author1hash=EAfamily=Einstein, familyi=E., given=Albert, giveni=A.,

author3hash=GMfamily=Goossens, familyi=G., given=Michel, giveni=M., hash=MFfamily=Mittelbach, familyi=M., given=Frank, giveni=F., hash=SAfamily=Samarin, familyi=S., given=Alexander, giveni=A.,

author1hash=KDfamily=Knuth, familyi=K., given=Donald, giveni=D.,

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Resumen

La caracterización automática de la señal ECG es de importancia crítica en el monitoreo y 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, funcional a tiempo real y cuyo próposito, 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 asi 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 unas anotaciones realizadas por expertos para comprobar si se han producido arritmias y así poder comprobar la eficacia de la replicación del programa a partir del prototipado en software. Además, la FPGA que se utiliza para probar el algoritmo en hardware es la Basys3.

Para concluir, en los resultados experimentales, se especifica la frecuencia ideal, el reporte de timing y el consumo en W 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.

1.1. Abstract

The automatic characterization of the ECG signal is of critical importance in patient monitoring and diagnosis. in the monitoring and diagnosis of the patient. 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, functional in real time and whose purpose is to detect the QRS peaks of a patient. QRS peaks of a patient and thus detect arrhythmias, for this the algorithm compares the distances that have the current peaks with the distance that formed the previous peaks.

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 prototype in

software. The software prototype will follow the steps of the 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 The tests will be performed by introducing electrocardiogram signals of a patient in a memory module for the program to process them, and in turn in another memory, annotations made by experts will be introduced to check if the tests have been performed. The tests will be performed by inserting electrocardiogram signals from a patient's electrocardiogram into a memory module for the program to process, while another memory module will be used to enter annotations made by experts to check whether arrhythmias have occurred and thus verify the effectiveness of the program replication from the software prototype. In addition, the FPGA used to test the algorithm in hardware is the Basys3.

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

Capítulo 2

Introducción

2.1. Concepto de arritmia

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

Una arritmia cardiaca es una alteración en el ritmo normal del corazón. Si se produce una arritmia, el corazón puede latir demasiado rápido, demasiado lento o de manera 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 para generar electrocardiogramas (ECG), que son diagramas que representan los latidos del corazón y con ellos son capaces de analizarlos y encontrar anomalias cardiacas. Entre esas anomalias están las contracciones prematuras del corazón, un tipo de arritmia que este proyecto se encargará de detectar.

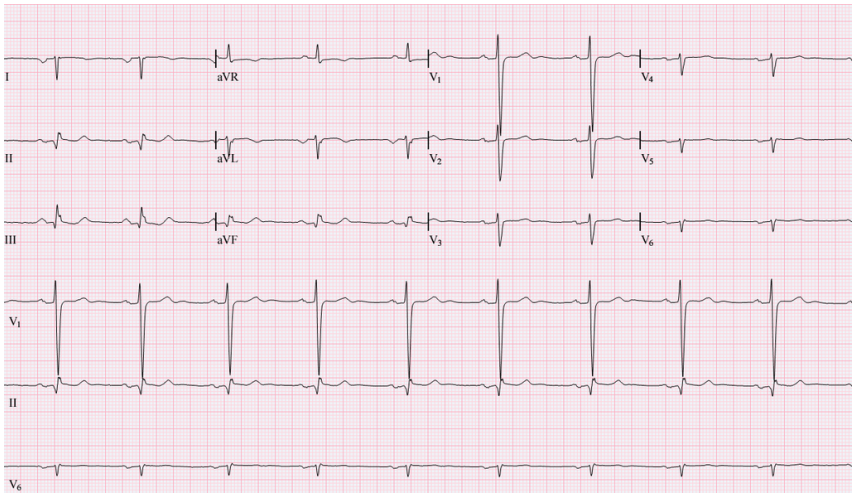

Figura 2.1: Imágen de varios electrocardiogramas recopilados por un Holter.

2.2. 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 2.2 en un electrocardiograma es causado por la contacción del ventriculo al bombear la sangre por las arterias. Este es el impulso eléctrico mas fuerte que el corazón produce en cada latido. En este proyecto utilizaremos estos picos para comparar la distancia entre ellos y poder determinar si se ha producido una arritmia.

2.2.1. Filtrado

Como se puede observar en la Figura 2.3 es conveniente hacer un filtrado de las tiras de ritmo para poder detectar mejor los picos QRS, ya que el filtrado centra la onda en el valor 0 y evita

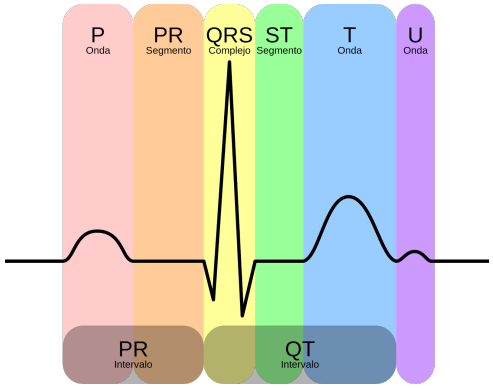

Figura 2.2: Complejo QRS

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.

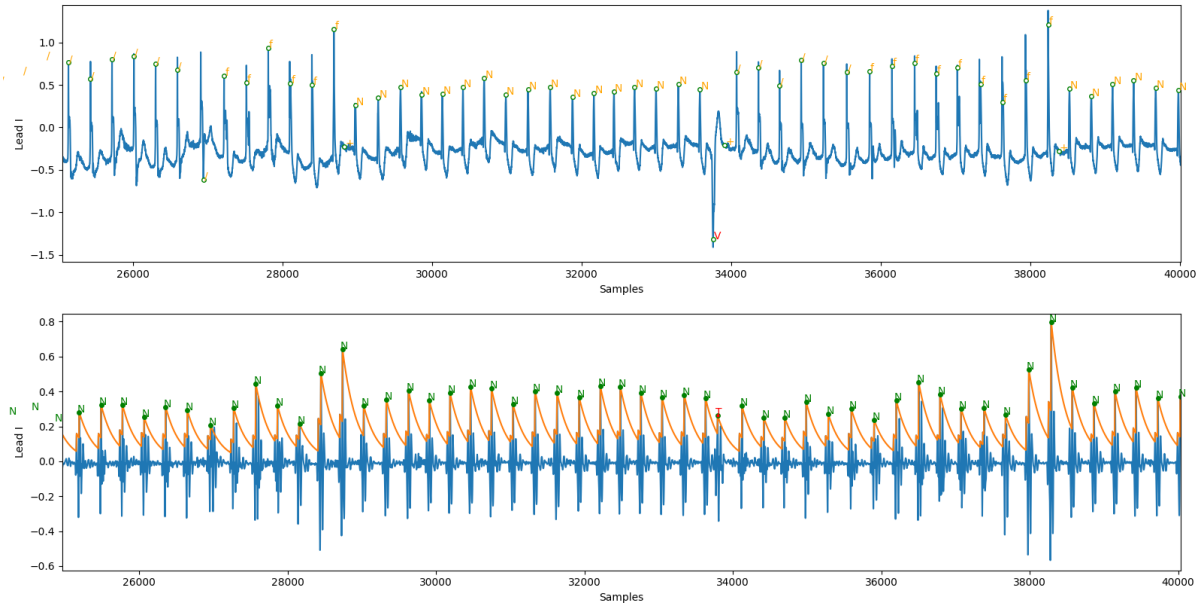

Figura 2.3: Ejemplo de electrocardiograma original y filtrado de paciente 102

2.3. 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) en el que se han realizado pruebas de media hora a varios pacientes con edades diversas y algunos de ellos llevan un marcapasos que actua 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 marcapasos para proporcionar el impulso electrico necesario.

Los datos consisten en electrocardiogramas que previamente han sido analizados por cardiologos y cuyas anotaciones indican cuando un paciente ha padecido una arritmia y cuando el ritmo es normal y cuando se ha producido en la lectura de la señal. 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.

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 2.4: Ejemplo con paciente 102

2.4. Utilización de las FPGAs

Este programa ha sido pensado para ejecutarse en un dispositivo pequeño, poco pesado y portable, es por ello, que es conveniente ejecutarlo en una FPGA, esto de debe a que una FPGA con una placa pequeña tiene el hardware suficiente para poder llegar a ejecutar este programa y por ello pueden ser llevados con facilidad, además el consumo producido por una FPGA al ejecutar el algoritmo es bastante bajo como se comprobará posteriormente.

Para este proyecto se usara la FPGA Artix-7 en la placa Basys3 para probar el funcionamiento del algoritmo. Aunque se debe considerar, según la cantidad de datos introducidos, que en este caso sería la longitud de la señal según el tiempo transcurrido, utilizar una FPGA cuyo hardware pueda soportar dicha cantidad de datos.

2.5. Objetivos del proyecto

El objetivo principal del proyecto es crear un algoritmo que detecte arritmias en los pacientes y que dicho algoritmo sea capaz de ejecutarse en un dispositivo portable que tenga un bajo consumo y que funcione a tiempo real. Para conseguir el objetivo principal es necesario alcanzar una serie de objetivos que son los siguentes:

1. Averiguar que son las arritmias, como se producen, que arritmias podría detectar el programa y que patrón siguen la mayoria 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 contraccion ventricular según aparecen en la señal filtrada, por ultimo la creación de un algoritmo encargado de detectar las arritmias segun 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 causado por el algoritmo y el tiempo que tarda en llevarse a cabo adecuándose a la llegada de la señal del paciente. Dichos resultados experimentales deberian tener una coherencia con lo que se buscaba en el objetivo principal.

2.6. Análisis y optimizacion del algoritmo

El algoritmo se centra en tres funciones principales.

1. Filtrado de la señal original: Lo que hace que la señal sea mas fácil de procesar para encontrar los picos QRS. Esto se realiza multiplicando los valores de la señal original por los valores 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.
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 calcula si hay una arritmia.

2.7. Implementacion 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 mas importantes son.

1. Módulo de filtrado: Se guardan los valores del filtrado en un módulo de memoria ROM y los valores de la señal original en una memoria RAM cuando los valores de la RAM son multiplicados, se escribe en la RAM y se pasan los valores 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 modulos para poder operar con los valores de la señal.
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 mas recientes, calcular segun la diferencia de estas distancias, si se ha producido una arritmia y mostrar como salida un flag indicandolo.

Además de estos módulos se debe de crear un módulo que los acompase y un testbench para probar el funcionamiento del programa en la simulación.

Capítulo 3

Planteamiento del algoritmo en software

3.1. Recopilacion de los datos

Para la recopilacion de los datos se utilizara la libreria wfdb 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 id (por ejemplo, 101) y hay 3 ficheros por paciente, con extensiones .dat, .atr y .hea

Se descarga la base de datos con la funcion de la libreria de wfdb, dldatabase que recoge la señal del paciente y las anotaciones de los cardiologos 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 mins 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 ultimo, para visualizar esta señal con las anotaciones de los cardiologos y poder comparar con las anotaciones que realiza el algoritmo se usara la libreria matplotlib.pyplot.

Con esto se mostrara la señal original con las anotaciones y la señal filtrada con las anotaciones del algoritmo como en Figura 2.3

```
#plot the signal
#add markers to the original signal
ax[0].plot(original_signal)
ax[1].plot(filtered_signal)
ax[0].set_xlabel('Samples')
ax[0].set_ylabel('Lead I')
ax[1].set_xlabel('Samples')
ax[1].set_ylabel('Lead I')


#Making the upper signal
for pos, sym in zip(annotation.sample, annotation.symbol):
    pos -= sampfrom
    ax[0].plot(pos, original_signal[pos], 'go', markersize=4, markerfacecolor='
        white')
    if(sym == "A" or sym == "V" or sym == "a"):
        ax[0].text(pos+10, original_signal[pos], sym, color='red')
    else:
        ax[0].text(pos+10, original_signal[pos], sym, color='orange')
```

3.2. Filtrado de la señal original

Este filtrado es llevado a cabo por el filtrado IIR.

El filtrado IIR, que significa Ïnfinite Impulse Response"(respuesta infinita al impulso), es un tipo de filtro utilizado en el procesamiento de señales digitales y analógicas.

La formula que se utilizara para el filtrado es

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

Con lo que b son los coeficientes y x la señal a filtrar

Los coeficientes se almacenan en un buffer de 99 valores en punto flotante simetricos que se iteran de forma circular, con lo que despues de ejecutar el ultimo valor vuelve de nuevo al primero.

Para el filtrado se usa la funcion lfilter de la libreria scipy.signal

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

3.3. Detección de picos QRS

El algoritmo de deteccion de picos esta representada en esta funcion que recibe la señal filtrada e intenta detectar los picos QRS.

Este algoritmo esta basado en el que se usa en el documento [**desai2021low**] donde en el 4.1.2 muestran una maquina de estados del proceso que realizan.

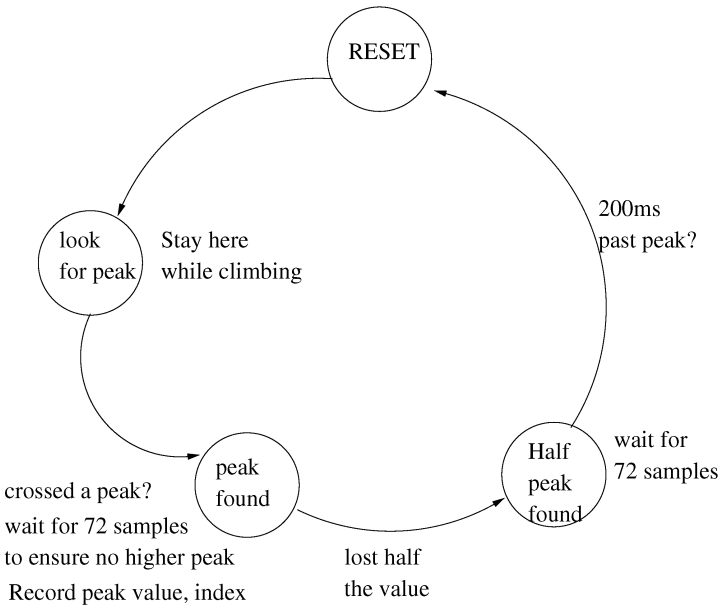

Figura 3.1: Maquina de estados de algoritmo de deteccion de picos de estudio de caracterizacion de señales usando polinomios de Hermite

Si bien nuestro algoritmo es distinto a ese, se replica el esperar a 72 muestras para asegurar de que no se encuentra un pico superior y asi 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 mas alto se necestia definir un pico en last_peak y si se encuentra otro pico se produce

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

Sin embargo hay un problema y es que cuando se detecte un pico QRS, es decir cuando se haya detectado el pico mas alto despues de haber pasado 72 samples se restauran los valores para empezar a detectar nuevos picos y al haber ruido el algoritmo podria detectar falsos picos QRS asi que por ello se implementa un cutoff.

El cutoff es representado como una funcion descendente que parte de cada pico localizado y mientras no se haya encontrado ningun pico, el valor de dicha funcion va decreciendo. La principal funcion del cutoff es evitar que el algoritmo detecte picos con el ruido y por ello se ha ajustado para que no ocurra el problema anterior y ser capaz de detectar todos los picos QRS.

La funcion del cutoff es la siguiente.

```
def calcular_cutoff(cutoff):
    cutoff = cutoff - cutoff/(256 - 64)
    return cutoff
```

Esta funcion es llamada cuando no se ha encontrado un nuevo pico y decrementa su valor, cuando se localiza un nuevo pico, el cutoff pasa a tener el valor del pico localizado.

Se han dado los valores (256 - 64) a la formula para que fuese mas facil la divison en hardware pero como al final se acabo haciendo en un modulo de division en punto flotante cualquier valor es valido para la division aunque debido al buen desempeño del valor en el programa se decidio dejar asi.

```
    def extract_peak_indices(signal, total_samples):
        samples_around_peak = total_samples // 2
        last_peak = None
        last_index = None
        peak_indices = []
        cutoff = 0
        for i in range(samples_around_peak-1, len(signal)):
            if last_peak == None:
                last_peak = signal[i]
                last_index = i
                cutoff = calcular_cutoff(cutoff)
            else:
                if signal[i] > last_peak and signal[i] > cutoff:
                    last_peak = signal[i]
                    last_index = i
                    cutoff= signal[i]
                else:
                    if (i - last_index) >= samples_around_peak and last_peak >
                        cutoff:
                        peak_indices.append(last_index)
                        cutoff = calcular_cutoff(cutoff)
                        last_peak = None
                        last_index = None
                    else:
                        cutoff = calcular_cutoff(cutoff)
            cutoff_plot.append(cutoff)
        ax[1].plot(range(samples_around_peak-1, len(signal)),cutoff_plot)
        return peak_indices
```

La salida de dicha funcion es un buffer de samples que sirven como indices para indicar donde se han encontrado los picos QRS y asi poder pasar al modulo de deteccion de arritmias.

3.4. Detección de arritmias

El algoritmo de deteccion de arritmias se encarga de ver si se ha producido una arritmia segun la distancia entre los picos.

En la deteccion 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 indice de aciertos producidos en las pruebas a cada paciente de las que se hablara 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 ultima distancia recogida y asi poder compararla con la distancia actual en calculos posteriores
- counter_buffer: utilizado para tener el valor de la posicion 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 mas 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 compararia 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 clasificaria como arritmia como se puede ver en la Figura 3.2, por ello se compara con un valor anterior que sea el ritmo normal del paciente.

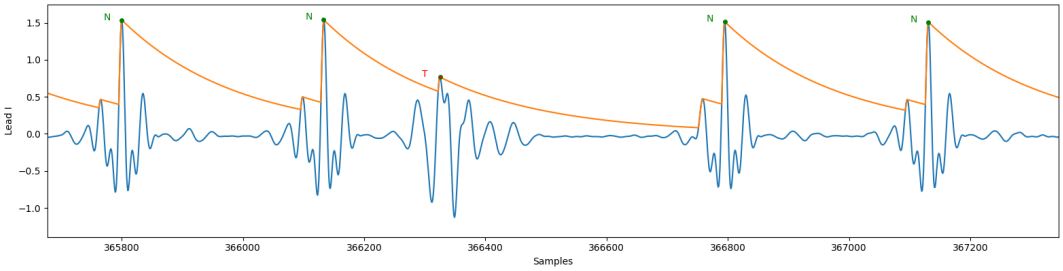

Figura 3.2: Cuando se detecta una arritmia, a veces, la siguiente distancia es considerablemente mas grande de lo normal. Para no detectar falsos positivos, se omite esa distancia

Por ello si se ha detectado una arritmia, la siguiente distancia se compara con la tercera ultima 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 funcion que compara las distancias devuelve un char que va a ser el que se vaya a plotear en la grafica, si el char es "N"significa que se ha detectado un ritmo normal y por tanto solo se plotea. Sin embargo si el resultado es "T"significa que la distancia es mas corta de lo normal, se detecta la arritmia y se ponen counter_arritmia a 1 para saber que la distancia es mas corta y TNRange a true para que el algoritmo sepa que la distancia que venga despues puede ser una ampliada.

```
    #init the first distance, the first beat doesn't count
    posant = peaks[0]
    last_distance = peaks[1] - peaks[0]

    pos_buffer = [last_distance]
    counterBuffer = 0
    #this var refers to the distance left in between an arrythmia and normal rythm
    # which tends to be longer. To avoid detection problems we will not compare
        this distance so the detection can be more precise
```

```
    TNRange = False
    counter_arrrythmia = 0
    for pos in peaks[1:]:
        act_distance = pos - posant
        pos_buffer.append(act_distance)
        counterBuffer += 1

        if(TNRange==True and counter_arrrythmia == 0):
            sym = get_frecuency_in_char(pos_buffer[counterBuffer - 3],act_distance
                )

            TNRange=False
        else:
            if(TNRange == True):
                counter_arrrythmia -= 1
            sym = get_frecuency_in_char(last_distance,act_distance)


        if(sym == "N"):
            ax[1].plot(pos, filtered_signal[pos], 'go', markersize=4,
                markerfacecolor='green')
            ax[1].text(pos-30, filtered_signal[pos], sym, color='green')
        elif(sym == "T"):
            ax[1].plot(pos, filtered_signal[pos], 'go', markersize=4,
                markerfacecolor='red')
            ax[1].text(pos-30, filtered_signal[pos], sym, color='red')
            TNRange = True
            counter_arrrythmia = 1

        posant = pos
        last_distance = act_distance
```

La funcion 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 el la distancia anterior con la actual. Despues se calcula el porcentaje de la diferencia de distancia con la distancia anterior que sabemos que va a ser un ritmo normal.

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.

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

```
def get_frecuency_in_char(last_distance,act_distance):

    gap = last_distance - act_distance

    percentaje = (gap / last_distance) * 100

    if(percentaje > 15):
        ret = "T"
    else:
        ret = "N"
    return ret
```

3.5. Pruebas con el algoritmo

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

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

Para desarrollar estas pruebas, se crea una clase Pair que contenga por cada iteracion de la deteccion de arritmias, el simbolo sacado por el algoritmo y la posicion del sample en la que se encuentre dicho pico QRS.

```
class Pair:
def __init__(self, sym, pos):
    self.sym = sym
    self.pos = pos

def __repr__(self):
    return f"Pair({self.sym},{self.pos})"
```

Dicho objeto se inserta en un buffer para luego poder comparar con las anotaciones de la señal original.

```
if(sym == "N" or sym == "T"):
    pair = Pair(sym,pos)
    produced_symbols.append(pair)
```

Una vez se rellena todo el buffer de Pares, 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 posicion aproximada.
2. Si,en el caso de que se haya detectado el pico, las anotaciones de los cardiologos 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 o otros problemas con la onda T. En la clase Annotation de la libreria wfdb, vienen explicadas todas las posibles anotaciones que puede haber.

```
    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 prestara atencion a la anotacion A y a la anotacion V que simbolizan las contacciones prematuras de la auricula y el ventriculo, las demas anotaciones sobre el pico QRS seran consideradas como ritmos normales.

Para poder ver donde se pueden producir posibles errores y el tipo de estos se ha creado un buffer donde en cada iteracion se hace push de un string con el resultado de la señal filtrada y la señal original.

Si por otro lado, el pico no se ha detectado donde tendria que haber un pico QRS puesto en la señal original, se pone ¨-¨para simbolizarlo.

Como se menciono anteriormente la deteccion de picos sobre a señal filtrada es aproximado, por lo que se cuenta si se ha detectado un pico 50 samples antes del pico de la señal original y 50 picos despues. El numero de aproximacion es moderadamente mas amplio para evitar problemas con las posibles imprecisiones del filtrado.

Otra prueba que se realiza es 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 registados.

```
def test_arrythmias(original_symbols,produced_symbols):
    sol = []
    #stats parameters
    detected = 0
    undetected = 0
    correctValue = 0
    incorrectValue = 0
    correctArrythmia = 0
    incorrectArrythmia = 0

    for i in range(len(original_symbols)):
        found = False
        aproximation = 5
        for j in range(len(produced_symbols)):
            if((produced_symbols[j].pos - 50) > original_symbols[i].pos -
                aproximation and (produced_symbols[j].pos - 50) < original_symbols
                [i].pos + aproximation):
                found = True
                sol.append(""+produced_symbols[j].sym + original_symbols[i].sym)
                detected += 1

                if produced_symbols[j].sym == 'N' and (original_symbols[i].sym ==
                    'N' or original_symbols[i].sym == '/' or original_symbols[i].
                    sym == 'f' or original_symbols[i].sym == 'L'):
                    correctValue += 1
                elif produced_symbols[j].sym == 'T' and (original_symbols[i].sym
                    == 'A' or original_symbols[i].sym == 'V' or original_symbols[i
                    ].sym == 'a'):
                    correctValue += 1
                    correctArrythmia += 1
                elif produced_symbols[j].sym == 'T' and (original_symbols[i].sym
                    == 'N' or original_symbols[i].sym == '/' or original_symbols[i
                    ].sym == 'f'or original_symbols[i].sym == 'L'):
                    incorrectValue += 1
                    incorrectArrythmia += 1
                elif produced_symbols[j].sym == 'N' and (original_symbols[i].sym
                    == 'A' or original_symbols[i].sym == 'V'):
                    incorrectValue += 1
                    incorrectArrythmia += 1
                else:
```

```
                incorrectValue += 1

        if(found==False):
            sol.append("--")
            undetected += 1
```

Con el conteo de las anotaciones se pueden sacar varias conclusiones aparte de las dichas anteriormente como los picos totales que tiene la señal original, el procentaje de picos detectados, el porcentaje de picos no detectados, el porcentaje de arritmias detectadas correcetamente, el porcentaje de falsos positivos o falsos negativos, y el porcentaje de exito de deteccion de arritmias segun todas las arritmias contando falsos positivos y negativos.

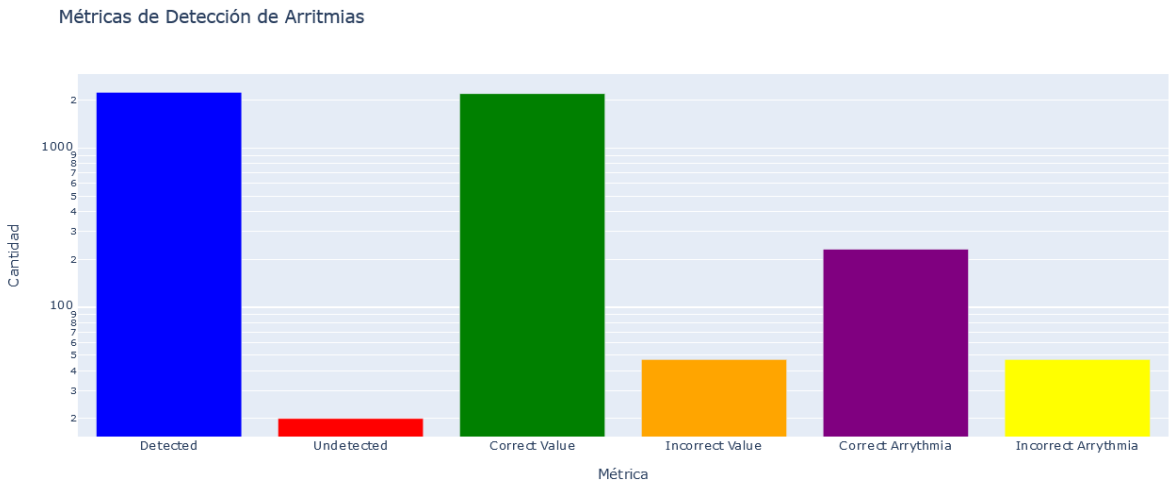

Figura 3.3: Métricas de detección de arritmias

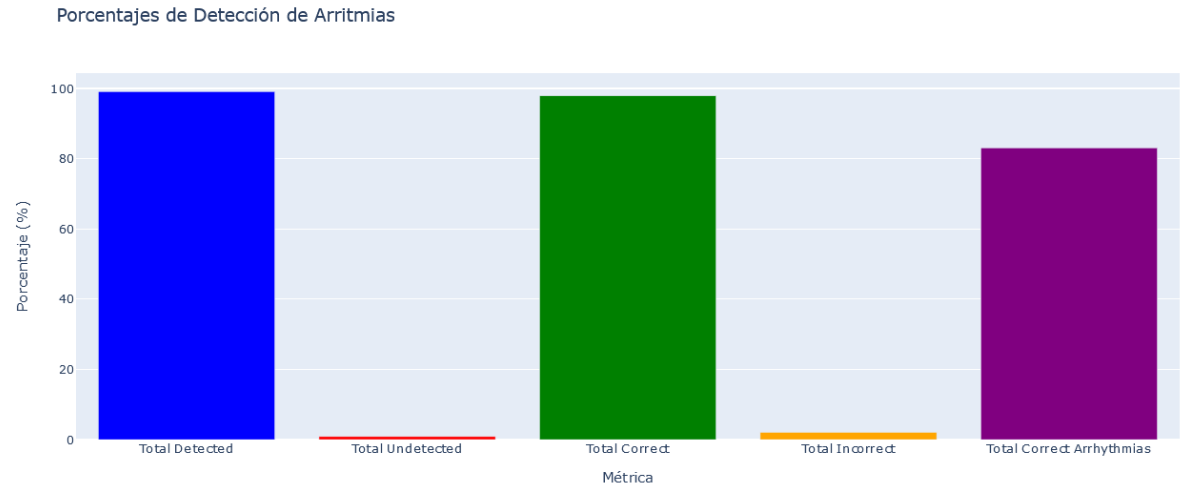

Figura 3.4: Porcentajes de detección de arritmias

Las pruebas que se han realizado se aplican solo para un paciente pero es posible aplicar estas pruebas a todos los pacientes. Para ello se ha creado un nuevo fichero de python que se encarga de realizar la misma prueba para los pacientes cuyo id esta almacenado en un buffer.

Este programa tiene 2 modos, uno procesa un paciente individualmente y el otro itera la lista definida procesandolos a todos. La logica del algoritmo esta contenida en una nueva funcion llamada calculations().

```
    mode = input("Introduce 1 to process only a patient or 2 to process all (
        monster mode):")
if mode == "1":
    patientNumber = input("introduce the patient number:")
    #select the data quantity (650000 sample intervals)
    sampfrom = 0
    sampto = 650000
    record = wfdb.rdsamp("mitdb/"+patientNumber, sampfrom=sampfrom, sampto=sampto)
    annotation = wfdb.rdann("mitdb/"+patientNumber, 'atr', sampfrom=sampfrom,
        sampto=sampto)
    calculations(sampfrom,sampto,record,annotation)
    perc = test_arrythmias(original_symbols,produced_symbols)
elif mode == "2":
    print(number_of_patients)
    print(len(number_of_patients))
    for pat in number_of_patients:
        #select the data quantity (650000 sample intervals)
        sampfrom = 0
        sampto = 650000
        record = wfdb.rdsamp("mitdb/"+str(pat), sampfrom=sampfrom, sampto=sampto)
        annotation = wfdb.rdann("mitdb/"+str(pat), 'atr', sampfrom=sampfrom,
            sampto=sampto)
        calculations(sampfrom,sampto,record,annotation)
        procesed_patients.append(pat)
        print(str(pat))
        perc = test_arrythmias(original_symbols,produced_symbols)
```

Las pruebas que se realizan para este algoritmo son iguales que en el fichero anterior pero tambien se han realizado las siguentes estadisticas.

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

```
if mode=="2":
    tdv = statistics.mean(detected_values)
    print("mean all patients detected values:"+str(tdv))
    tcv = statistics.mean(correct_values)
    print("mean all patients correct values:"+str(tcv))
    print(procesed_patients)
```

Capítulo 4

Implementación hardware

Para implementar el algoritmo en hardware dividimos en módulos el algoritmo de filtrado, el algoritmo de deteccion de picos y el algoritmo de deteccion de arritmias, estos los unificamos en un super módulo y probamos la simulacion con un testbench.

Como los valores de las señales estan en punto flotante para operar con ellos es necesario utilizar módulos hardware que permitan hacer dichas operaciones, en este proyecto utilizaremos módulos de resta, division y comparacion de numeros en punto flotante.

4.1. Módulo de filtrado

Este módulo utiliza una ROM con los coeficientes, una RAM con las muestras y un módulo de multiplicacion de numeros en punto flotante.

Este módulo se compone de una maquina de estados que va multiplicando cada elemento de la RAM muestras con los elementos de la ROM coeficientes con el módulo de multiplicacion de punto flotante.

4.1.1. 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: son 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 indices de cada valor de la señal filtrada
- output_valid y output_ready: se encargan de sincronizar el módulo del filtrado con el módulo de deteccion de picos

```
entity filter is
port (
    -- Seniales de reloj y de reset
    clk                 : in  std_logic;
    reset               : in  std_logic;

    -- bus AXI Stream de entrada
    input_signal_data   : in  std_logic_vector(31 downto 0);
    input_valid         : in  std_logic;
    input_ready         : out std_logic;

    -- bus AXI Stream de salida
    output_filter_data  : out std_logic_vector(31 downto 0);
    output_filter_index : out std_logic_vector(31 downto 0);
    output_valid        : out std_logic;
```

```
        output_ready          : in  std_logic
    );
end filter;
```

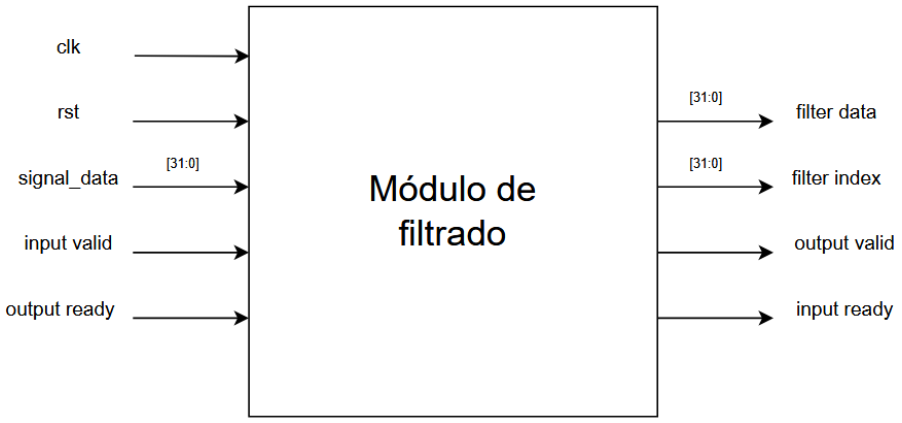

Figura 4.1: Entradas y salidas del módulo de filtrado

4.1.2. Maquina de estados

El envio de datos al output se realiza de forma asincrona

```
    output_filter_data <= acumulado;
    output_filter_index <= cont_indice;
```

- Estado de espera: En el estado de espera se activa la señal de ready y se espera a que se envie un valor de la señal sin filtrar, se borra el valor de la solucion de la multiplicacion anterior en caso de haberla, se activan las señales de escritura de la RAM y se establece el indice donde se va a escribir la muestra.
- Estado lectura: Se activa la lectura de los coeficientes y de las muestras.
- Estado para ordenar el cálculo: en este estado se activa el flag del módulo de multiplicacion.
- Estado de espera del calculo: se espera a que termine el módulo de multiplicacion esperando la señal de ready_muladd y se almacena el resultado, tambien se actualiza el contador de los coeficientes, de las muestras y dependiendo de si el indice de coeficientes es menor que 98 se va al estado de lectura o el estado de enviar un nuevo dato al siguiente módulo.
- Estado de envio de nuevo dato: este estado sincroniza el siguientemódulo, activa el bit de valid a 1 y espera el bit de ready del siguiente módulo para poder enviar el dato.

4.1.3. Módulos utilizados

Se utilizo una ROM para almacenar los coeficientes y poder leerlos

```
    -- ROM que contiene los coeficientes
    ROM_coeficientes_i : entity work.ROM_coeficientes port map (
        clka    => clk,
        ena     => ROM_coeficientes_ena,
        addra   => ROM_coeficientes_addra,
        douta   => ROM_coeficientes_douta
    );
```

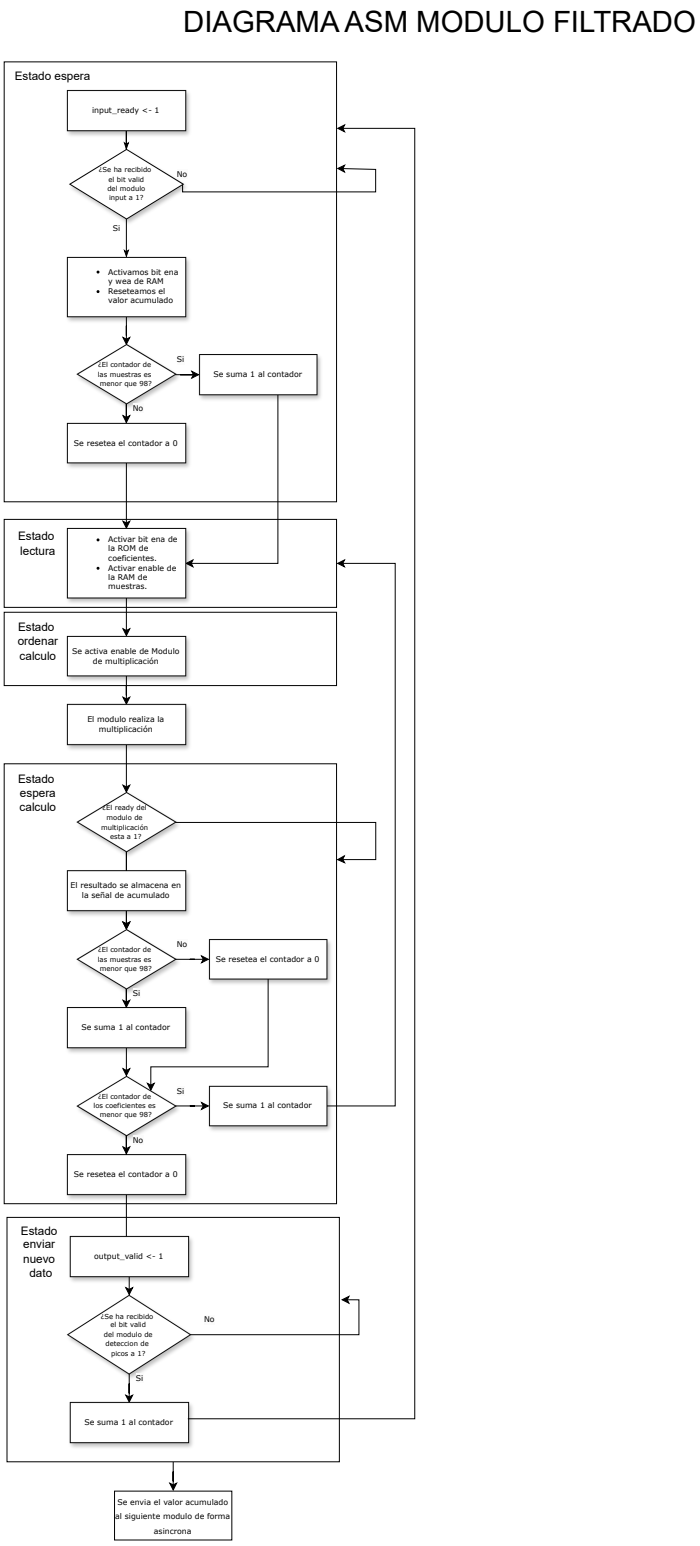

Figura 4.2: Diagrama asm de Módulo de filtrado de señal

```
ROM_coeficientes_addra <= cont_coeficientes;
```

Se usa una RAM para poder leer y escribir en las muestras de la señal original.

```
-- ROM que contiene las muestras
RAM_muestras_i : entity work.RAM_muestras port map (
    clka    => clk,
    ena     => RAM_muestras_ena,
    wea     => RAM_muestras_wea,
    addra   => RAM_muestras_addra,
    dina    => RAM_muestras_dina,
    douta   => RAM_muestras_douta
);

RAM_muestras_addra <= cont_muestras;
RAM_muestras_dina <= input_signal_data;
```

Se usa un módulo de multiplicacion para poder multiplicar los valores de las muestras con los valores de los coeficientes.

```
-- Multiplicacion y suma
mul_add_i : entity work.muladdpf port map (
    aclk                    => clk,
    s_axis_a_tvalid         => enable_muladd,
    s_axis_a_tdata          => RAM_muestras_douta,
    s_axis_b_tvalid         => enable_muladd,
    s_axis_b_tdata          => ROM_coeficientes_douta,
    s_axis_c_tvalid         => enable_muladd,
    s_axis_c_tdata          => acumulado,
    m_axis_result_tvalid    => ready_muladd,
    m_axis_result_tdata     => result_muladd
);
```

4.2. Módulo de deteccion de picos

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

4.2.1. Señales de entrada y salida

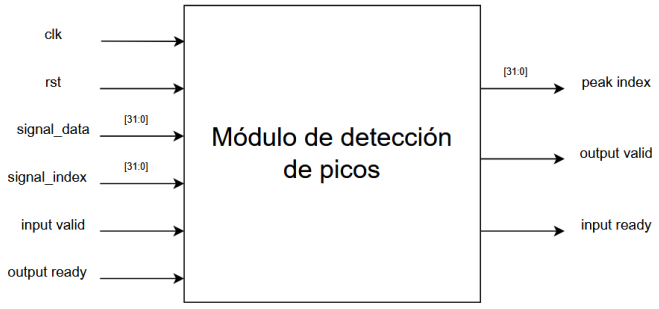

Figura 4.3: Entradas y salidas del módulo de deteccion de picos

- clk y reset
- input_signal_data: señal que recibe las muestras de la señal filtrada
- input_signal_index: señal que recibe lso indices 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 indices de los picos detectados
- output_valid y output_ready: Se encargan de sincronizar el módulo de deteccion de picos con el módulo de deteccion de arritmias.

```
port (
    -- Seniales de reloj y de reset
    clk                 : in  std_logic;
    reset               : in  std_logic;

    -- bus AXI Stream de entrada
    input_signal_data   : in  std_logic_vector(31 downto 0);
    input_signal_index  : in  std_logic_vector(31 downto 0); -- vamos a dejarlo en
        32 (aunque no sean necesarios tantos bits) para sean todos los buses
        iguales
    input_valid         : in  std_logic;
    input_ready         : out std_logic;

    -- bus AXI Stream de salida
    output_peak_index   : out std_logic_vector(31 downto 0); -- vamos a dejarlo en
        32 (aunque no sean necesarios tantos bits) para sean todos los buses
        iguales
    output_valid        : out std_logic;
    output_ready        : in  std_logic
);
```

4.2.2. Maquina de estados

- Estado de espera: En el estado de espera se activa la señal de ready y se espera a que se envie un valor de la señal sin filtrar, se borra el valor de la solucion de la multiplicacion anterior en caso de haberla, se activan las señales de escritura de la RAM y se establece el indice donde se va a escribir la muestra.
- Estado de comprobar indice: si no hay pico, o lo que es lo mismo que la señal de last_peak este a 0 este se asigna a la señal y ademas se registra el indice, despues pasa al estado de actualizar el cutoff activando por tanto la señal de division para que empiecen los módulos de division y resta de valores en punto flotante a calcular el valor. Si por otro lado si que hay pico, se ordena hacer la comparacion signal_data ¿last_peak pasando las señales correspondientes al módulo de comparacion en punto flotante. Ademas se anticipa y se hace la comparacion last_peak ¿cutoff para en dado caso de que no se cumpla la condicion anterior ya esta la comparacion hecha y se pude pasar directamente al estado siguiente. Tambien activamos las señales del módulo de comparacion correspondiente. El siguiente estado es el estado de espera a la condicion en la que la señal es mayor que el pico maximo.
- Estado de actualizar cutoff: este estado espera a la señal ready del módulo de la resta ya que es la ultima operacion que se realiza para calcular el cutoff. Primero se ejecuta el módulo de la division para calcular cutoff/192 y luego la resta cutoff - cutoff/192. cuando la señal ready_sub sea '1' se actualiza el cutoff y pasa al estado de espera terminando la iteracion.
- Estado de espera a la condicion en la que la señal es mayor que el pico maximo: cuando las señales ready de los comparadores esten a '1' se podra ejecutar las funcionalidades del estado. este tiene 3 condiciones:
    - si se ha encontrado un valor mas alto que last_index, este pico pasa a ser el nuevo last_peak y el nuevo cutoff, el index tambien se actualiza.

- la señal result_signal_index_sub_last_index_gt_or_eq_samples_around_peak se calcula de forma asincrona, por lo que si esa condicion que indica que han pasado 72 muestras sin encontrar un valor mas alto que last_peak y ademas last_peak es mayor que el cutoff, se pasa directamente al estado de envio de nuevo pico para enviar el pico QRS.
- Sino simplemente se ordena la actualizacion del cutoff activando el enable del módulo de la division y pasando al estado correspondiente.

- Estado de envio de nuevo pico: se activa la señal de valid a 1 y espera a que el módulo de deteccion de arritmias mande la señal de ready para resetear las señales de last peak y last index a '0' ademas se actualiza el cutoff.

De manera asincrona se pasa como output el last index pero el módulo de deteccion de arritmias se activa cuando input_valid se activa usando asi el last index correspondiente

```
output_peak_index <= last_index;
```

4.3. Módulo de deteccion de arritmias

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

4.3.1. 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 deteccion 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 sincronizacion con el módulo output.

```
Port (
    clk: in std_logic;
    rst: in std_logic;

    input_peak_index: in std_logic_vector(31 downto 0);
    input_valid: in std_logic;
    input_ready: out std_logic;

    output_arrythmia_detected: out std_logic; -- ya que saca 0 si es ritmo
        normal y 1 si es arritmia
    output_arrythmia_index: out std_logic_vector(31 downto 0);
    output_valid: out std_logic;
    output_ready: in std_logic
   );
```

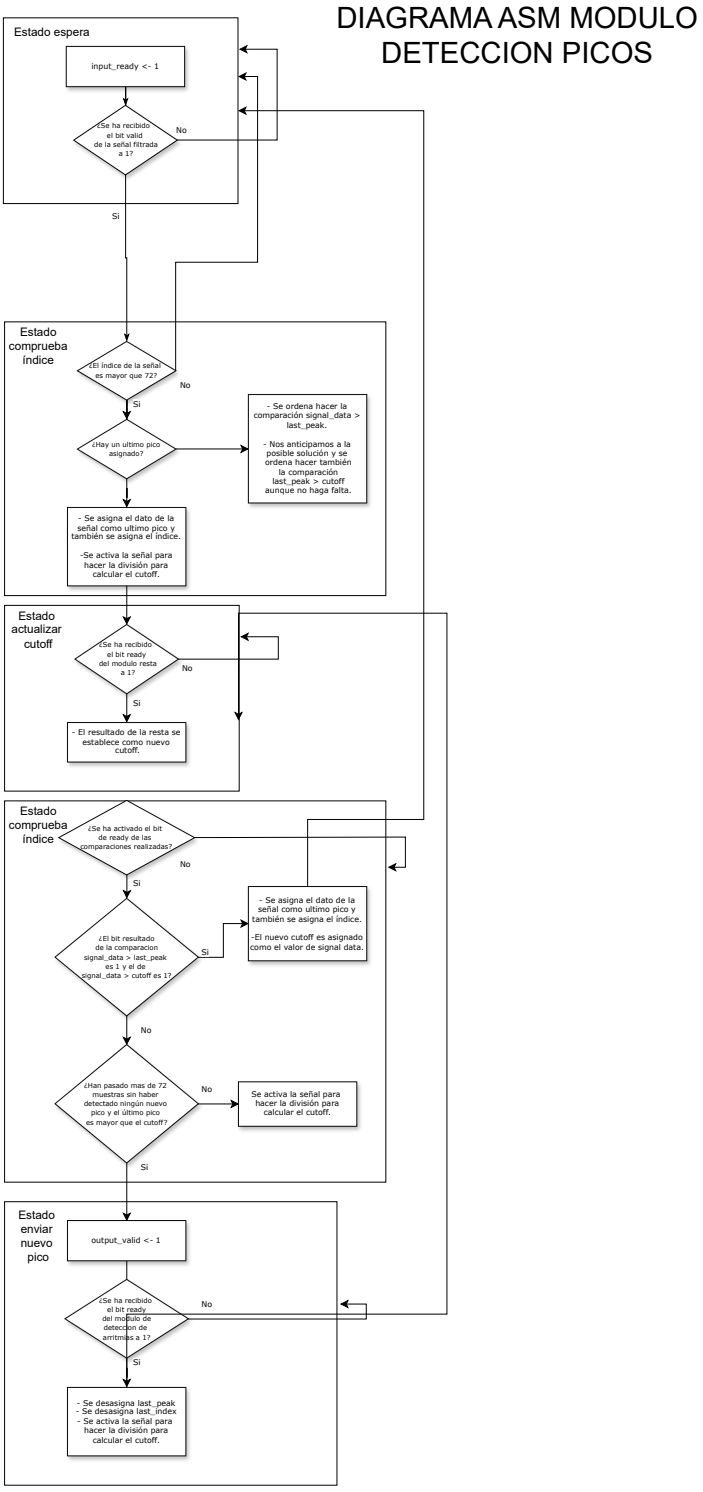

Figura 4.4: Diagrama asm de Módulo de deteccion de picos

4.3.2. Maquina de estados

- Estado de espera: En el estado de espera se activa la señal de ready y se espera a que se envie un pico QRS, despues pasa al estado S0

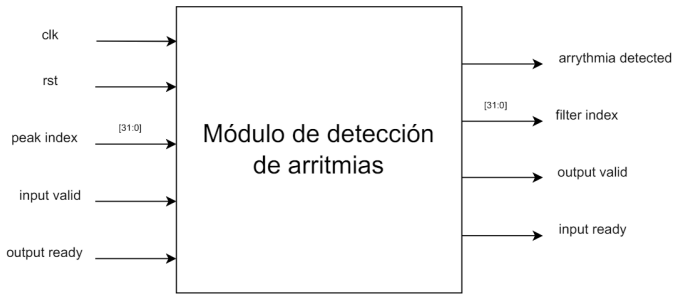

Figura 4.5: Entradas y salidas del módulo de deteccion de arritmias

- Estado S0: Si el contador es 0 significa que se recibe el primer pico registrado por lo que se guarda para mas 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 despues pasa al estado S1. Si no se cumple ninguna condición se pasa al estado S1.
- Estado S1: aumenta el contador en 1, y se calcula la distancia actual.
- Estado S2: se actualizan las variables que crean un buffer ficticio y los valores se mueven una posicion cuando se añade la distancia actual como si fuese una cola.
- Estado S3: Como ya se explico en la parte de la implementacion del algoritmo TNRange es una señal que simbiliza la distancia entre el pico detectado como arritmia y el pico normal actual, este flag se activa cuando ha habido una arritmia ya que last_distance puede ser mas grande de lo normal, es por eso que para calcular el gap cuando este flag esta activo y el de arritmia detectada no, se compara la distancia actual con la ultima distancia sino con una 3 veces anterior a la última. Si esta condicion no se cumple, para calcular el gap se compara con la ultima distancia, ademas si justo la anterior distancia era la de una arritmia, se desactiva el flag de arritmia detectada.
- Estado S4: Se calcula el procentaje de forma asincrona, si el flag de porcentaje es igual a 1 significa que el porcentaje calculado es mayor de lo esperado y por tanto se activan las flags de TNRange y counter_arrythmias. Independientemente despues el indice y la distancia actual pasan a ser last_distance y last_index
- Estado S5: Se activa la señal de valid y se espera a la señal de ready para que se envie el dato al módulo de output.

Se calcula el porcentaje del gap entre las 2 distancias con una distancia normal de forma asincrona.

- Estas señales estan en complemento A2 por ello al salir numeros negativos, el bit mas significativo se cambia a 1, es por ello que como solo se consideran los numeros positivos, se considera solamente los numeros cuyo bit mas significativo sea 0
- como al principio la señal de la ultima distancia es x"00000000.ªl comparar esta distancia con la actual saldra una distancia enorme que activara el bit del porcentaje que se calcula de forma asincrona asiq ue quitramos ese caso especifico de por medio.
- en vez de hacer una division del gap entre distance_for_calc se sigue esta fórmula:

$$gap/distanceforcalc > 0{,}15$$

$$gap > 0{,}15 \cdot distanceforcalc$$

$$gap > distanceforcalc \gg 5 + distanceforcalc \gg 3 + distanceforcalc$$

```
percentage <= '1' when gap(31) = '0' and last_distance > x"00000000" and (
    std_logic_vector(shift_right(unsigned(distance_for_calc), 3) + shift_right
    (unsigned(distance_for_calc), 5)) <= std_logic_vector(unsigned(gap))) else
     '0';
```

4.4. Módulos input y output

Estos módulos se componen de un estado de lectura y uno de escritura donde 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 posicion de la ROM

4.4.1. Módulo input

- Estado de lectura: Primero se asegura de que el contador no hay llegado a la cantidad de muestras máximas en este caso 40625, después pone el bit de enable a 1 y 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 cundo la señal de ready se activa.

4.4.2. Módulo output

- Estado de lectura: Primero se asegura de que el contador no hay llegado a la cantidad de muestras máximas en este caso 144, después pone el bit de enable a 1 y pasa al estado de espera. Si se han leido 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 anotacion del pico coincide con la anotacion de la BRAM que es la anotacion original. Ademas se asegura que la anotacion pertenece al indice correcto, si esta condicion se cumple sigue con la ejecucion, si no pasa al estado de error.
- Estado error: pone la señal de error a 1 y se para la ejecucion ya que un resultado no coincide.
- Estado correcto: pone la señal de correcto a 0 que indica que el programa ha sido replicado con exito.

Este módulo cuenta con una ROM con las anotaciones de los cardiologos de casa pico QRS, valores que se van leyendo cundo la señal de valid se activa.

4.5. Módulo principal y testbench

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

Tambien se ha definido un testbench donde se definen los ciclos de reloj, ademas del reset al principio de la ejecucion. Como salida tiene los estados correcto y error para ver los resultados de la ejecucion.

DIAGRAMA ASM MODULO DETECCION ARRITMIAS

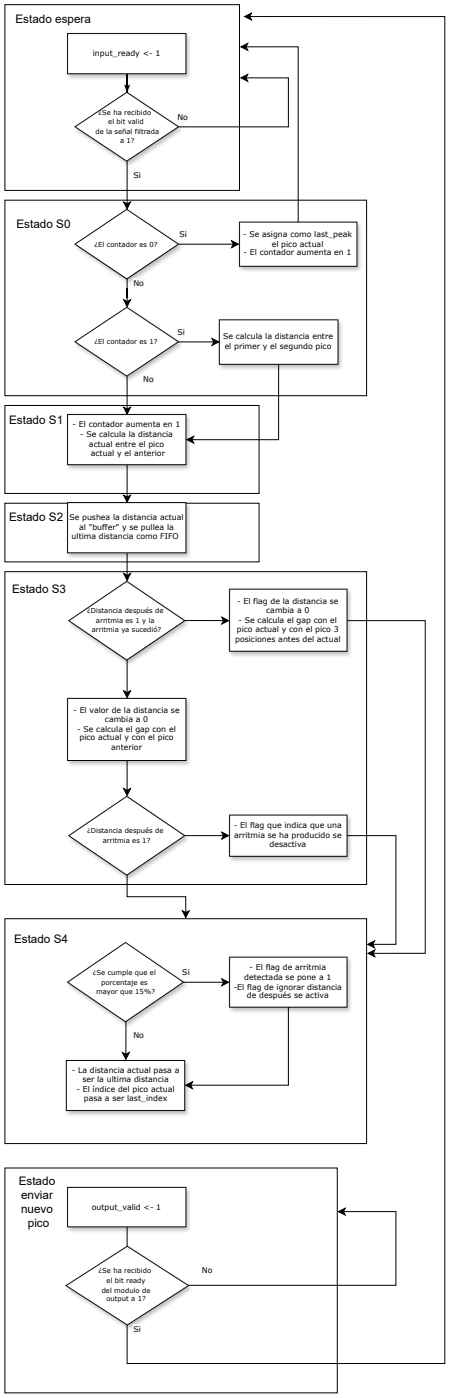

Figura 4.6: Diagrama asm de Módulo de filtrado de señal

4.6. Otros módulos

explicar módulos de punto flotante y ROM/RAM pero mover arriba para que sean los primeros módulos que se expliquen

4.6.1. Módulos ROM y RAM

ROM muestras

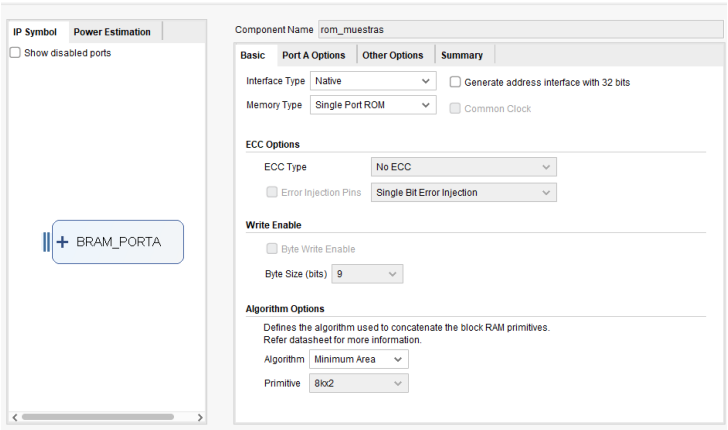

Figura 4.7: Seleccion de la opcion simple block ROM y

Es importante desactivar l a opcion de primitive output para que no se añada un registro extra al principio y la simulacion se ejecute en cada tiempo correspondiente.

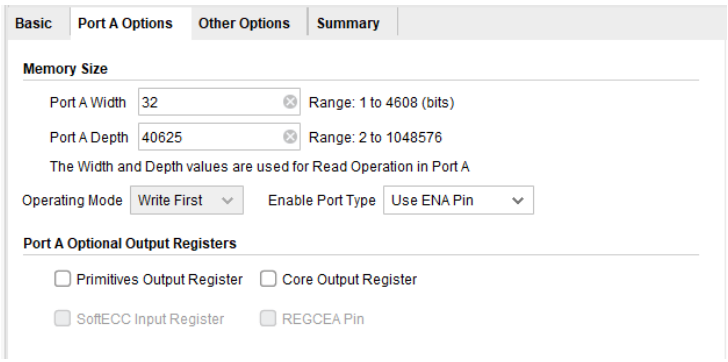

Figura 4.8: Se establecen las filas de la BROM y la longitud de estas

El módulo de filtrado utiliza 1 ROM y una RAM

- La ROM se configura igual que la ROM del módulo de input, este tiene 99 filas y de anchura tiene 32 bits.
- La RAM se configura como single port RAM y se mantiene desactivado el valor de primitive output. Ahora bien los valores asignados son los siguientes.

4.6.2. 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 para estas operaciones.

Como se operan con valores en punto flotante simple, las señales tienen que ser de 32 bits y por simplificar el codigo, casi todos los valores tanto en punto flotante como enteros se declaran con señales de 32 bits.

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

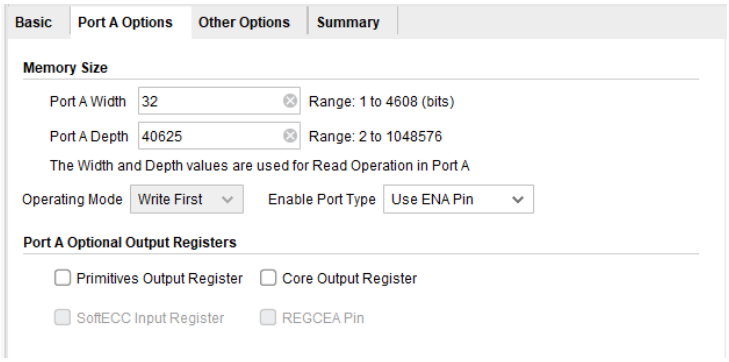

Figura 4.9: Se establecen las filas de lectura y escritura de la RAM y las longitud de dichas filas

- Módulo comparador mayor que: se utiliza para comparar varias señales en el módulo de deteccion de picos como son:
  - signal_data gt last_peak
  - signal_data gt cutoff
  - last_peak gt cutoff

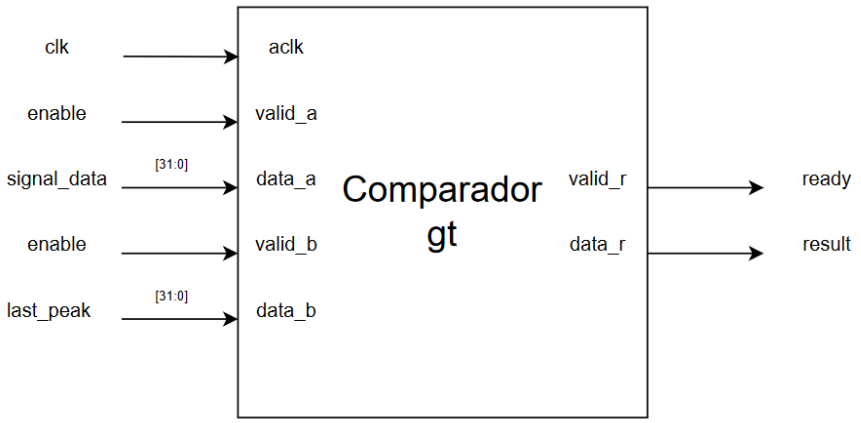

Figura 4.10: Entrada y salida del módulo de comparador

- Módulo divisor y resta: se utilizan en conjunto para calcular el cutoff que tiene la operacion:

$$cutoff = cutoff - cutoff/192$$

- Modulo multiplicacion: Se usa para poder multiplicar los valores de las muestras con los valores de los coeficientes en el modulo de filtrado.

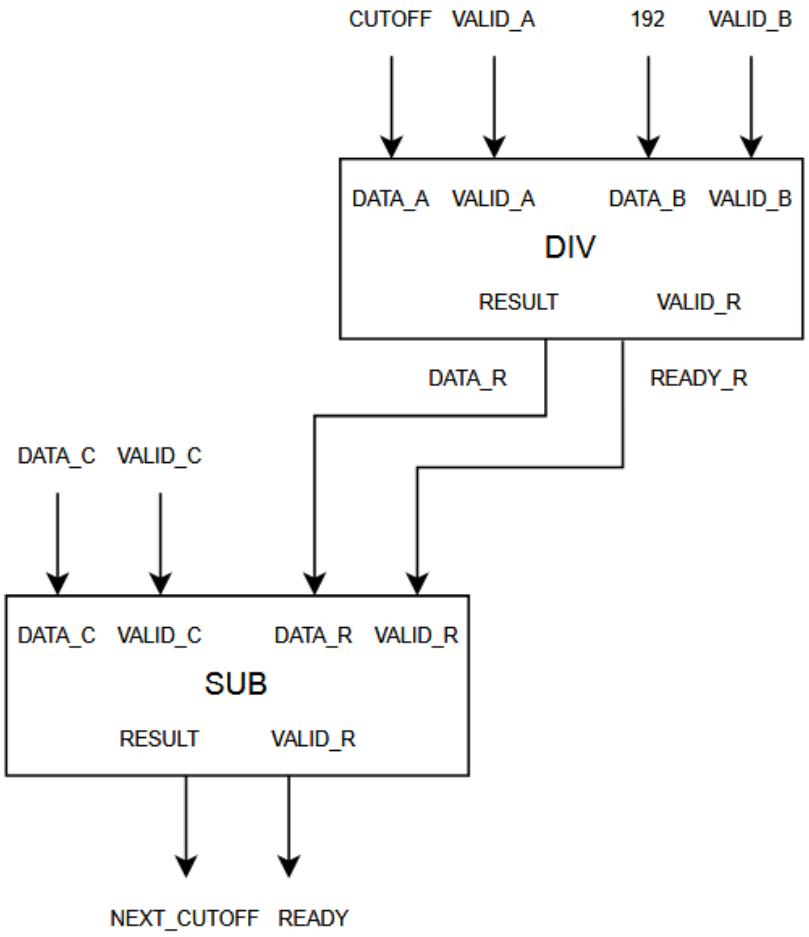

Figura 4.11: Funcionamiento de la conexion de los módulos de divisor y restador

Capítulo 5

Resultados Experimentales

5.1. Entorno de pruebas

Para hacer las pruebas en la placa se ha utilizado la Basys3 de virtex ya que se utiliza la misma en el estudio en el que se basa el proyecto. El problema que se encontro con el uso de la placa es que la ROM no podria almacenar 650000 filas de valores de punto flotante por lo que se probo un dieciseisavo de las pruebas totales que equivale a 40625. Porlo que, para hacer la prueba con todos los samples, se requiere utilizar una FPGA con mas recursos como la Virtex-7 VC709 Evaluation Platform.

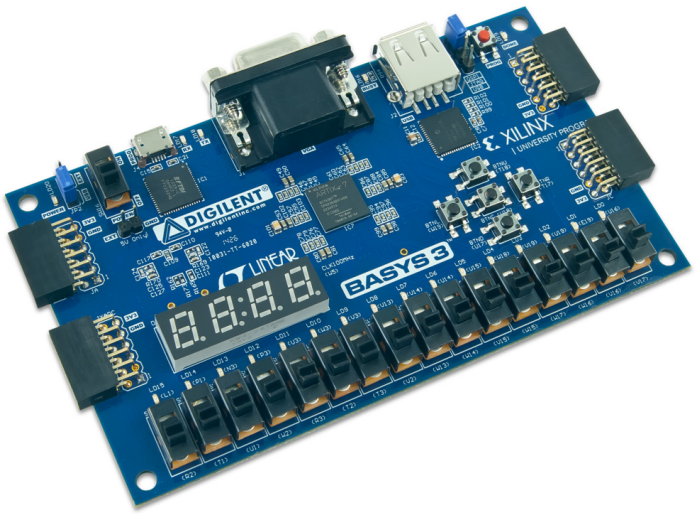

Figura 5.1: Basys3 Artix-7 FPGA

5.2. Consumo

Para evaluar el consumo se tendra en cuenta los resultados sacados del análisis de síntesis, del reporte de timing y de el reporte de power de el modulo principal que contiene el modulo de filtrado de señal, el módulo de deteccion de picos y el modulo de deteccion de arritmias.

5.2.1. Análisis de síntesis

En el análisis de síntesis podemos ver las sigueintes caracteristicas:

- Luts as logic: 195
- Luts as memory: 0
- Slice registers: Hay 279 slice registers de los cuales todos son flip flops y no hay ningun latch por la arquitectura seguida en la creacion del programa haciendo que al pasar de estado se cambien todas las señales nuevas por las actuales

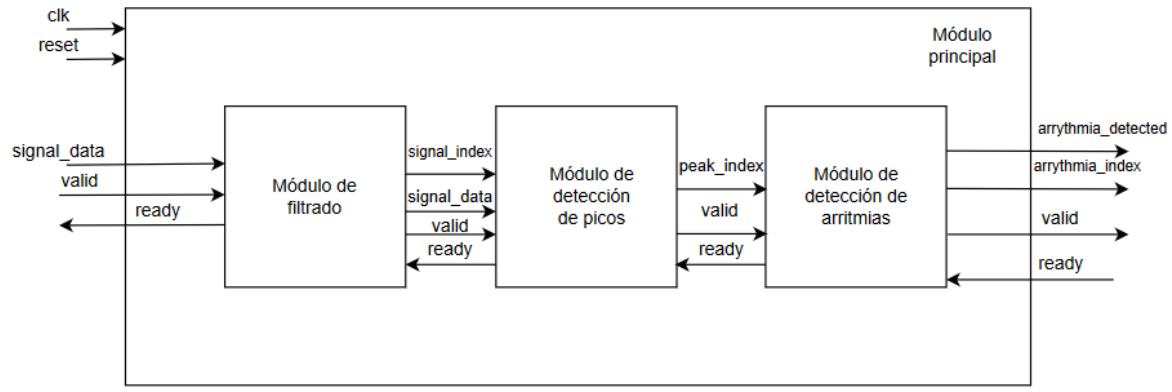

Figura 5.2: Diagrama principal de todos los modulos a evaluar

- No se ha usado ningún DSP
- No se ha usado ninguna block RAM tile
- Hay un total de 474 total slices
- La frecuencia de funcionamiento configurada en el .xdc es de 640800 pero para las pruebas se usara una frecuencia de 540000

La frecuencia de funcionamiento se ha calculado segun la referencia de el articulo [**desai2021low**] que nos indica que las muestras van a 360sps (samples per second) por lo que es equivalente a 360Hz. Tambien se calcula el numero de ciclos que tarda en ejecutarse el modulo de filtrado que resulta ser el mas critico de todos, este da un total de 1780 ciclos pero para hacer las pruebas se usaran 1500 ciclos. Si se multiplican ambos valores da una frecuencia de 540000 cuyo periodo es de 1851,85 que redondeando es de 1852. En el xdc se pone lo siguiente:

```
## 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 1852.00 -waveform {0 926} [
            get_ports clk]
```

El waveform oscila desde 0 a 926 para que sea simétrico.

5.2.2. Análisis de timing

En el análisis de timing se vera cual es el worst negative slack y se calculara la frecuencia minima necesaria. Este reporte de timing muestra lo siguiente.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1845,353 ns	Worst Hold Slack (WHS):	0,073 ns	Worst Pulse Width Slack (WPWS):	925,020 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:	3901	Total Number of Endpoints:	3901	Total Number of Endpoints:	3219

All user specified timing constraints are met.

Figura 5.3: Imagen que muestra el reporte de timing generado

Para calcular la frecuencia minima necesaria se resta la frecuencia actual menos el worst negative slack dando como resultado 6,49 ns de frecuencia mínima de funcionamiento.

5.2.3. Análisis de power

En el análisis de power se evalua la potencia que necesita la FPGA para poder llevar a cabo las instrucciones.

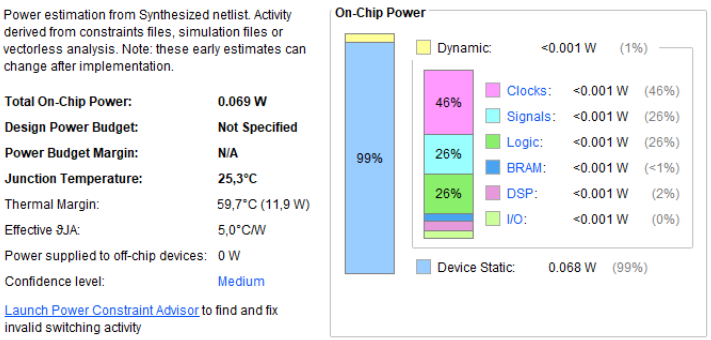

Figura 5.4: Imagen que muestra el reporte de power generado

Comparando este proyecto con otros estudios, por ejemplo con el de caracterizacion de señales usando polinomios de hermite presentan unos resultados de dynamic power de 28mW. Sin embargo, el dynamic power de este proyecto es menor que 0,001W.

Capítulo 6

Conclusión

Este proyecto trata de buscar una solucion simple para la deteccion de arritmias de una señal de un electrocardiograma, Para la elaboracion 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 observo que la inmensa mayoria de las arritmias que ocurrian eran dadas por una contraccion prematura del corazon, por tanto el proyecto, aunque inicialmente se penso detectar el mayor tipo de arritmias posibles, al no ver ningun ejemplo claro de arritmia no producida por una contraccion prematura el proyecto solo se centro en detectar dichas arritmias.

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

Para la implementacion de hardware se usaran 3 modulos principales que son el modulo de filtado, el modulo de deteccion de picos y el modulo de deteccion de arritmias. Ademas estos modulos estan contenidos en un modulo principal. Para hacer las pruebas sobre estos modulos, se añaden 2 modulos adicionales de input de señal y output donde se comparan los resultados de las anotaciones. Ademas se evaluan los resultados mediante una simulacion al crear un testbench.

Para las pruebas en hardware se utiliza la FPGA 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 especifico teniendo en cuenta la frecuencia de las muestras que es de 365 sps y da un periodo de 1852 ns. Gracias al reporte de timing se halla que la minima frecuencia de funcionamiento es de 6,49 ns.

Segun el reporte de power el consumo de la placa es de 0,069 W lo que resulta en un consumo bajo incluso para un uso continuo de este. Comparandolo con otros proyectos similares, el consumo dinámico es menor.

6.1. Conclusion

This project tries to find a simple solution for the detection of arrhythmias from an electrocardiogram signal. arrhythmias from an electrocardiogram signal. For the elaboration of this project, the behavior of the arrhythmias has been studied, looking at the database of MIT and studying the behavior of the the behavior of the noted arrhythmias, it was observed that the vast majority of the arrhythmias that occurred the vast majority of the arrhythmias that occurred were due to premature contraction of the heart. initially intended to detect as many arrhythmias as possible, but did not see any clear examples of arrhythmias not caused by premature of arrhythmia not caused by premature contraction, the project only focused 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 the IIR filtering. Next, a QRS peak detection algorithm is applied to the filtered signal. QRS peak detection algorithm is then

applied to the filtered signal to find the highest peak that also exceeds the established dynamic cutoff. Finally 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. previous distance of a normal rhythm.

For the hardware implementation, 3 main modules will be used, which are the filtering module, the peak detection module and 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. In addition the results are evaluated through a simulation by creating a testbench.

For the hardware tests, the Basys3 FPGA is used since it is the FPGA used in the study and although it is not capable of Although it is not able to hold the 30 minutes of tests in RAM, with less tests it has a good performance.

In the .xdc a specific period has been established taking into account the frequency of the samples which is 365 sps and gives a period of 1852 ns. Thanks to the .xdc, a period of 1852 ns has been established. Thanks to the timing report we found that the minimum operating frequency is 6.49 ns.

According to the power report the power consumption of the board is 0.069 W which results in a low power consumption even for a continuous use of the board. Compared to other similar projects, the dynamic power consumption is lower.

Bibliografía