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

Introducción

1.1. Arritmias

Las enfermedades cardiovasculares son la primera causa de muerte en el mundo y una de las causas mas 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 cardiologos utilizan dispositivos como un Holter para generar tiras de ritmo o electrocardiogramas, que es un diagrama que representa los latidos del corazon y con eso pueden llegar a detectar arritmias.

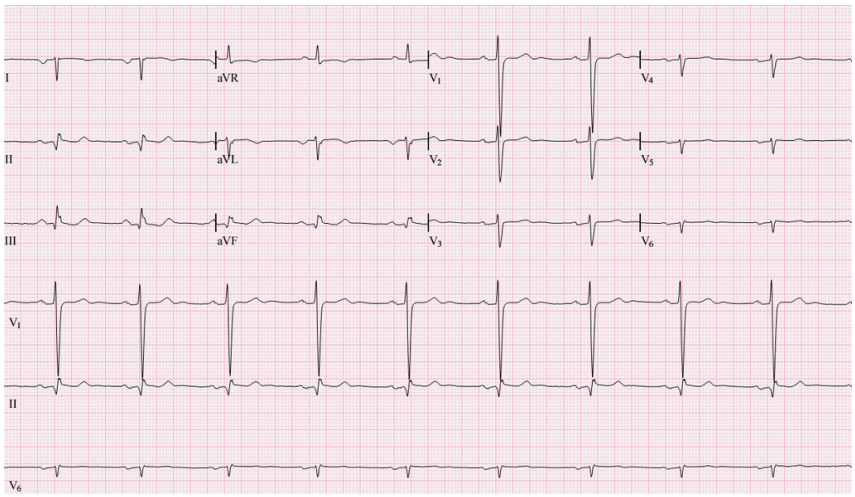


Figura 1.1: Electrocardiogramas

En este proyecto se tratara de solucionar las arritmias en las que se produce una contraccion prematura del corazon como las contracciones prematuras del corazón. Estas arritmias se pueden detectar con un electrocardiograma (ECG) que es un diagrama de los latidos del corazon.

1.2. Algoritmo de deteccion

Dado que para detectar arritmias correctamente se necesitan varios años de cardiologia,el algoritmo de deteccion que se utilizara consistira en detectar las arritmias unicamente usando los picos QRS del electrocardiograma.

Un pico QRS como se muestra en la Figura 1.2 en un electrocardiograma es causado por la contaccion del ventriculo al bombear la sangre por las arterias. Este es el impulso electrico mas fuerte que el corazon produce en cada latido. En este proyecto utilizaremos estos picos para comparar la distancia entre ellos y poder ver si se ha producido una arritmia.

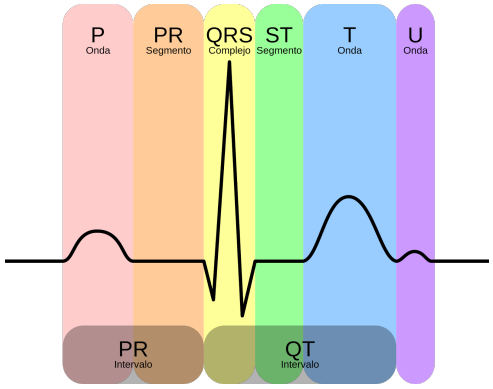


Figura 1.2: Complejo QRS

1.2.1. Filtrado

Como se puede ver en las imagenes 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 0 y evita fallos en el algoritmo de deteccion de picos del que se hablará mas adelante.

En la creacion 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.

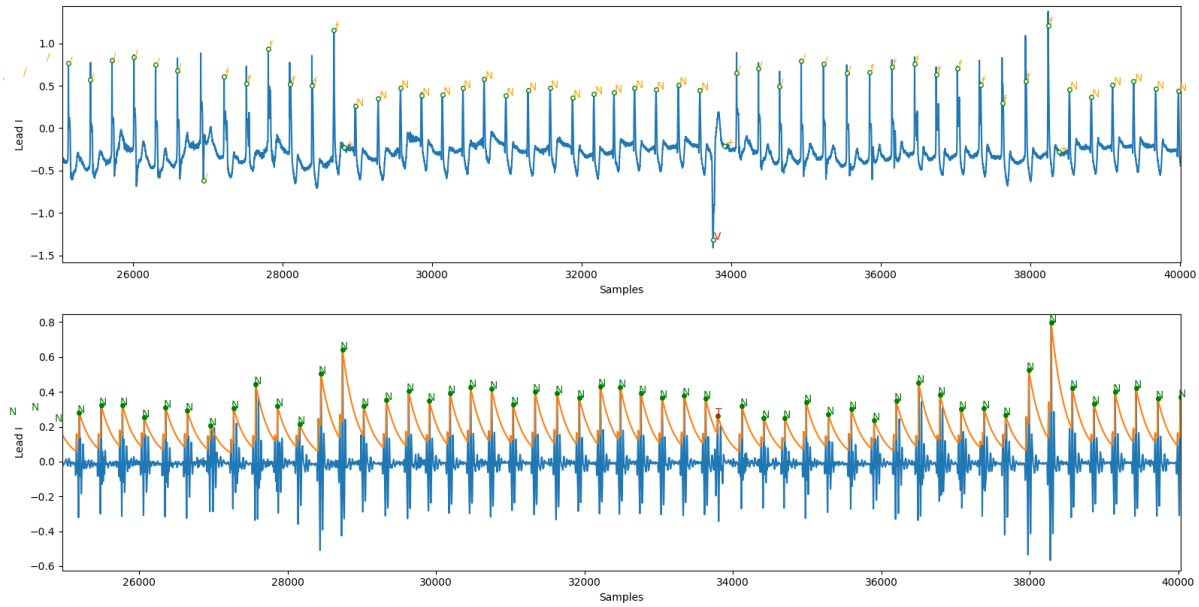
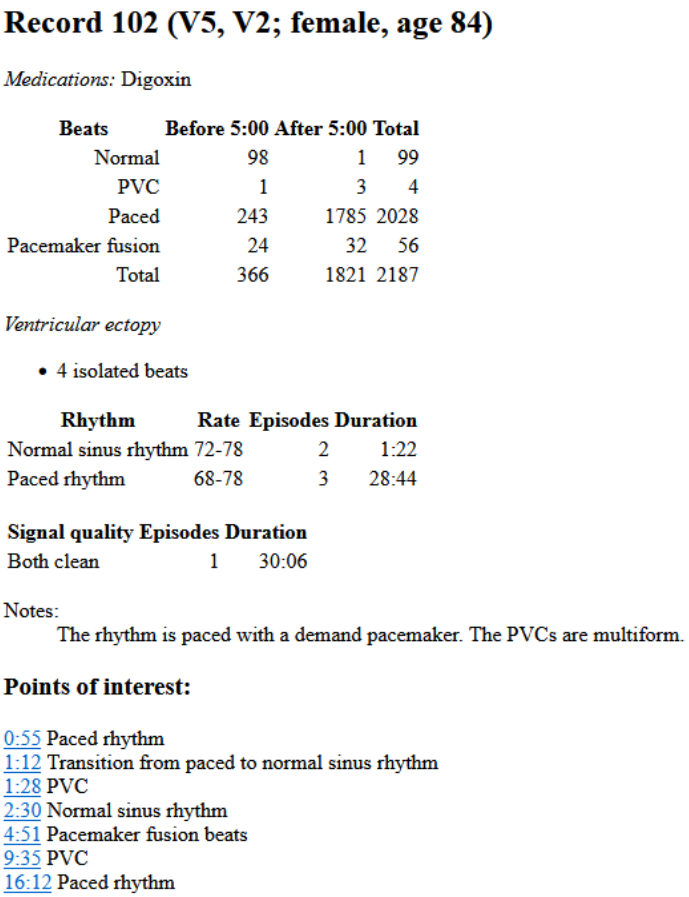


Figura 1.3: Ejemplo de electrocardiograma original y filtrado de paciente 102

1.3. Pruebas con pacientes

Se han realizado las pruebas con unos resultados del Instituto de Tecnología de Massachusetts (MIT) en el que se han recogido tiras de ritmo de media hora de 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.

Estas pruebas han sido analizadas por cardiologos y se ha indicado donde el paciente padece una arritmia y donde el ritmo es normal y donde se ha producido un error en la lectura de la señal. Tambien muestra informacion menos relevante como la activacion 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 1.4: Ejemplo con paciente 102

1.4. Utilizacion de las FPGAs

Este proyecto requiere un gran procesamiento de señales, una alta cantidad de calculos y un eficiente paralelismo entre modulos por ello la mejor forma de optimizar el algoritmo es utilizando una FPGA.

Los motivos son los siguientes:

- Las FPGA pueden procesar datos a velocidades muy altas, lo que lo hace indispensable para esta aplicacion que esta pensada para ejecutarse en tiempo real.
- Las FPGA son dispositivos de hardware programable que permite diseñar circuitos digitales personalizados, y por ello pueden reconfigurarse para adaptarse a tareas específicas. Ademas son susceptibles a cambios en el algoritmo para una posible mejora de este.
- El alto paralelismo que ofrecen las FPGA es perfecto para las multitareas que realiza el algoritmo.
- Puesto que las FPGA pueden ser diseñadas para realizar una tarea en concreto, estas son mas energeticamente eficientes que otros dispositivos como los portatiles.

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

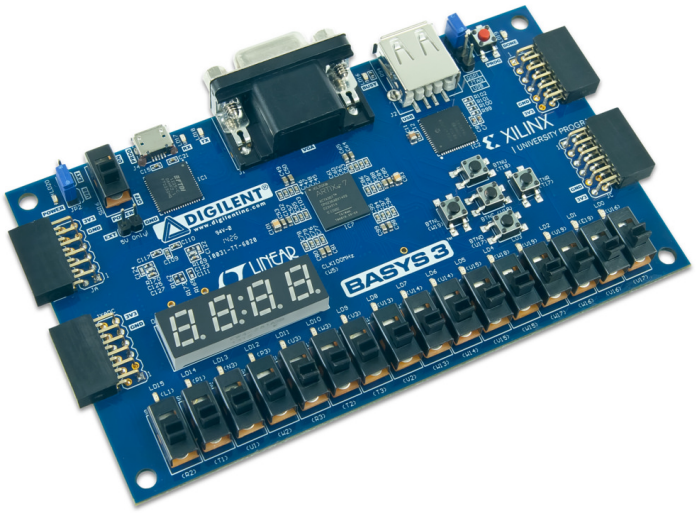


Figura 1.5: Basys3 Artix-7 FPGA

1.5. Objetivos del proyecto y organización

Los objetivos de este proyecto es tener una solucion para detectar contracciones prematuras ventriculares a tiempo real en un largo periodo de tiempo y optimizar el algoritmo para que se ejecute de una forma mas eficiente y menos costosa en una FPGA

Para ello la organizacion de este proyecto comienza con la creacion de el prototipado del algoritmo en software para facilitar la manera de probar el algoritmo con la solucion proporcionada por la base de datos y poder ver resultados graficos, para mejorar la velocidad de compilacion y depuracion del algoritmo, para aumentar la claridad del algoritmo que se quiere conseguir en el prototipado y para validar la funcionalidad y eficacia del algoritmo.

1.6. Analisis y optimizacion del algoritmo

Para lograr los objetivos del algoritmo se centra en tres funciones.

1. Filtrado de la señal original: Lo que hace que la señal sea mas facil de procesar para encontrar los picos QRS. Esto se realiza multiplicando los valores de la señal original por los valores de filtrado.

2. Deteccion de picos sobre la señal filtrada: Se analiza cada señal y comparandola con otras señales anteriores se deduce si puede ser un posible pico y si lo es, se comprueba si es un pico QRS.

3. Deteccion de arritmias comparando la posicion de los picos: una vez se tienen los picos QRS se calcula la distancia de el pico actual con el pico anterior y dependiendo de las otras distancias se calcula si hay una arritmia.

1.7. Implementacion en la FPGA

Para implementar el codigo en la FPGA se implementara varios modulos para tratar de imitar el proyecto creado en software los modulos mas importantes son.

1. Modulo de filtrado: Lo que hace que la señal sea mas facil de procesar para encontrar los picos QRS. Esto se realiza multiplicando los valores de la señal original por los valores de filtrado.
2. Deteccion de picos sobre la señal filtrada: Se analiza cada señal y comparandola con otras señales anteriores se deduce si puede ser un posible pico y si lo es, se comprueba si es un pico QRS.
3. Deteccion de arritmias comparando la posicion de los picos: una vez se tienen los picos QRS se calcula la distancia de el pico actual con el pico anterior y dependiendo de las otras distancias se calcula si hay una arritmia.

Estos modulos tratan de replicar las funcionalidades que realiza el algoritmo de software y se convertiran en la parte esencial de dicho programa.

Ademas de estos modulos se debe de crear un modulo que acompase a estos tres y un testbench para probar el funcionamiento del programa en la simulación.

1.8. Plantilla para usos de la herramienta

The document is divided into `chapters`, `sections`, and `subsections`.

Some important references are [**einstein**, **latexcompanion**, **knuthwebsite**].

To add paragraphs in the document, one line break is not enough,

two line breaks are needed.

An itemized list:

- An item.
- Another item.
- Final item.

An enumerated list:

1. First item.
2. Second item.
3. Third item.

A figure with an image is presented in Figura 1.6. Note that it floats away and latex places it where convenient.

Figura 1.6: Sample figure

Tables work in the same way, as seen in Tabla 1.1

Row	English	Español
1	One	Uno
2	Two	Dos

Tabla 1.1: Sample table

Capítulo 2

Planteamiento del algoritmo en software

2.1. 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 1.3

```
#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')
```

2.2. 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 trata de 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)
```

TODO a y formula completa ademas de una mejor explicacion

2.3. 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 https://www.mdpi.com/2079-9292/10/19/2324 donde en el 4.1.2 muestran una maquina de estados del proceso que realizan.

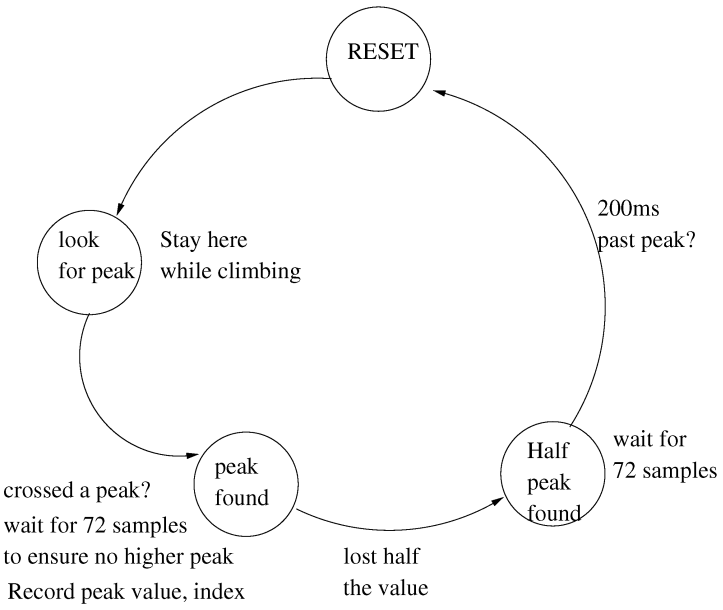


Figura 2.1: Maquina de estados de algoritmo de deteccion de picos de estudio de caracterizacion de señales usando polinomios de Hermite

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.

2.4. 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 2.2, por ello se compara con un valor anterior que sea el ritmo normal del paciente.

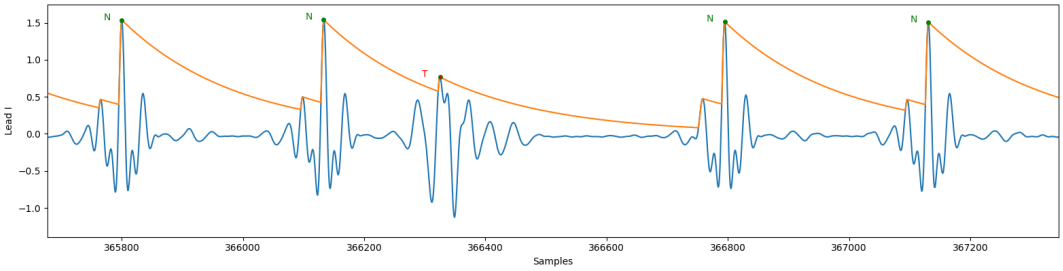


Figura 2.2: Cuando se detecta una arritmia, a veces, la siguiente distancia es considerablemente mas grande de lo normal. Para no detectar falsos positivos, se omite esa distancia

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 arrimtia.

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
```

2.5. 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 arrimtias 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 arrimtias contando falsos positivos y negativos.

```
print("detected:" + str(detected))
print("undetected:" + str(undetected))
print("correctValue:"+ str(correctValue))
print("incorrectValue:"+ str(incorrectValue))
print("correctArrythmia:"+ str(correctArrythmia))
print("incorrectArrythmia:"+ str(incorrectArrythmia))
print("————————————")
totalValues = undetected + detected
print("total values"+ str(totalValues))

totalDetected = detected / totalValues * 100
print("total detected"+ str(totalDetected))

totalUndetected = undetected / totalValues * 100
print("total undetected"+ str(totalUndetected))

totalCorrect = correctValue / detected * 100
print("total correct"+ str(totalCorrect))

totalIncorrect = incorrectValue / detected * 100
print("total incorrect"+ str(totalIncorrect))
if(incorrectArrythmia == 0):
    totalCorrectArrythmias = 100
else:
    totalCorrectArrythmias = correctArrythmia / (correctArrythmia +
        incorrectArrythmia) *100
print("total correct arrythmias" + str(totalCorrectArrythmias))
```

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 3

Implementación hardware

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

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

Por tanto en esta imagen quedan representados todos los modulos.

(IMAGEN DE MODULOS Y QUIEN LOS IMPLEMENTA)

3.1. Modulo de filtrado

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

Este modulo 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 modulo de multiplicacion de punto flotante.

3.1.1. 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 modulo 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 modulo del filtrado con el modulo 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;
```

3.1.2. Maquina de estados

Se realiza el proceso de sincronizacion de los estados, donde las señales siguientes pasan a las señales actuales.

```
    sync: process(clk)
    begin
        if ( rising_edge(clk) ) then
            if ( reset = '1' ) then
                state <= estado_espera;
                cont_muestras <= (others=>'0');
                cont_coeficientes <= (others=>'0');
                cont_indice <= (others=>'0');
                acumulado <= (others=>'0');
            else
                state <= next_state;
                cont_muestras <= next_cont_muestras;
                cont_coeficientes <= next_cont_coeficientes;
                cont_indice <= next_cont_indice;
                acumulado <= next_acumulado;
            end if;
        end if;
    end process sync;
```

Se realiza el proceso de actualizacion de las señales donde se le asignan nuevos valores para el siguiente ciclo de reloj.

- 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 modulo de multiplicacion.
- Estado de espera del calculo: se espera a que termine el modulo 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 modulo.
- Estado de envio de nuevo dato: este estado sincroniza el siguientemodulo, activa el bit de valid a 1 y espera el bit de ready del siguiente modulo para poder enviar el dato.

```
    cmb: process(state, cont_coeficientes, cont_muestras, cont_indice, acumulado,
        input_valid, ready_muladd, result_muladd, output_ready)
    begin
        -- Registros de estado y de seniales
        next_state <= state;
        next_cont_coeficientes <= cont_coeficientes;
        next_cont_muestras <= cont_muestras;
        next_acumulado <= acumulado;
        next_cont_indice <= cont_indice;
```

```
-- Seniales de control de los buses
input_ready <= '0';
output_valid <= '0';

-- Seniales de control para la ruta de datos
ROM_coeficientes_ena <= '0';
RAM_muestras_ena <= '0';
RAM_muestras_wea <= "0";
enable_muladd <= '0';

case state is

    when estado_espera =>
        input_ready <= '1';
        if ( input_valid = '1' ) then
            next_acumulado <= (others=>'0');
            -- Almacenamos el valor de entrada
            RAM_muestras_ena <= '1';
            RAM_muestras_wea <= "1";
            if ( to_integer(unsigned(cont_muestras)) < 98 ) then
                next_cont_muestras <= std_logic_vector(unsigned(
                    cont_muestras) + 1);
            else
                next_cont_muestras <= (others=>'0');
            end if;
            next_state <= estado_lectura;
        end if;

    when estado_lectura =>
        -- Hacemos la lectura de ambas memorias
        ROM_coeficientes_ena <= '1';
        RAM_muestras_ena <= '1';
        next_state <= estado_ordenar_calculo;

    when estado_ordenar_calculo =>
        enable_muladd <= '1';
        next_state <= estado_espera_calculo;

    when estado_espera_calculo =>
        if ( ready_muladd = '1' ) then
            next_acumulado <= result_muladd;

            if ( to_integer(unsigned(cont_muestras)) < 98 ) then
                next_cont_muestras <= std_logic_vector(unsigned(
                    cont_muestras) + 1);
            else
                next_cont_muestras <= (others=>'0');
            end if;

            if ( to_integer(unsigned(cont_coeficientes)) < 98 ) then
                next_cont_coeficientes <= std_logic_vector(unsigned(
                    cont_coeficientes) + 1);
            else
                next_cont_coeficientes <= (others=>'0');
            end if;

            if ( to_integer(unsigned(cont_coeficientes)) < 98 ) then
                next_state <= estado_lectura;
            else
                next_state <= estado_enviar_nuevo_dato;
            end if;

        end if;
```

```
            when estado_enviar_nuevo_dato =>
                output_valid <= '1';
                if ( output_ready = '1' ) then
                    next_cont_indice <= std_logic_vector(unsigned(cont_indice) +
                        1);
                    next_state <= estado_espera;
                end if;

        end case;
    end process cmb;
```

El envio de datos al output se realiza de forma asincrona

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

3.1.3. Modulos 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
    );

    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 modulo 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
    );
```

3.2. Modulo de deteccion de picos

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

3.2.1. 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 lso indices de las muestras de la señal filtrada
- input_valid e input_ready: son flags que sirven para sincronizar el modulo 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 modulo de deteccion de picos con el modulo 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
);
```

3.2.2. Maquina de estados

Se realiza el proceso de sincronizacion de los estados, donde las señales siguientes pasan a las señales actuales.

```
    sync: process(clk)
    begin
        if ( rising_edge(clk) ) then
            if ( reset = '1' ) then
                state <= estado_espera;
                signal_data <= (others=>'0');
                signal_index <= (others=>'0');
                last_peak <= (others=>'0');
                last_index <= (others=>'0');
                cutoff <= (others=>'0');
            else
                state <= next_state;
```

```
                signal_data <= next_signal_data;
                signal_index <= next_signal_index;
                last_peak <= next_last_peak;
                last_index <= next_last_index;
                cutoff <= next_cutoff;
            end if;
        end if;
    end process sync;
```

Se realiza el proceso de actualizacion de las señales donde se le asignan nuevos valores para el siguiente ciclo de reloj.

- 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 modulos 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 modulo 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 modulo 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 modulo de la resta ya que es la ultima operacion que se realiza para calcular el cutoff. Primero se ejecuta el modulo 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 modulo 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 modulo 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.

```
    cmb: process(state, signal_data, signal_index, last_peak, last_index, cutoff,
        input_valid, input_signal_data, input_signal_index,
        ready_signal_data_gt_cutoff, result_signal_data_gt_cutoff, ready_sub,
        result_sub, ready_signal_data_gt_last_peak,
        result_signal_data_gt_last_peak,
        result_signal_index_sub_last_index_gt_or_eq_samples_around_peak,
        result_last_peak_gt_cutoff, output_ready, samples_around_peak)
```

```
begin
    -- Registros de estado y de seniales
    next_state <= state;
    next_signal_data <= signal_data;
    next_signal_index <= signal_index;
    next_last_peak <= last_peak;
    next_last_index <= last_index;
    next_cutoff <= cutoff;

    -- Seniales de control de los buses
    input_ready <= '0';
    output_valid <= '0';

    -- Seniales de control para la ruta de datos
    enable_signal_data_gt_cutoff <= '0';
    enable_div <= '0';
    enable_signal_data_gt_last_peak <= '0';
    enable_last_peak_gt_cutoff <= '0';

    case state is

        -- Esperamos a que llegue una nueva muestra
        when estado_espera =>
            -- El modulo esta listo para recibir nuevos datos
            input_ready <= '1';
            if ( input_valid = '1' ) then
                -- Almaceno los valores de entrada
                next_signal_data <= input_signal_data;
                next_signal_index <= input_signal_index;
                next_state <= estado_comprueba_indice;
             end if;

        when estado_comprueba_indice =>
            if ( to_integer(unsigned(signal_index)) < unsigned(
               samples_around_peak)) then  -- Las primeras 300 muestras las
                utilizamos para fijar un cutoff inicial
                next_state <= estado_espera;
            else
                if ( last_peak = x"00000000" ) then
                    next_last_peak <= signal_data;
                    next_last_index <= signal_index;
                    -- ordenar la actualizacion del cutoff (division)
                    enable_div <= '1';
                    next_state <= estado_actualizar_cutoff;
                else
                    -- Ordenamos hacer la comparacion 'signal[i] > last_peak'
                    enable_signal_data_gt_last_peak <= '1';
                    enable_signal_data_gt_cutoff <= '1';

                    enable_last_peak_gt_cutoff <= '1';
                    next_state <=
                        estado_espera_condicion_signal_data_gt_last_peak;
                end if;
            end if;

        when estado_actualizar_cutoff =>
            -- Ha terminado la resta
            if ( ready_sub = '1' ) then
                next_cutoff <= result_sub;
                next_state <= estado_espera;
            end if;

         when estado_espera_condicion_signal_data_gt_last_peak =>
            if ( ready_signal_data_gt_last_peak = '1' and
                ready_signal_data_gt_cutoff = '1') then
```

```
                    if ( result_signal_data_gt_last_peak(0) = '1' and
                        result_signal_data_gt_cutoff(0) = '1') then
                        next_last_peak <= signal_data;
                        next_last_index <= signal_index;
                        next_cutoff <= signal_data;
                        next_state <= estado_espera;
                    elsif (
                        result_signal_index_sub_last_index_gt_or_eq_samples_around_peak
                        ='1' and result_last_peak_gt_cutoff(0) = '1' ) then
                        -- Hemos encontrado un nuevo pico
                        next_state <= estado_envio_nuevo_pico;
                    else
                        -- ordenar la actualizacion del cutoff (division)
                        enable_div <= '1';
                        next_state <= estado_actualizar_cutoff;
                    end if;
                end if;

            when estado_envio_nuevo_pico =>
                    -- output_peak_index <= last_index;
                    output_valid <= '1';
                    if ( output_ready = '1' ) then
                        next_last_peak <= (others => '0');
                        next_last_index <= (others => '0');
                        enable_div <= '1';
                        next_state <= estado_actualizar_cutoff;
                    end if;

        end case;
    end process cmb;
```

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

```
    output_peak_index <= last_index;
```

3.2.3. Módulos utilizados

Se usa un modulo de comparador para 2 comparaciones, y uno de division y otro de resta para calcular el cutoff (TODO IMAGEN DE LOS MODULOS)

```
-- Comparador 'signal[i] > cutoff'
signal_data_gt_cutoff : entity work.gtFP port map (
    aclk                  => clk,
    s_axis_a_tvalid       => enable_signal_data_gt_cutoff,
    s_axis_a_tdata        => signal_data,
    s_axis_b_tvalid       => enable_signal_data_gt_cutoff,
    s_axis_b_tdata        => cutoff,
    m_axis_result_tvalid  => ready_signal_data_gt_cutoff,
    m_axis_result_tdata   => result_signal_data_gt_cutoff
);

-- Divisor 'cutoff / 192'
divisor : entity work.divFP port map (
    aclk                  => clk,
    s_axis_a_tvalid       => enable_div,
    s_axis_a_tdata        => cutoff,
    s_axis_b_tvalid       => enable_div,
    s_axis_b_tdata        => x"43400000",                    -- 192 en decimal
    m_axis_result_tvalid  => ready_div,
    m_axis_result_tdata   => result_div
);
```

```
-- Restador 'cutoff - cutoff/192'
enable_sub <= ready_div;
restador : entity work.subfp port map(
    aclk                => clk,
    s_axis_a_tvalid     => enable_sub,
    s_axis_a_tdata      => cutoff,
    s_axis_b_tvalid     => enable_sub,
    s_axis_b_tdata      => result_div,
    m_axis_result_tvalid => ready_sub,
    m_axis_result_tdata  => result_sub
);

-- Comparacion 'signal[i] > last_peak'
signal_data_gt_last_peak: entity work.gtFP port map(
    aclk                => clk,
    s_axis_a_tvalid     => enable_signal_data_gt_last_peak,
    s_axis_a_tdata      => signal_data,
    s_axis_b_tvalid     => enable_signal_data_gt_last_peak,
    s_axis_b_tdata      => last_peak,
    m_axis_result_tvalid => ready_signal_data_gt_last_peak,
    m_axis_result_tdata  => result_signal_data_gt_last_peak
);

-- Comparacion 'last_peak > cutoff'
last_peak_gt_cutoff: entity work.gtFP port map(
    aclk                => clk,
    s_axis_a_tvalid     => enable_last_peak_gt_cutoff,
    s_axis_a_tdata      => last_peak,
    s_axis_b_tvalid     => enable_last_peak_gt_cutoff,
    s_axis_b_tdata      => cutoff,
    m_axis_result_tvalid => ready_last_peak_gt_cutoff,
    m_axis_result_tdata  => result_last_peak_gt_cutoff
);
```

3.3. Modulo de deteccion de arritmias

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

3.3.1. Señales de entrada y salida

las señales de entrada de este modulo 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 modulo con el modulo 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 modulo output.

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

        input_peak_index: in std_logic_vector(31 downto 0); --por coherencia con
            el modulo anterior lo dejamos asi
        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
       );
```

3.3.2. Maquina de estados

Se realiza el proceso de sincronizacion de los estados, donde las señales siguientes pasan a las señales actuales.

```
sync: process(clk,rst)
begin
    if rising_edge(clk) then
        if(rst = '1') then
            state <= estado_espera;
            peak_index <= (others=>'0');
            last_peak_index <= (others=>'0');
            act_distance <= (others=>'0');
            last_distance <= (others=>'0');
            pos_buffer0 <= (others=>'0');
            pos_buffer1 <= (others=>'0');
            pos_buffer2 <= (others=>'0');
            pos_buffer3 <= (others=>'0');
            TNRange <= '0';
            counter_arrythmias <= (others=>'0');
            counter <= (others=>'0');
            --para que no de arritmia al principio inicializo el gap a 1
            gap <= (others => '0');
            distance_for_calc <= (others => '0');
            --percentage <= '0';
        else
            state <= next_state;
            peak_index <= next_peak_index;
            last_peak_index <= next_last_peak_index;
            act_distance <= next_act_distance;
            last_distance <= next_last_distance;
            pos_buffer0 <= next_pos_buffer0;
            pos_buffer1 <= next_pos_buffer1;
            pos_buffer2 <= next_pos_buffer2;
            pos_buffer3 <= next_pos_buffer3;
            TNRange <= next_TNRange;
            counter_arrythmias <= next_counter_arrythmias;
            counter <= next_counter;
            -- frequency calc
            gap <= next_gap;
            distance_for_calc <= next_distance_for_calc;
            --percentage <= next_percentage;
        end if;
    end if;
end process;
```

Se realiza el proceso de actualizacion de las señales donde se le asignan nuevos valores para el siguiente ciclo de reloj.

- 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
- 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 modulos 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 modulo 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 modulo 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 modulo de la resta ya que es la ultima operacion que se realiza para calcular el cutoff. Primero se ejecuta el modulo 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:
- Estado de envio de nuevo pico: se activa la señal de valid a 1 y espera a que el modulo 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.

```
fsm: process(distance_for_calc,TNRange,state,peak_index,last_peak_index,
    act_distance,last_distance,pos_buffer0,pos_buffer1,pos_buffer2,pos_buffer3,
    counter_arrythmias,counter,gap,percentage,input_valid,input_peak_index,
    output_ready)
begin
        next_state <= state;
        next_peak_index <= peak_index;
        next_last_peak_index <= last_peak_index;
        next_act_distance <= act_distance;
        next_last_distance <= last_distance;
        next_pos_buffer0 <= pos_buffer0;
        next_pos_buffer1 <= pos_buffer1;
        next_pos_buffer2 <= pos_buffer2;
        next_pos_buffer3 <= pos_buffer3;
        next_TNRange <= TNRange;
        next_counter_arrythmias <= counter_arrythmias;
        next_counter <= counter;
        -- frequency calc
        next_gap <= gap;
        next_distance_for_calc <= distance_for_calc;
        --next_percentage <= percentage;
        -- sync with other modules
        input_ready <= '0';
        output_valid <= '0';
case state is
        when estado_espera =>
         -- El modulo esta listo para recibir nuevos datos
            input_ready <= '1';
            if ( input_valid = '1' ) then
                -- Almaceno los valores de entrada
```

```
                next_peak_index <= input_peak_index;
                next_state <= S0;
            end if;
        when S0 =>
            if(to_integer(unsigned(counter)) = 0) then
                next_last_peak_index <= peak_index;
                next_counter <= std_logic_vector(unsigned(counter)+1);
                next_state <= estado_espera;
            elsif(to_integer(unsigned(counter)) = 1) then
                next_last_distance <= std_logic_vector(unsigned(peak_index) -
                    unsigned(last_peak_index));
                next_state <= S1;
            else
                next_state <= S1;
            end if;
        when S1 =>
            next_counter <= std_logic_vector(unsigned(counter)+1);
            next_act_distance <= std_logic_vector(unsigned(peak_index) - unsigned(
                last_peak_index));
            next_state <= S2;
        when S2 =>
            next_pos_buffer3 <= pos_buffer2;
            next_pos_buffer2 <= pos_buffer1;
            next_pos_buffer1 <= pos_buffer0;
            next_pos_buffer0 <= act_distance;
            next_state <= S3;
        when S3 =>
            if(TNRange = '1' and counter_arrythmias = x"00000000") then
                next_TNRange <= '0';
                next_gap <= std_logic_vector(signed(pos_buffer3) - signed(
                    act_distance));
                next_distance_for_calc <= pos_buffer3;
                --next_state <= S4;
            else
                next_gap <= std_logic_vector(signed(last_distance) - signed(
                    act_distance));
                next_distance_for_calc <= last_distance;
                if TNRange = '1' then
                    next_counter_arrythmias <= std_logic_vector(unsigned(
                        counter_arrythmias) - 1);
                end if;                                          --signed o
                    unsigned?

            end if;

            next_state <= S6;

        when S6 =>
            if(percentage = '1') then
                next_TNRange <= '1';
                next_counter_arrythmias <= x"00000001";
            end if;
            next_last_peak_index <= peak_index;
            next_last_distance <= act_distance;
            next_state <= S7;
        when S7 =>
            output_valid <= '1';
            if(output_ready = '1') then
                next_state <= estado_espera;
            end if;
    end case;
end process;
```

3.4. Modulos input y output

3.5. Modulo principal y testbench

3.6. Otros modulos

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

3.6.1. Modulos ROM y RAM

ROM muestras

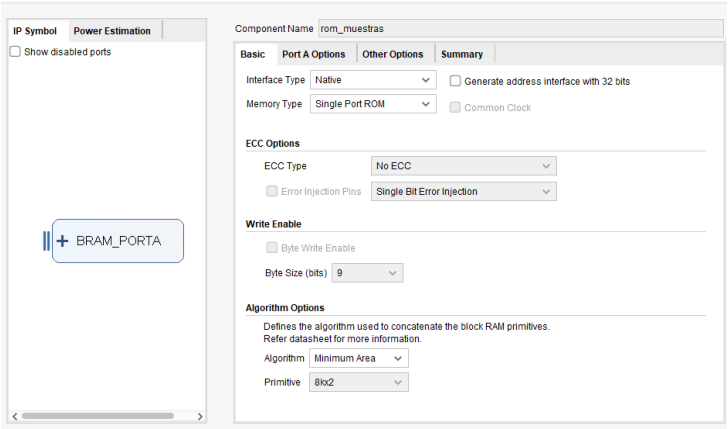


Figura 3.1: Seleccion de la opcion simple block ROM y

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.

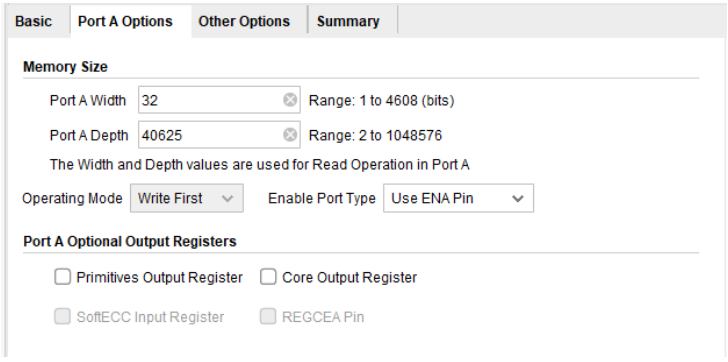


Figura 3.2: Se establecen las filas de la BROM y la longitud de estas

El modulo de filtrado utiliza 1 ROM y una RAM

- La ROM se configura igual que la ROM del modulo 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.

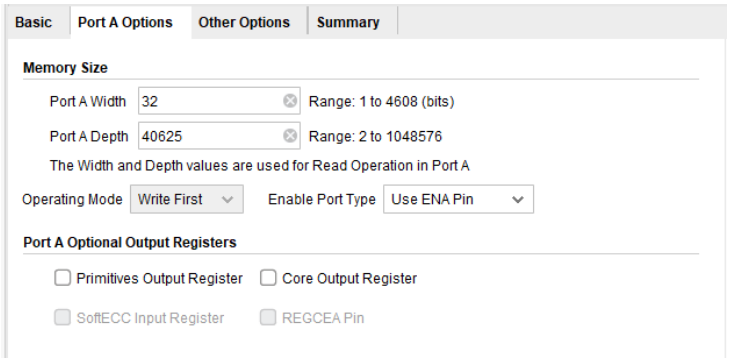


Figura 3.3: Se establecen las filas de lectura y escritura de la RAM y las longitud de dichas filas

Capítulo 4

Resultados Experimentales

Se han hecho pruebas con 15 pacientes y los 15 tienen un porcentaje de exito por encima del 50 %

4.1. Placa utilizada

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 mas potente como (FPGA DE LA FACULTAD MAS POTENTE)

4.2. Consumo

analisis de sintesis -¿

que hace falta poner del syntesis report?

en la parte de power..

como comparo la el consumo de la placa con otros dispositivos?

hay que hacer el analisis de sintesis de cada modulo, para eso lo meto en un proyecto aparte y lo hago? no se puede coger compilandolo todo junto

en el analisis de sintesis que es el cell usage? Que son los LUTs? hay que indicar cuantos luts se utilizan?

Para que sirve el noise? hay que poner cuanto noise se genera?

hace falta poner el timing summary? qhace falta indicar cosas como cual es el worst negative slack?

Capítulo 5

Results

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Capítulo 6

Conclusiones y trabajo futuro

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Conclusions and future work

Translate the previous chapter

Bibliografía