of total)	peak_detection_i (peak_detection)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	divisor (divfp)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	restador (subfp)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (370)									
<0.001 W (<1% of total)	signal_data_gt_cutoff (gtfp)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	signal_data_gt_last_peak (gtfp)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	last_peak_gt_cutoff (gtfp)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	filter_i (filter)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	mul_add_i (muladdpf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (121)									
<0.001 W (<1% of total)	RAM_muestras_i (ram_muestras)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	ROM_coeficientes_i (rom_coeficientes)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	arrythmia_detection_i (arrythmia_detection)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (7)									

Figura 4.8: Muestra el total consumo de todos los componentes generados en *hardware*

Power Supply

Supply Source	Voltage (V)	Total (A)	Dynamic (A)	Static (A)	Budget (A)	Margin (A)
Vccint	1.000	0.011	0.000	0.011	Unspecified	NA
Vccaux	1.800	0.013	0.000	0.013	Unspecified	NA
Vcco33	3.300	0.001	0.000	0.001	Unspecified	NA
Vcco25	2.500	0.000	0.000	0.000	Unspecified	NA
Vcco18	1.800	0.000	0.000	0.000	Unspecified	NA
Vcco15	1.500	0.000	0.000	0.000	Unspecified	NA
Vcco135	1.350	0.000	0.000	0.000	Unspecified	NA
Vcco12	1.200	0.000	0.000	0.000	Unspecified	NA
Vccaux_io	1.800	0.000	0.000	0.000	Unspecified	NA
Vccbram	1.000	0.001	0.000	0.001	Unspecified	NA
MGTAVcc	1.000	0.000	0.000	0.000	Unspecified	NA
MGTAVtt	1.200	0.000	0.000	0.000	Unspecified	NA
Vccadc	1.800	0.020	0.000	0.020	Unspecified	NA

Figura 4.9: Muestra el power supply

Analizando más a fondo el consumo, se puede comprobar que el *signal rate* de la lógica no traspasa los 0,5 Mtr/s, el de la BRAM y los DSP también son bajos.

Utilization	Name	Type	Clock (MHz)	Clock Name	Signal Rate (Mtr/s)	% High
<0.001 W (<1% of total)	main					
<0.001 W (<1% of total)	CHAIN_GEN[0].C_MUX.CARRY_MUX_i_1__0 (LUT6)	LUT	N/A	Async	0.458	49.939
<0.001 W (<1% of total)	i_no_versal_es1_workaround.DSP_i_24 (LUT6)	LUT	N/A	Async	0.458	49.939
<0.001 W (<1% of total)	i___1/opt_has_pipe.first_q[0]_i_2 (LUT6)	LUT	N/A	Async	0.284	61.856
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999
<0.001 W (<1% of total)	CHAIN_GEN[12].C_MUX.CARRY_MUX_CARRY4 (CARRY4)	CARRY4	N/A	Async	0.270	49.999

Figura 4.10: Imagen que muestra la el signal rate de la lógica

Utilization	Name	Mode	Signal Rate	Clock Name A	Clock A (MHz)
<0.001 W (<1% of total)	main				
<0.001 W (<1% of total)	DEVICE_7SERIES.NO_BMM_INFO.SP.WIDE_PRIM18.ram (RAMB18E1)	RAMB18	0.034	clk_IBUF_BUFG	0.540
<0.001 W (<1% of total)	DEVICE_7SERIES.NO_BMM_INFO.SP.WIDE_PRIM18.ram (RAMB18E1)	RAMB18	0.000	clk_IBUF_BUFG	0.540

Figura 4.11: Imagen que muestra la el signal rate de BRAM

Utilization	Name	Clock A (MHz)	Clock Name	MULT Used?	MREG Used?	Pre-Adder Used?	Signal Rate
<0.001 W (<1% of total)	main						
<0.001 W (<1% of total)	i_no_versal_es1_workaround.DSP (DSP48E1)	0.540	clk_IBUF_BUFG	Yes	Yes	No	0.111
<0.001 W (<1% of total)	i_no_versal_es1_workaround.DSP (DSP48E1)	0.540	clk_IBUF_BUFG	Yes	Yes	Yes	0.087
<0.001 W (<1% of total)	use_prim.appDSP48[0].bppDSP48[1].use_dsp.use_dsp48e1.iDSP48E1 (DSP48E1)	0.540	clk_IBUF_BUFG	Yes	Yes	No	0.037
<0.001 W (<1% of total)	use_prim.appDSP48[0].bppDSP48[0].use_dsp.use_dsp48e1.iDSP48E1 (DSP48E1)	0.540	clk_IBUF_BUFG	Yes	Yes	No	0.021

Figura 4.12: Imagen que muestra la el signal rate de los DSP

Comparando este proyecto con otros estudios, por ejemplo con el de caracterización de señales usando polinomios de Hermite [4] presentan unos resultados de potencia de 28mW. Lo que hace nuestro algoritmo significativamente mejor en términos de consumo.

Capítulo 5

Conclusión

Este proyecto trata de buscar una solución simple para la detección de arritmias de una señal de un electrocardiograma, Para la elaboración de este proyecto, se ha estudiado el comportamiento de las arritmias, viendo la base de datos de MIT y estudiando el comportamiento de las arritmias anotadas se observó que la inmensa mayoría de las arritmias que ocurrían eran dadas por una contracción prematura del corazón, por tanto el proyecto, aunque inicialmente se pensó detectar el mayor tipo de arritmias posibles, al no ver ningún ejemplo claro de arritmia no producida por una contracción prematura el proyecto solo se centró en detectar dichas arritmias.

Se realizó un prototipo en *python* que sirvió para crear el algoritmo y probarlo con facilidad. Este prototipo inicia con un filtrado de la señal original aplicando el filtrado FIR. Seguidamente se aplica un algoritmo de detección de picos QRS sobre la señal filtrada que busca el pico más alto que además sobrepase el *cutoff* dinámico establecido. Finalmente se aplica el algoritmo de detección de arritmias calculando la distancia entre el pico actual con el anterior y comparándola con una distancia anterior de un ritmo normal.

Para la implementación de *hardware* se utilizaron 3 módulos principales que son el módulo de filtrado, el módulo de detección de picos y el módulo de detección de arritmias. Además estos módulos están contenidos en un módulo principal. Para hacer las pruebas sobre estos módulos, se añaden 2 módulos adicionales de input de señal y output donde se comparan los resultados de las anotaciones. Además se evalúan los resultados mediante una simulación al crear un *testbench*.

Para las pruebas en *hardware* se utiliza la FPGA *Artix 7* en la placa *Basys3*, ya que es la FPGA que se usa en el estudio y aunque no sea capaz de albergar los 30 minutos de pruebas en la RAM, con menos pruebas tiene un buen desempeño.

En el fichero .xdc se ha establecido un periodo específico teniendo en cuenta la frecuencia de las muestras que es de 360 sps y da un periodo de 1852 ns. Gracias al reporte de *timing* se halla que el mínimo periodo de funcionamiento es de 6,49 ns.

Según el reporte de potencia el consumo de la placa es de 0,069 vatios, y solamente el algoritmo consume menos de 0,001 vatios, lo que resulta en un consumo bajo incluso para un uso continuo de este. Comparándolo con otros proyectos similares, el consumo dinámico es menor.

Capítulo 6

Trabajo futuro

Como trabajo futuro, se ha propuesto sincronizar el proyecto con un dispositivo capaz de detectar latidos del corazón para mostrar pruebas en tiempo real y demostrar la eficacia del proyecto en cualquier persona. Además, se considera la posibilidad de implementar todo lo anterior en un dispositivo similar a un reloj para crear un producto que cumpla todos los objetivos mencionados en el proyecto.

También se ha pensado en realizar un trabajo de investigación para explorar alternativas en el ámbito médico y compararlas en términos de consumo y eficacia. Este trabajo de investigación proporcionaría información valiosa para mejorar y optimizar el proyecto actual, así como para identificar oportunidades para futuras investigaciones y desarrollos.

Por último, se ha contemplado la posibilidad de ampliar el proyecto para detectar más tipos de arritmias siguiendo los patrones que estas presentan, lo que permitiría desarrollar un algoritmo más completo y versátil para la detección de arritmias cardíacas.

Conclusion

This project tries to find a simple solution for the detection of arrhythmias from an electrocardiogram signal, For the development of this project, The behavior of arrhythmias has been studied, looking at the MIT database and studying the behavior of the annotated arrhythmias it was observed that the vast majority of arrhythmias that occurred were given by a premature contraction of the heart, therefore the project, although initially it was thought to detect the largest possible type of arrhythmias, not seeing any clear example of arrhythmia not produced by a premature contraction made the project only to focus on detecting these arrhythmias.

A prototype was made in python which was used to create the algorithm and test it easily. This prototype starts with a filtering of the original signal by applying FIR filtering. Next, a QRS peak detection algorithm is applied to the filtered signal, searching for the highest peak that also exceeds the established dynamic cutoff. Finally the arrhythmia detection algorithm is applied by calculating the distance between the current peak and the previous peak and comparing it with a previous distance of a normal rhythm.

For the hardware implementation, 3 main modules were used, which are the filtering module, the peak detection module and the arrhythmia detection module. In addition these modules are contained in a main module. In order to test on these modules. These modules are tested by adding 2 additional modules for signal input and output where the results of the annotations are compared. Furthermore the results are evaluated by means of a simulation when creating a testbench.

For the tests on hardware the FPGA Artix 7 on the Basys3 board is used, since it is the FPGA used in the study and although it is not able to hold the 30 minutes of tests in RAM, with fewer tests it performs well.

In the .xdc file a specific period has been established taking into account the frequency of the samples which is 360 sps and gives a period of 1852 ns. Thanks to the timing report it is found that the minimum operating period is 6.49 ns.

According to the power report the power consumption of the board is 0.069 watts, and only the algorithm consumes less than 0.001 watts, which results in a low power consumption even for a continuous use of the algorithm. Compared to other similar projects, the dynamic power consumption is lower.

Introduction

Arrhythmia concept

Cardiovascular diseases are the leading cause of death in the world [10] and one of the most common causes of these diseases is arrhythmias.

A cardiac arrhythmia is a result of a disturbance in the initiation or propagation of the heart's rhythm [12]. If an arrhythmia occurs, the heart may have a rhythm that is too fast, too slow or irregular. This can cause symptoms such as palpitations, dizziness, shortness of breath and even fainting, all of these can become life-threatening.

Cardiologists use devices such as a Holter[1] to generate electrocardiograms (ECGs), which are diagrams that represent the heartbeat, and with them they are able to analyze and find cardiac abnormalities.

Among these anomalies are premature contractions of the heart, a type of arrhythmia that this project will detect.

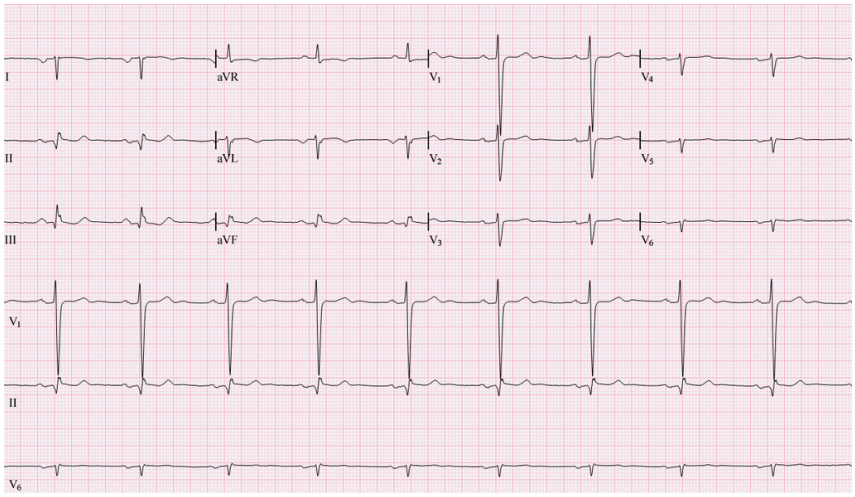


Figura 6.1: Image of several electrocardiograms generated by a Holter *Holter* [9]

.

Detection algorithm

The arrhythmia detection algorithm used in this project is based on the detection of the QRS peaks produced in the electrocardiogram. QRS peaks produced in the electrocardiogram.

A QRS spike, as shown in the Figura 6.2, on an electrocardiogram is caused by the contraction of the ventricle as it pumps blood through the arteries. This is the strongest electrical impulse that the heart produces in each beat [**wiki:complejoQRS**]. In this project we will use these peaks to compare the distance between them to determine if an arrhythmia has occurred.

This algorithm is of particular interest[5] because the development of efficient techniques to automate ECG analysis is essential to assist cardiologists in their diagnoses, especially in the detection of arrhythmias. The cardiologist would have to spend several minutes to find

[1]**Holter**:A portable medical device used to monitor and record the electrical activity of the heart over an extended period of time

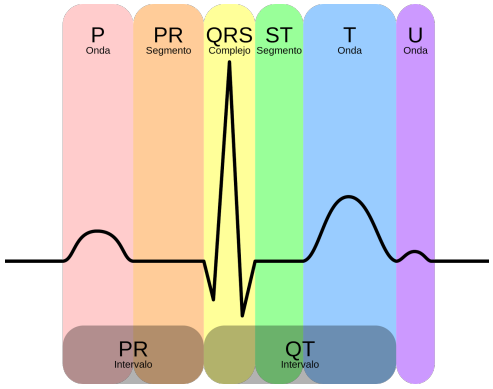


Figura 6.2: QRS Complex [4]

arrhythmias in an ECG of, for example, 30 minutes. This algorithm is not intended to be a definitive solution for the detection of arrhythmias, as several years of study are required to fully understand them. Therefore, it focuses on detecting premature ventricular contractions.

Filtrado

As it can be seen in the Figura 6.3 it is convenient to filter the rhythm strips in order to better detect the QRS peaks, since filtering centers the signal wave on the 0 value and avoids failures in the peak detection algorithm, which will be discussed later.

In the creation of the project we have tried not to filter the waveform to check if better results are obtained than without such filtering, but this has not been the case due to the irregularities of the waveform.

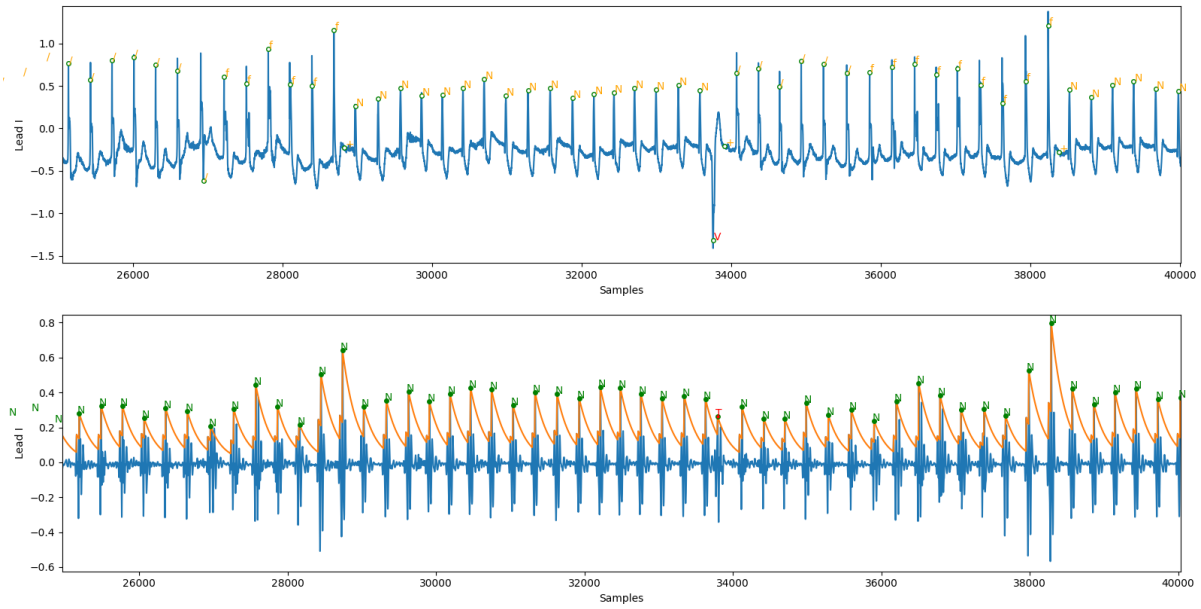


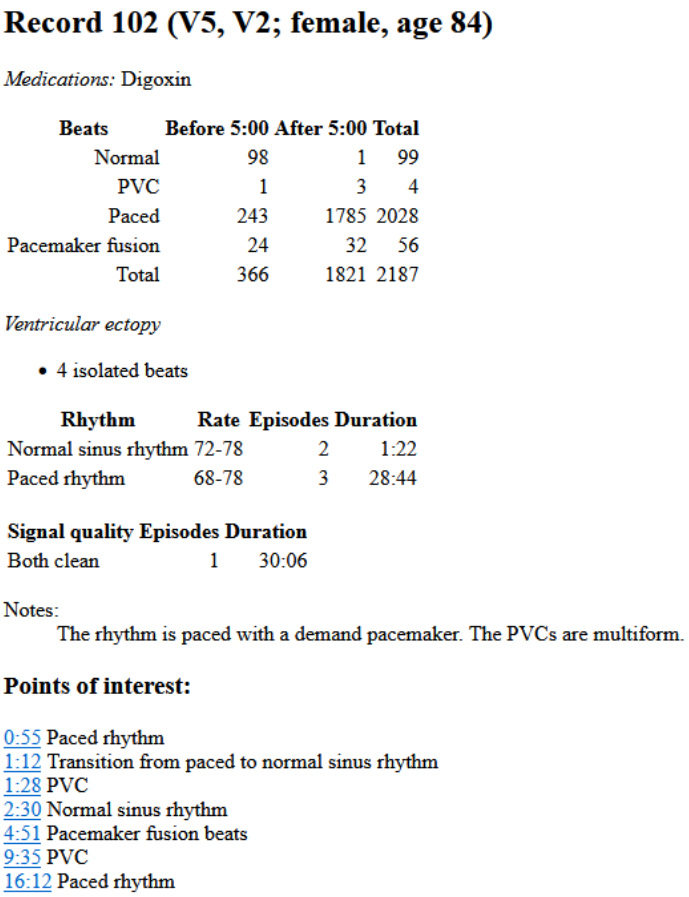
Figura 6.3: Example of an original electrocardiogram and a filtered one of patient 102

Reference data

The reference data for testing the algorithm were obtained from a study conducted at the Massachusetts Institute of Technology (MIT)[6] in which half-hour tests were performed on a

number of patients of varying ages, some of whom were implanted with pacemakers.

The data consist of electrocardiograms that have previously been analyzed by cardiologists and whose annotations indicate when a patient has suffered an arrhythmia, when the rhythm is normal and when there has been a failure in the reading of the signal, as seen in Figura 6.4. Less relevant information for our study is also shown, such as when an error has occurred. study, such as when there has been an error in the signal reading and pacemaker activation.[2].



Record 102 (V5, V2; female, age 84)

Medications: Digoxin

Beats	Before 5:00	After 5:00	Total
Normal	98	1	99
PVC	1	3	4
Paced	243	1785	2028
Pacemaker fusion	24	32	56
Total	366	1821	2187

Ventricular ectopy

- 4 isolated beats

Rhythm	Rate	Episodes	Duration
Normal sinus rhythm	72-78	2	1:22
Paced rhythm	68-78	3	28:44

Signal quality	Episodes	Duration
Both clean	1	30:06

Notes:
The rhythm is paced with a demand pacemaker. The PVCs are multiform.

Points of interest:

0:55 Paced rhythm
1:12 Transition from paced to normal sinus rhythm
1:28 PVC
2:30 Normal sinus rhythm
4:51 Pacemaker fusion beats
9:35 PVC
16:12 Paced rhythm

Figura 6.4: Example with patient 102 [8]

Use of FPGAs

This algorithm has been designed to run on a small, lightweight and portable device, which is why it is convenient to run it on an FPGA [1]. This is because there are small FPGAs that have enough hardware to be able to execute this algorithm, in addition the consumption produced by an FPGA when executing this algorithm is quite low as will be shown later.

For this project we used the Artix-7 FPGA [14] on the Basys3 board to test the operation of the algorithm. However, for testing, due to the amount of test values pulled from the database per patient, it was considered to use another FPGA whose hardware can support such amount of data.

[2]**Pacemaker**: A device implanted in the heart that acts when the heart is not pumping blood strongly enough, i.e. the QRS peak is not as prominent and the help of such a device is needed to provide the necessary electrical impulse.

Project objectives

The main objective of the project is to create an algorithm based on the study of signal characterization using Hermite polynomial [4] that detects arrhythmias in patients and that this algorithm is capable of running on a portable device, which has low power consumption and works in real time. In order to achieve the main objective, it is necessary to achieve a series of objectives which are as follows:

1. Finding out what arrhythmias are, how they occur, what arrhythmias the program could detect and what pattern most of them follow.
2. The creation of a software algorithm capable of reading and filtering the original signal of the patients in the database, the creation of an algorithm to detect the peaks where ventricular contraction occurs as they appear in the filtered signal and, finally, the creation of an algorithm to detect the arrhythmias according to the pattern found in the previous objective.
3. The replication of the program in hardware according to the prototype previously created in software. Using the necessary hardware for the algorithm to work in an FPGA and the necessary hardware to be able to test in simulation and on the board.
4. The observation of the experimental results of the amount of resources used, the resources used by the algorithm and the time it takes to be carried out according to the arrival of the patient's signal. These experimental results should be consistent with the main objective.

Work plan

1. Research and theoretical basis:
   a) Review the book[12] and consult with a first aid medic to understand what arrhythmias are and how they occur.
   b) Use the information obtained to establish the medical basis for the project.
2. Data collection and preparation:
   a) Locate and download the project database used in the signal characterization study using Hermite polynomials. [4].
   b) Represent the signals in Python and perform the filtering as described in the aforementioned study.
3. Algorithm Development:
   a) Implement the QRS peak detection algorithm and the arrhythmia detection algorithm.
   b) Perform tests and adjustments to the algorithms to obtain satisfactory results.
4. Hardware implementation:
   a) Design the main modules for each part of the algorithm: peak detection, arrhythmia detection, filtering and a main module.
   b) Create additional modules to perform simulations in Vivado and verify program performance.
   c) Develop a testbench to generate the clock frequency and signals needed to simulate the system.

5. Analysis of experimental results:
   a) Evaluate the results obtained from the analyses performed at Vivado.
   b) Draw relevant conclusions to the project objectives.

Algorithm analysis and optimization

The algorithm focuses on three main functions:

1. Filtering of the original signal: FIR filtering is used[2], which makes the signal easier to process to find the QRS peaks. This is done by multiplying the original signal values by the filter coefficients stored in a memory.
2. Peak detection on the filtered signal: Each value of the filtered signal is analyzed and if that value is higher than its previous values, it is considered a possible peak. If after 72 values it remains as the highest value, then that value is considered a QRS peak. Additionally, a dynamic cutoff is established in which, if the signal peaks do not exceed that value, they are not taken into account.
3. Arrhythmia detection by comparing the position of the peaks: Once the QRS peaks are obtained, the distance of the current peak from the previous peak is calculated and, depending on the previous distances, it is determined if there is an arrhythmia.

FPGA Implementation

To implement the code in the FPGA, several modules will be implemented, which generally try to replicate the functionalities performed by the software algorithm and will become the most important part of the program.

The main modules are.

1. Filtering module: The filter coefficients are stored in a ROM memory module and the values of the original signal in a RAM memory, then each sample is multiplied by its corresponding coefficient and the result is accumulated. Once the multiplication and accumulation process is finished, the calculated value is transmitted to the next main module.
2. Peak detection module on the filtered signal: A finite state machine is implemented which analyzes each value of the filtered signal and calculates if it is a possible peak, or a QRS peak. As the signal values are real numbers, several modules are implemented to be able to operate with the signal values in floating point.
3. Arrhythmia detection module: A finite state machine is implemented to receive a QRS peak as input, store the most recent distances, determine if an arrhythmia has occurred according to the difference of these distances, and display a flag as output.

In addition to these modules, a main module must be created to encapsulate them and a testbench module to test the operation of the program with the signals from the database. [4].

Organization of the memory

Chapter 2 will show how the algorithm was prototyped in software. Section 2.1 shows the course of the algorithm. In section 2.2, the libraries used for data collection will be shown. In section 2.3, the type of filtering used for the original signal will be explained. In section 2.4,

the QRS peak detection algorithm and the methodologies followed will be presented. In section 2.5, the arrhythmia detection algorithm will be explained and in section 2.6, the algorithm for testing the prototype and obtaining statistics of the tests performed.

In chapter 3, the different modules created for the hardware implementation will be discussed. In section 1, the modules for floating-point operations and the memories used will be specified. From section 2 to 4, the operation of the signal filtering module, the peak detection module and the arrhythmia detection module will be explained. In section 5, the creation of the main module and the used testbench will be discussed.

In chapter 4, the experimental results obtained will be shown, such as the FPGA used, hardware utilization analysis, timing analysis and consumption.

Finally, in chapter 5, a conclusion will be given on this project and future work will be suggested in case there is a desire to extend the project.

Bibliografía

[1] First Author, Second Author y Others. «Title of the Article». En: *Sensors* 14.4 (2014). Accessed: 2024-05-21, págs. 6247-6264. URL: https://www.mdpi.com/1424-8220/14/4/6247.

[2] Wikipedia contributors. *FIR (Finite Impulse Response)*. Accessed: 2024-05-27. 2024. URL: https://es.wikipedia.org/wiki/FIR_(Finite_Impulse_Response).

[3] Wikipedia contributors. *QRS complex*. Accessed: 2024-05-27. 2024. URL: https://en.wikipedia.org/wiki/QRS_complex.

[4] Madhav P Desai et al. «A low-latency, low-power FPGA implementation of ECG signal characterization using hermite polynomials». En: *Electronics* 10.19 (2021), pág. 2324.

[5] Serkan Kiranyaz et al. «Personalized long-term ECG classification: A systematic approach». En: *Expert Systems with Applications* 38.4 (2011), págs. 3220-3226.

[6] Massachusetts Institute of Technology. *MIT - Massachusetts Institute of Technology*. Accedido: 22 de mayo de 2024. 2024. URL: https://web.mit.edu/.

[7] Matplotlib Development Team. *Matplotlib — Visualization with Python*. Accedido: 22 de mayo de 2024. 2024. URL: https://matplotlib.org/.

[8] PhysioNet. *MIT-BIH Arrhythmia Database Directory*. Accessed: 2024-05-21. URL: https://archive.physionet.org/physiobank/database/html/mitdbdir/.

[9] Xavier Santiago Ramírez. *Electrocardiograma Normal*. Accedido: 22 de mayo de 2024. 2013. URL: https://fisiologiaconxavy.blogspot.com/2013/03/electrocardiograma-normal.html.

[10] Gregory A Roth, George A Mensah y Valentin Fuster. *The global burden of cardiovascular diseases and risks: a compass for global action*. 2020.

[11] SciPy Community. *scipy.signal.lfilter*. Accedido: 22 de mayo de 2024. 2024. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html.

[12] D. Vélez Rodríguez. *ECG - Handbook*. ISBN: 9788417184988 Autor: D. Vélez Rodríguez Encuadernación: Flexilibro. Sin fecha.

[13] Waveform Database. *WFDB - Waveform Database*. Accedido: 22 de mayo de 2024. 2024. URL: https://wfdb.io/.

[14] Xilinx. *Artix-7 FPGAs*. Accessed: 2024-05-21. 2024. URL: https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html.

[15] *Xilinx IP Cores*. https://www.xilinx.com/products/intellectual-property.html. Accedido en 24 de mayo de 2024.