ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL DA TESE DEFENDIDA POR Alexandre Tomazati Oliveira E APROVADA PELA COMISSÃO JULGADORA EM 22 / 12 / 2009

ORIENTADOR

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Detecção do complexo QRS em sinais cardíacos utilizando FPGA

Autor: Alexandre Tomazati Oliveira

Orientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega

01/2010

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

Detecção do complexo QRS em sinais cardíacos utilizando FPGA

Autor: Alexandre Tomazati Oliveira
Orientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega

Curso: Engenharia Mecânica
Área de Concentração: Mecânica dos Sólidos e Projeto Mecânico

Dissertação de mestrado acadêmico apresentada à comissão de Pós Graduação da Faculdade de Engenharia Mecânica, como requisito para a obtenção do título de Mestre em Engenharia Mecânica.

Campinas, 2009

S.P. - Brasil

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Oliveira, Alexandre Tomazati
OL41d Detecção do complexo QRS em sinais cardíacos utilizando FPGA / Alexandre Tomazati Oliveira. -- Campinas, SP: [s.n.], 2009.

Orientador: Eurípedes Guilherme de Oliveira Nóbrega.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Eletrocardiograma. 2. Wavelet (Matematica). 3. VHDL (Linguagem descritiva de *hardware*). I. Nóbrega, Eurípedes Guilherme de Oliveira. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

Título em Inglês: QRS complex detection in cardiac signals using FPGA
Palavras-chave em Inglês: Electrocardiography, Wavelets (Mathematics), VHDL (Computer *hardware* description language)
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico
Titulação: Mestre em Engenharia Mecânica
Banca examinadora: Niederauer Mastelari, Takashi Yoneyama
Data da defesa: 22/12/2009
Programa de Pós Graduação: Engenharia Mecânica

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

DISSERTAÇÃO DE MESTRADO ACADÊMICO

Detecção do complexo QRS em sinais cardíacos utilizando FPGA

Autor: Alexandre Tomazati Oliveira
Orientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega, Presidente
Universidade Estadual de Campinas

Prof. Dr. Niederauer Mastelari
Universidade Estadual de Campinas

Prof. Dr. Takashi Yoneyama
Instituto Tecnológico de Aeronáutica

Campinas, 22 de dezembro de 2009.

Dedicatória:

Dedico este trabalho à minha família pelo apoio irrestrito em todos os momentos e pela referência na formação dos meus princípios éticos e morais, indispensáveis na minha formação social e acadêmica.

Aos meus amigos verdadeiros que conquistei no docorrer da vida.

Agradecimentos:

Agradeço inicialmente a DEUS por ter me dado a oportunidade e capacidade física e mental para a realização deste trabalho de pós-graduação.

Ao meu orientador Eurípedes Guilherme de Oliveira Nóbrega por confiar e acreditar no meu trabalho, conduzindo-me na direção correta e contribuindo para o enriquecimento da pesquisa.

À Biosensor Ind. e Com. LTDA que financiou o meu mestrado, contribuindo para a realização das pesquisas e pela confiança depositada nesta parceria com a Unicamp e no meu trabalho.

A todos os professores que participaram desta minha caminhada, sempre dispostos a ajudar, transferindo o conhecimento e auxiliando na formação de novas idéias.

À minha família pelo carinho e apoio, sempre me fortalecendo e amparando nos momentos difíceis.

À minha namorada pela compreensão, paciência, apoio e amparo nos momentos difíceis, contribuindo enormemente para a realização deste trabalho.

Aos meus amigos pelo incentivo, auxílio e companheirismo.

“A arte da vida consiste em fazer da vida uma obra de arte.”

MAHATMA GANDHI

Resumo:

OLIVEIRA, Alexandre Tomazati, *Detecção do complexo QRS em sinais cardíacos utilizando FPGA*. 2009. 136 p.

Dissertação (Mestrado) – Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, Campinas.

O eletrocardiograma (ECG) é uma ferramenta utilizada para o diagnóstico de cardiopatias e outras doenças. Este trabalho tem como objetivo a detecção do complexo QRS, com foco na onda R, que representa a contração dos ventrículos. Para isso, são apresentadas duas técnicas de processamento do sinal de ECG. A primeira utiliza o algoritmo proposto por Pan & Tompkins que consiste em um banco de filtros digitais. A segunda faz uso da transformada *wavelet* discreta, que permite a localização de características de sinais tanto no tempo quanto na frequência. É apresentado um comparativo da eficácia dos dois algoritmos com base na sua implementação através de FPGA, utilizando dois métodos, o processamento serial em microcontrolador programado em C e o paralelo inteiramente em VHDL, com o intuito de comparar os tempos de processamento. Os resultados sugerem que trabalhos futuros poderão ser baseados na investigação de outras famílias *wavelets* para a detecção do complexo QRS em sinais de ECG, bem como explorar outros métodos de implementação de filtros em FPGA.

Palavras Chave

- ECG, FPGA, complexo QRS, VHDL, *Wavelet*, Filtro Digital.

Abstract:

OLIVEIRA, Alexandre Tomazati, *QRS complex Detection in cardiac signals using FPGA*. 2009. 136 p.

Dissertation (Master Degree in Mechanical Engineering): Faculty of Mechanical Engineering, State University of Campinas, Campinas.

The electrocardiogram (ECG) is a tool used for diagnosis of diseases related to the heart. This work has the purpose of detecting QRS complex, focusing on the R wave, which represents the ventricles'contraction. It is presented two techniques of processing ECG signals. The first uses Pan & Tompkins algorithm based on digital filtering. The second uses the discrete wavelet transform, which represents the characteristics of the signal simultaneously in time and frequency. It is presented a comparison of the efficacy of both algorithms, which are implemented in FPGA, using serial processing based on a C programmed microcontroller, and parallel processing entirely in VHDL, with the purpose of comparing the time of processing. The results suggest that future work can be based on the investigation of other wavelets family for detecting QRS complex in ECG signals and other methods of implementing filters in FPGA.

Key Words

- ECG, FPGA, QRS complex, VHDL, Wavelet, Digital Filter.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

Nomenclatura

Siglas

AMR - Análise Multi-resolução

ASIC - Application Specific Integrated Circuit

CI - Circuito Integrado

CPLD - Complex Programmable Logic Device

CWT - Transformada *Wavelet* Contínua (Continuous Wavelet Transform)

DFT - Transformada de Fourier Discreta (Discrete Fourier Transform)

DSP - Processador Digital de Sinais (Digital Signal Processor)

DSTFT - Transformada Janelada de Fourier Discreta (Discrete Short Time Fourier Transform)

DWT - Transformada *Wavelet* Discreta (Discrete Wavelet Transform)

ECG - Eletrocardiograma

EDA - Electronic Design Automation

EDK - Embedded Development Kit

EEPROM - Electrically Erasable Programmable Read-Only Memory

FFT - Transformada Rápida de Fourier (Fast Fourier Transform)

FIR - Filtro de Resposta ao Impulso Finita (Finite Impulse Response)

FN - Falso Negativo

FP - Falso Positivo

FPGA - Field Programmable Gate Array

FSL - Fast Simplex Link

FT **-** Transformada de Fourier (Fourier Transform)

HDL - *Hardware* Description Language

IEEE - Institute of Electrical and Electronics Engineers

IIR - Filtro de Resposta ao Impulso Infinita (Infinite Impulse Response)

ISE - Integrated Software Environment

LUT - Look Up Table

MIT/BIH - Massachusetts Institute of Technology / Beth Israel Hospital

MPGA - Mask Programmable Gate Array

PAL - Programmable Array Logic

PLA - Programmable Logic Array

PLD - Programmable Logic Device

RAM - Random Acess Memory

RISC - Reduced Instruction Set Computer

SoC - System on Chip

SPLD - Simple Programmable Logic Device

SRAM - Static Random Acess Memory

STFT - Transformada Janelada de Fourier (Short Time Fourier Transform)

TWE - Transformada Wavelet Estacionária

VHDL - Very-High-Speed Integrated Circuit *Hardware* Description Language

VLSI - Very Large Scale Integration

VP - Verdadeiro Positivo

WT - Transformada Wavelet (Wavelet Transform)

XPS - Xilinx Plataform Studio

Capítulo 1

1 Introdução

O avanço tecnológico decorrente do surgimento de sistemas microprocessados tem contribuído enormemente para o processamento, tratamento e análise de sinais biomédicos. Este recurso tem auxiliado no aumento da sobrevida de pacientes que sofrem de doenças crônicas, como as de origem cardiovascular, que causam um terço das mortes no Brasil (Portal do Coração).

O ECG (eletrocardiograma), registro dos fenômenos elétricos na superfície do corpo gerados durante a atividade cardíaca, é bastante utilizado, pois fornece informações rápidas e seguras sobre alterações como as arritmias, isquemias e distúrbios eletrolíticos que podem afetar a função cardíaca. Os eventos identificados no ECG são suas ondas (P, Q, R, S e T), intervalos (PR, ST, QT e RR) e segmentos (P-R e S-T) característicos, os quais apresentam flutuações batimento a batimento. Por meio da análise da morfologia, amplitude, duração e polaridade dos diferentes eventos eletrocardiográficos, dentre outros aspectos, consegue-se estabelecer o diagnóstico da condição patológica ou de normalidade do coração. As três ondas Q, R e S formam o chamado complexo QRS e representam a despolarização (contração) dos ventrículos. O ECG pode ser registrado não só em repouso, mas também durante o esforço físico (teste ergométrico), a fim de avaliar o estado do coração, uma vez que o paciente pode ter um ECG de repouso sem alterações, porém quando submetido ao esforço, podem aparecer alterações significativas.

A detecção do complexo QRS é importante em uma análise automática, pois pode ajudar a determinar a taxa cardíaca do paciente, bem como outras características do ECG, de forma rápida e precisa, sendo um instrumento rápido dos processos de diagnóstico para o médico especialista.

A detecção precisa do complexo QRS é de vital importância em instrumentos clínicos e pode ser obtida automaticamente através de algoritmos específicos. Mesmo sendo o complexo QRS a característica dominante do sinal de ECG, com esta detecção podendo ser feita facilmente pelo olho treinado de um cardiologista, a automatização deste processo encontra algumas dificuldades, tais como a variação da morfologia das ondas, condições fisiológicas e a presença de ruídos. Em geral, a detecção automática é dividida em três etapas: aquisição dos sinais eletrocardiográficos, pré-processamento do sinal e detecção da onda R. Para isso é necessário o uso de ferramentas matemáticas robustas, como por exemplo, o uso de filtros digitais e transformadas, cuja principal função é possibilitar análises do sinal (série temporal) no domínio da frequência.

Dentre as principais transformadas, podemos citar:

- A transformada de Fourier (FT), desenvolvida por Jean Baptiste Joseph Fourier em 1822 (HSU, 1973);
- A transformada rápida de Fourier (FFT), executada inicialmente através do algoritmo de Cooley-Turkey (HSU, 1973);
- A transformada janelada de Fourier (QIAN, 2002);
- A transformada *wavelet* (WT), desenvolvida por Jean Morlet em 1984 (QIAN, 2002).

Jean Baptiste Joseph Fourier demonstrou que qualquer função $f(t)$ periódica pode ser representada como uma soma infinita de senos e co-senos, ou seja, uma soma infinita de funções periódicas exponenciais complexas ($e^{-2j\pi ft}$). Este tratamento matemático, conhecido como transformada de Fourier (FT), tornou possível a análise no domínio da frequência, porém não permitindo a análise das informações no domínio do tempo (HSU, 1973).

A FT foi e ainda é largamente utilizada graças ao desenvolvimento do algoritmo de Cooley-Turkey, conhecido como transformada rápida de Fourier (FFT). Entretanto, a FFT contém as mesmas limitações da FT.

Na intenção de superar as limitações da FT para sinais cujo índice de frequência varia com o tempo (sinais não-estacionários), foi desenvolvida a transformada janelada de Fourier (STFT – do inglês *Short Time Fourier Transform*). O sinal, na STFT, é dividido em segmentos (janelas) de tempo fixos e pequenos, onde, presume-se, que o sinal seja estacionário. Note que esta técnica permite uma dupla localização, no domínio do tempo e no domínio da frequência, uma vez que podemos conhecer o conteúdo espectral em intervalos de tempo definidos. Mas como a STFT possui janelas de comprimento fixo no tempo, em alguns casos pode não ser possível obter uma boa resolução em ambos os domínios (tempo e frequência), pois se o tamanho de uma janela for menor que o período de determinada componente espectral do sinal, não será possível detectá-la (QIAN, 2002).

A transformada *wavelet* (WT) também se baseia no janelamento do sinal. Entretanto, este janelamento não é fixo e utiliza um conjunto de funções base obtido a partir de translações e escalonamentos da *wavelet* mãe. A função da escala é permitir a variação do comprimento da "janela" e a função da translação é a de deslocar esta "janela" pelo sinal em análise. Assim, é possível conhecer as frequências que compõem um sinal e sua localização no domínio do tempo. As propriedades das *wavelets* serão vistas com mais detalhes no Capítulo 2.

O termo "*wavelet*" pode ser entendido como pequena onda, e foi originariamente introduzido por J. Morlet (MORETTIN, 1999). Desde então, a teoria das *wavelets* tem atraído a atenção de diversos pesquisadores, encontrando aplicações nas mais diferentes áreas, como por exemplo, geofísica, análise e compressão de sinais, compressão de imagens, economia e processamento de sinais biomédicos. No que envolve o processamento de sinais de eletrocardiograma, características como a boa localização no tempo e na frequência, assim como a análise multi-resolução, fazem da transformada *wavelet* instrumento de ótimo desempenho na detecção de ondas e intervalos.

O termo "filtro digital", ou simplesmente filtro, é usado frequentemente para denotar um sistema de tempo discreto. Um filtro digital foi definido por J. F. Kaiser como um "...processo

computacional ou algoritmo pelo qual um sinal amostrado ou seqüência de números (atuando como entrada) é transformado em uma segunda seqüência de números chamada sinal de saída. O processo computacional pode ser de filtragem passa-baixa (suavização), filtragem passa-faixa, interpolação, geração de derivadas, etc" (HAYES, 1999).

As técnicas de processamento de sinal citadas acima precisam ser implementadas em sistemas computacionais. Tradicionalmente decide-se entre implementações em *hardware* customizado (fixo) ou *software*. Em alguns sistemas, toma-se esta decisão para cada sub-tarefa, alocando algumas em *hardware* e outras em *software* em processadores de uso geral. Os projetos em *hardware* oferecem alto desempenho, pois:

- São customizados para o problema em particular;
- São muito rápidos devido à execução espacial.

Já as implementações em *software*, utilizando processadores gerais (DSP, microprocessador), interpreta um determinado fluxo de dados como instruções que indicam quais operações devem realizar. Como resultado, o *software* é:

- Flexível, pois uma tarefa pode ser mudada simplesmente trocando-se o código de instruções em uma memória regravável;
- Lento devido principalmente à execução seqüencial dos comandos;
- Ineficiente, uma vez que os operadores podem ser inadequados à tarefa a ser realizada.

Em implementações espaciais, cada operador existe em um ponto diferente na área do *chip*, permitindo que a computação explore o paralelismo e alcance uma alta taxa de transferência de dados e baixa latência. Em implementações seqüenciais, um pequeno número de recursos computacionais é reutilizado ao longo do tempo, permitindo que a computação seja implementada de forma compacta (RIBEIRO, 2002).

Esforços para se obter um maior poder de processamento é a utilização de PLDs (*Programmable Logic Devices*) para a construção de *hardwares* reconfiguráveis. Estes dispositivos contêm módulo configurável que pode ser reprogramado para a aplicação a ser

executada, tornando-se possível obter um desempenho comparável aos ASICs (*Application Specific Integrated Circuits*) sem o custo e risco inerentes a esta tecnologia. Os PLDs mais empregados na construção de *hardwares* reconfiguráveis são os FPGAs (*Field Programmable Gate Arrays*), que contêm arranjos de blocos lógicos reconfiguráveis interligados por recursos de roteamento (CHAN, MOURAD, 1994).

Além do benefício do paralelismo espacial, permitindo a realização de mais operações por ciclo, destaca-se também a ótima flexibilidade dos dispositivos FPGAs, a alta capacidade lógica, a facilidade de programação pelo usuário final, que reduz o tempo de desenvolvimento para minutos, o reduzido tempo de projeto, que possibilita um rápido *time-to-market* e o baixo custo dos dispositivos e custos fixos de projeto.

Nos últimos anos, a tecnologia de dispositivos FPGAs tem evoluído significativamente, alcançando elevados níveis de densidade, altos índices de desempenho e menores custos de fabricação. Esta evolução tem tornado cada vez menor a distância entre FPGAs e CIs (circuitos integrados) customizados. Além dos avanços em capacidade, desempenho e custo, os fabricantes de FPGAs têm introduzido, no decorrer dos anos, cada vez mais recursos de reconfigurabilidade.

Em termos básicos, a computação reconfigurável combina a velocidade do *hardware* com a flexibilidade do *software*, permitindo uma eficiência muito maior do que a normalmente encontrada em processadores de uso geral.

Neste trabalho abordaremos dois algoritmos que detectam o complexo QRS. O primeiro deles é o algoritmo proposto por Pan & Tompkins (TOMPKINS, 1993), composto por um encadeamento de filtros que extrai e enfatiza o sinal de interesse e o segundo utiliza o conceito da transformada *wavelet*, que utiliza uma função base (*wavelet* mãe) para segmentar o sinal com dois mecanismos de controle de janela, a escala e a translação. O resultado da transformação são os coeficientes que representam a relação entre o sinal original e a *wavelet* mãe. Ambos algoritmos serão implementados em FPGA através do processamento seqüencial utilizando um microcontrolador como ferramenta de processamento e através do processamento paralelo, devido ao fato dos blocos serem implementados em *hardware*, tendo sido usada a linguagem VHDL (*Very High Speed Integrated Circuits Hardware Description Language*).

1.1 Objetivos

A proposta desta dissertação é o estudo de métodos de detecção do complexo QRS através de duas técnicas de processamento de sinal implementadas em dispositivos lógicos programáveis, fornecendo um comparativo da eficácia e as características destas técnicas. Este estudo considera a filtragem do sinal e a extração das características que representa a despolarização dos ventrículos. Também faz parte da proposta o estudo da relação entre implementação e desempenho dos algoritmos em *hardware*, uma vez que se faz uso de dispositivos lógicos programáveis que possibilita a implementação dos filtros em *software* utilizando a linguagem C através de microcontrolador e em linguagem VHDL, que explora o processamento paralelo. Dessa forma é possível comparar os resultados dos dois algoritmos.

1.2 Justificativa

A análise do sinal de ECG é amplamente usada para diagnosticar doenças cardíacas, que são uma das principais causas de mortalidade em países desenvolvidos e em desenvolvimento. O aumento da utilização de métodos mais rápidos de processamento do sinal de ECG vem se tornando cada vez mais importante à medida que informações clinicamente úteis podem ser obtidas a partir desses dados, especialmente em registros de longa duração, como os sistemas *Holter* que registra os sinais de ECG de pacientes em memórias *flash*. Uma importante aplicação que utiliza diretamente os resultados da identificação dos batimentos é a análise da variabilidade da frequência cardíaca, que corresponde à oscilação dos intervalos entre batimentos. A identificação das outras ondas do ECG, bem como os intervalos e segmentos, muito úteis na identificação de cardiopatias, também utiliza, na maioria das vezes, a localização do complexo QRS como referência.

O avanço da computação e da microeletrônica associado à área de Engenharia Biomédica e Processamento Digital de Sinais tornou possível o tratamento e processamento de informações biológicas através de programas de computador, possibilitando viabilizar soluções para as necessidades metodológicas existentes, tais como o estudo mais elaborado de parâmetros significativos à integridade do organismo, como as ondas que compõem o sinal de eletrocardiograma, suas morfologias, intervalos e segmentos. Dessa forma, torna-se real a idéia

de diagnóstico e/ou detecção precoce de patologias, bem como de morte súbita, gerando informações fundamentais para uma bem-sucedida atuação dos médicos especialistas em cada caso.

1.3 Organização do trabalho

O trabalho está dividido nos seguintes tópicos:

- Introdução e apresentação da estrutura do texto (Capítulo 1);
- Anatomia e a atividade elétrica do coração, o eletrocardiograma e suas diversas ondas, as técnicas de processamento de sinais eletrocardiográficos utilizadas neste trabalho e um breve histórico evolutivo dos PLDs até os FPGAs, apresentando os aspectos mais relevantes da arquitetura FPGA, arquitetura das células (blocos lógicos) e as estruturas de roteamento (Capítulo 2);
- Metodologia de desenvolvimento onde são apresentados os algoritmos para detecção do complexo QRS e as técnicas de programação em *hardware*, utilizando processamento seqüencial e paralelo (Capítulo 3);
- Resultados obtidos, comparando a eficácia dos algoritmos propostos neste trabalho e o tempo de processamento utilizando lógica seqüencial e paralela (Capítulo 4);
- As conclusões deste trabalho onde são apresentadas algumas sugestões para trabalhos futuros (Capítulo 5).
- Por fim, o fluxo de projeto utilizando FPGA é mostrado no Anexo A, a interface entre o microcomputador e o FPGA encontra-se no Apêndice A e os códigos-fonte utilizados neste trabalho no Apêndice B.

Capítulo 2

2 Fundamentos técnicos e matemáticos

Encontram-se explanados nesta seção os temas fundamentais relacionados ao escopo do trabalho desenvolvido: anatomia funcional e atividade elétrica do coração, o eletrocardiograma, processamento digital de sinais e FPGA.

2.1 Anatomia Funcional do Coração

O coração humano é um órgão muscular que possui a forma de um cone invertido pesando cerca de 0,5% do peso total de um indivíduo. Ocupa uma pequena região localizada entre a terceira e a sexta costela na porção central da cavidade torácica do corpo apoiando-se no diafragma e na parte mais baixa dos dois pulmões e possui uma inclinação, da base para o ápice, para o lado esquerdo do corpo e ligeiramente para frente (BRONZINO, 2000), conforme a Figura 2.1.

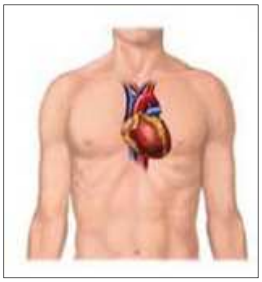

Figura 2.1: Localização do coração no corpo humano

Este órgão desempenha a função de uma bomba que ejeta sangue dentro de uma rede de vasos sanguíneos assegurando circulação por todos os órgãos do corpo humano. É formado por quatro câmaras ocas dispostas de tal forma que o divide em duas porções, superior e inferior, separadas por uma camada fibrosa. As superiores denominadas de átrios e as inferiores denominadas ventrículos possuem o aspecto conforme a Figura 2.2. Os átrios se comunicam com os ventrículos através de válvulas que deixam o sangue passar para os ventrículos na contração dos átrios, mas impedem o seu retorno.

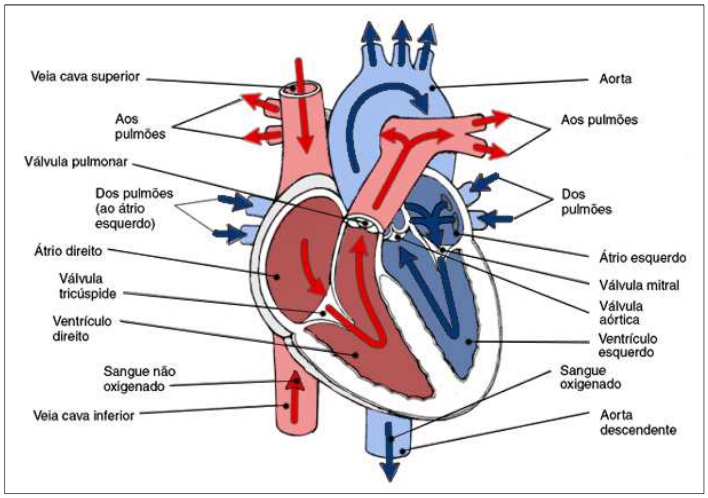

Figura 2.2: Fluxo sanguíneo no coração

Os batimentos cardíacos de um indivíduo normal e em repouso encontram-se em torno de 80 batimentos/min, ou seja, o ciclo cardíaco é de aproximadamente 754 ms (BRONZINO, 2000).

2.1.1 Atividade Elétrica

O coração tem uma capacidade de auto-excitação, ou seja, o miocárdio possui determinado tipo de célula capaz de gerar e propagar, de forma organizada, um estímulo elétrico por todas as câmaras do coração, que acarreta a contração organizada dos átrios e ventrículos e, como conseqüência, o batimento cardíaco, conforme ilustra a Figura 2.3.

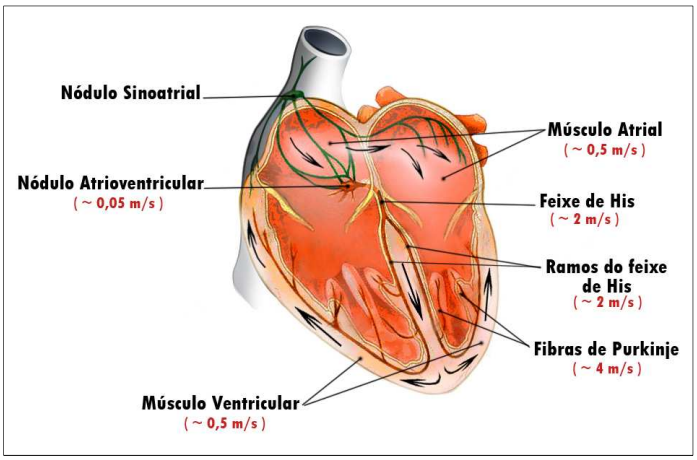

Figura 2.3: Sistema de condução elétrica cardíaca

As estruturas envolvidas neste processo são [1]:

- nódulo sinoatrial ou sinusal, no qual o impulso auto-excitador rítmico é gerado;

- vias internodais, que conduzem o impulso do nódulo sinoatrial para o nódulo atrioventricular;

- nódulo atrioventricular, no qual o impulso proveniente dos átrios é retardado antes de passar para os ventrículos;

- feixe de His, que conduz o impulso dos átrios aos ventrículos;

- feixes esquerdo e direito das fibras de Purkinje, que conduzem o impulso a todas as partes dos ventrículos (MOHRMAN, HELLER, 2006).

2.1.2 Eletrofisiologia Celular

Considere a representação de uma célula viva em repouso conforme a Figura 2.4. Existe uma diferença de potencial entre as regiões interna e externa da célula devido às diferentes concentrações iônicas estabelecidas de acordo com a permeabilidade seletiva da membrana. A principal fonte do potencial transmembrana é a distribuição desigual dos íons inorgânicos, sódio (Na^+) e potássio (K^+) entre os lados da membrana. A concentração de sódio predomina no lado externo da membrana, enquanto que no meio intracelular a concentração de potássio é predominante. O exterior da membrana celular possui mais cargas elétricas positivas

(positividade relativa) do que a região interna correspondente. Esta diferença de potencial denomina-se potencial de repouso.

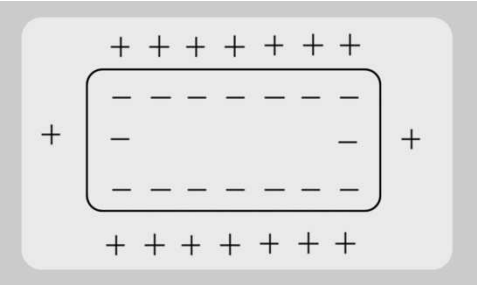

Figura 2.4: Célula em repouso (polarizada)

Quando a célula miocárdica em repouso é excitada, produzem-se fluxos iônicos através de sua membrana, que ao modificar as relações de concentrações iônicas existentes, alteram a diferença de potencial transmembrana. O registro destas variações é a curva denominada de potencial de ação, que registra o potencial intracelular em relação ao potencial neutro e que pode ser dividida em cinco fases, conforme a Figura 2.5.

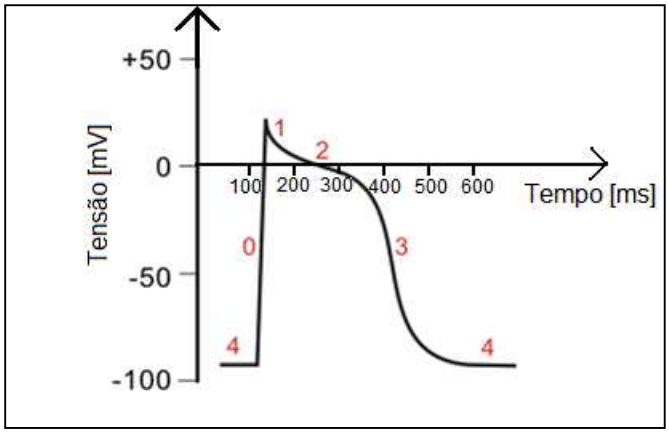

Figura 2.5: Potencial de ação

Fase 0: Rápida despolarização.

Fase 1: Pequena rápida repolarização.

Fase 2: Platô prolongado.

Fase 3: Rápida repolarização.

Fase 4: Célula polarizada.

Imagine a mesma célula representada pela Figura 2.4. Quando esta sofre um estímulo, inicia-se a despolarização e logo em seguida a repolarização. A Figura 2.6 retrata a variação do potencial no lado externo da membrana em vários pontos em relação a um potencial neutro. Observe na figura que o estímulo "S" é aplicado no lado esquerdo da célula e a despolarização inicia-se no estímulo e percorre a célula da esquerda para a direita, sendo que na repolarização ocorre o inverso. Em cada uma das três representações, o eletrodo "E" é colocado em posições diferentes da célula e, conseqüentemente, vê-se uma mudança de sinal no potencial (LENGYEL 1974).

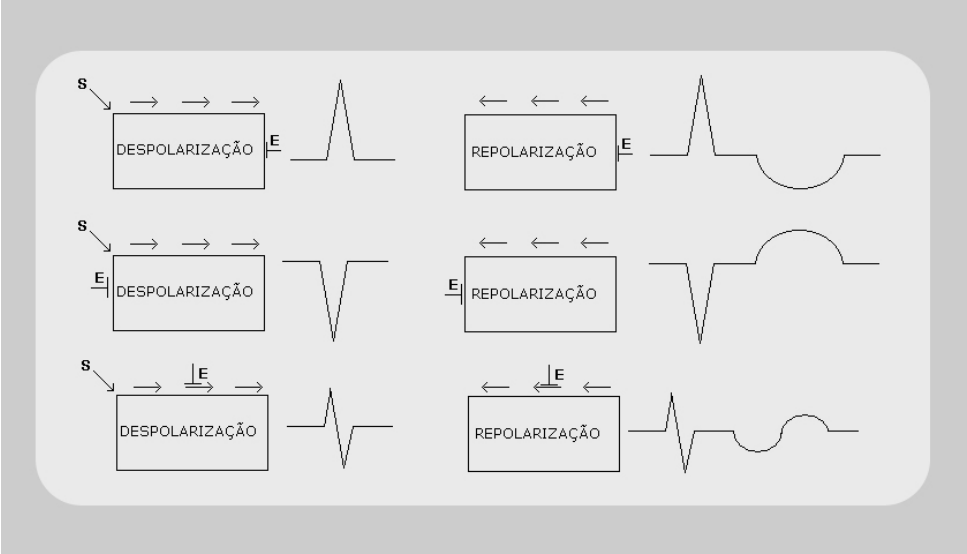

Figura 2.6: "S" representa o local do estímulo; "E" assinala a localização do eletrodo

2.1.3 Excitação Natural do Coração

Conforme citado em (2.1.1), o estímulo elétrico originado no nódulo sinoatrial se propaga pelos dois átrios causando a contração dos mesmos e, conseqüentemente, o bombeamento do sangue para os ventrículos. Entre os átrios e os ventrículos, existe uma estrutura fibrosa que apresenta baixa condutividade elétrica, impedindo a passagem dos impulsos elétricos dos átrios para os ventrículos, a não ser através de um grupo de células conhecido como nódulo atrioventricular.

As células do nódulo atrioventricular apresentam uma baixa velocidade de condução, permitindo que a contração dos ventrículos ocorra apenas após a ejeção do sangue contido nos átrios. A extremidade inferior do nódulo atrioventricular se prolonga pelo feixe de His até a extremidade inferior dos ventrículos por meio das fibras de Purkinje. Logo após, o potencial elétrico é conduzido para o miocárdio ventricular causando a contração dos ventrículos.

Cada estrutura envolvida na condução gera potenciais de ação que acarretam na forma de onda gerada em cada ciclo cardíaco, conhecidas como complexo P-QRS-TU, conforme ilustrado na Figura 2.7.

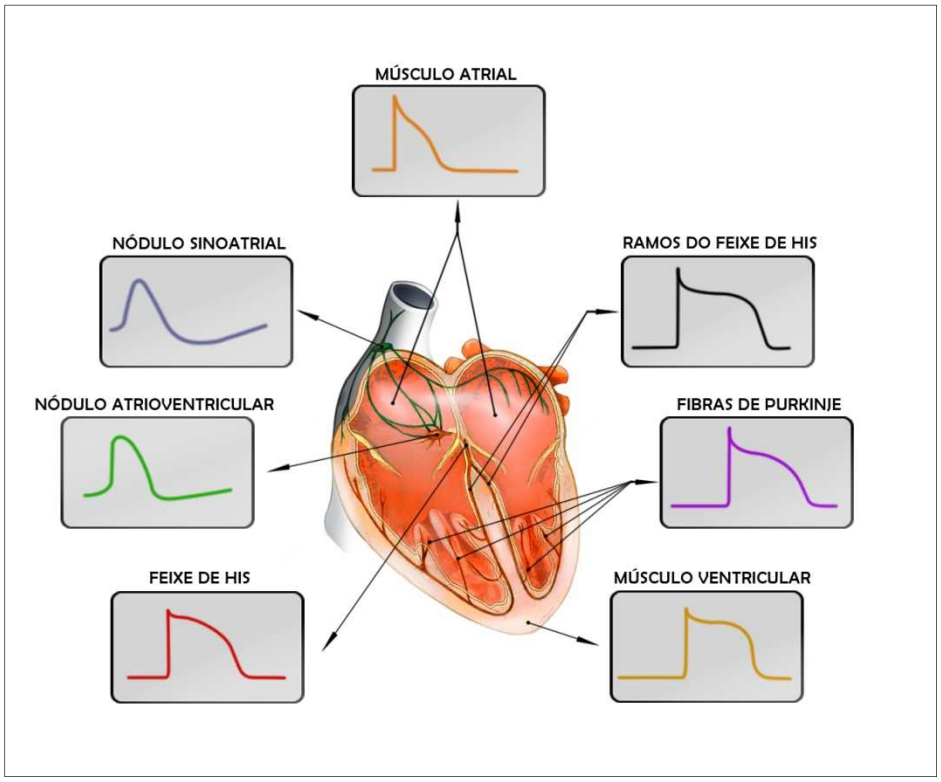

Figura 2.7: Potenciais de ação das diversas estruturas do coração

Cada potencial de ação contribui para a formação do sinal de eletrocardiograma, sendo que a maior contribuição se dá pelos músculos atriais e ventriculares por representar a maior parte do tecido do coração. A Figura 2.8 representa o sinal de eletrocardiograma destacando as cinco ondas características provenientes dos diversos potenciais de ação.

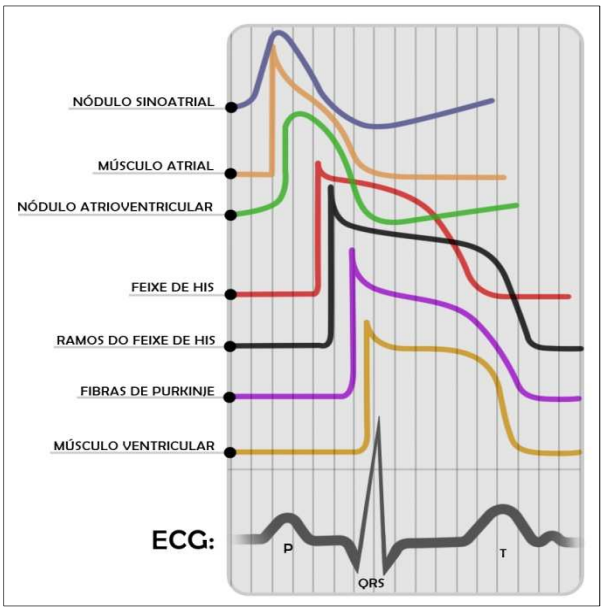

Figura 2.8: Contribuição de cada onda para a formação do ECG

2.2 O Eletrocardiograma

O ECG é o registro das diferenças de potencial elétrico entre diferentes pontos do corpo, geradas durante a atividade cardíaca e está relacionada à ação do músculo cardíaco. À medida que a excitação percorre o coração, correntes elétricas não apenas fluem pelo coração, mas também pelos tecidos circunvizinhos e uma fração desta corrente atinge a superfície do corpo. O fluxo de correntes acarreta uma diferença de potencial entre diferentes sítios do corpo. Esta diferença pode ser medida por meio de eletrodos aplicados à pele, em localizações pré-definidas e expressa o denominado eletrocardiograma, que pode ser definido como o registro gráfico da atividade elétrica do coração captada ao longo do tempo na superfície corporal.

A seqüência de eventos repetitivos de contração (despolarização) e relaxamento (repolarização) dos músculos dos átrios e ventrículos, durante o ciclo cardíaco, gera eventos característicos. Tais eventos são identificados no ECG e denominados ondas P, Q, R, S e T, conforme ilustra a Figura 2.8, além de intervalos e segmentos derivados das mesmas.

Onda P

A onda P representa a despolarização atrial, contendo as seguintes características:

Duração: representa o tempo de excitação total dos átrios e varia com a idade do indivíduo e com a frequência cardíaca. A duração da onda P tende a ser maior quanto maior for a idade, variando de 0,06 a 0,09s nas crianças e 0,08 a 0,11s nos adultos e é tanto menor quanto maior é a frequência cardíaca (LENGYEL, 1974).

Morfologia: A onda P normal é arredondada, monofásica e de inscrição lenta. Nos indivíduos de pouca idade e nos aumentos de frequência cardíaca, podemos observar ondas P pontiagudas, mas com tensão ainda dentro dos limites da normalidade. Nos recém-nascidos a onda P, freqüentemente pontiaguda, tem como característica normal ser assimétrica, com o ramo ascendente mais lento do que o descendente.

Amplitude: A tensão máxima da onda P em indivíduos normais é considerada entre 0,25 a 0,30 mV, medida em D2. Dependendo da orientação do vetor de ativação atrial, podemos ter onda P isoelétrica em uma dada derivação. Em casos de taquicardia a tensão de P aumenta, não ultrapassando, contudo, o limite máximo normal (LENGYEL, 1974).

Polaridade: As ondas P na superfície corporal podem ser positivas, negativas ou bifásicas, dependendo da orientação do vetor que representa a ativação global dos átrios. A Figura 2.9 ilustra a orientação do vetor e as ondas na superfície corpórea.

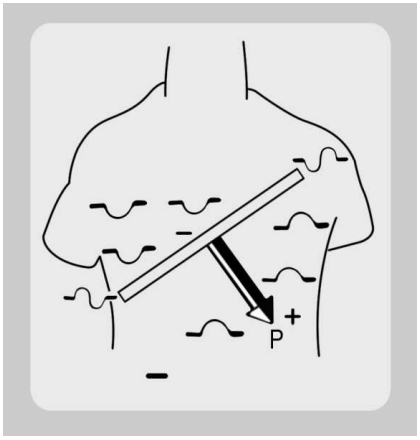

Figura 2.9: Vetor P representativo da ativação global dos átrios

A orientação média da ativação atrial normal aponta para baixo e para a esquerda, com pequena ou nenhuma inclinação para frente ou para trás. Por esta razão, as ondas P são

normalmente negativas nas regiões superiores e direitas do tórax, tanto na fase anterior como na posterior, e positivas nas regiões inferiores esquerdas. Entre estas duas zonas de positividades e negatividades, existe uma estreita faixa onde se registram ondas P bifásicas demarcando o plano zero, que é perpendicular à orientação espacial do vetor resultante P, conforme a Figura 2.9. Geralmente este plano passa ao nível de derivação precordial V1, onde são registradas, em geral, ondas P bifásicas.

Na Figura 2.10 segue a onda P destacada no eletrocardiograma.

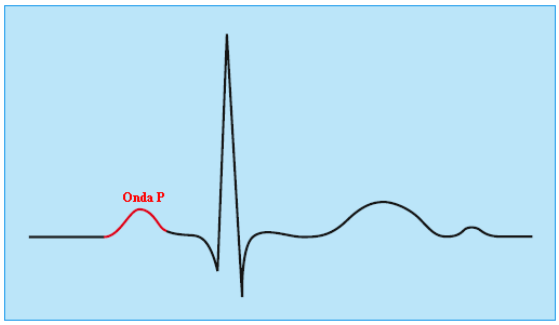

Figura 2.10: Onda P no eletrocardiograma (MORRIS, 2003)

Segmento P-R e intervalo P-R

Quando a onda de ativação acaba de excitar toda musculatura atrial (onda P), desaparecem as diferenças de potencial antes existente e, em conseqüência, o aparelho registra a linha isoelétrica. Essa linha isoelétrica acaba ao principiar a ativação ventricular, isto é, com o início da inscrição da onda Q ou, na ausência desta, da onda R. A porção da linha isoelétrica entre o fim da onda P e o início do complexo ventricular chama-se segmento P-R. Esse segmento não é medido habitualmente. Mede-se, entretanto, o intervalo P-R, que é definido pelo espaço compreendido entre o início da onda P e a inscrição da primeira onda do complexo QRS.

A duração do intervalo P-R, em adultos normais, está entre 0,12 s e 0,21 s. A duração depende da idade e da frequência cardíaca e, em certos casos, pode ser patológico. Em geral, duração aumentada significa dificuldade de condução do estímulo através do nódulo atrioventricular, ao passo que duração mais curta é sinal de ritmos ectópicos ou de condução aberrante, como na síndrome de Wolff-Parkinson-White. O segmento P-R, isto é, a porção

isoelétrica do intervalo P-R, é normalmente reto, situado na linha de base. Ele pode ser, todavia, infradesnivelado, com concavidade superior, por exemplo, nas taquicardias. A Figura 2.11 retrata no eletrocardiograma o segmento e o intervalo P-R (LENGYEL, 1974).

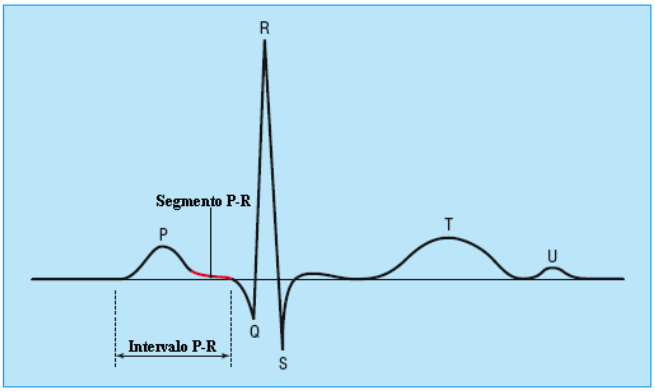

Figura 2.11: Intervalo e segmento P-R no eletrocardiograma (MORRIS, 2003)

Complexo QRS

O complexo QRS compreende a contração (despolarização) dos ventrículos, contendo as seguintes características:

Duração: O período de tempo durante o qual se inscreve o complexo QRS representa a duração total da despolarização dos ventrículos, desde o início da onda Q até o final da onda S. Da mesma maneira que para a onda P, a duração do QRS tende a ser tanto maior quanto mais idoso é o indivíduo e quanto menor é a frequência cardíaca. Entretanto a duração do QRS aumenta com a idade, mesmo sem modificação da frequência cardíaca com os incrementos da superfície corporal, ou mais diretamente com os aumentos do tamanho do coração. A duração é maior em atletas e nos indivíduos que apresentam certo retardo na porção final do complexo QRS.

Normalmente o complexo QRS se inscreve de 0,05 a 0,10 s com uma duração média de 0,07 s e um desvio padrão de 0,016 s. A duração de 0,11 s em adultos ou 0,09 s em crianças é um dado suspeito e aparece muito raramente em indivíduos normais, sendo mais freqüentemente relacionada a transtornos da condução intraventricular do estímulo ou a crescimento ventricular.

Morfologia: É extremamente variável a morfologia do complexo QRS normal nas diferentes derivações. A projeção, em uma dada derivação, dos três vetores principais do fenômeno de ativação ventricular, produz deflexões de tamanho e polaridade variáveis, segundo as orientações espaciais da linha da derivação e de cada um dos vetores. A posição anatômica do coração influi no registro do complexo QRS normal. A Figura 2.12 ilustra a morfologia do complexo QRS nas diversas derivações precordiais (MORRIS, 2003).

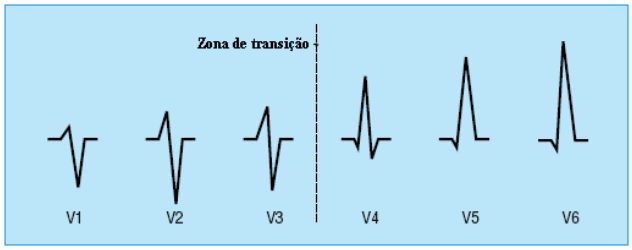

Figura 2.12: Morfologias do complexo QRS (MORRIS, 2003)

Amplitude: A tensão do complexo QRS varia muito e depende de condições cardíacas e extra cardíacas, como por exemplo, do meio condutor que rodeia o coração. Nas crianças de parede torácica delgada, a tensão é relativamente ampla. Onda R de grande amplitude (2 a 3 mV) pode ser registrada normalmente em D2, sendo que a hipertrofia ventricular acarreta, habitualmente, aumento da tensão em D1 e D3.

Polaridade: A despolarização ventricular média, dominada pela grande voltagem dos fenômenos elétricos que se processam durante a atividade elétrica do ventrículo esquerdo, orienta-se normalmente, em indivíduos adultos e normolíneos, para trás, para a esquerda e para baixo. A Figura 2.13 destaca o complexo QRS no eletrocardiograma (DALE, 2004).

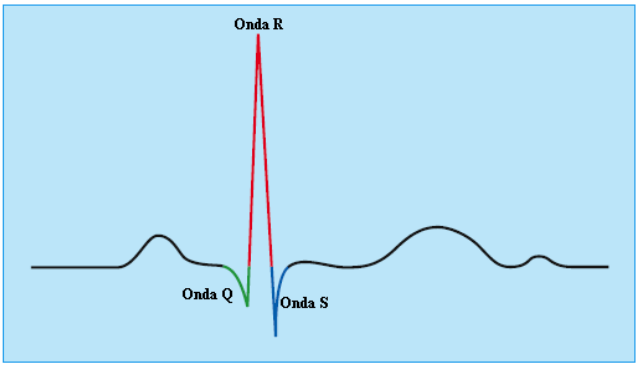

Figura 2.13: Complexo QRS no eletrocardiograma (MORRIS, 2003)

Segmento S-T

O segmento S-T se inicia no fim do complexo QRS e vai até o início da onda T. Denomina-se segmento S-T independentemente de o complexo ventricular acabar na onda R ou onda S. O início da onda T, em traçados normais, é imperceptível e por isso a medição da duração do S-T é difícil. Esse segmento representa o tempo que decorre desde o fim da despolarização até o início da repolarização da musculatura ventricular. Como durante esse tempo não há diferença de potencial a registrar, o segmento S-T está situado na linha isoelétrica. No entanto, desvios pequenos são freqüentes e, quando não ultrapassam 0,05 ou no máximo 0,1 mV, são considerados como normais, exceto em pacientes com dor precordial. Neste último caso, o desvio pode ser um sinal patológico, embora duvidoso quando permanecer dentro dos limites mencionados. A passagem do segmento S-T para a onda T, como já mencionado, é imperceptível; assim sendo, a onda T continua diretamente, elevando-se do segmento S-T. A duração do S-T depende da frequência cardíaca, variando de 0,10 s (quando a frequência é de 100 por minuto) até 0,15 s (frequência de 50 por minuto). Em derivações precordiais o S-T é muito curto ou mesmo ausente, misturando com a onda T. Isto, supostamente, é assim porque o início da repolarização gera poucas diferenças de potencial que ficam sem registro em derivações distantes do coração, mas se registram em derivações próximas, como são as precordiais. O desvio do segmento S-T deve ser comparado à linha isoelétrica, usando para este fim o intervalo T-P, entre a onda T e a próxima onda P. Este, por sua vez, precisa ser registrado sem distorções a fim de que possa ser usado para tal comparação (DALE, 2004). A Figura 2.14 ilustra no eletrocardiograma o segmento S-T e o intervalo T-P.

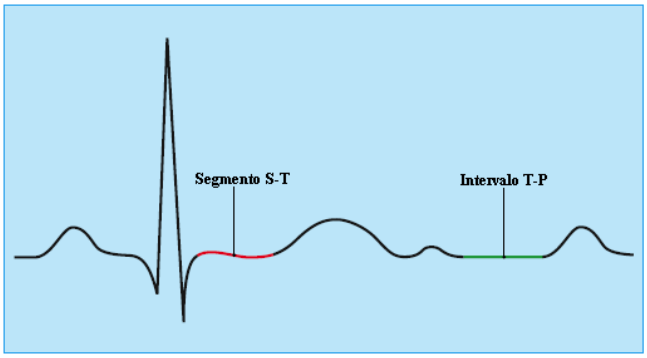

Figura 2.14: Segmento S-T e intervalo T-P no eletrocardiograma (MORRIS, 2003)

Onda T

A onda T representa a repolarização ventricular. Ela tem as seguintes características:

Morfologia: é arredondada, de inscrição lenta. O ramo ascendente é mais lento que o descendente, sendo a onda T normal, desta maneira, assimétrica.

Duração: não se mede isoladamente, estando a medida inclusa no cálculo do intervalo Q-T.

Polaridade: bastante variável, geralmente localizada em torno de +45º. Sua polaridade é semelhante a do QRS, sendo positiva nas três derivações clássicas, como também em aVF e aVL, sendo sempre negativa em aVR.

Eixo elétrico: está situado em + 45º, em média, paralelo ao plano frontal, um pouco para trás em jovens e um pouco para frente em adultos. Ângulo superior a 90º é suspeito de patologia ventricular, embora possa haver casos em que o ângulo é obtuso, sem anormalidade.

Amplitude: não se mede na prática. As amplitudes são: D1=0,2 mV, D2 = 0,3 mV e D3 = 0,1 mV. Nas precordiais, ela varia entre 0,3 e 0,8 mV. Ultrapassando a altura de 0,7 mV em derivações clássicas, 0,5 mV em derivações unipolares dos membros ou 2 mV em precordiais, denomina-se “alta tensão”. Baixa tensão significa que a T não ultrapassa 0,1 mV em nenhuma das derivações no plano frontal ou é menor que 0,2 mV em todas as precordiais.

A onda T, em geral, apresenta polaridade igual à maior onda do complexo QRS. A Figura 2.15 ilustra a onda T no eletrocardiograma (DALE, 2004).

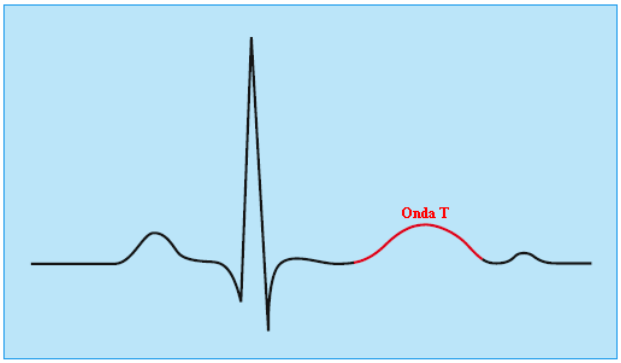

Figura 2.15: Onda T no eletrocardiograma (MORRIS, 2003)

Intervalo Q-T

Esse intervalo, no traçado, abrange o espaço que vai do início do complexo ventricular até o término da onda T. Ele representa a "sístole" elétrica, incluindo a despolarização e repolarização ventricular.

A Figura 2.16 ilustra o intervalo Q-T na derivação aVL.

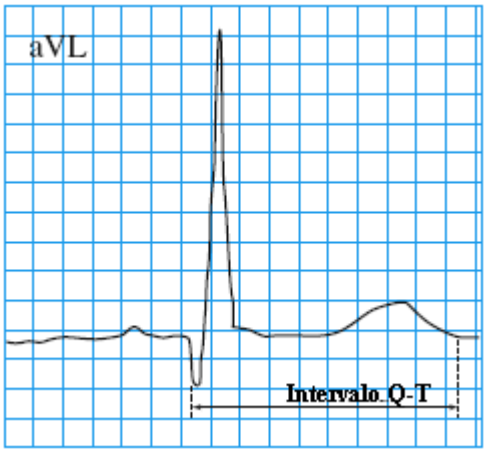

Figura 2.16: Intervalo Q-T no eletrocardiograma

Onda U

A onda U se apresenta como uma onda arredondada e larga, de baixa tensão. Nas derivações clássicas não ultrapassa 0,05 mV, sendo sua duração de aproximadamente 0,20 s. Essa onda é mais visível nas precordiais médias, onde tem amplitude maior, principalmente nos bradicárdicos. Quanto à sua polaridade, a onda U é positiva em D1, D2 e nas precordiais V2-V6. Onda U negativa nessas derivações é patológica, ocorrendo com maior frequência em hipertensão arterial, hipertrofia ventricular esquerda e no infarto.

A onda U é resultado da repolarização das células mid-miocárdicas, ou seja, aquelas localizadas entre o endocárdio e o epicárdio e o sistema His-Purkinje. Na Figura 2.17 é possível observar ondas U em pacientes com hipokalaemia (MORRIS, 2003).

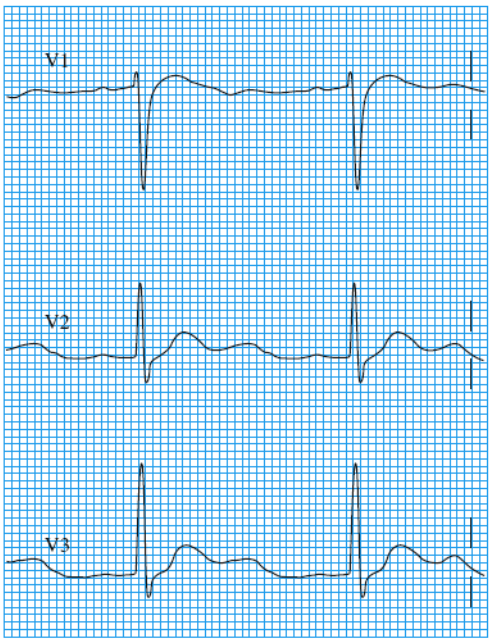

Figura 2.17: Ondas U nas derivações V1 a V3 em pacientes com hypokalaemia

2.2.1 O Eletrocardiograma Normal

O eletrocardiograma normal é composto pelas ondas P, Q, R S e T; pelos segmentos P-R e S-T; e pelos intervalos PR, ST e QT. De modo geral, refere-se às ondas Q, R e S, como sendo um único parâmetro, denominado complexo QRS. A Figura 2.18 ilustra um registro de eletrocardiograma normal, enquanto que a Tabela 2.1 descreve seus valores normais de amplitude e duração (DALE, 2004).

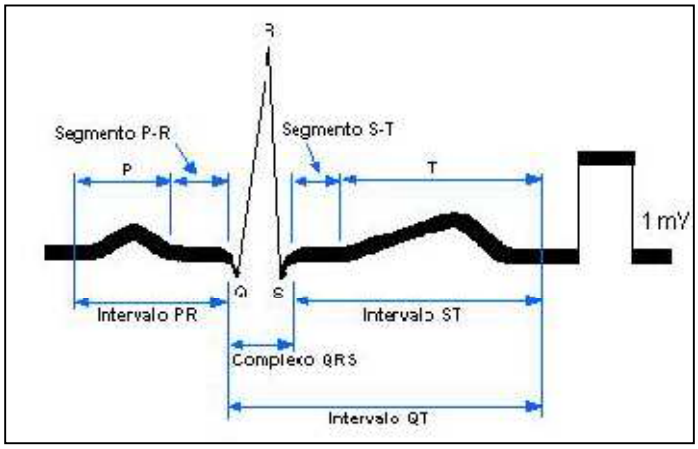

Figura 2.18: Eletrocardiograma normal, intervalos e segmentos

Tabela 2.1: Valores normais de amplitude e duração dos parâmetros do ECG

PARÂMETRO	AMPLITUDE	PARÂMETRO	DURAÇÃO
Onda P	0,25 mV	Intervalo PR	0,12 a 0,20 s
Onda R	1,60 mV	Intervalo QT	0,35 a 0,44 s
Onda Q	25% da onda R	Intervalo ST	0,05 a 0,15 s
Onda T	0,10 a 0,50 mV	Intervalo da onda P	0,11 s
		Intervalo de QRS	0,09 s

2.2.2 Derivações Eletrocardiográficas

Na superfície do corpo existem diferenças de potencial conseqüentes aos fenômenos elétricos gerados durante a excitação cardíaca. Estas diferenças podem ser medidas e registradas de acordo com o tipo e a intensidade das forças elétricas do coração. Os pontos do corpo a serem explorados são ligados ao aparelho de registro por meio de fios condutores. Dessa forma, obtêm-se as chamadas derivações que podem ser definidas de acordo com a posição dos eletrodos.

Por convenção, registram-se curvas ditas positivas (para cima da linha considerada como isoelétrica), quando um dos eletrodos, admitido como o explorador, está orientado para as áreas que se comportam como positivas em relação às que se encontram voltadas para o outro eletrodo.

A resultante dos fenômenos elétricos, em um dado momento, pode ser definida como uma grandeza vetorial. Ela pode ser representada, também convencionalmente, por um vetor cuja flecha está orientada para as regiões da superfície corporal que têm potencial positivo em relação àquelas para as quais estiver voltada para a cauda (negativa) do referido vetor.

As derivações do plano frontal

As derivações na superfície do corpo são infinitas. Foi estabelecida uma convenção para que os registros obtidos pudessem ser comparados. Einthoven estabeleceu três derivações, dispostas a formar os lados de um triângulo eqüilátero – "Triângulo de Einthoven", conforme a Figura 2.19.

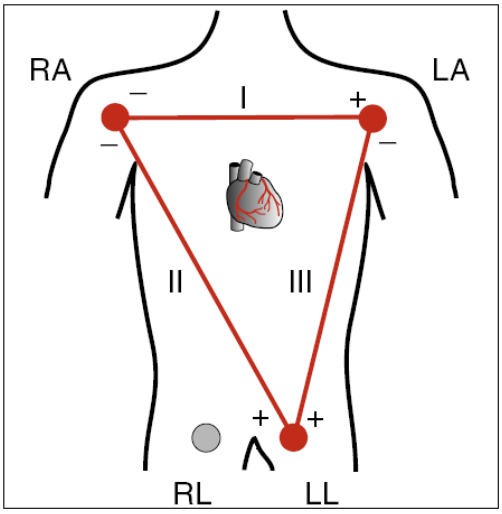

Figura 2.19: Triângulo de Einthoven

Assim, a primeira derivação (I ou D1) mede a diferença de potencial (V) entre o braço esquerdo (LA) e o braço direito (RA).

A segunda derivação (II ou D2) mede a diferença de potencial entre a perna esquerda (LL) e o braço direito.

A terceira derivação (III ou D3) mede a diferença de potencial entre a perna esquerda e o braço esquerdo.

As derivações bipolares dos membros são também chamadas “derivações clássicas” ou “*standards*”.

O centro do triângulo eqüilátero corresponde ao centro elétrico do coração ou centro aparente de origem dos vetores.

Outro grupo de derivações, denominadas “derivações unipolares dos membros” ou “de Goldberger” são designadas pelas seguintes abreviações: aV_R, aV_L, aV_F, correspondendo ao braço direito, esquerdo e perna esquerda, respectivamente, conforme a Figura 2.20.

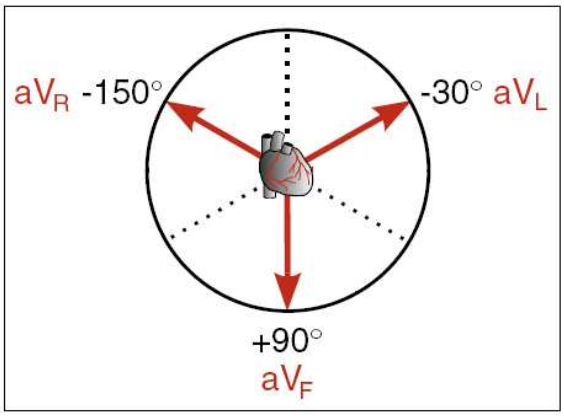

Figura 2.20: Derivações unipolares dos membros

A derivação aV_R usa o braço direito como positivo e os outros dois eletrodos (braço esquerdo e perna esquerda) são ligados no terminal negativo.

A derivação aV_L usa o braço esquerdo como positivo e os outros dois eletrodos referenciados como negativo.

Por sua vez, a derivação aV_F utiliza o eletrodo da perna esquerda como positivo e os outros dois eletrodos referenciados como negativo.

As derivações do plano horizontal

O coração é um órgão tridimensional e por isso os vetores cardíacos têm uma orientação espacial. Precisamos de dois planos perpendiculares para orientar um vetor no espaço. Assim, para localizar um vetor no espaço, basta conhecer as projeções deste vetor nos planos frontal e horizontal.

Além das seis derivações já descritas, são registradas na prática outras seis derivações unipolares. Elas exploram o fenômeno elétrico a partir da face anterior do tórax e são denominadas de “precordiais”. Os seus pontos de exploração são vistos na Figura 2.21.

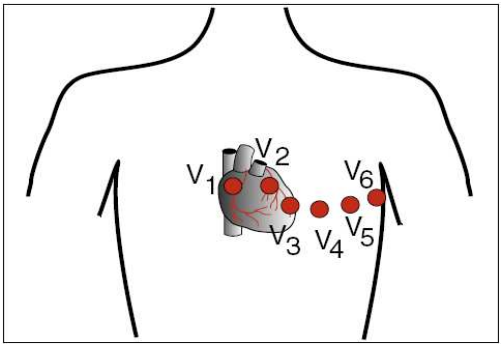

Figura 2.21: Derivações precordiais

As derivações precordiais (V_1 a V_6) são consideradas como derivações do plano horizontal. No entanto elas não estão situadas no mesmo plano. V_1 e V_2 estão no plano horizontal que passa pelo quarto espaço intercostal; V_3 situa-se no plano horizontal intermediário entre o quarto espaço e o plano que passa pelo quinto (onde estão localizadas as derivações V_4, V_5 e V_6). Cada derivação precordial "vê" o fenômeno elétrico cardíaco de um determinado ângulo do espaço, resultando uma inscrição diferente segundo a derivação, conforme ilustra a Figura 2.22 (MORRIS, 2003).

A Tabela 2.2 descreve todas as 12 derivações eletrocardiográficas citadas acima.

Tabela 2.2: As doze derivações eletrocardiográficas

AUTOR	DESCRIÇÃO	DERIVAÇÃO
Einthoven	Bipolares	I = LA – RA
		II = LL – RA
		III = LL – LA
Frank Wilson	Unipolares Precordiais	$V_1 = v_1 - (RA+LA+LL) / 3$
		$V_2 = v_2 - (RA+LA+LL) / 3$
		$V_3 = v_3 - (RA+LA+LL) / 3$
		$V_4 = v_4 - (RA+LA+LL) / 3$
		$V_5 = v_5 - (RA+LA+LL) / 3$
		$V_6 = v_6 - (RA+LA+LL) / 3$
Emanuel Goldberg	Unipolares Aumentadas	$aV_R = RA – ½(LA+LL)$
		$aV_L = LA – ½(LL+RA)$
		$aV_F = LL – ½(LA+RA)$

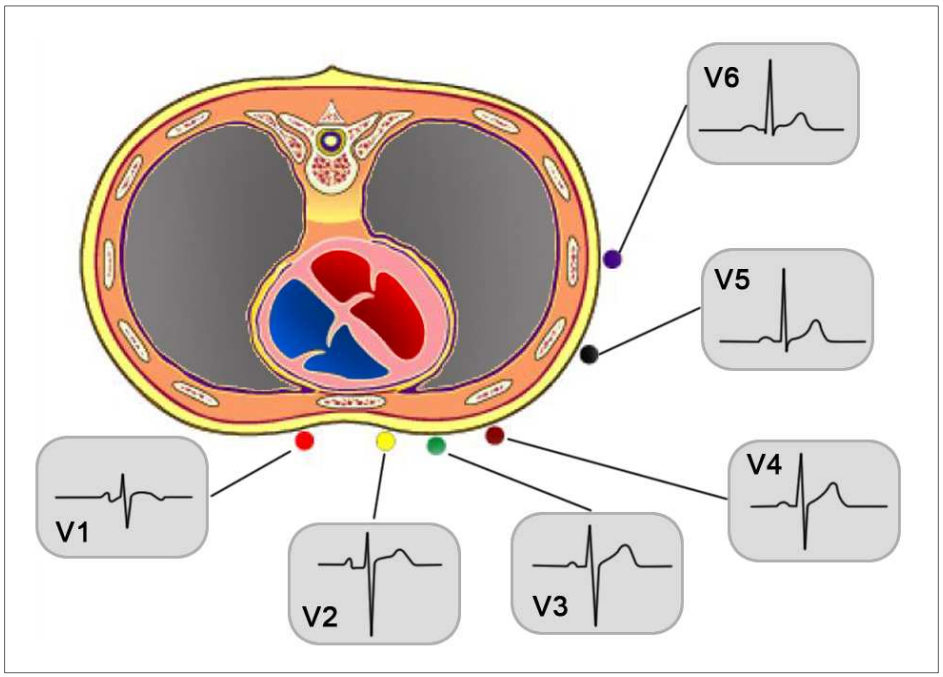

Figura 2.22: Inscrição das ondas nas diversas derivações precordiais

2.3 Métodos matemáticos sobre o sinal de ECG

Existem vários algoritmos desenvolvidos e em desenvolvimento para detectar o complexo QRS da curva de ECG. Isso ocorre devido ao fato de ainda não existir um algoritmo que detecte a onda R com 100% de eficácia. As ondas de ECG apresentam uma infinidade de morfologias, mudando de pessoa para pessoa, da situação (enfermidade) da pessoa analisada, de quanto tempo ela tem sido analisada (isto afeta a eficiência dos eletrodos), de ruído provindo do contato do eletrodo com a pele, contração muscular de ruído provindo da rede elétrica, de variação da impedância do eletrodo (variação da linha de base do ECG e queda da relação sinal/ruído), modulação da amplitude do ECG de acordo com a respiração e ruído de instrumentos elétrico-cirúrgicos.

A Figura 2.23 ilustra o espectro de potência relativo do sinal de ECG, complexo QRS, ondas P e T, atrito do eletrodo com a pele e ruídos musculares. Observa-se que o complexo QRS apresenta um maior espectro de potência relativo para frequências em torno de 10 Hz (TOMPKINS, 1993).

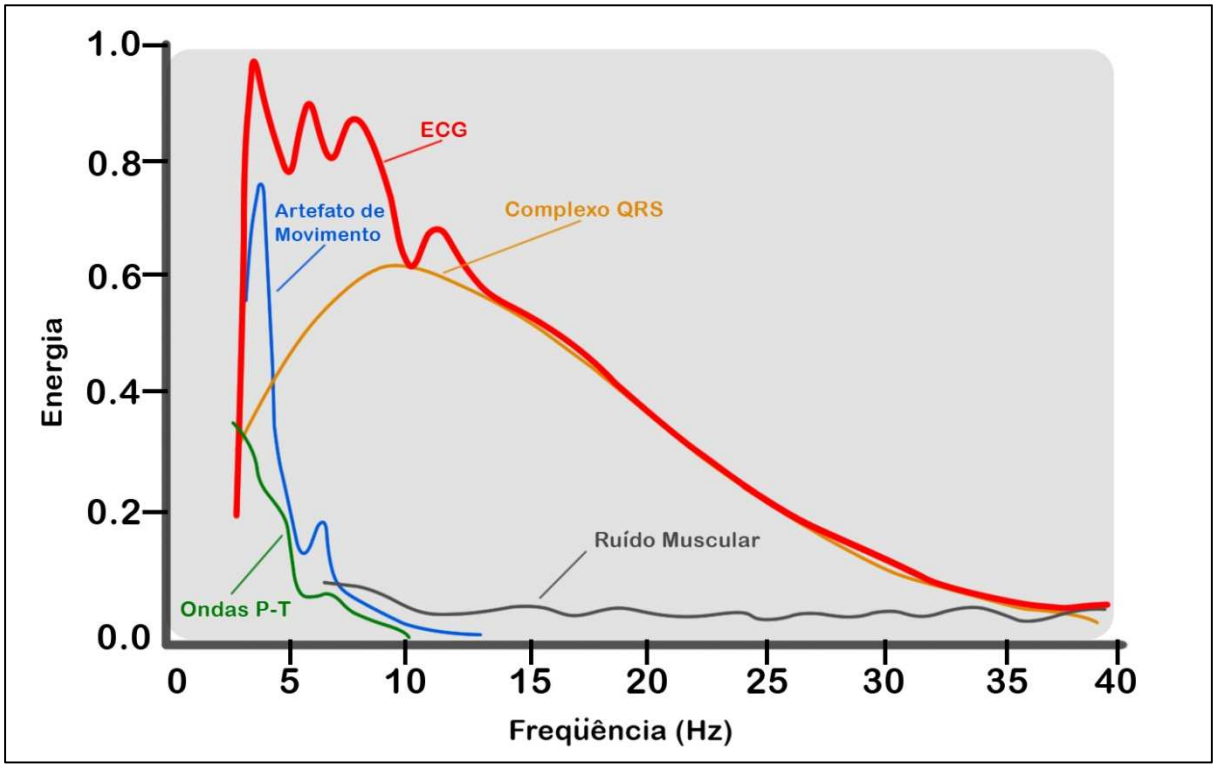

Figura 2.23: Espectro de potência relativo do complexo QRS, ondas P e T, atrito do eletrodo e ruído muscular

Conforme descrito na introdução deste trabalho, pretende-se analisar algoritmos que gerem parâmetros capazes de informar a ocorrência de complexos QRS em sinais de eletrocardiograma. Para isso é necessário o uso de ferramentas matemáticas mais robustas, como por exemplo, as transformadas, cuja principal função é possibilitar análises do sinal (série temporal) no domínio da frequência.

Dentre as diversas técnicas disponíveis, abordaremos algumas delas nos tópicos abaixo.

2.3.1 Transformada Janelada de Fourier

Dada uma função $x(t)$ periódica, com período T_0, esta função pode ser representada por uma série trigonométrica da forma:

$$x(t) = \frac{1}{2}a_0 + \sum_{n=-\infty}^{\infty}(a_n \cos(n\omega_0 t) + b_n sen(n\omega_0 t)) \qquad (2.1)$$

onde $\omega_0 = 2\pi/T_0$ e os coeficientes a_n e b_n são as constantes que definem a importância (amplitude) de cada frequência do sinal analisado. Esta série tem a propriedade de se repetir para todo ciclo

2π. É baseado nestas duas propriedades (análise por frequência e capacidade de repetição a cada batimento) que a série de Fourier parece um meio interessante de interpretar cada ciclo do batimento cardíaco e analisar as diferenças que ocorrem no ECG entre batimentos. Com a descrição do batimento cardíaco em série de Fourier, pode-se determinar as componentes no espectro de frequências do sinal de ECG. Os parâmetros a_n e b_n para uma função contínua e periódica são definidos como:

$$a_n = \frac{2}{T}\int_{-T_0/2}^{T_0/2} x(t)\cos(n\omega_0 t)dt \quad (\text{n} = 0, 1, 2, ...) \tag{2.2}$$

$$b_n = \frac{2}{T}\int_{-T_0/2}^{T_0/2} x(t)sen(n\omega_0 t)dt \quad (\text{n} = 1, 2, ...) \tag{2.3}$$

Outra forma, mais concisa, de se declarar a série de Fourier é através da relação exponencial $e^{jn\omega_0 t}$, $j = \sqrt{-1}$, de forma que a função dos valores da curva analisada fica:

$$x(t) = \sum_{n=-\infty}^{+\infty}(c_n e^{jn\omega_0 t}) \tag{2.4}$$

onde o parâmetro c_n é definido como:

$$c_n = \frac{1}{T}\int_{-T_0/2}^{T_0/2} x(t)e^{-jn\omega_0 t}dt, \quad \text{n}=0, \pm1, \pm2, ... \tag{2.5}$$

Como lidamos com sinal amostrado, não podemos utilizar a forma contínua da série de Fourier. Dado um sinal no tempo discreto *x(nT)* de duração finita, o sinal periódico $x_p(nT)$ com período *NT*, onde *N* é um inteiro constante, representando o número total de amostras, pode ser formado como:

$$x_p(n) = \sum_{r=-\infty}^{\infty} x(n + rN) \tag{2.6}$$

onde omitiu-se o intervalo de tempo *T*, que corresponde ao período de amostragem do sinal, que será adotado no restante do texto por simplicidade.

A Transformada de Fourier Discreta de $x_p(n)$ é definida por:

$$X_p(k\Omega) = \sum_{n=0}^{N-1} x_p(n) e^{-kj2\pi n/N} \qquad (2.7)$$

onde a resolução na frequência é dada por $\Omega = \omega_s/N$ e $\omega_s = 2\pi/T$, e será também omitida no restante do texto.

Entretanto este método de análise não traz bons resultados para a análise de eventos não repetitivos ou evolutivos nos sinais. Para possibilitar tais análises, utilizando uma técnica de janelamento do sinal no tempo, a transformada discreta de Fourier permite observar uma pequena secção do sinal no tempo. A idéia central do algoritmo é que, ao se ajustar a largura da janela, pode-se detectar as componentes de frequência nela contidas, uma vez que o sinal $x_p(n)$ é periódico.

A transformada janelada de Fourier, ou STFT, contínua é definida pela Equação (2.8),

$$STFT(t, \omega) = \int_{-\infty}^{\infty} x(\tau - t) g(\tau) e^{-j\omega\tau} d\tau \qquad (2.8)$$

onde $x(t)$ é o sinal contínuo e $g(t)$ é a função de janelamento da STFT.

Para se obter a forma discreta, ou DSTFT, utiliza-se a DFT, conforme a Equação (2.7), aplicada à STFT contínua, como mostrado na Equação (2.9).

$$DSTFT(m, k) = \sum_{n=0}^{L-1} x_p(n - m) g(n) e^{-j2\pi nk/L} \qquad 0 \leq m \leq N\text{-}1 \qquad (2.9)$$

onde a função janela possui L pontos e, portanto, $\Omega = \frac{2\pi}{LT}$.

A DSTFT divide o plano tempo-frequência em porções de dimensões regulares e constantes, apresentando para as frequências altas a mesma resolução que para as frequências baixas. A Figura 2.24 ilustra tal resultado para análise no plano tempo-frequência para a DSTFT, onde a amplitude de cada raia pode corresponder a uma cor ou a uma visão em perspectiva. Neste caso, a amplitude estaria representada em um eixo saindo do plano do papel.

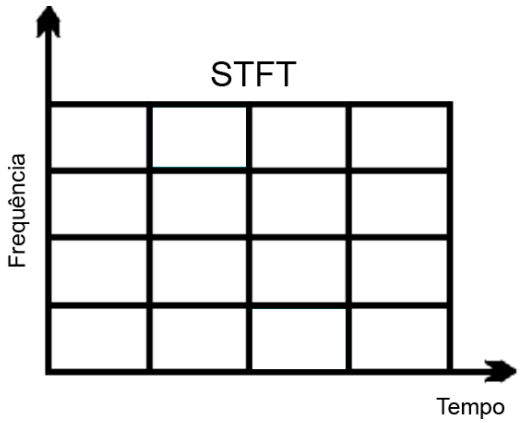

Figura 2.24: STFT no plano tempo-frequência

2.3.2 Transformada *Wavelet*

Wavelets são funções obtidas a partir de uma função protótipo – a *wavelet* mãe – $\Psi(t) \in L^2(R)$, por meio de dilatações (ou contrações no tempo, ditas escalamento) e translações (deslocamentos) da *wavelet* mãe.

A Transformada *Wavelet* é uma alternativa para o uso da Transformada Discreta de Fourier, quando há necessidade da análise tempo-frequência. Diferentemente da DSTFT, a resolução não é constante, gerando um diagrama conforme ilustrado na Figura 2.25. Ou seja, quanto menor o tamanho da janela de tempo utilizada, maior é a resolução no domínio da frequência, fazendo com que para análise de baixas frequências a resolução seja melhor do que para altas frequências. Notar que o eixo que indica a escala corresponde à frequência.

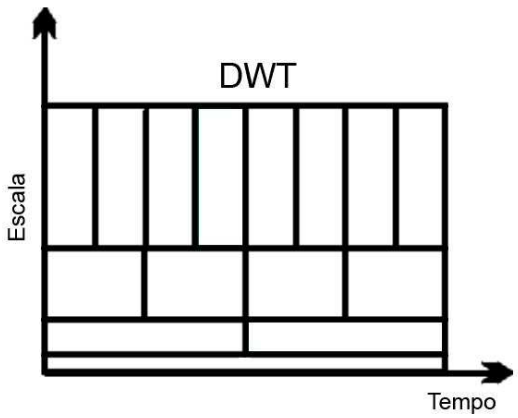

Figura 2.25: Resolução tempo-frequência da Transformada *Wavelet*

Da mesma forma que a Transformada Discreta de Fourier, a análise por *wavelets* procura estabelecer o espectro de frequências de um sinal ao longo do tempo. As diferenças principais entre os dois processos são:

1. Não é tomada a Transformada de Fourier da janela de tempo analisada;
2. A largura da janela é alterada para cada componente no espectro de frequências analisado.

Através da Figura 2.26, tem-se uma idéia de como o sinal é analisado, onde se observa um sinal característico utilizado como base para verificação das componentes de frequência do sinal analisado. Quanto maior for a componente de frequência procurada em um sinal, menor será a escala deste sinal característico utilizado para verificação e menor a janela de tempo necessária para a comparação. Desta forma, através de várias interações discretas (em janelas de tempo específicas), compara-se o sinal característico (em uma determinada escala) com o sinal de análise para o cálculo dos coeficientes de decomposição. Esses pesos informam se as componentes de frequência do sinal característico na escala analisada são ou não encontradas no sinal observado.

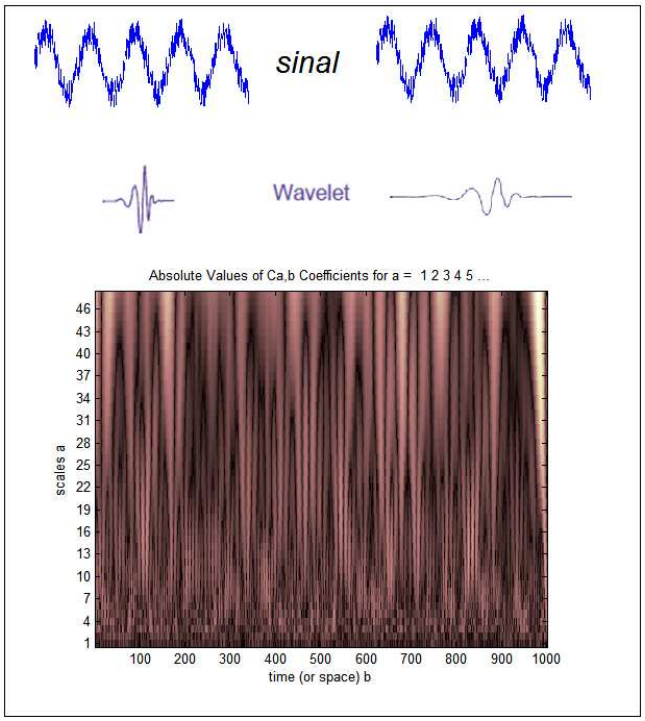

Figura 2.26: Convolução da *wavelet* com o sinal

Transformada *Wavelet* Contínua

A Transformada *Wavelet* Contínua (CWT) é definida a partir de um espaço de funções ortonormais[1], denominado baixas *wavelets*, as quais formam uma base de funções da Transformada *Wavelet*. Assim, a Transformada *Wavelet* de um sinal de tempo contínuo, *x(t)*, é definida como:

$$T(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{+\infty} x(t) \Psi^* \left(\frac{t-b}{a}\right) dt \quad \text{a} \neq 0 \tag{2.10}$$

onde $\Psi^*(t)$ é o complexo conjugado da função *wavelet* $\Psi(t)$. A constante $\frac{1}{\sqrt{|a|}}$ é usada para normalização da energia da função *wavelet* em diferentes escalas, *x(t)* é o sinal investigado, *a* é o fator de escala e *b* é o fator de translação. A função *wavelet* deve satisfazer certos critérios matemáticos, que são:

(1) Deve possuir energia finita:

$$E = \int_{-\infty}^{+\infty} |\Psi(t)|^2 dt < \infty \tag{2.11}$$

(2) Se $\widehat{\Psi}(f)$ é a Transformada de Fourier de $\Psi(t)$, i.e.

$$\widehat{\Psi}(f) = \int_{-\infty}^{+\infty} \Psi(t) e^{-j(2\pi f)t} dt \tag{2.12}$$

então a seguinte condição deve ser satisfeita:

$$C_g = \int_0^{\infty} \frac{|\widehat{\Psi}(f)|^2}{f} df < \infty \tag{2.13}$$

Isto implica que a *wavelet* não tem componente de frequência zero, i.e. $\widehat{\Psi}(0) = 0$, ou seja, deve possuir média zero. A Equação (2.13) é conhecida como condição de admissibilidade e C_g é chamado de constante de admissibilidade. O valor de C_g depende da *wavelet* escolhida.

A função $\Psi(t)$ é chamada de *wavelet* básica ou *wavelet* mãe. Quando os termos *a* e *b* variam de forma apropriada, estas formam a base de funções para $L^2(R)$ da transformada. Note

[1] Da algebra linear, funções ortonormais são funções ortogonais entre si com módulo igual a um.

então que a resposta da *CWT* será proporcional ao grau de similaridade entre o sinal de entrada e a família de funções definida pela *wavelet* mãe $\Psi(t)$.

As *wavelets* $\Psi_{a,b}(t)$ são versões escalonadas pelo parâmetro "*a*" e transladadas pelo parâmetro "*b*" da função básica $\Psi(t)$ ou *wavelet* mãe.

A Transformada *Wavelet* Contínua Inversa é definida como:

$$x(t) = \frac{1}{C_g}\int_{-\infty}^{\infty}\int_{0}^{\infty} T(a,b)\Psi_{a,b}(t)\frac{da\ db}{a^2} \qquad (2.14)$$

A recuperação de $x(t)$ de $T(a,b)$ só é possível se a constante C_g acima representada for finita, donde deriva a ***condição de admissibilidade*** expressa na Equação (2.13).

As *wavelets* mãe podem assumir diversas formas (db2, symlet, morlet, cubic-spline, Haar, etc) e sua escolha é um fator importante.

A Figura 2.27 ilustra um trecho de um sinal de eletrocardiograma, representando de forma clara as ondas P, T e o complexo QRS. A utilização da primeira derivada da função gaussiana (Equação (2.15)) como *wavelet* mãe, para a análise do sinal, resulta no gráfico da Figura 2.28, onde se observam claramente os coeficientes de mais alta ordem representando o complexo QRS.

$$G' = -2.x.e^{(-x)^2} \qquad (2.15)$$

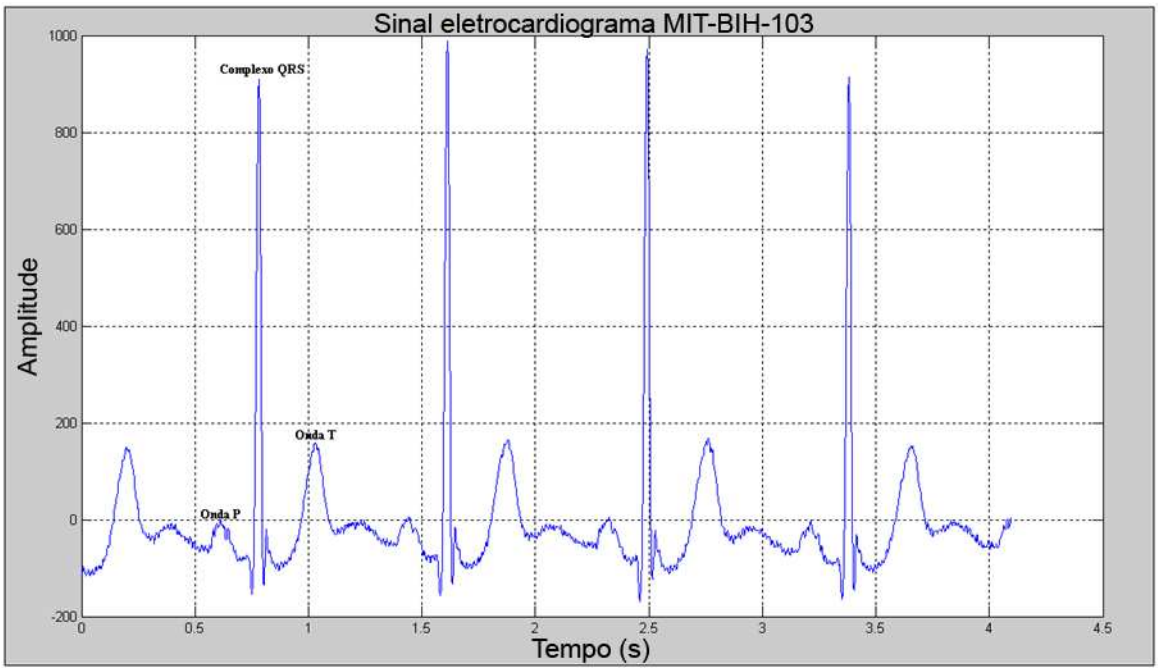

Figura 2.27: Trecho do sinal 103 da base de dados MIT-BIH

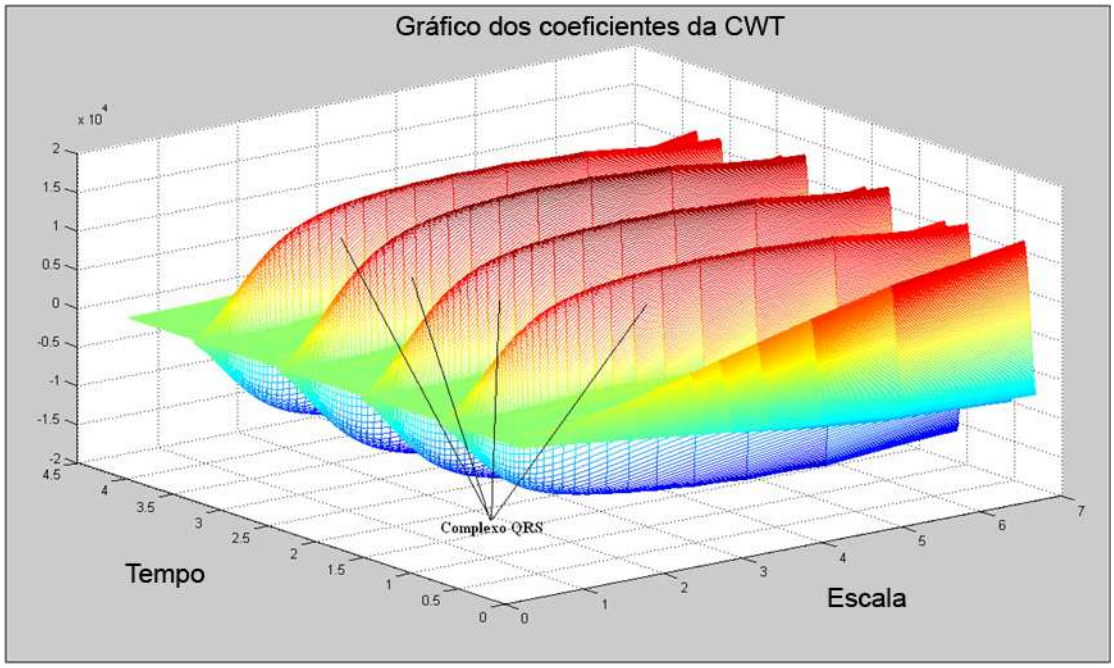

Figura 2.28: Coeficientes da CWT para diferentes valores de tempo e escala

Transformada *Wavelet* Discreta (DWT)

A CWT utiliza os parâmetros de variação do tempo e escala contínuos. No entanto, na prática, para o cálculo da WT, estes parâmetros são discretizados.

As *wavelets* são geradas de uma função Ψ, chamada *wavelet* mãe, através de dilatações (ou compressões) e translações em passos discretos, conforme a Equação (2.16).

$$\Psi_{m,n}(t) = \frac{1}{\sqrt{a_0^m}} \Psi\left(\frac{t - n b_0 a_0^m}{a_0^m}\right) \qquad (2.16)$$

onde os inteiros m e n controlam a dilatação e translação da *wavelet*, respectivamente; $a_0 > 1$ é especificado como o parâmetro da dilatação (ou compressão) e $b_0 > 0$ é o parâmetro da localização.

No caso discreto, surge a questão sobre a possibilidade de se representar $x(t)$ em termos de $T(a,b)$ e de se recuperar tais coeficientes. Isto é possível desde que satisfeitas algumas condições sobre o suporte e a regularidade da *wavelet*, e atendidos alguns requisitos matemáticos. Essencialmente, a condição de admissibilidade permanece válida. Existem formas diferentes de se trabalhar com *wavelets* discretas e de se implementar a transformada discreta. Uma delas é através da utilização de banco de filtros organizados num esquema piramidal, que levará também a uma representação em multiresolução do sinal.

O fator a_0 não pode ser arbitrário. Diferentes valores de a_0 levam a *wavelets* diferentes e bases ortonormais de *wavelets* só são conhecidas para valores racionais de a_0. Fazendo $a_0 = 2$ e $b_0 = 1$, a Equação (2.16) fica:

$$\Psi_{m,n}(t) = 2^{-m/2} \Psi(2^{-m} t - n) \qquad (2.17)$$

A *wavelet* da Equação (2.17) é conhecida por *wavelet* diádica. O plano tempo-escala (frequência) neste caso fica amostrado por uma ***grade diádica***, conforme a Figura 2.29.

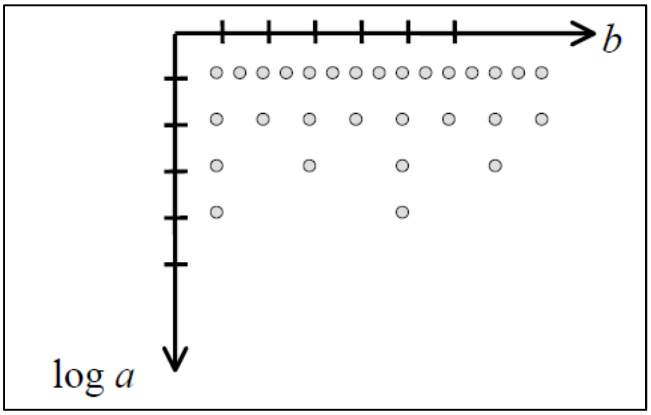

Figura 2.29: Discretização do plano tempo - escala

Wavelets diádicas constituem bases ortonormais e permitem a caracterização de um sinal $x(t)$ sem redundância. A prova deste fato pode ser desenvolvida utilizando-se a análise em multiresolução como ferramenta, mostrando que qualquer função $x(t)$ pode ser aproximada com uma precisão arbitrária por combinações lineares de *wavelets* ortonormais.

Para computar a Transformada *Wavelet* Discreta (DWT), pode-se utilizar a aplicação da CWT com parâmetros de escala e translação discretos e a aplicação da técnica de multiresolução desenvolvida por MALLAT (1999).

A Equação (2.18) mostra o cálculo da DWT utilizando como base a CWT com parâmetros de escala e translação discretos.

$$T(m,n) = \int_{-\infty}^{\infty} x(t)\Psi_{m,n}(t)dt \qquad (2.18)$$

onde $x(t)$ é o sinal contínuo, $\Psi_{m,n}(t)$ é a função *wavelet* com os fatores inteiros de escala (m) e translações (n) discretizados, $T(m,n)$ é a Transformada *Wavelet* Discreta (coeficientes da *wavelet*).

A análise multirresolução consiste na decomposição de um sinal em funções que formam uma sequência de espaços aninhados V_m, ou seja, ... $V_{m-2} \subset V_{m-1} \subset V_m$. Isto possibilita a representação dos dados em múltiplos níveis, em termos da união dos conjuntos: $V_m = V_{m-1} \oplus W_{m-1} = V_{m-2} \oplus W_{m-2} \oplus W_{m-1} =...$, em que m representa o nível menos refinado e W_{m-1} é um conjunto ortogonal a V_{m-1}, contendo os detalhes da representação em V_m que não podem ser

representados em V_{m-1}. Este conceito tem aplicação imediata para implementar a forma discreta da transformada *wavelet*.

Com base na multiresolução, o uso de banco de filtros para o cálculo da DWT foi apresentado por MALLAT (1989), gerando uma decomposição em múltiplas bandas de frequência. A função *wavelet* $\Psi(t)$ é associada a um filtro passa-alta que produz os coeficientes de detalhes da decomposição; e uma função escala $\Phi(t)$ é associada a um filtro passa-baixa que produz os coeficientes de aproximação da decomposição da *wavelet*.

O processo de decomposição da *wavelet* por multiresolução é ilustrado na Figura 2.30. Trata-se assim de um processo de codificação em sub-bandas, onde o sinal é filtrado (filtros passa-baixa (Lo) e passa-alta (Hi)) e sofre uma dizimação, dando origem aos coeficientes *wavelet* de aproximação (cA) e detalhe (cD), cada um com metade da dimensão do sinal original.

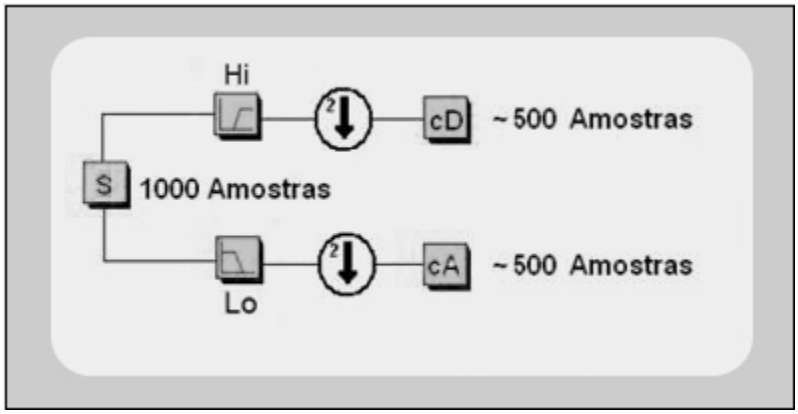

Figura 2.30: Processo de filtragem e dizimação (Lo – filtro passa-baixa; Hi – filtro passa-alta)

Na realidade este processo de decomposição do sinal em suas sub-bandas é um processo iterativo com sucessivas decomposições nos coeficientes de aproximação. Portanto, o que se obtém é uma árvore de decomposição da *wavelet* em sub-bandas que pode ser vista sendo uma estrutura de banco de filtros chamados de Filtros Espelhados de Quadratura (ou *Quadrature Mirror Filters* (QF)), conforme ilustra a Figura 2.31.

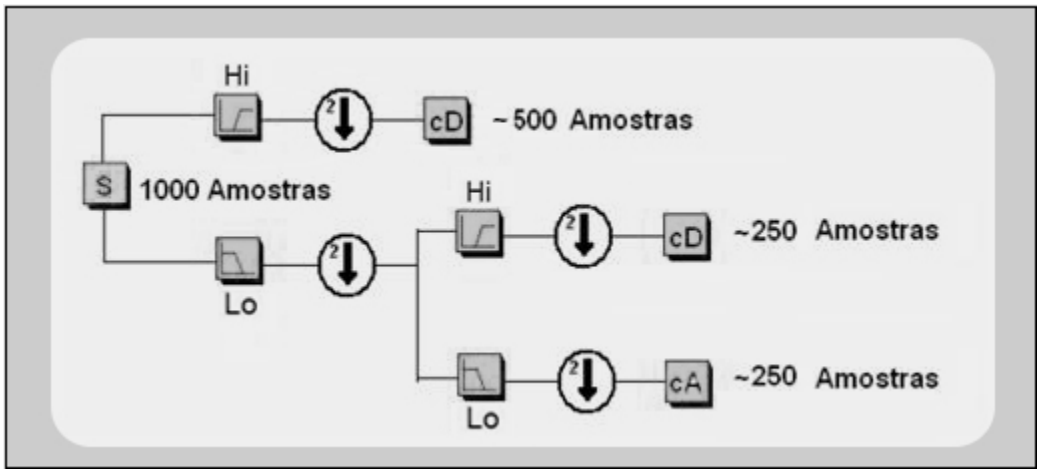

Figura 2.31: Decomposição DWT em sub-bandas

Para que esta decomposição seja possível, é necessária a aplicação de uma função de escalonamento $\Phi_{m,n}(t)$ no sinal. Esta função deve ser contínua, geralmente real e ainda satisfazer a condição de admissibilidade, ou seja:

$$\int_{-\infty}^{\infty} \Phi_{0,0}(t)dt = 1 \qquad (2.19)$$

A função escala está associada com a suavização do sinal e tem a mesma forma da função *wavelet*, dado por:

$$\Phi_{m,n}(t) = 2^{-\frac{m}{2}}\Phi(2^{-m}t - n) \qquad (2.20)$$

A função escala pode ser convoluída com o sinal para produzir coeficientes de aproximação, conforme abaixo:

$$cA_{m,n} = \int_{-\infty}^{\infty} x(t)\Phi_{m,n}(t)dt \qquad (2.21)$$

Da mesma forma, os coeficientes de detalhe são dados por:

$$cD_{m,n} = \int_{-\infty}^{\infty} x(t)\Psi_{m,n}(t)dt \qquad (2.22)$$

Desta forma, os coeficientes de aproximação e detalhe, gerados pela aplicação dos filtros descritos anteriormente, podem ser calculados como mostram as Equações (2.23) e (2.24).

$$cA_{m+1}(n) = \sum_i cA_m(i)Lo(i-2n) \quad (2.23)$$

$$cD_{m+1}(n) = \sum_i cA_m(i)Hi(i-2n) \quad (2.24)$$

onde, *x(t)* é o sinal, *Lo* e *Hi* são os filtros passa-baixa e passa-alta, respectivamente, *m* é o nível de decomposição (ou escalamento), cA_m e cD_m são os coeficientes da DWT (aproximação e detalhe) e *i* é número de elementos no nível respectivo.

Transformada *Wavelet* Estacionária ou Transformada *Wavelet* Discreta não dizimada

A operação de dizimação (*downsampling*) por um fator de 2, nas saídas dos filtros *Lo* e *Hi* do processo da AMR, representada pelo símbolo ⑵↓ conforme ilustrado na Figura 2.31, realiza o ajuste da dimensão do sinal para o processamento do próximo nível de decomposição. Deste modo, o sinal de um dado nível de decomposição tem sempre a metade do número de amostras do sinal de um nível subseqüente a este nível, o que nem sempre pode trazer uma adequada representação do sinal. Para resolver este problema, foi proposta a utilização da Transformada *Wavelet* Estacionária (TWE) ou DWT sem dizimação.

O algoritmo da TWE é simples e muito semelhante ao processo da DWT. O primeiro nível da TWE de um dado sinal pode ser obtido por um processo de filtragem do sinal através de filtros passa-baixa e passa-alta apropriados como no caso da análise multiresolução. Porém, sem a etapa de dizimação, de modo a fornecer os coeficientes de aproximação e de detalhe com o mesmo número de amostras do sinal original (Figura 2.32). Para se obter os coeficientes de detalhe e de aproximação em um nível *j* de decomposição, devem-se utilizar filtros com escalamento $Lo^j(n)$ e $Hi^j(n)$. Esses filtros são obtidos de *Lo(n)* e *Hi(n)*, intercalando-se $(2^{j-1} - 1)$ zeros entre cada um dos coeficientes desses filtros (Figura 2.33). Deve-se observar que a resposta em frequência destes filtros fornece, obviamente, as mesmas bandas de frequência obtidas no caso da utilização do dizimador na saída dos filtros.

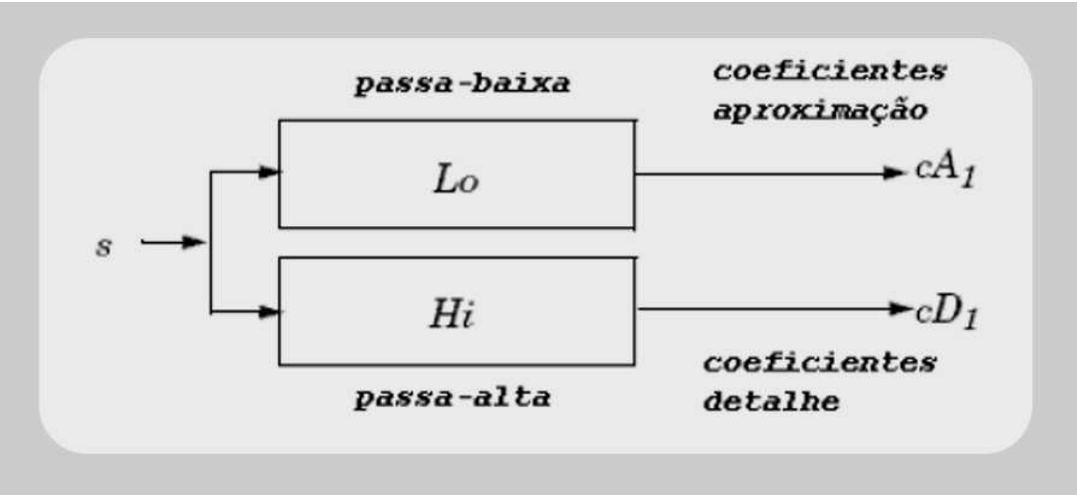

Figura 2.32: Processo de decomposição em nível 1 via TWE

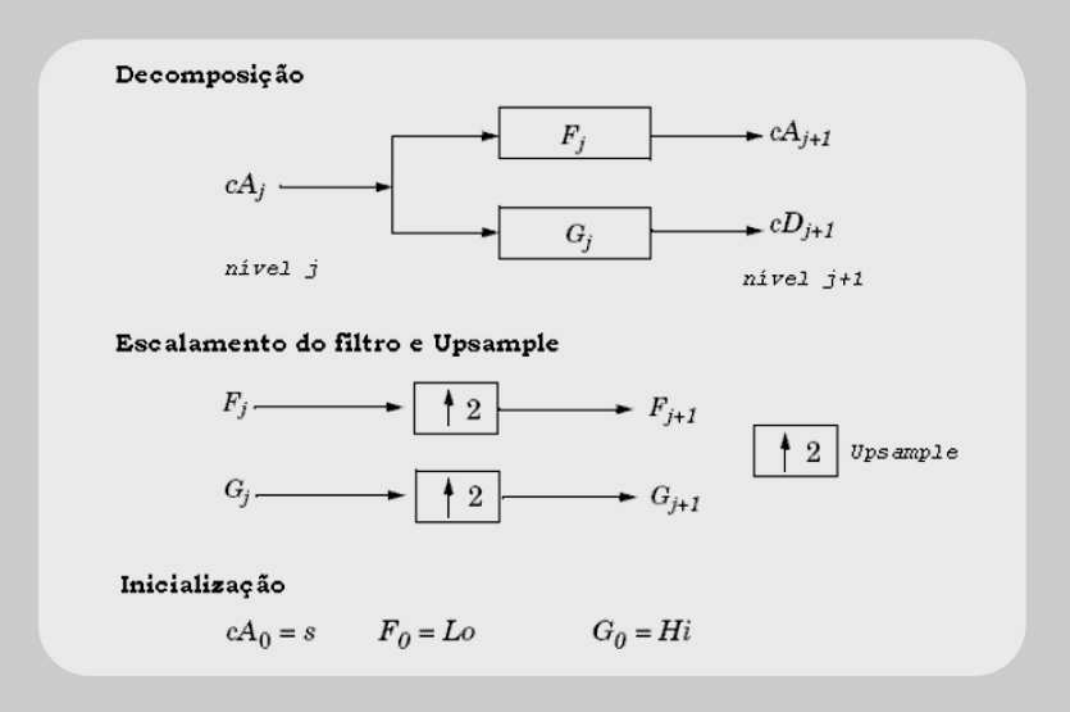

Figura 2.33: Processo de decomposição em nível j via TWE

Ressalta-se que a utilização da TWE implica em um maior esforço computacional, bem como na introdução de redundância nos sinais de saída. Porém, dependendo do nível de decomposição requerido, o número de amostras para a reconstrução do sinal pode ser insuficiente para uma boa representação quando da utilização da análise multiresolução com dizimação. Fato

este que pode ser atenuado utilizando a TWE, uma vez que sempre teremos o mesmo número de amostras do sinal original.

Escolha da base *wavelet*

São vários os sinais característicos que podem ser utilizados na Transformada *Wavelet*, como Haar, Daubechies, Meyer, Mexican Hat, dentre outros, que, dependendo da aplicação, podem proporcionar melhor ou pior resultado.

A escolha de determinada *wavelet* para análise ainda é arbitrária, apesar dos vários estudos sobre a otimização de funções. A Transformada *Wavelet* pode atuar, dentre outras formas, ressaltando localmente os detalhes do sinal (efetuando um "zoom"), detectando bordas e singularidades ou reconhecendo padrões.

Considerando a simplicidade da *wavelet* de Haar, qualquer processo baseado nela poderá ser rápido e utilizando um *hardware* simples.

2.3.3 Filtro digital

Um filtro digital é um sistema dinâmico discreto projetado para alterar o perfil do conteúdo espectral de um sinal de entrada em uma determinada banda de frequências. Com base na resposta da sua função de transferência, os filtros são classificados em quatro tipos: passa-baixa, passa-alta, passa-faixa e rejeita-faixa.

Uma ampla classe de filtros digitais é descrita por uma equação linear a diferenças, com coeficientes constantes, que pode ser escrita como:

$$\sum_{i=0}^{N} b_i . y(n-i) = \sum_{i=0}^{N} a_i . x(n-i) \qquad (2.25)$$

onde a_i e b_i são os coeficientes da função de transferência e N é a ordem do filtro. Apesar de esta equação ser geral, apenas os sistemas causais serão discutidos, com y_n e x_n iguais a zero para $n < 0$.

Um sistema causal refere-se a um sistema que é fisicamente realizável. Este sistema, em um dado tempo t, produz uma saída que é dependente somente das entradas presentes e passadas, $n \leq$

t, e saídas passadas, $n < t$. Isto sempre será verdadeiro para uma resposta ao impulso unitário, que é zero para $n < 0$.

A Equação (2.26) expressa a saída presente em termos das entradas presentes e passadas e saídas passadas, admitindo-se o coeficiente $b_0 = 1$.

$$y(n) = \sum_{i=0}^{N} a_i . x(n-i) - \sum_{i=1}^{N} b_i . y(n-i) \qquad (2.26)$$

A Equação (2.26) pode ser implementada como um conjunto de multiplicações, somatórios e atrasos.

A tarefa de projetar filtros digitais consiste na determinação dos coeficientes da Equação (2.26) para preencher os critérios de entrada e saída. Existem duas classes principais de filtros digitais, que são: filtros de resposta ao impulso finita (FIR) e filtros de resposta ao impulso infinita (IIR).

Filtros de resposta finita ao impulso (FIR)

Os filtros podem ser classificados como lineares ou não lineares, sendo que os primeiros são preferíveis, pois seu tratamento pode ser analisado de uma maneira mais simples.

A reação *y(n)* de um sistema linear discreto invariante no tempo quando aplicada uma sequência de entrada *x(n)* é determinada pela Equação 2.27.

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) \qquad (2.27)$$

onde a_i são as constantes e *N* é a ordem do filtro.

Uma implementação de filtros FIR pode ser diretamente derivada a partir da Equação (2.27), e consiste de elementos somadores, multiplicadores (triângulos) e registradores para o armazenamento dos sinais atrasados sucessivamente com o operador z^{-1}, conforme a Figura 2.34.

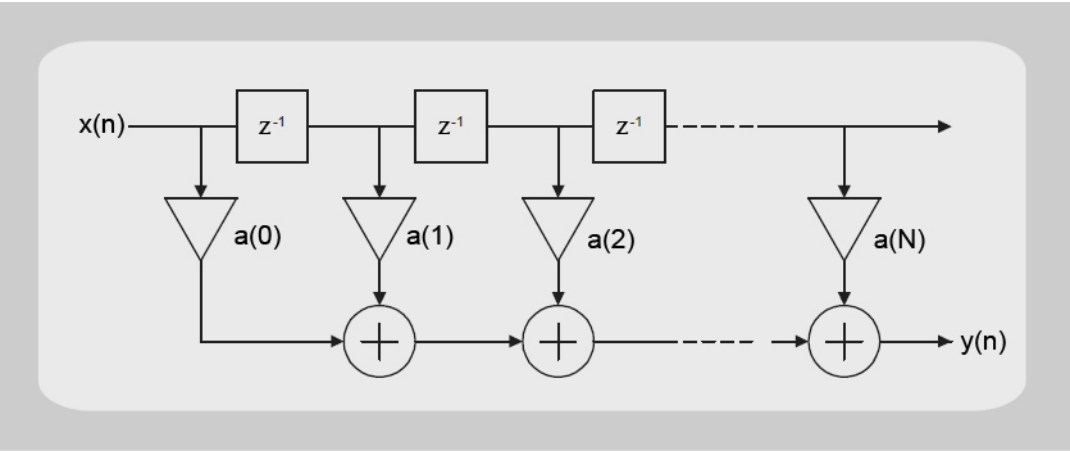

Figura 2.34: Fluxo de sinais de um filtro FIR

Devido ao fato de que os valores de saída passados não influenciam no cálculo dos valores de saída presentes, este filtro também é chamado de filtro não-recursivo. Uma característica importante é que estes filtros são sempre estáveis, por não apresentarem polos.

Filtros de resposta infinita ao impulso (IIR)

Em geral, filtros recursivos podem cumprir determinadas tarefas com um número de operações mais baixo que seus equivalentes FIR. Quanto menos coeficientes forem necessários, menos são os componentes para sua implementação na forma direta. Em contrapartida, filtros recursivos podem apresentar instabilidade, devido à possibilidade de pólos fora do círculo de raio unitário.

Filtros recursivos podem ser realizados a partir da soma de uma parte puramente recursiva e uma parte não-recursiva. Isto significa que um filtro recursivo geral pode ser implementado por um filtro FIR não recursivo e um filtro IIR (*Infinite Impulse Response*) puramente recursivo (chamado de filtro *autorregressivo*) conectados em série. Filtros IIR normalmente são representados no domínio do tempo por equações a diferenças, como:

$$y(n) = \sum_{i=1}^{N} \left(b_i y(n-i) \right) + u(n) \qquad (2.28)$$

onde *u(n)* representa o filtro FIR, expresso por:

$$u(n) = \sum_{i=0}^{N} a_i x(n-i) \qquad (2.29)$$

A Figura 2.35 mostra a implementação de um filtro autorregressivo com base na Equação (2.28). É possível verificar que os filtros IIR utilizam os mesmos elementos estruturais dos filtros FIR (somadores, multiplicadores e registradores), tendo como diferença apenas a maneira como esses blocos são conectados.

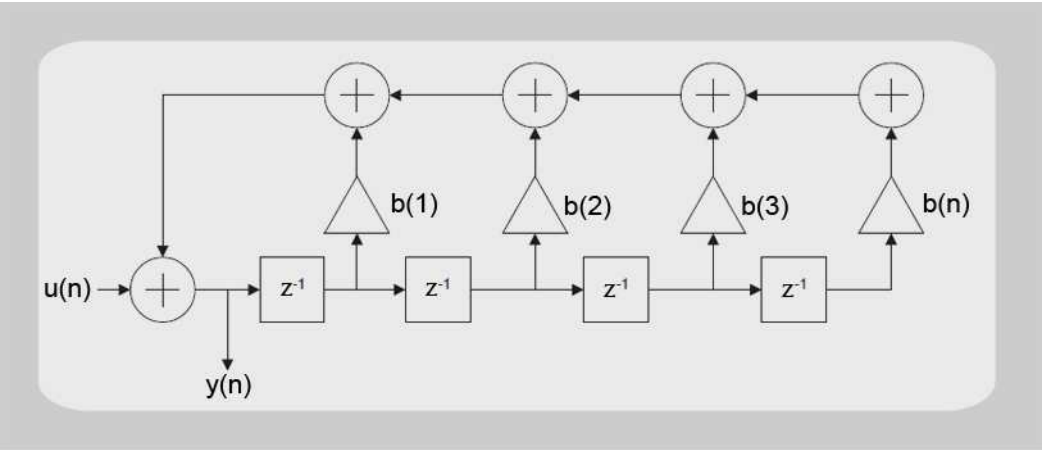

Figura 2.35: Filtro autorregressivo na forma direta

2.4 Dispositivos lógicos programáveis

2.4.1 Circuitos digitais

Os circuitos digitais têm sofrido grande evolução nas últimas décadas. As constantes mudanças na tecnologia têm transformado de forma radical todo o processo de projeto de *hardware*.

Os componentes dos circuitos digitais evoluíram de válvulas termo-iônicas, passando para transistores individuais até circuitos integrados VLSI (*Very Large Scale Integration*). A utilização de ferramentas EDA (*Electronic Design Automation*) tem simplificado e acelerado todo o ciclo de projeto. Atualmente, não é mais necessário desenhar portas lógicas individuais e planejar todas suas interconexões. As linguagens de descrição de *hardware* (HDLs) estão hoje

consolidadas nos meios acadêmicos e industriais como forma padrão na elaboração de projetos. Existem também, ferramentas de síntese lógica automática, disponíveis para mapear circuitos em diversas tecnologias.

Todas essas mudanças na tecnologia exigem uma prototipação cada vez mais rápida, pois o ciclo de vida dos produtos modernos está se tornando cada vez mais curto em relação ao tempo necessário para o projeto e desenvolvimento dos mesmos.

Os CIs digitais podem ser construídos utilizando-se de diversas tecnologias diferentes, sendo que a escolha da tecnologia adequada deve ser realizada com base no tipo de projeto que se pretende executar. A Figura 2.36 ilustra as tecnologias para projetos de sistemas digitais.

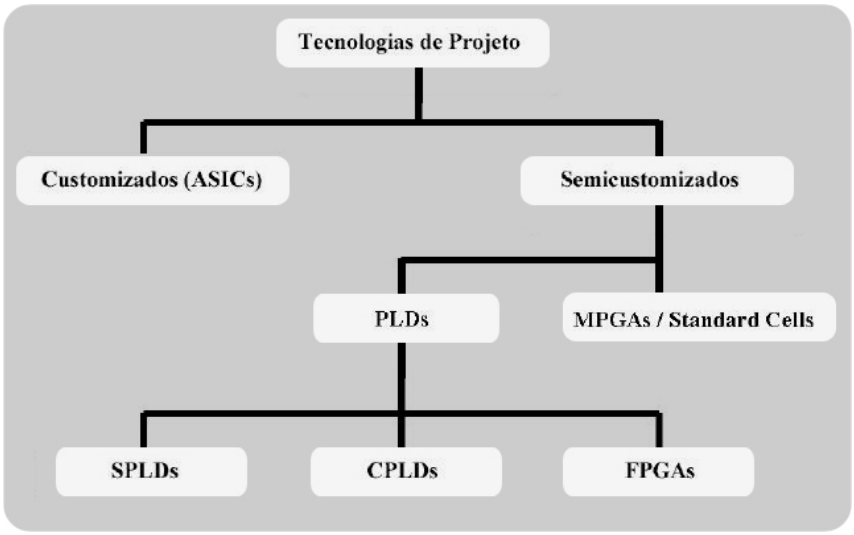

Figura 2.36: Tecnologias para projetos de sistemas digitais

As implementações de circuitos podem ser agrupadas em diversas categorias de projeto:

- CIs customizados ou ASICs (*Application Specific Integrated Circuit*): São aqueles que necessitam de um processo de fabricação especial, que requer máscaras específicas para cada projeto. Características desse tipo de implementação são os custos de projeto extremamente altos e o tempo de desenvolvimento longo. Em aplicações que requerem um grande volume de produção, o alto custo do projeto e dos testes é amortizado;

- MPGAs (*Mask Programmable Gate Arrays*): Nesse tipo de implementação, o processo de fabricação é agilizado pelo uso de máscaras genéricas de módulos pré-projetados, mas ainda necessita de máscaras específicas para a interconexão dos módulos. O projeto é normalmente facilitado por uma biblioteca de células, proporcionando um tempo de desenvolvimento mais curto e custos mais baixos em relação aos CIs cutomizados;

- *Standard Cells*: Essa tecnologia se assemelha muito a das MPGAs, o projeto também é facilitado pelo uso de módulos pré-projetados. Os módulos (*standard cells*) são geralmente salvos em bancos de dados. Os projetistas selecionam as células desejadas (nesses bancos de dados) para realizar seus projetos. Em comparação aos CIs customizados, os circuitos implementados em *standard cells* são menos eficientes em tamanho e desempenho, entretanto, seu custo de desenvolvimento é mais baixo;

- PLDs (*Programmable Logic Devices*): Essa tecnologia possui como principal carcterística a capacidade de programação (configuração) pós-fabricação pelo usuário, facilitando assim as mudanças de projetos. Em comparação com outras tecnologias, os PLDs apresentam ciclo de projeto muito curto com baixos custos de desenvolvimento.

O mercado de PLDs se encontra em plena expansão, de forma que atualmente existem diversos fabricantes e modelos de dispositivos desse tipo. Uma das principais tarefas do projetista hoje é pesquisar e selecionar, dentre as opções disponíveis no mercado, qual a que melhor atende suas necessidades.

2.4.2 Evolução dos PLDs

O primeiro tipo de chip programável pelo usuário, que podia implementar circuitos lógicos, foi a memória PROM (*Programmable Read Only Memory*), em que cada linha de endereço poderia ser usada como entrada do circuito e as linhas de dados como saídas. As funções lógicas, entretanto, raramente requerem mais que alguns termos de produto, e uma PROM contém um decodificador completo para seus endereços de entradas. A PROM, portanto, é uma arquitetura ineficiente para a realização de circuitos lógicos, e são raramente utilizadas na prática para esse fim.

O primeiro dispositivo desenvolvido especificamente para a implementação de circuitos lógicos foram os PLAs (*Programmable Logic Arrays*). Um PLA consiste de dois níveis de portas lógicas, conforme ilustrado na Figura 2.37, sendo um plano de portas *wired-AND* seguido por um plano de portas *wired-OR*, ambos programáveis. Um PLA é estruturado de tal forma que cada saída do plano de portas *AND* pode corresponder a qualquer termo de produto das entradas. Similarmente, cada saída do plano *OR* pode ser configurada para produzir a soma lógica de qualquer saída do plano *AND*. Com essa estrutura, os PLAs são bem adequados para implementação de funções lógicas na forma de soma de produtos. Eles são muito versáteis, pois tanto os termos *AND* como os termos *OR* podem ter muitas entradas.

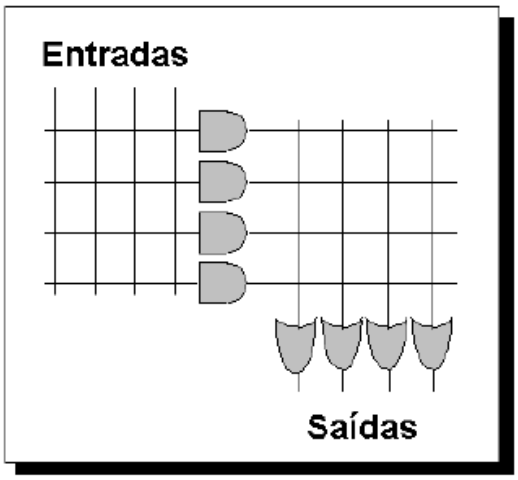

Figura 2.37: Estrutura de um PLA

Quando os PLAs foram introduzidos no início dos anos 70, pela Philips, suas principais deficiências eram o alto custo de fabricação e o pobre desempenho em termos de velocidade. Ambas as desvantagens eram devido aos níveis de lógica configurável, porque os planos lógicos programáveis eram difíceis de serem fabricados e introduziam atrasos de propagação significantes. Para superar essas deficiências, dispositivos PAL (*Programmable Array Logic*) foram desenvolvidos. Os PALs possuem um único nível de programação, consistindo de um plano de portas *wired-AND* programáveis que alimenta portas *OR* fixas, conforme a Figura 2.38. Para compensar a falta de generalidade devido ao plano *OR* fixo, diversos modelos de PALs foram produzidos, com diferentes números de entradas e saídas e vários tamanhos de portas *OR*.

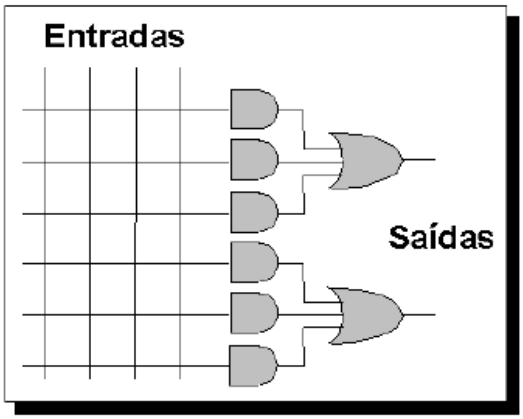

Figura 2.38: Estrutura de um PAL

Os PALs geralmente contêm *flip-flops* conectados às saídas das portas *OR* para que circuitos seqüenciais possam ser implementados. Dispositivos PAL foram importantes porque, quando introduzidos, tiveram um profundo efeito no projeto de *hardware* digital, e também foram a base para algumas das novas e mais sofisticadas arquiteturas. Variantes da arquitetura básica do PAL são encontradas em outros produtos conhecidos por diferentes siglas. Todos os pequenos PLDs, como PLAs, PALs, e outros dispositivos similares são agrupados em uma única categoria chamada SPLDs (*Simple PLDs*), cujas características mais importantes são o baixo custo e alto desempenho.

Com o avanço da tecnologia, tornou-se possível a produção de dispositivos com maior capacidade que os SPLDs. A dificuldade de aumentar a capacidade da arquitetura de SPLDs consiste na estrutura dos planos lógicos programáveis que aumenta muito rapidamente com o aumento do número de entradas. O único modo viável de produzir dispositivos com maior capacidade baseados nas arquiteturas de SPLDs foi integrar múltiplos SPLDs em um único chip e prover interconexões programáveis para conectar os blocos SPLDs. Muitos produtos PLDs são encontrados no mercado atualmente com essa estrutura básica, e são coletivamente chamados de CPLDs (*Complex PLDs*).

Os CPLDs foram introduzidos pela ALTERA Co., inicialmente com sua família de chips chamada Classic EPLDs (*Erasable PLDs*), e em seguida com a série MAX. Devido ao rápido crescimento do mercado para grandes PLDs, outros fabricantes desenvolveram dispositivos na categoria CPLD e há atualmente diversas opções disponíveis. Os CPLDs provêem capacidade

lógica de até 50 dispositivos SPLDs típicos, mas é difícil estender essa arquitetura para densidades maiores. Para construir PLDs com capacidade lógica muito alta, uma abordagem diferente é necessária.

A alta capacidade lógica em dispositivos de propósito geral começou a ser alcançada com os MPGAs. MPGAs consistem em um *array* de transistores pré-fabricados que podem ser customizados pelo usuário, através da definição das interligações e conexões a serem realizadas ainda na fase de manufatura do dispositivo, o que implica em alto custo de configuração e longo ciclo de manufatura. Assim como os MPGAs, os FPGAs incluem um arranjo de elementos de circuitos não conectados, chamados blocos lógicos, e recursos de interconexão, mas a configuração de FPGAs é realizada através de programação pelo usuário final. Sendo o único tipo de PLD que suporta capacidade lógica elevada, os FPGAs têm sido responsáveis pela principal mudança no modo em que os circuitos digitais são projetados.

O primeiro FPGA disponível comercialmente foi desenvolvido pela Xilinx Inc., surgindo no mercado em 1984. Desde então, vários dispositivos têm sido desenvolvidos por diversos fabricantes e rapidamente tem se disseminado, o que se pode atribuir ao reduzido tempo de projeto e ao relativo baixo custo destes dispositivos programáveis de alta capacidade.

Cada tipo de PLD apresenta vantagens que tornam mais adequados para algumas aplicações do que outros. Um projetista hoje depara com a difícil tarefa de pesquisar os diferentes tipos de *chips*, entender qual sua melhor utilização, escolher um fabricante específico, aprender a utilizar as ferramentas EDA, para só então começar a projetar o *hardware*.

Serão apresentados abaixo os aspectos mais relevantes relativos às arquiteturas de FPGAs, detalhando suas principais características.

2.4.3 Arquitetura de FPGAs

FPGA é um dispositivo semicondutor formado principalmente por um bloco de circuitos eletrônicos básicos que podem ser reconfigurados para exercer diversas funções lógicas. A célula básica deste bloco completamente programável é formada por um conjunto de memórias, que pode ser arranjado em várias matrizes diferentes através de sua programação. Estas células

permitem a implementação de qualquer tipo de circuito lógico, limitado apenas pelo número de células. Além dos blocos completamente programáveis, os dispositivos atuais incorporam vários outros circuitos especializados, que permitem incorporar funções adicionais já implementadas em *hardware*, como entradas e saídas digitais, multiplicadores e interfaces padronizadas. Por outro lado, os FPGA's são razoavelmente fáceis de programar, há ótimas ferramentas de alto nível e de baixo custo para o desenvolvimento de aplicações, produzidas pelos fabricantes, e a evolução neste setor tem sido muito acentuada nos últimos anos. Dentre outras vantagens asseguradas para a sua aplicação, pode-se citar, na comparação com o leque de opções disponíveis ao engenheiro projetista: a diminuição dos custos pela aceleração do tempo de desenvolvimento de produtos devido ao uso de ferramentas avançadas; o aumento da confidencialidade uma vez que uma dada arquitetura projetada para uma aplicação não pode ser facilmente duplicada pelos concorrentes; o atendimento facilitado a restrições para sistemas embarcados tais como consumo energético, condições térmicas e ambientais severas, etc; e um melhor desempenho devido a projetos dedicados a uma dada aplicação, e sua reconfigurabilidade em atendimento a condições operacionais que possam variar.

Por todas estas razões, os FPGA's têm se tornado muito populares nos últimos anos nos meios especializados, sendo usados, dentre outras coisas, para aplicações tão diversas como telecomunicações, processamento de vídeo e imagem, equipamentos médicos, robótica, automobilística, aeroespacial, sistemas embarcados em geral, controle de processos industriais, controle de potência elétrica e processamento de sinais (Monmasson, 2007).

Um FPGA consiste de um grande arranjo de células configuráveis (ou blocos lógicos) contidos em um único *chip*. Cada uma dessas células contém certa capacidade computacional para implementar funções lógicas, e/ou realizar roteamento para permitir a comunicação entre células. Essas operações podem acontecer simultaneamente no arranjo das células.

A arquitetura de um FPGA é bastante semelhante à arquitetura convencional de um MPGA. A principal diferença é que no MPGA as interconexões são feitas durante o processo de fabricação, como em circuitos integrados, enquanto os FPGAs são programados via comutadores programáveis, assim como os PLDs.

A arquitetura básica de um FPGA, ilustrada na Figura 2.39, consiste de um arranjo 2-D de blocos lógicos. A comunicação entre blocos é feita através de recursos de interconexão. A borda externa do arranjo consiste de blocos especiais capaz de realizar operações de entrada e saída (I/O).

Existem várias arquiteturas de FPGAs comercialmente disponíveis, por aproximadamente 10 fabricantes. Três aspectos principais definem a arquitetura de um FPGA:

- tipo de tecnologia de programação;

- arquitetura de células;

- estrutura de roteamento.

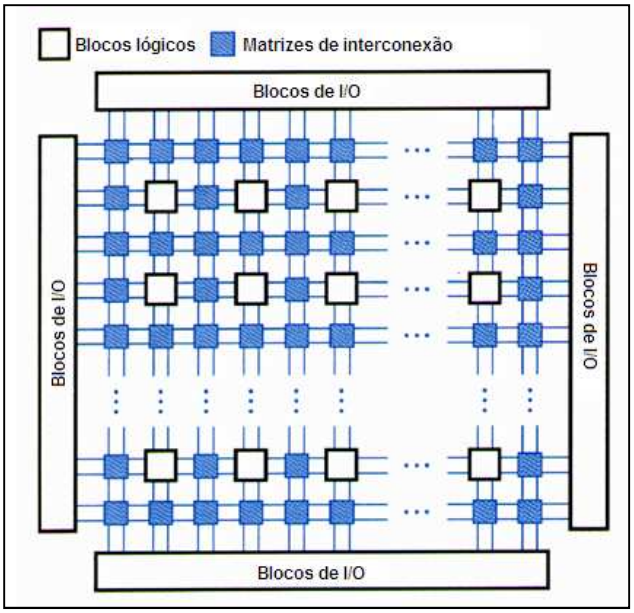

Figura 2.39: Estrutura de um FPGA

Estes aspectos influenciam diretamente o desempenho e a densidade das diferentes arquiteturas de FPGAs, entretanto não se pode afirmar que há uma melhor arquitetura, e sim a mais adequada para uma determinada aplicação.

As próximas seções descrevem com maiores detalhes as diferentes tecnologias de programação, complexidades dos blocos lógicos e arquiteturas de roteamento.

2.4.4 Tecnologia de programação

Um FPGA é programado usando comutadores programáveis eletricamente. As propriedades desses comutadores, tais como tamanho, resistência, capacitância, tecnologia, afetam principalmente o desempenho e definem características como volatilidade e capacidade de reprogramação, que devem ser avaliadas na fase inicial do projeto para a escolha do dispositivo.

Em todos os tipos de FPGAs, os comutadores programáveis ocupam uma grande área e apresentam valores de resistência e capacitância muito maior que as de um contato físico típico. Como conseqüência, o desempenho de um FPGA, se comparado a um MPGA de mesma tecnologia de fabricação, é menor.

Basicamente existem três tipos de tecnologias de programação:

- *Antifuse*: é originalmente um circuito aberto, que quando programado forma um caminho de baixa resistência;

- *Gate* flutuante: o comutador é um transistor com *gate* flutuante;

- SRAM (*Static Random Acess Memory*): o comutador é um transistor de passagem controlado pelo estado de um *bit* de SRAM.

2.4.5 Arquitetura de blocos lógicos

Os blocos lógicos dos FPGAs variam muito de tamanho e capacidade de implementação lógica. A construção dos blocos lógicos pode ser tão simples como um transistor ou tão complexa como um microcontrolador. Geralmente são capazes de implantar uma ampla variedade de funções lógicas combinacionais e seqüenciais.

Os blocos lógicos dos FPGAs comerciais são baseados em um ou mais dos seguintes componentes:

- pares de transistores;

- portas básicas do tipo *NAND* ou *XOR* de duas entradas;

- multiplexadores;

- *Look Up Tables* (LUTs);

- estruturas *AND* e *OR* de múltiplas entradas;

- *flip-flops* associados com diversos tipos de portas lógicas.

A fim de classificar os FPGAs quanto à capacidade lógica dos blocos lógicos, pode-se dividi-los em três categorias de granularidade: fina, média e grossa. A primeira categoria designa os blocos simples e pequenos, a segunda os blocos mais complexos e a terceira, sistemas completos em um único bloco lógico.

2.4.6 Arquitetura de roteamento

A arquitetura de roteamento de um FPGA é a maneira pela qual os comutadores programáveis e segmentos de trilhas são posicionados para permitir a interconexão dos blocos lógicos. As arquiteturas de roteamento podem ser descritas a partir de um modelo geral, conforme a Figura 2.40. São necessários alguns conceitos para um melhor entendimento desse modelo.

- pinos: são as entradas e saídas dos blocos lógicos, é válido ressaltar que estes pinos são internos aos FPGAs e não devem ser confundidos com os pinos externos do encapsulamento que são ligados aos blocos de I/O;

- conexão: é a ligação elétrica entre um pino e um segmento de trilha, as conexões são realizadas pelo bloco de conexão;

- bloco de conexão: é o dispositivo utilizado para conectar eletricamente um pino e um segmento de trilha, estes dispositivos utilizam alguma tecnologia de programação;

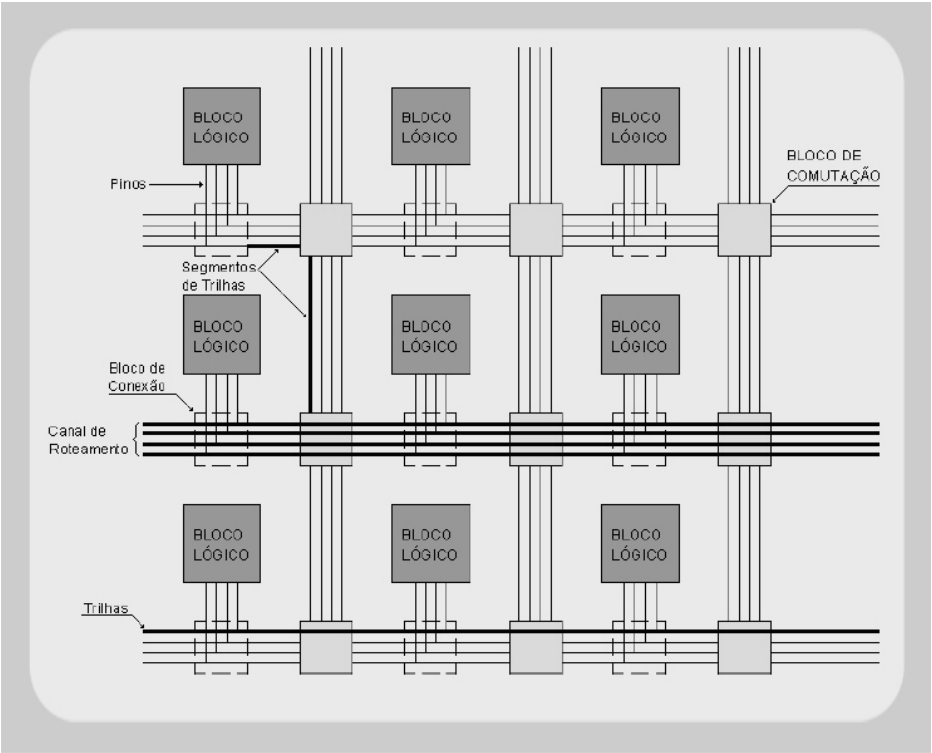

Figura 2.40: Arquitetura geral de roteamento de um FPGA

- segmentos de trilha: são os fios entre dois blocos de comutação;

- trilhas: é uma sequência de um ou mais segmentos de trilha em uma direção, estendendo-se por todo o comprimento de um canal de roteamento, pode ser composta de segmentos de tamanhos variados;

- bloco de comutação (*switch*): é o dispositivo utilizado para conectar eletricamente dois segmentos de trilha, estes dispositivos também utilizam alguma tecnologia de programação;

- canal de roteamento: é a área entre duas linhas ou colunas de blocos lógicos, sendo que um canal é formado por várias trilhas paralelas.

O modelo da Figura 2.40 contém duas estruturas básicas. A primeira é o bloco de conexões que aparece em todas as arquiteturas. O bloco de conexão permite a conectividade das entradas e saídas (pinos) de um bloco lógico com os segmentos de trilhas nos canais. A segunda estrutura é o bloco de comutação que permite a conexão entre os segmentos de trilhas horizontais e verticais. Em algumas arquiteturas, o bloco de comutação e o bloco de conexões são distintos, em outras estão combinados numa mesma estrutura. Nem todas as arquiteturas seguem esse modelo, pois há

arquiteturas que apresentam uma estrutura hierárquica e outras que possuem somente canais horizontais.

Uma importante questão a ser considerada é se a arquitetura de roteamento permite que se alcance um roteamento completo e, ao mesmo tempo, uma alta densidade lógica. Sabe-se que usando um grande número de comutadores programáveis, torna-se fácil alcançar um roteamento completo, mas esses comutadores consomem área, que é desejável minimizar. Algumas pesquisas estabelecem uma relação entre a flexibilidade da arquitetura de roteamento, a capacidade de roteamento e uso eficiente da área.

2.4.7 Arquiteturas comerciais de FPGA

O mercado oferece uma ampla gama de dispositivos FPGAs, de diversos fabricantes, em geral a estrutura de roteamento, os blocos lógicos e a forma de programação variam. Nota-se, entretanto, a existência de arquiteturas semelhantes que, em geral, podem ser classificadas em quatro classes, conforme ilustra a Figura 2.41. Dependendo da aplicação, uma arquitetura pode ter características mais desejáveis para a aplicação em questão.

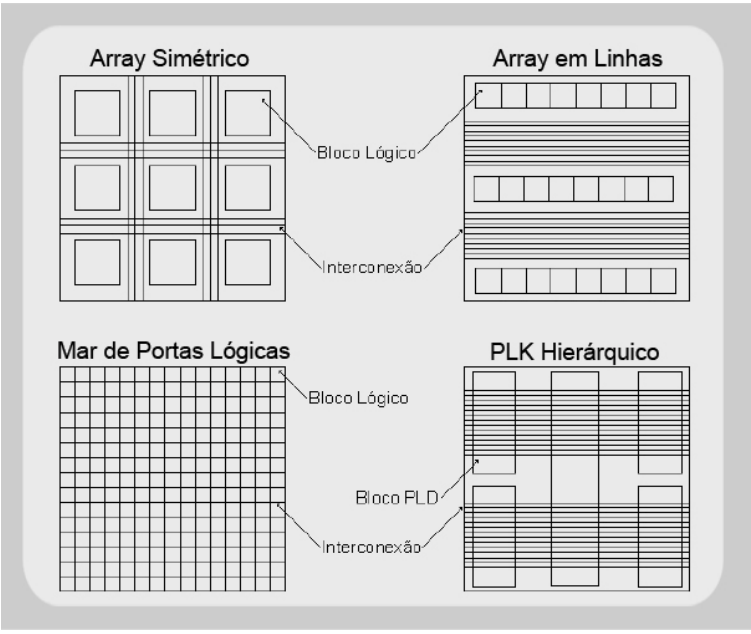

Figura 2.41: Tipos de arquiteturas

A Tabela 2.3 apresenta os principais fabricantes de FPGAs e suas tecnologias de programação.

Tabela 2.3: Fabricantes de FPGA x Tecnologia de Programação

Fabricantes	Tecnologia de Programação
Actel Corp.	Antifuse / EEPROM
Altera Corp.	SRAM / EEPROM
Atmel Corp.	SRAM
Cypress Semicondutor Corp.	EEPROM
Integrated Circuit Technology (ICT) Corp.	EEPROM
Lattice Semiconductor Corp.	SRAM / EEPROM
QuickLogic Corp.	Antifuse
Xilinx Corp.	SRAM / EEPROM

Os dois maiores fabricantes de arquiteturas FPGA atualmente são Xilinx Inc. e ALTERA CO.. A seguir são apresentadas, de forma sucinta, algumas famílias destes dois fabricantes.

A Xilinx Inc., fundada em 1984, tornou-se a primeira fabricante de FPGAs do mercado, lançando em 1985 a família XC2000. A cada ano ela oferece novas gerações, introduzindo inovações tecnológicas que aumentam a capacidade lógica e a velocidade dos dispositivos, que já ultrapassam o patamar de 700.000 células lógicas com o Virtex-6. Algumas famílias fornecidas pela Xilinx Inc. são a Virtex-6, Virtex-5, Spartan-6, Spartan-3, etc.

Enquanto a Xilinx Inc. foi a primeira fabricante de FPGAs, com a família XC2000, lançada em 1985, a ALTERA Co. inovou em 2000, fornecendo uma completa solução SoC (*System on Chip*), que consiste no desenvolvimento de sistemas completos em um único *chip*, para FPGAs da família APEX. Algumas famílias fornecidas pela ALTERA Co. são Stratix-III, Stratix-IV, Cyclone-III, Cyclone-IV, etc.

2.4.8 Ferramentas de Projeto ISE e EDK

Para a execução do trabalho, utilizou-se a plataforma *Integrated Software Environment* (ISE), como ferramenta para o ambiente de projeto. Para a utilização do microcontrolador MicroBlaze, utilizou-se uma outra ferramenta de integração de componentes de software embarcados – a EDK (*Embedded Development Kit*).

Ambas as ferramentas englobam as etapas necessárias para a implementação de sistemas reconfiguráveis: síntese, *floorplaning*, mapeamento, roteamento e criação dos *bitstreams*. Cada

etapa é realizada por uma ferramenta específica da Xilinx Inc., e estas executam segundo o fluxo definido para as ferramentas ISE e EDK (para esta última ferramenta, o fluxo é definido no *script* XFLOW). Para mais informações, ver Anexo A.

Capítulo 3

3 Metodologia para detecção do complexo QRS

3.1 Algoritmos desenvolvidos

Tem crescido com o passar dos anos as pesquisas sobre processamento do eletrocardiograma (ECG) usando microcontroladores. A literatura indica que sistemas baseados em microcomputadores estão sendo utilizados de maneira eficiente no auxílio de serviços médicos, tais como a implementação de técnicas de processamento de sinais na análise das 12 derivações do sinal de ECG, análise dos sinais armazenados pelo equipamento *Holter* e monitoramento de pacientes em tempo real. Todas essas aplicações requerem uma precisa detecção do complexo QRS de um ECG. Por exemplo, o monitoramento de arritmias em pacientes através da análise do ECG é capaz de indicar o tempo preciso da ocorrência de uma arritmia. Esse tipo de monitoramento requer uma boa capacidade de reconhecimento do complexo QRS. Dessa forma, a detecção do complexo QRS é uma parte importante de muitos sistemas de processamento de sinais de ECG (TOMPKINS, 1993).

O uso de dispositivos lógicos programáveis tem crescido bastante nos últimos anos, inclusive substituindo os microcontroladores convencionais, uma vez que estes dispositivos possuem grande flexibilidade de se adequar e otimizar uma aplicação específica através da sintetização em *hardware* e a facilidade e comodidade da implementação de microcontroladores utilizando propriedades intelectuais fornecidas também pelo próprio fabricante do dispositivo.

3.1.1 Método baseado em filtragem digital

O primeiro algoritmo utilizado neste trabalho para a detecção do complexo QRS foi desenvolvido por Pan & Tompkins (TOMPKINS, 1993) e os detecta baseado na análise da inclinação, amplitude e largura do sinal.

Através da análise do espectro de potência do sinal de ECG visto na seção 2.3, observa-se que a energia do sinal referente ao complexo QRS se destaca para frequências entre 10 e 15 Hz, sendo que um estudo revela que o valor máximo da relação sinal-ruído é obtido para uma frequência de 17 Hz (TOMPKINS, 1993). Assim, Pan & Tompkins desenvolveu um algoritmo baseado em filtros digitais que seleciona o complexo QRS do sinal de ECG utilizando filtros com coeficientes inteiros a uma frequência de amostragem de 200Hz.

O projeto consiste de dois estágios principais, que são o pré-processamento e a detecção dos picos.

Estágio de pré-processamento

O estágio de pré-processamento consiste de um filtro passa-baixa, um filtro passa-alta, um filtro derivativo, um estágio onde se eleva ao quadrado o resultado do filtro derivativo e por último um integrador de janela móvel. O propósito principal desta etapa é localizar complexos QRS (que geralmente ocorre numa banda de frequência em torno de 17 Hz) e, portanto, identificar os batimentos cardíacos corretamente. A Figura 3.1 ilustra o encadeamento de filtros (TOMPKINS, 1993).

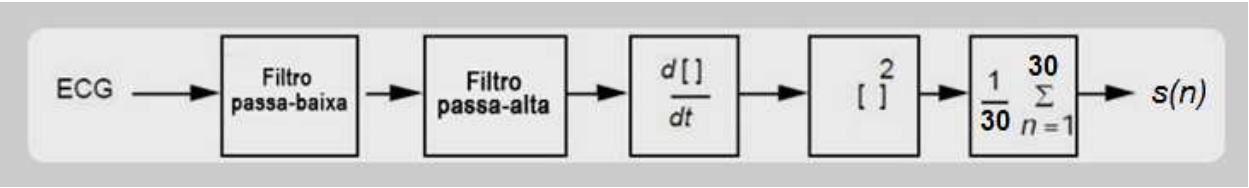

Figura 3.1: Banco de filtros para a detecção do QRS

Os filtros passa-baixa e passa-alta juntos formam um filtro passa-banda de largura de banda que varia de 5 a 11 Hz. O filtro passa-baixa atenua o ruído de alta frequência enquanto o filtro passa-alta atenua as ondas P e T do sinal de ECG e o sinal proveniente do atrito do eletrodo com

a pele (artefatos de movimento). O filtro derivativo calcula uma derivada de cinco pontos do sinal que não só minimiza o ruído, mas também enfatiza as informações provenientes do complexo QRS. A próxima etapa é um processamento não linear onde os valores resultantes do filtro derivativo são elevados ao quadrado. Este estágio enfatiza ainda mais os complexos QRS e atenuam os sinais provenientes de ruído. O próximo e último estágio refere-se ao integrador de janela móvel que calcula a média das últimas 30 amostras do sinal. A saída de cada um destes filtros é mostrada na Figura 3.2.

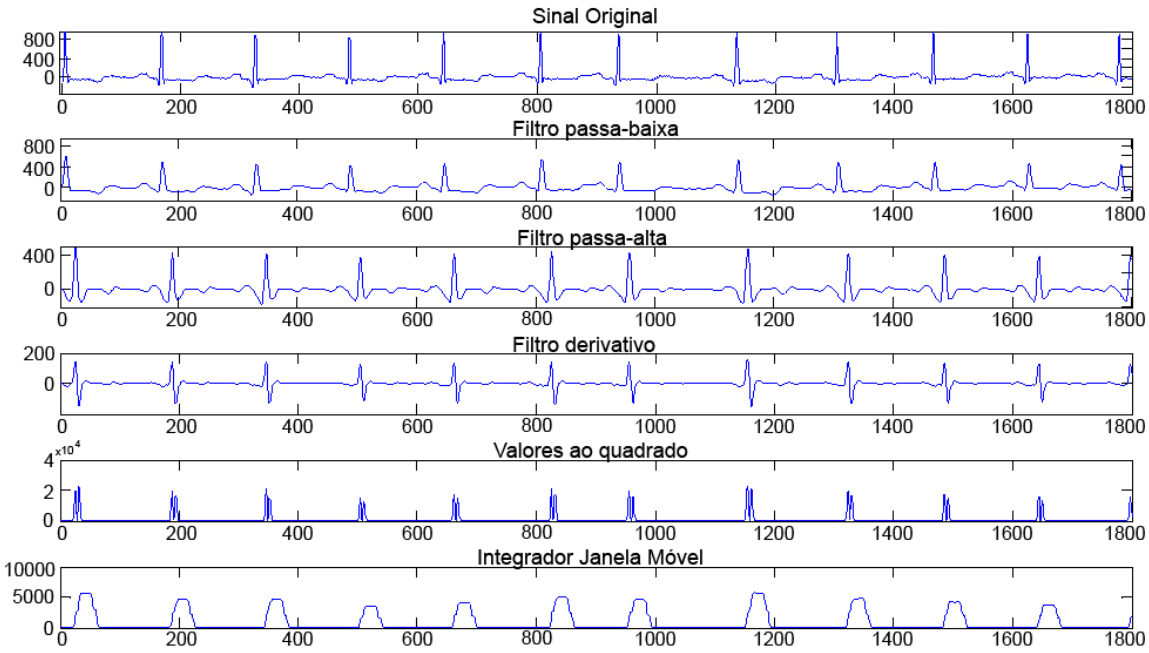

Figura 3.2: Resultado das saídas de cada estágio do pré-processamento

As funções de transferência e os gráficos de amplitude e fase dos filtros estão ilustrados abaixo. A função de transferência da Equação 3.1 refere-se ao filtro passa-baixa ilustrado na Figura 3.1.

$$H(z) = \frac{(1-z^{-6})^2}{(1-z^{-1})^2} \qquad (3.1)$$

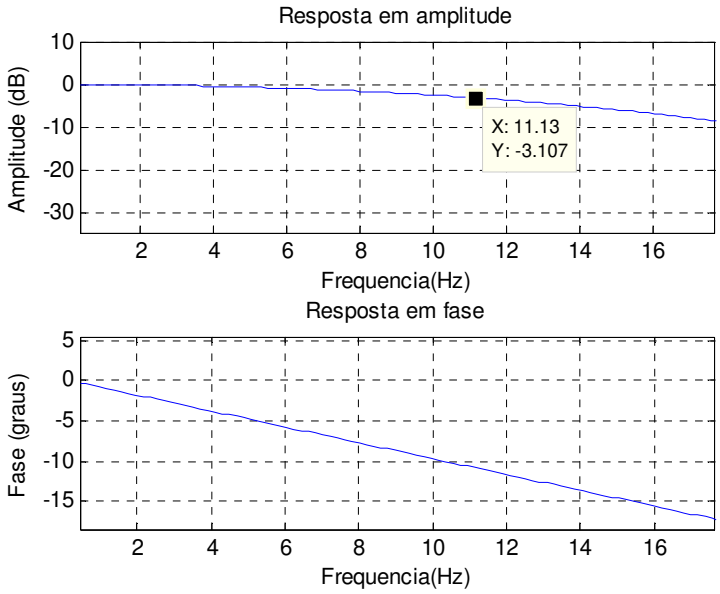

Figura 3.3: Resposta do ganho e fase do filtro passa-baixa

Conforme mencionado acima, o algoritmo de Pan & Tompkins foi desenvolvido para uma frequência de amostragem de 200Hz, sendo o tempo de amostragem de 5ms.

O filtro passa-alta é composto por um filtro passa-tudo menos um filtro passa-baixa, conforme ilustra as Equações (3.2) e (3.3). A função de transferência e os gráficos de amplitude e fase do filtro passa-alta estão ilustrados abaixo.

$$H_{lp}(z) = \frac{Y(z)}{X(z)} = \frac{(1-z^{-32})}{(1-z^{-1})} \tag{3.2}$$

$$H_{hp}(z) = \frac{P(z)}{X(z)} = z^{-16} - \frac{H_{lp}(z)}{32} \tag{3.3}$$

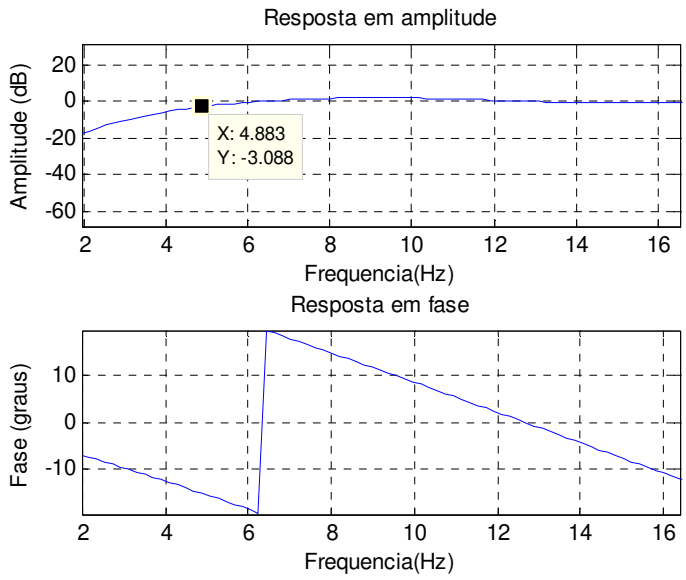

Figura 3.4: Resposta do ganho e fase do filtro passa-alta

A função de transferência e os gráficos de amplitude e fase do filtro derivativo estão ilustrados abaixo.

$$H(z) = 0{,}1(2 + z^{-1} - z^{-3} - 2z^{-4}) \qquad (3.4)$$

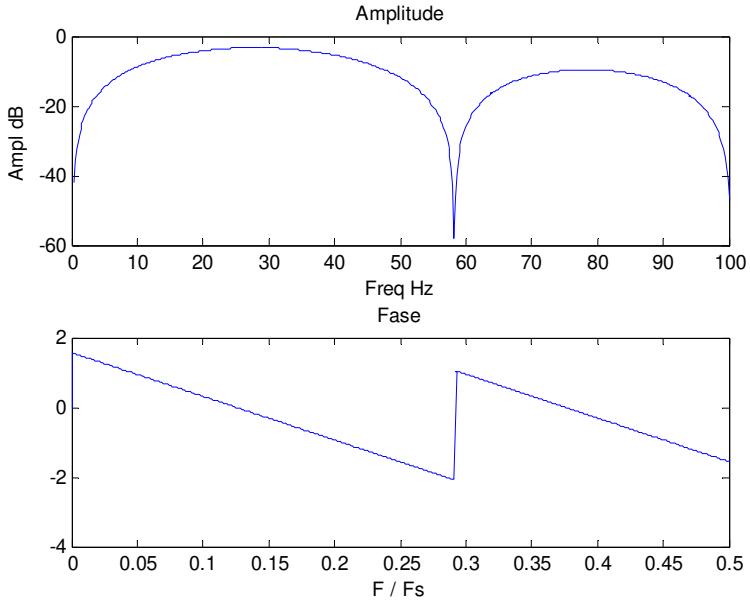

Figura 3.5: Resposta do ganho e fase do filtro derivativo

A equação a diferenças da função ao quadrado e da janela móvel estão ilustradas abaixo.

$$y(nT) = [x(nT)]^2 \quad (3.5)$$

$$y(nT) = \frac{1}{N}[x(nT - (N-1)T) + x(nT - (N-2)T) + \cdots + x(nT)]_{N=30} \quad (3.6)$$

Estágio de detecção do pico

Este estágio tem como entrada a saída do integrador de janela móvel. O propósito deste estágio é identificar o pico do sinal obtido no estágio de pré-processamento e utilizam-se limites, chamados aqui de *thresholds*, que possibilitam separar os picos provenientes de ruídos dos picos que representam o complexo QRS.

Dois *thresholds* são utilizados, sendo o primeiro (Th1) definido como sendo 25% da média dos oito últimos picos (MPEAK) correspondentes ao complexo QRS. O threshold 2 (Th2) é definido como sendo 50% do valor de pico em análise. Abaixo se encontram as equações de Th1 e Th2.

$$Th1 = 0.25 * MPEAK \quad (3.7)$$

$$Th2 = 0.5 * PEAK \quad (3.8)$$

onde,

MPEAK: Média aritmética dos últimos oito picos;

PEAK: A maior amostra representando a existência do complexo QRS. Precisa ser maior que Th1.

Todos os valores maiores que o Th1 são comparados entre si para determinar o máximo valor. No entanto, o máximo valor não é reportado enquanto não for encontrado um valor que possua 50% do máximo valor. Isto ocorre na inclinação negativa do sinal, conforme ilustrado na Figura 3.6.

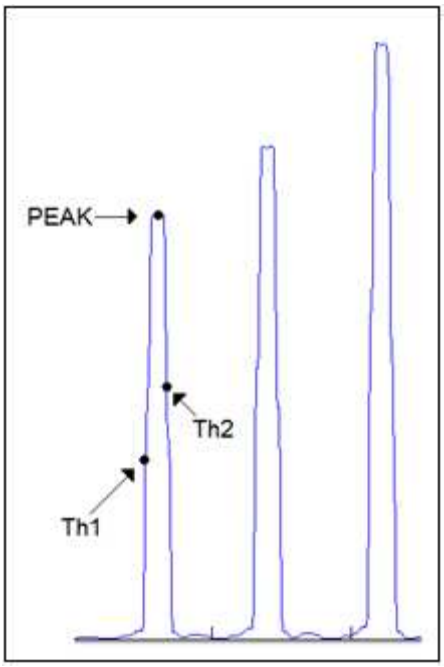

Figura 3.6: Posicionamento dos *thresholds* na detecção do QRS

O instante onde o pico é detectado (peak) é considerado como referência para a localização do complexo QRS.

Cada filtro utilizado na etapa de pré-processamento contribui com um atraso, que é inerente dos filtros e é chamado de *group delay*. O atraso total que corresponde à somatória dos atrasos dos filtros passa-baixa, passa-alta e derivativo equivale a 23 amostras ou 115ms para a frequência de amostragem utilizada. O integrador de janela móvel também contribui para o atraso, resultando assim em um atraso total de 35 amostras. Este atraso é considerado e descontado do instante de tempo onde ocorre a detecção do pico no último estágio do algoritmo.

3.1.2 – Método baseado na Transformada *Wavelet*

Características como a boa localização no tempo e na frequência, assim como a análise multi-resolução, são responsáveis pelo ótimo desempenho das Transformadas *Wavelets* na detecção de ondas e intervalos. De fato, essa abordagem permite a determinação das principais ondas e intervalos relevantes à análise do eletrocardiograma, tais como: complexo QRS; ondas P e T, intervalos e segmentos.

Ressaltam-se os níveis de acuidade obtidos na detecção automática do complexo QRS (valores de 98,8% e 99,8% já foram obtidos na análise de alguns sinais mais bem comportados das bases de dados MIT/BIH). Comparada com outros algoritmos de determinação do QRS, as Transformadas *Wavelets* se revelaram robustas a ruídos, flexíveis na análise de flutuações da morfologia do ECG e menor número de falsos negativos detectados (KADAMBE, MURRAY, BOUDREAUX-BARTELS, 1999).

O primeiro passo para a detecção dos picos do complexo QRS é o cálculo dos coeficientes da Transformada *Wavelet* a eles relacionados.

A *wavelet* mãe escolhida para este trabalho foi a de Haar, que corresponde à *wavelet* mais simples e de fácil processamento.

Levando em consideração o espectro de frequências do complexo QRS, onde Tompkins (1993) descreve que a melhor relação sinal-ruído ocorre para a frequência de 17 Hz, e as larguras de banda da TWE de Haar, escolheu-se a escala que representa o filtro passa-baixa de ordem 3 (Lo_3) e o filtro passa-alta de ordem 4 (Hi_4) como a ideal para a detecção do complexo QRS, pois esta escala apresenta largura de banda de 11,25 Hz a 22,50 Hz para a frequência de amostragem de 360Hz, conforme a Tabela 3.1.

Tabela 3.1: Faixa de freqüências da Transformada *Wavelet* de Haar

Filtro passa-alta		Filtro passa-baixa	
Hi_1	90,00 Hz		
Hi_2	45,00 Hz	90,00 Hz	Lo_1
Hi_3	22,50 Hz	45,00 Hz	Lo_2
Hi_4	11,25 Hz	22,50 Hz	Lo_3
Hi_5	5,63 Hz	11,25 Hz	Lo_4

Os coeficientes dos filtros passa-baixa e passa-alta utilizando a TWE, bem como suas respectivas funções de transferência estão ilustradas na Tabela 3.2:

Tabela 3.2: Filtros passa-baixa e passa-alta utilizados no banco de filtros da TWE

Filtros da TWE	Função de Transferência	Coeficientes
Lo_1	$H(z) = 0{,}7071 + 0{,}7071z^{-1}$	[0,7071 0,7071]
Lo_2	$H(z) = 0{,}7071 + 0{,}7071z^{-2}$	[0,7071 0 0,7071 0]
Lo_3	$H(z) = 0{,}7071 + 0{,}7071z^{-4}$	[0,7071 0 0 0 0,7071 0 0 0]
Lo_4	$H(z) = 0{,}7071 + 0{,}7071z^{-8}$	[0,7071 0 0 0 0 0 0 0 0,7071 0 0 0 0 0 0 0 0]
Lo_5	$H(z) = 0{,}7071 + 0{,}7071z^{-16}$	[0,7071 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,7071 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Hi_1	$H(z) = -0{,}7071 + 0{,}7071z^{-1}$	[-0,7071 0,7071]
Hi_2	$H(z) = -0{,}7071 + 0{,}7071z^{-2}$	[-0,7071 0 0,7071 0]
Hi_3	$H(z) = -0{,}7071 + 0{,}7071z^{-4}$	[-0,7071 0 0 0 0,7071 0 0 0]
Hi_4	$H(z) = -0{,}7071 + 0{,}7071z^{-8}$	[-0,7071 0 0 0 0 0 0 0 0,7071 0 0 0 0 0 0 0 0]
Hi_5	$H(z) = -0{,}7071 + 0{,}7071z^{-16}$	[-0,7071 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,7071 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Segue abaixo o gráfico da amplitude do filtro passa-baixa Lo_1 em azul e o passa-alta Hi_1 em vermelho. Observa-se a frequência de corte em -3dB de 565 rad/s, ou seja, aproximadamente 90Hz.

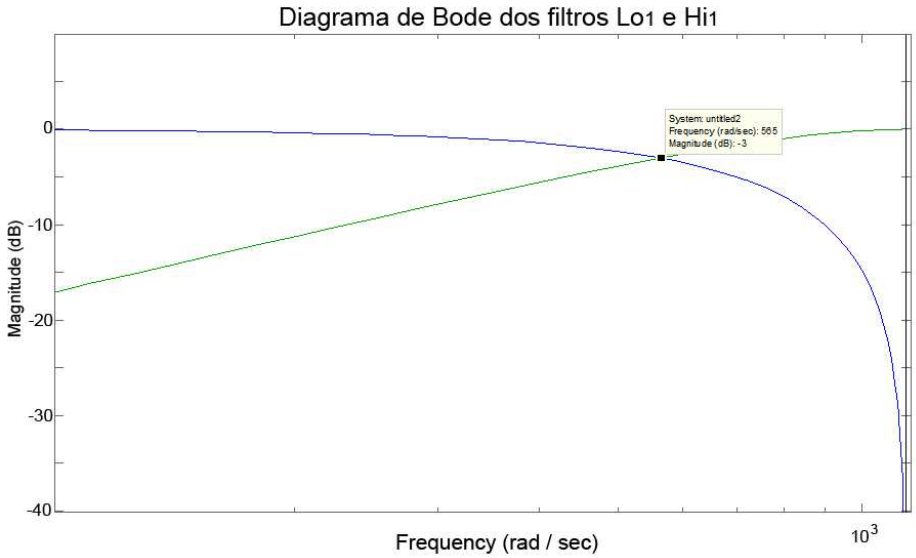

Figura 3.7: Frequência de corte dos filtros passa-baixa e passa-alta do nível 1

O segundo filtro passa-baixa recebe uma interpolação do primeiro filtro, ou seja, o acréscimo de "zeros" entre cada coeficiente. Observa-se no gráfico abaixo a frequência de corte deste filtro, que para uma queda de -3dB equivale a 282 rad/s ou aproximadamente 45Hz.

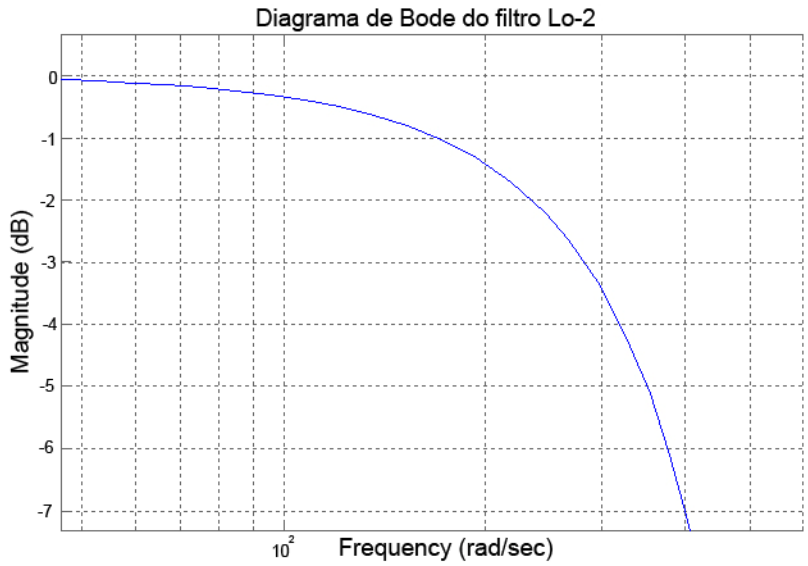

Figura 3.8: Frequência de corte do filtro passa-baixa do nível 2

O terceiro filtro passa-baixa por sua vez recebe a interpolação do segundo filtro, diminuindo pela metade a frequência de corte. Observe na figura abaixo que para uma queda de -3dB, a frequência é de aproximadamente 141 rad/s que equivale a 22,5Hz.

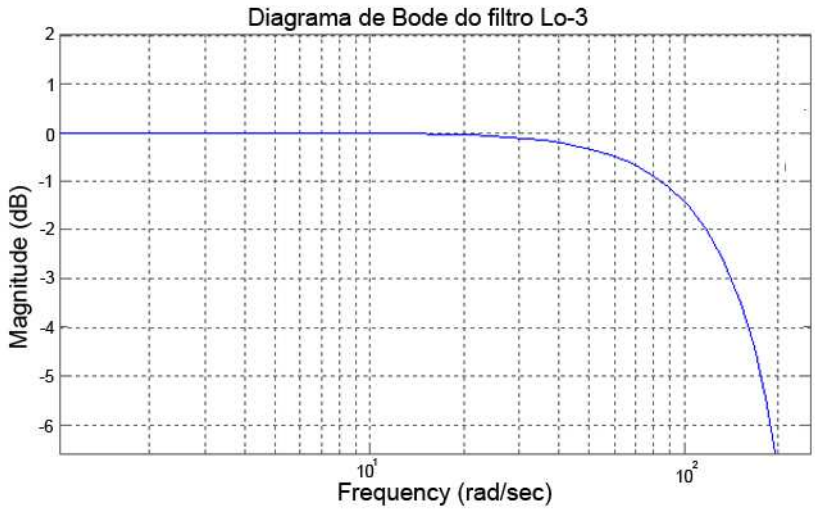

Figura 3.9: Frequência de corte do filtro passa-baixa do nível 3

Observa-se nas Figuras 3.7, 3.8 e 3.9 que os coeficientes dos filtros interpolados respeitam o conceito de *quadrature mirror filter*.

O algoritmo utilizado para a detecção das ondas R do sinal de ECG consiste da etapa de pré-processamento, que está fundamentado nos conceitos de pontos de máximo do módulo da

TWE e da etapa de detecção do pico, que consiste na definição de *thresholds* adaptativos para a validação dos picos. O algoritmo compreende os seguintes passos:

Passo 1: Cálculo da TWE através da filtragem do sinal pelos três filtros passa-baixa (Lo_1, Lo_2 Lo_3), seguido da filtragem pelo filtro passa-alta (Hi_4);

Passo 2: Elevar ao quadrado o resultado da TWE;

Passo 3: Localizar os pontos de máximo que excedam um determinado limiar, escolhido como sendo 50% da média dos últimos oito picos.

Passo 4: Os pontos que chegam mais próximos de zero após a localização de um ponto de máximo é considerado como sendo a localização de uma onda R.

A Figura 3.10 ilustra um trecho do sinal de eletrocardiograma e os resultados de cada etapa de pré-processamento.

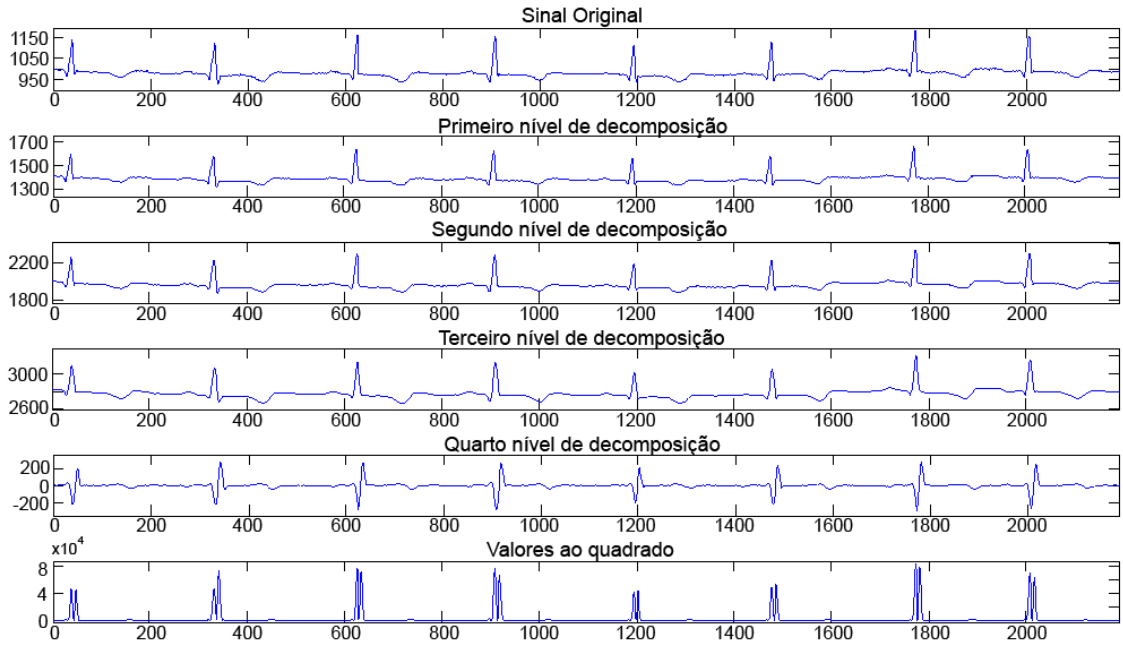

Figura 3.10: Resultado de cada etapa do pré-processamento através da TWE

Observa-se que no quarto nível de decomposição, o sinal original foi filtrado e apresenta frequências entre 11,25 Hz e 22,5 Hz. O resultado apresenta uma característica peculiar, onde no

instante da ocorrência do complexo QRS o sinal apresenta um ponto de mínimo local, seguido de um ponto de máximo local, representando o ponto onde cruza o eixo das abscissas o local da ocorrência do complexo QRS, conforme ilustra a Figura 3.11.

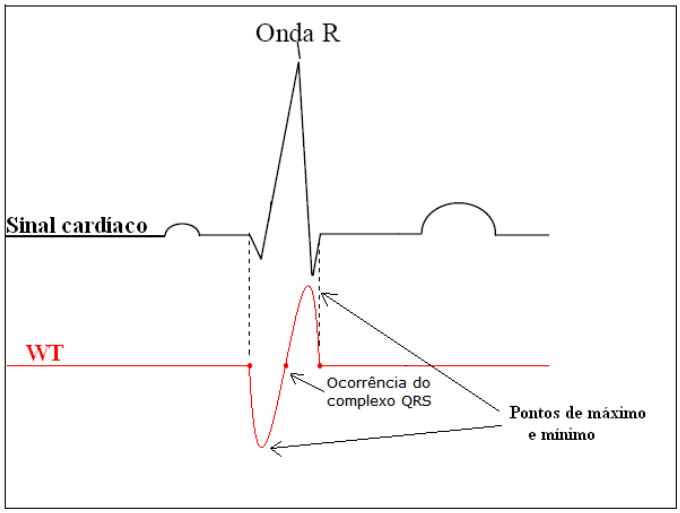

Figura 3.11: Ponto de ocorrência do complexo QRS utilizando WT

Optou-se por elevar ao quadrado o resultado da WT a fim de ressaltar o complexo QRS, resultando assim em um gráfico conforme ilustra a Figura 3.12.

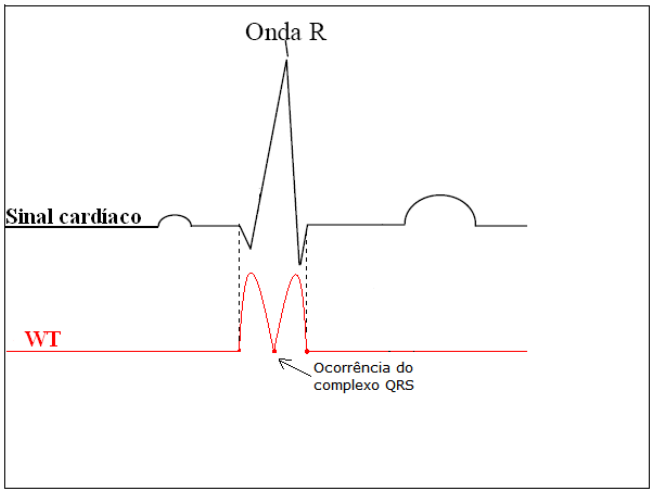

Figura 3.12: WT elevada ao quadrado

Estágio de detecção do pico

Esta etapa consiste em definir um limiar, a partir do qual é considerada a ocorrência do complexo QRS. A partir deste limiar, chamado de *threshold* 1 e que equivale a 50% da média dos oito últimos picos, o algoritmo detecta o pico de maior valor e então aguarda até que o valor corrente seja 50% do maior valor. Quando isso ocorre, é detectado o valor de pico e em seguida o valor mais próximo de zero. Neste instante mais próximo de zero é considerado o ponto de ocorrência do complexo QRS. A Figura 3.13 ilustra a localização de um complexo QRS.

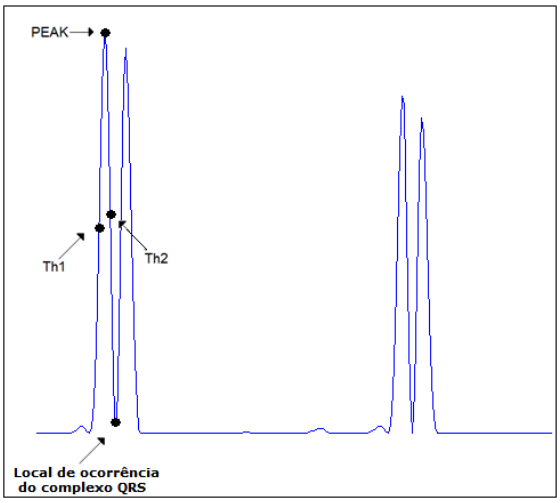

Figura 3.13: Posicionamento dos *thresholds* e local de ocorrência do complexo QRS

3.2 Implementação dos filtros utilizando FPGA

Para a implementação dos filtros, utilizou-se um kit de desenvolvimento contendo um FPGA da família Spartan-3E fornecido pela Xilinx Inc.. O kit contempla uma plataforma de *hardware* avançada que pode ser usada para criar complexos sistemas, sendo a família Spartan-3E destinada especificamente para atender as necessidades de relativa complexidade e baixo custo, conforme ilustrado na Figura 3.14.

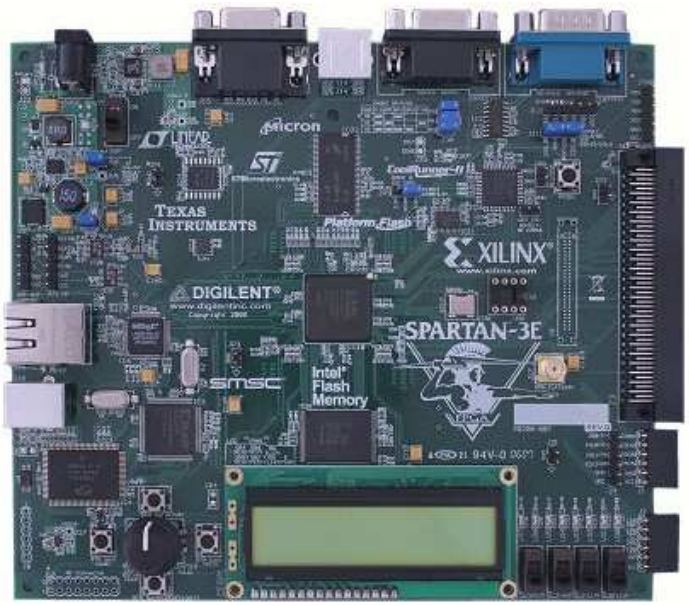

Figura 3.14: Kit de desenvolvimento Spartan 3E Starter Kit

Este kit possibilita ao projetista a implementação do microcontrolador MicroBlaze disponibilizado pela Xilinx Inc. como uma propriedade intelectual, além da possibilidade de utilização dos diversos periféricos.

Este kit contém as seguintes características:

- Baixo custo e alto desempenho;
- Oito leds;
- Quatro chaves on/off;
- Quatro chaves tipo *push-button*;
- FPGA Spartan-3E XC3S500E da Xilinx: 320 pinos no encapsulamento, 232 pinos de I/O e mais de 10.000 células lógicas;
- Suporte completo ao sistema Xilinx ISE Webpack;
- Display LCD de 2 linhas x 16 caracteres;
- Porta VGA;
- Porta PS/2 para mouse ou teclado;
- Interface de programação e debug USB;
- Oscilador de clock de 50 MHz;

- Conectores de expansão;
- Duas portas RS-232 (DTE e DCE);
- Conversores A/D e D/A;

Incluindo as funções acima, o FPGA em questão apresenta as seguintes características específicas:

- Configuração *flash*;
- Configuração *multiboot* da PROM *flash*;
- Configuração SPI *serial flash*;
- Implementação do processador MicroBlaze de 32 bits RISC;
- Controlador embarcado KCPSM3 8-bits;

Com relação à programação do FPGA, os fabricantes dos dispositivos fornecem um conjunto de ferramentas de software associado a um fluxo de projeto que permite ao projetista focar no algoritmo que se deseja implementar, ao invés de se preocupar com os circuitos que serão implementados. Para mais informações sobre o fluxo de projeto, consultar o Anexo A.

Foram utilizados dois métodos para o processamento do sinal. Primeiramente os filtros foram programados em linguagem C utilizando o microcontrolador MicroBlaze fornecido pela própria Xilinx Inc.. Em seguida foram programados os mesmos filtros em linguagem VHDL, utilizando o microcontrolador apenas como gerenciador.

3.2.1 **Implementação em linguagem C**

O MicroBlaze é um microcontrolador RISC de 32 bits, que apresenta arquitetura *Harvard* e é disponibilizado pela Xilinx Inc. como uma propriedade intelectual. Possui incorporadas diversas bibliotecas que possibilita o controle dos diversos barramentos e dispositivos periféricos, conforme a Figura 3.15.

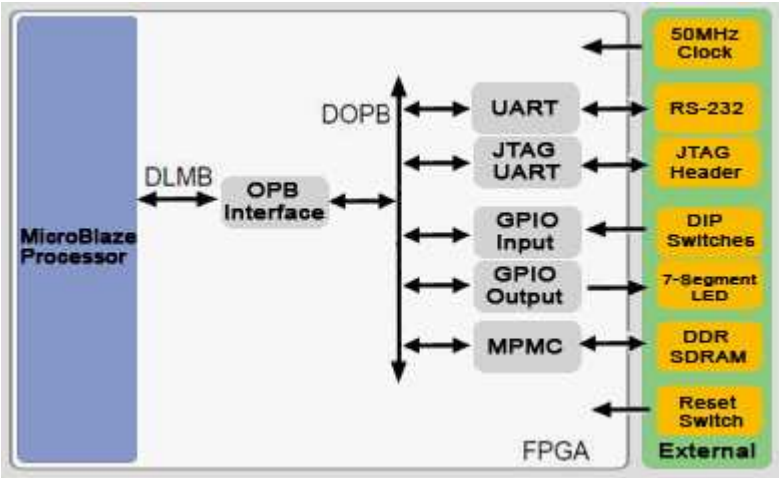

Figura 3.15: Implementação do microcontrolador MicroBlaze em FPGA

Dentre as diversas ferramentas disponibilizadas pela Xilinx Inc., a plataforma EDK foi a utilizada neste trabalho, uma vez que é através dela que se cria projetos embarcados com o microcontrolador MicroBlaze. Esta ferramenta também possibilita a inserção de diversos periféricos, tais como controladores, interface entre barramentos e blocos periféricos dedicados implementados em linguagem de *hardware*. Esta ferramenta trabalha em conjunto com a plataforma ISE, utilizada na estruturação e desenvolvimento do projeto.

A implementação dos filtros no microcontrolador MicroBlaze é semelhante à implementação em um microcontrolador de uso geral, onde se desenvolve a programação em linguagem C. A única diferença é que se utilizou uma propriedade intelectual que representa um microcontrolador e um dispositivo programável para a implementação deste *softcore*.

Os filtros implementados em linguagem C são processados de forma seqüencial, necessitando de um ciclo de *clock* para cada etapa do programa, conforme ilustra a Figura 3.16.

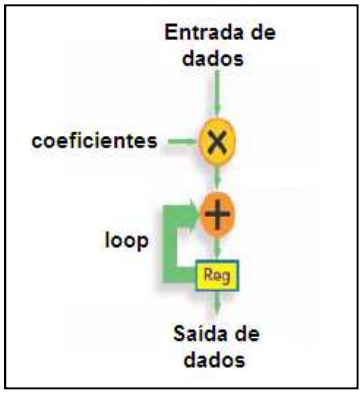

Figura 3.16: Implementação serial

Os algoritmos envolvem uma seqüência de filtros, sendo que a saída de um filtro é utilizada como entrada para o próximo filtro, conforme ilustra a Figura 3.17.

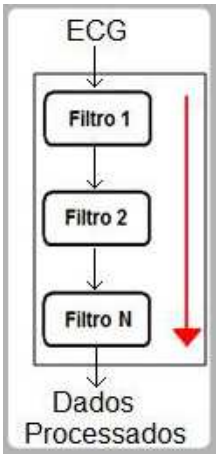

Figura 3.17: Processamento serial

3.2.2 Implementação em VHDL

Uma grande vantagem do uso de FPGA's é a capacidade da implementação de circuitos lógicos independentes, possibilitando o processamento paralelo, uma vez que se trata de um dispositivo programável e consistem de blocos lógicos configuráveis, blocos de entrada e saída e blocos de interconexões. Dessa forma propõe-se a implementação em *hardware* da detecção de complexos QRS em FPGA, fazendo uso de sua capacidade de paralelismo.

O MicroBlaze é utilizado apenas como gerenciador, uma vez que se faz necessário a transferência dos dados pela interface serial, o armazenamento em memória e o processamento dos dados no bloco VHDL. A Figura 3.18 ilustra a implementação.

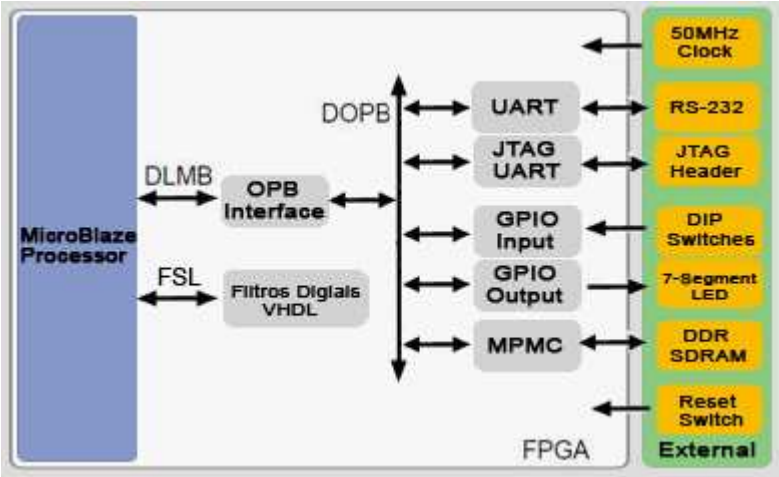

Figura 3.18: Interface entre microcontrolador e bloco VHDL

Os mesmos filtros foram implementados em VHDL, sendo que cada filtro representa um processo e cada processo é executado de forma concorrente ou paralela, porque se trata de circuitos eletrônicos. Foi utilizado o barramento *Fast Simplex Link* (FSL) para a transferência dos dados entre os filtros digitais e o microcontrolador. A Figura 3.19 ilustra o conceito do processamento paralelo de cada filtro utilizado.

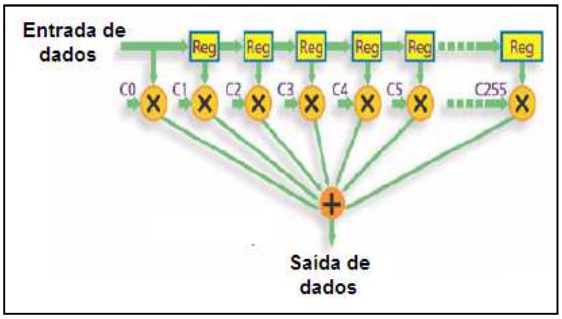

Figura 3.19: Implementação de forma paralela

Cada filtro de um determinado algoritmo foi implementado em um *process* utilizando a linguagem VHDL, sendo que a saída de um filtro, como na implementação em linguagem C, é utilizada como entrada para o próximo filtro. O que difere nesta implementação é que cada filtro

é processado de forma independente e paralela, resultando em um tempo de processamento menor, conforme ilustra a Figura 3.20.

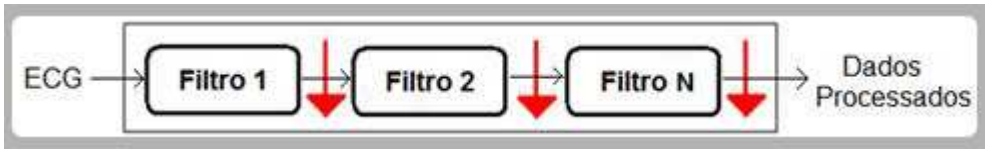

Figura 3.20: Processamento paralelo

A decisão de uma arquitetura ótima para uma aplicação particular deve ser tomada levando-se em conta o compromisso entre a área ocupada pelo projeto, o desempenho e o tempo de resposta, uma vez que arquiteturas seriais ocupam pouco espaço do dispositivo, mas tempo de processamento maior. Já a arquitetura paralela ocupa um espaço considerável no dispositivo e tempos de resposta muito rápidos.

Capítulo 4

4 Análise dos resultados

Os resultados apresentados são baseados no que foi produzido nas etapas do projeto em *hardware*, objetivando o comparativo de tempo de processamento através da implementação dos filtros em linguagem C e VHDL, bem como ressaltar as características de cada algoritmo para a detecção do complexo QRS.

Os dados utilizados nesta dissertação foram obtidos a partir de uma base de dados do MIT/BIH que é uma das principais bases de validação utilizada pelos pesquisadores de sistemas de análise de ECG.

A base de dados MIT/BIH foi compilada pelo *Massachusetts Institute of Technology* e extraída a partir de um conjunto de mais de 4000 registros de *holter* de longa duração obtidos pelo *Arrhythmia Laboratory* do *Beth Israel Hospital*, no período de 1975 a 1979 (MIT/BIH). Aproximadamente 60% dos registros de ECG da base foram obtidos de pacientes internados, e ao todo, a base está constituída de 48 amostras selecionadas do conjunto coletado, contendo aproximadamente 30 minutos de duração e amostrados na frequência de 360Hz.

As 48 amostras da base de dados são constituídas de 23 registros escolhidos aleatoriamente, identificadas entre 100 e 124. Os demais 25 registros identificados de 200 a 234 também são selecionados a partir do mesmo conjunto, portando uma variedade de fenômenos clínicos importantes, para a validação dos algoritmos de detecção. Os registros foram coletados a partir de 25 homens de faixa etária entre 32 e 89 anos e 22 mulheres de 23 a 89 anos.

Essa diversidade presente no conjunto de dados se torna importante para a avaliação da capacidade de generalização de um algoritmo detector, uma vez que, em registros de longa duração, podem ser observadas alterações de morfologia das ondas do ECG. Os registros foram feitos nas derivações MLII, que é uma modificação da derivação D2 adquirida com eletrodos posicionados no peito do paciente, V5 e V1, que permitem analisar a propagação do impulso elétrico nos planos frontal (MLII) e transversal (V5 e V1).

O primeiro grupo de 23 registros possui um conjunto representativo de formas de onda e artefatos, que um mecanismo de detecção poderá encontrar em um ambiente clínico. O segundo grupo contém amostras com ocorrências de diversas anormalidades de condução.

Muitos desses registros foram selecionados devido às suas características de ritmicidade, variação morfológica do complexo QRS ou qualidade do sinal, que visam validar e conferir dificuldades aos algoritmos de detecção de arritmias cardíacas. Desta forma, ambos os conjuntos oferecem uma base de registros de ECG satisfatória para a validação de algoritmos de detecção de ondas de ECG, tal como os modelos de análise proposto nesta dissertação para a localização do complexo QRS.

Os resultados apresentados nesta seção se referem ao reconhecimento dos complexos QRS utilizando cinco arquivos da base de dados do MIT/BIH, denominados pelos números 100, 105, 108, 203 e 222, de modo a se obter um comparativo com o trabalho de Shukla e Macchiarulo (2008). Fez-se uso do *software* Matlab para a validação dos algoritmos e em seguida os algoritmos foram implementados em linguagem C através do microcontrolador MicroBlaze e em descrição de *hardware* através da linguagem VHDL. Os resultados obtidos, como eram de se esperar, foram os mesmos, sendo que a velocidade de processamento foi maior no processamento paralelo, com os filtros digitais codificados em VHDL.

Uma vez que não existe um algoritmo com uma eficácia de 100% para a detecção do complexo QRS, torna-se importante salientar as potencialidades e deficiências de cada algoritmo utilizado e para quais sinais os algoritmos tiveram melhor resultado. Outro ponto importante deste trabalho foi comparar os tempos de processamento através da implementação serial e paralela e comprovar a potencialidade e as vantagens em se trabalhar com implementação em *hardware*.

4.1 Algoritmo proposto por Pan & Tompkins

Conforme descrito na seção 3.1.1, o algoritmo proposto por Pan & Tompkins é composto por um banco de filtros que ressalta o complexo QRS e define limiares que auxiliam na detecção do mesmo.

Segue abaixo um trecho do sinal 100 – derivação MLII - seguido dos resultados dos diversos filtros da fase de pré-processamento. Observa-se a ocorrência dos complexos QRS representados pela letra "R", possuindo neste trecho um total de 21 complexos QRS.

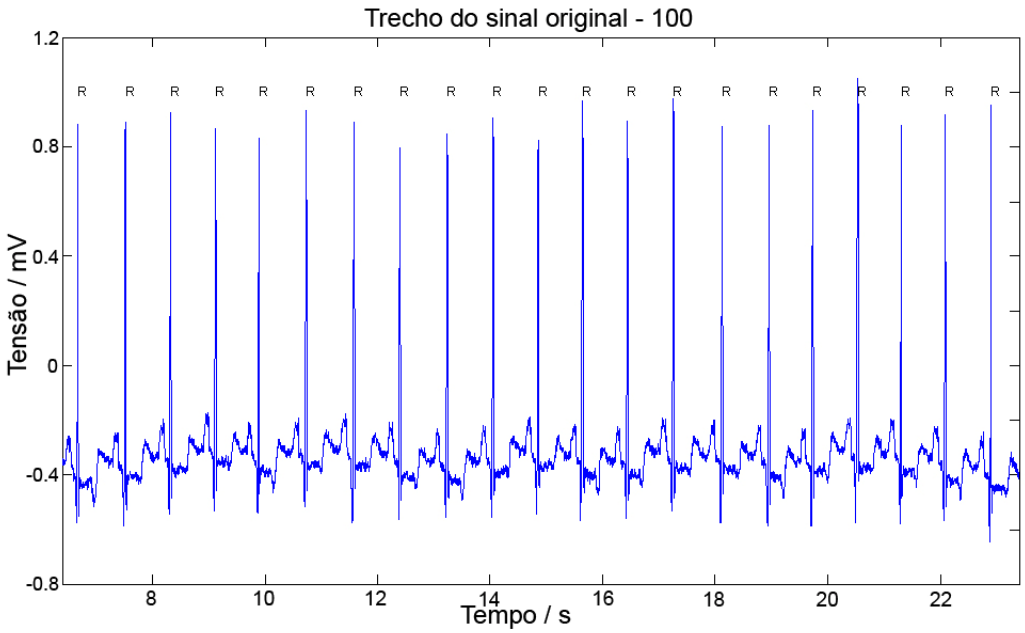

Figura 4.1: Trecho do sinal 100 da base de dados MIT-BIH

As diversas etapas do processamento estão ilustradas nos gráficos abaixo, onde o gráfico superior representa o trecho do sinal original e os seguintes os resultados do filtro passa-baixa, passa-alta, derivativo, função ao quadrado e integrador janela móvel, respectivamente.

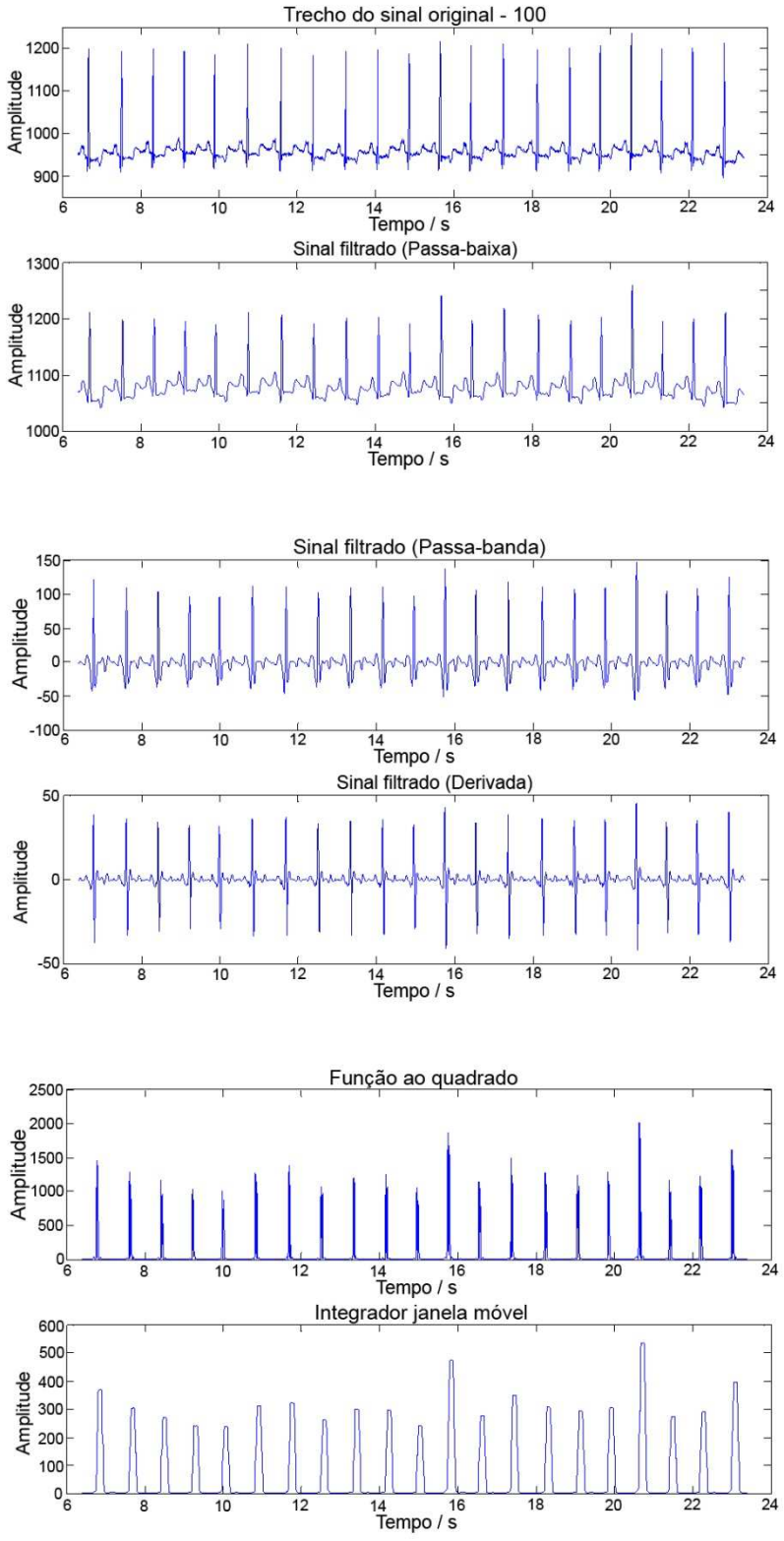

Figura 4.2: Etapas de pré-processamento do algoritmo de Pan & Tompkins

Após a etapa de pré-processamento, o resultado é processado pelo segundo estágio, que define os limiares a partir do qual são considerados prováveis complexos QRS. A Figura 4.3 ilustra as detecções, sendo as letras "R" anotações que representam as verdadeiras localizações dos complexos QRS. Observa-se um pequeno retardo na detecção em virtude dos atrasos dos diversos filtros.

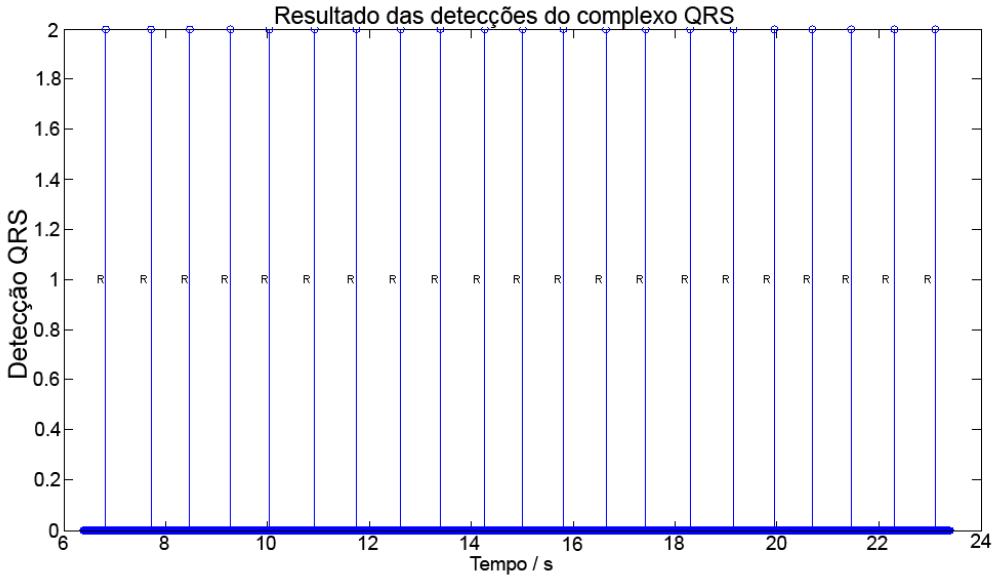

Figura 4.3: Resultado do algoritmo de Pan & Tompkins

Para a implementação no FPGA Spartan-3E, foi necessária a transferência do sinal de ECG para a memória DDR SDRAM do kit de desenvolvimento. Esta transferência foi feita através da comunicação serial RS-232, utilizando o *Hyper Terminal* do ambiente Windows. Todo o controle da transmissão, processamento e recepção dos dados foram feitos pela interface serial através de comandos enviados do teclado do computador para o FPGA. Para mais detalhes, consultar o Apêndice A.

Conforme citado anteriormente, foram utilizados neste trabalho cinco arquivos da base de dados do MIT/BIH, sendo que cada arquivo apresenta duas derivações. Todos os cinco arquivos apresentam em comum o sinal da derivação MLII, sendo que V5 é a outra derivação do arquivo 100 e V1 a outra derivação dos arquivos 105, 108, 203 e 222. Os resultados obtidos foram comparados com as anotações e, como existe uma diferença da localização da ocorrência do complexo QRS entre as anotações e o resultado do algoritmo, foi definido um intervalo válido da

ocorrência do complexo QRS como sendo de 10 amostras da posição reportada pelas anotações como sendo o verdadeiro pico para os sinais amostrados a 200Hz e 18 amostras para os sinais amostrados em 360Hz. Assim, o intervalo total, para os sinais a 200 e 360Hz corresponde a 20 e 36 amostras, respectivamente, sendo o centro do intervalo o local reportado pelas anotações, conforme ilustra a Figura 4.4.

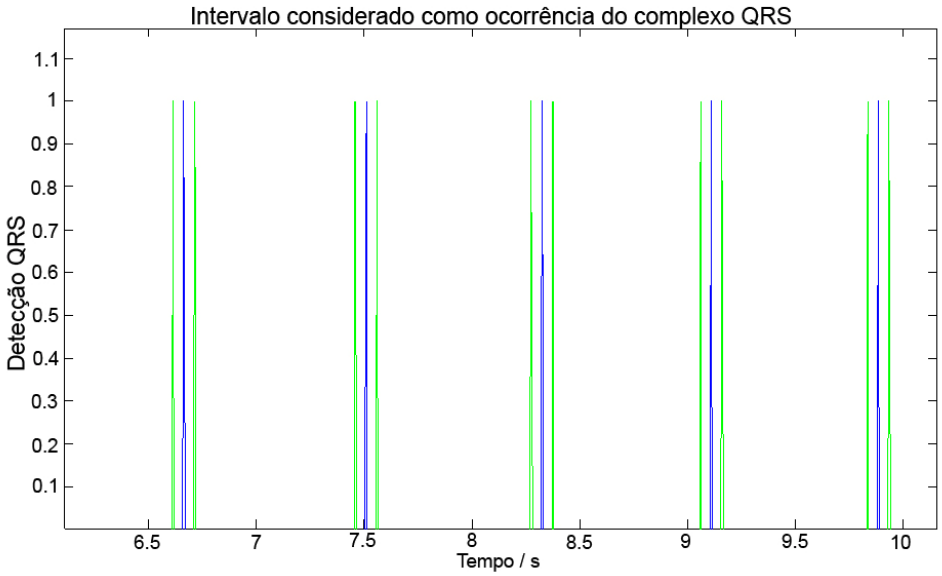

Figura 4.4: Gráfico azul: localização das anotações; Gráfico verde: intervalo considerado como localização do complexo QRS

Foi realizada uma análise quantitativa dos resultados confrontando as detecções realizadas pelo algoritmo implementado e os arquivos de anotações da base de dados do MIT/BIH que indicam o instante da ocorrência do pico da onda R.

A Tabela 4.1 exibe a detecção de batimentos cardíacos através das ondas R dos cinco sinais analisados, sendo possível fazer algumas análises mais detalhadas do ponto de vista estatístico. Os resultados podem ser classificados como verdadeiros positivos (detecções corretas), falsos positivos (acusação de um batimento de forma incorreta) e falsos negativos (falha na detecção de um batimento existente). Obtiveram-se resultados diferentes das duas derivações de cada arquivo. Assim, os dados exibidos referem-se ao melhor resultado entre as duas derivações de cada arquivo.

Tabela 4.1: Detecções dos diversos arquivos por Pan & Tompkins

Arquivo / Derivação	Número total de batimentos	Falso Positivo (FP)	Falso Negativo (FN)	Falhas de detecção (batimentos)	Porcentagem de falhas de detecção
100 / MLII	2273	0	1	1	0%
105 / MLII	2572	81	29	110	4,3%
108 / V1	1763	92	42	134	7,6%
203 / MLII	2980	43	181	224	7,5%
222 / V1	2483	0	6	6	0,2%
TOTAL	**12071**	**216**	**259**	**475**	**3,9%**
$\%\ \boldsymbol{de\ falhas} = \frac{\boldsymbol{falhas\ de\ detecção}}{\boldsymbol{total\ de\ batimentos}}$					

Durante a implementação dos filtros, uma determinada quantidade de componentes do FPGA foi utilizada. Segue abaixo os recursos de *hardware* utilizados para a implementação do algoritmo de Pan & Tompkins tanto em linguagem C quanto em VHDL.

Como ambas as implementações (C e VHDL) utilizam recursos em conjunto, tais como interface serial, controlador de memória, barramentos, etc, optou-se por exibir os componentes utilizados para a implementação do microcontrolador MicroBlaze no processamento em linguagem C e acrescentando para o processamento em VHDL o bloco dos filtros, pois apenas estes blocos são alterados quando se implementam os dois algoritmos.

A Tabela 4.2 representa os componentes utilizados para a implementação do microcontrolador MicroBlaze em linguagem C.

Tabela 4.2: Recursos do FPGA para implementação do filtro em microcontrolador

Dispositivo selecionado: xc3s500efg320-4	**Microblaze 32 bits soft processor**
Número de slices:	716 de 4656 15%
Número de slice flip flops:	904 de 9312 9%
Número de 4 input LUTs:	1392 de 9312 14%
Número de MULT18X18s:	3 de 20 15%

A Tabela 4.3 representa os componentes utilizados para a implementação do microcontrolador MicroBlaze e o banco de filtros em linguagem VHDL.

Tabela 4.3: Recursos do FPGA para implementação do filtro em VHDL

Dispositivo selecionado: xc3s500efg320-4	**Microblaze 32 bits soft processor**	**Barramento FSL**	**Filtros em VHDL**
Número de slices:	750 de 4656 16%	44 de 4656 1%	1320 de 4656 28%
Número de slice flip flops:	947 de 9312 10%	14 de 9312 0%	1395 de 9312 14%
Número de 4 input LUTs:	1410 de 9312 15%	88 de 9312 1%	2044 de 9312 21%
Número de MULT18X18s:	3 de 20 15%	N/A	3 de 20 15%

Observa-se o uso maior de componentes para a implementação dos filtros em *hardware*, uma vez que se faz necessário o acréscimo dos filtros implementados em *hardware* e do barramento *fast simplex link* para a comunicação com o bloco que representa os filtros.

4.1.1 Comparativo do tempo de processamento

Conforme explicado no item 3.2, existe uma grande diferença nas implementações em linguagem C e VHDL devido às formas de processamento. Mediram-se os tempos de processamento dos cinco sinais, considerando as operações de leitura e escrita dos dados na memória externa DDR SDRAM e observou-se a maior velocidade de processamento para o algoritmo implementado em VHDL, conforme ilustrado na Tabela 4.4.

Tabela 4.4: Comparativo dos tempos de processamento

Arquivo / Derivação	Tempo de processamento (ms)	
	Linguagem C	Linguagem VHDL
100 / MLII	19523	816
105 / MLII	20852	839
108 / V1	19158	813
203 / MLII	20739	832
222 / V1	20109	823

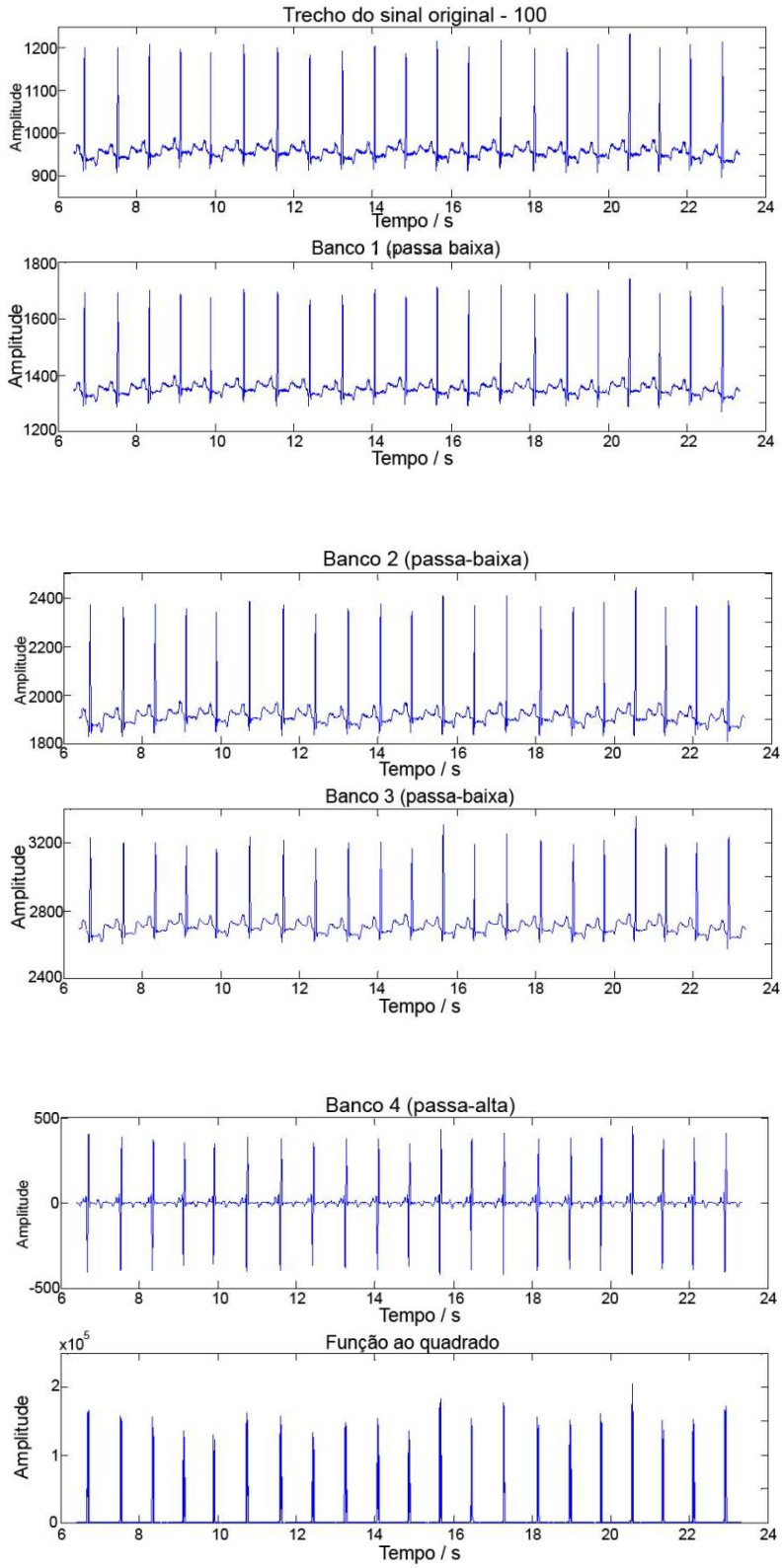

Figura 4.5: Resultado das diversas etapas do banco de filtros da Transformada *Wavelet* de Haar

4.2 Algoritmo utilizando Transformada Wavelet

O algoritmo desenvolvido através da Transformada *Wavelet* de Haar, conforme descrito na seção 3.1.2, é composto por um banco de filtros de forma que, para a faixa de frequências de interesse, utilizou-se o resultado do filtro passa-alta de ordem 4. Este filtro apresenta sinais filtrados na faixa de frequência de 11,25 a 22,5Hz.

A Figura 4.5 retrata o mesmo trecho do arquivo 100 / MLII utilizado na seção 4.1 para ilustrar o resultado das diversas fases de processamento.

O próximo passo é o estágio de detecção dos picos, onde são definidos limiares a partir do qual são consideradas prováveis localizações do complexo QRS. O gráfico da Figura 4.6 ilustra as detecções dos complexos QRS feitas pelo algoritmo e as anotações ilustradas com a letra "R" representam as verdadeiras localizações dos complexos QRS.

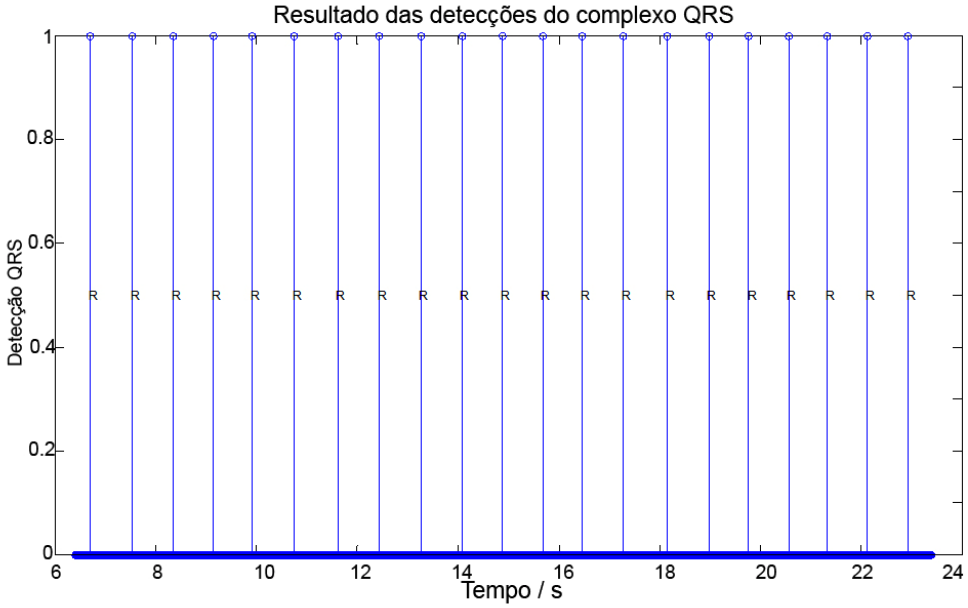

Figura 4.6: Resultado do algoritmo da Transformada *Wavelet* de Haar

Da mesma forma que no item 4.1, os cinco arquivos foram utilizados no processamento, sendo que para este caso, onde a frequência de amostragem foi de 360Hz, a largura do intervalo foi ajustada para 36 amostras. A Tabela 4.5 exibe o resultado das detecções dos diversos arquivos.

Tabela 4.5: Resultado das detecções utilizando a Transformada *Wavelet* de Haar

Arquivo / Derivação	Número total de batimentos	Falso Positivo (FP)	Falso Negativo (FN)	Falhas de detecção (batimentos)	Porcentagem de falhas de detecção
100 / MLII	2273	1	0	1	0%
105 / MLII	2572	73	27	100	3,9%
108 / V1	1763	78	121	199	11,3%
203 / MLII	2980	62	198	260	8,7%
222 / V1	2483	1	45	46	1,9%
TOTAL	**12071**	**215**	**391**	**606**	**5%**
$\%\ \boldsymbol{de\ falhas} = \frac{\boldsymbol{falhas\ de\ detecção}}{\boldsymbol{total\ de\ batimentos}}$					

Os recursos utilizados pelo FPGA para a implementação de cada algoritmo são apresentados nas duas tabelas seguintes.

Tabela 4.6: Recursos do FPGA para implementação do filtro em Microblaze

Dispositivo selecionado: xc3s500efg320-4	**Microblaze 32 bits soft processor**
Número de slices:	716 de 4656 15%
Número de slice flip flops:	904 de 9312 9%
Número de 4 input LUTs:	1392 de 9312 14%
Número de MULT18X18s:	3 de 20 15%

Tabela 4.7: Recursos do FPGA para implementação do filtro em VHDL

Dispositivo selecionado: xc3s500efg320-4	**Microblaze 32 bits soft processor**	**Barramento FSL**	**Filtros em VHDL**
Número de slices:	750 de 4656 16%	44 de 4656 1%	971 de 4656 20%
Número de slice flip flops:	947 de 9312 10%	14 de 9312 0%	564 de 9312 6%
Número de 4 input LUTs:	1410 de 9312 15%	88 de 9312 1%	1852 de 9312 19%
Número de MULT18X18s:	3 de 20 15%	N/A	3 de 20 15%

Fazendo um comparativo da utilização do dispositivo entre as implementações em VHDL do algoritmo de Pan & Tompkins e de Haar, observa-se que a Transformada *Wavelet* de Haar apresentou um melhor resultado, visto que utilizou uma menor quantidade de *slices*, *flip-flops* e *LUTs*. Já a implementação em linguagem C, como era de se esperar, utilizou a mesma quantidade de recursos do FPGA.

4.2.1 Comparativo de tempo de processamento

Da mesma forma que no item 4.1.1, os tempos de processamento em VHDL e linguagem C foram tomados, conforme ilustra a Tabela 4.8.

Tabela 4.8: Comparativo do tempo de processamento para o algoritmo da TW de Haar

Arquivo / Derivação	Tempo de processamento (ms)	
	Linguagem C	Linguagem VHDL
100 / MLII	6175	1411
105 / MLII	6944	1421
108 / V1	8134	1408
203 / MLII	8010	1429
222 / V1	6284	1414

Se comparado o tempo de processamento em linguagem VHDL dos dois algoritmos, observa-se um tempo maior no processamento da Transformada *Wavelet* em virtude da frequência de amostragem que é de 360Hz, se comparado com a frequência de amostragem de 200Hz para o processamento do algoritmo de Pan & Tompkins.

4.3 Comparativo dos resultados das detecções

Nesta seção serão analisados trechos dos arquivos utilizados neste trabalho com o intuito de comparar e levantar as potencialidades e deficiências de cada algoritmo.

O arquivo 100 apresenta poucos problemas de interferência, com poucas variações da linha de base e poucos artefatos de movimento. Dessa forma, nota-se a grande eficácia dos dois algoritmos, apresentando 0% de falhas nas detecções. Segue na Figura 4.7 a ilustração das

detecções dos dois algoritmos, onde se observa no gráfico em vermelho o arquivo 100-MLII, o sinal em verde representa o intervalo para o qual é considerado como verdadeiro a ocorrência do complexo QRS, o sinal em azul o resultado do algoritmo de Pan & Tompkins e o em preto o resultado do algoritmo da Transformada *Wavelet* de Haar. Esta ordem dos sinais e suas respectivas cores serão utilizadas em todos os gráficos seguintes desta seção.

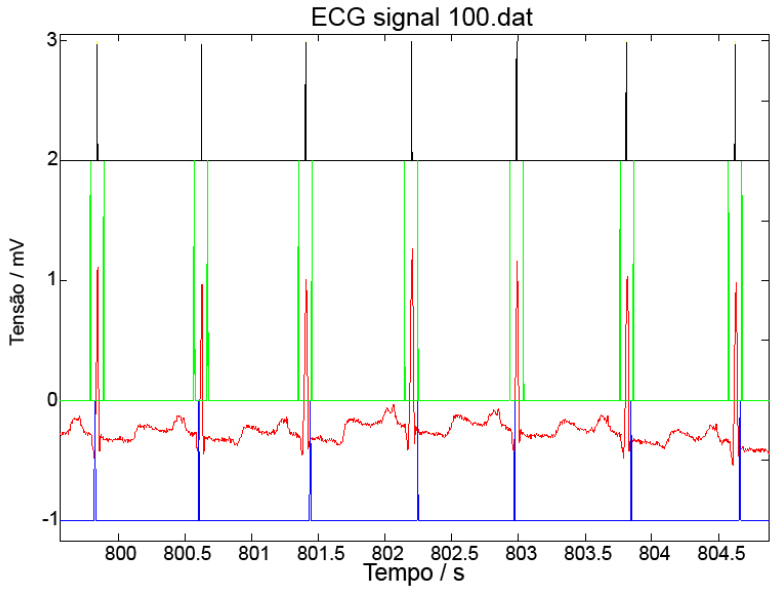

Figura 4.7: Local de ocorrência dos complexos QRS e as detecções dos dois algoritmos utilizados

Já o arquivo 105 apresenta trechos com elevado ruído provenientes de artefatos de movimento. Observou-se experimentalmente que para este sinal, o algoritmo da TW de Haar apresentou melhor resultado, pois durante ruído proveniente de artefatos de movimento, o algoritmo suprimiu de maneira mais satisfatória estas variações.

Outra observação foi que o algoritmo da TW de Haar se mostrou mais preciso nas detecções, uma vez que não é utilizado o integrador de janela móvel. A Figura 4.8 ilustra um trecho deste sinal, onde é possível observar a diferença da precisão mencionada acima.

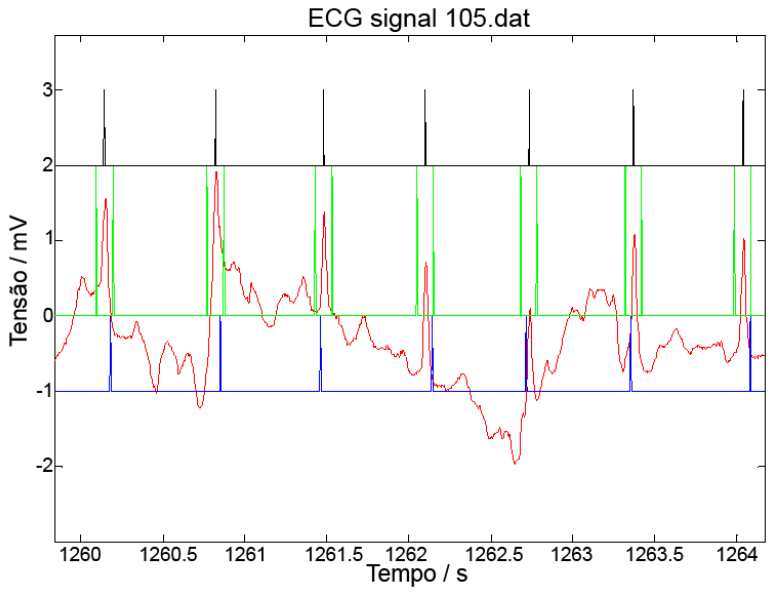

Figura 4.8: Resultado da detecção dos dois algoritmos de um trecho do arquivo 105-MLII

Já na Figura 4.9 observa-se a ocorrência de dois falsos-positivos e dois falsos-negativos pelo algoritmo de Pan & Tompkins e um falso-positivo pelo algoritmo da TW de Haar.

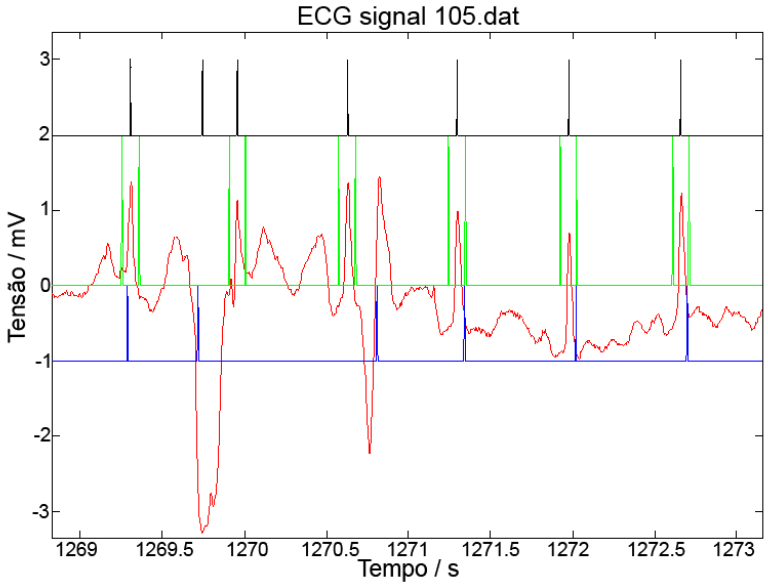

Figura 4.9: Trecho do arquivo 105 mostrando FP e FN.

A fim de entender melhor os falsos positivos da Figura 4.9, observa-se que para o tamanho de 30 amostras utilizado para a função integrativa do algoritmo de Pan & Tompkins, o resultado

apresenta um amortecimento considerável e, em conseqüência disso, pontos onde ocorre o complexo QRS acabam sendo mascarados por grandes variações do sinal em virtude de ruídos de artefatos de movimento. Já a resposta do algoritmo da TW de Haar é mais rápida e menos susceptível a artefatos de movimento e, em conseqüência disso, melhor são os resultados. A Figura 4.10 ilustra as respostas da fase de pré-processamento dos dois algoritmos.

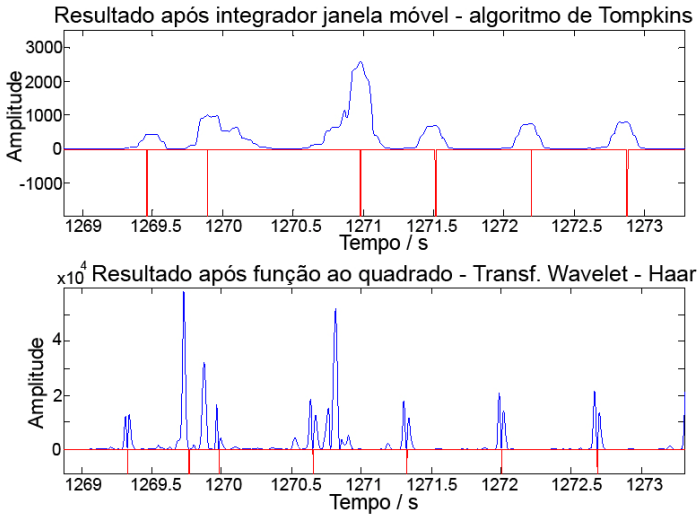

Figura 4.10: Trecho do último estágio do pré-processamento do arquivo 105

Outro trecho do mesmo arquivo 105 é mostrado na Figura 4.11, onde se observam sinais que se confundem com complexos QRS e ambos os algoritmos apresentaram falsos positivos.

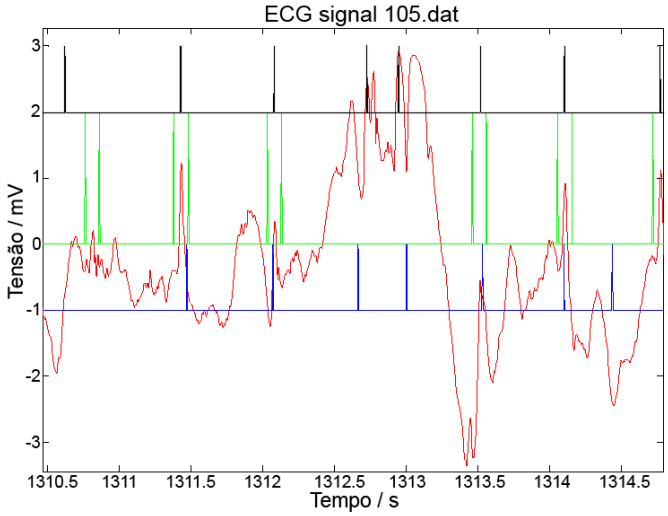

Figura 4.11: Ocorrência de FP e FN por ambos os algoritmos

O trecho abaixo ilustra o sinal sem ruído e sem variação da linha de base e a boa eficácia dos algoritmos.

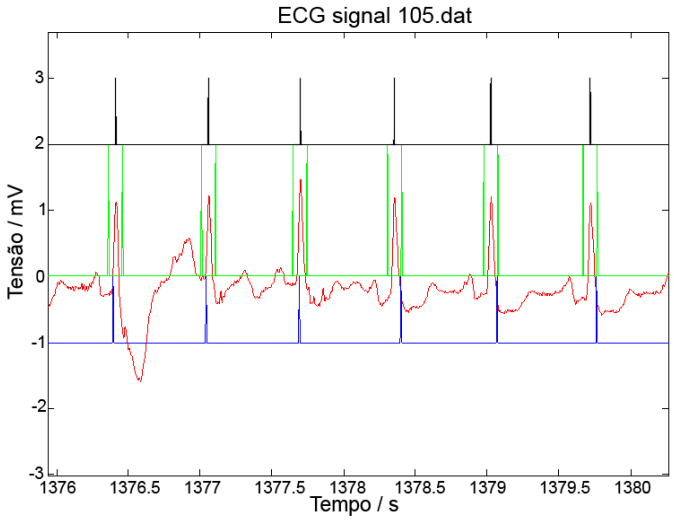

Figura 4.12: Trecho com 100% de eficácia nas detecções

O arquivo 108 apresenta considerável ruído e variação na linha de base, sendo que a maior eficácia dos algoritmos se deu para a derivação V1, pois a derivação MLII apresenta considerável amplitude da onda P e, conseqüentemente, uma grande quantidade de falsos-positivos e negativos. Para este arquivo, o algoritmo de Pan & Tompkins se mostrou mais satisfatório, com 7,6% de falhas de detecção, contra 11,3% do algoritmo utilizando a Transformada *Wavelet* de Haar. A Figura 4.13 ilustra um trecho do sinal da derivação MLII e as detecções dos algoritmos.

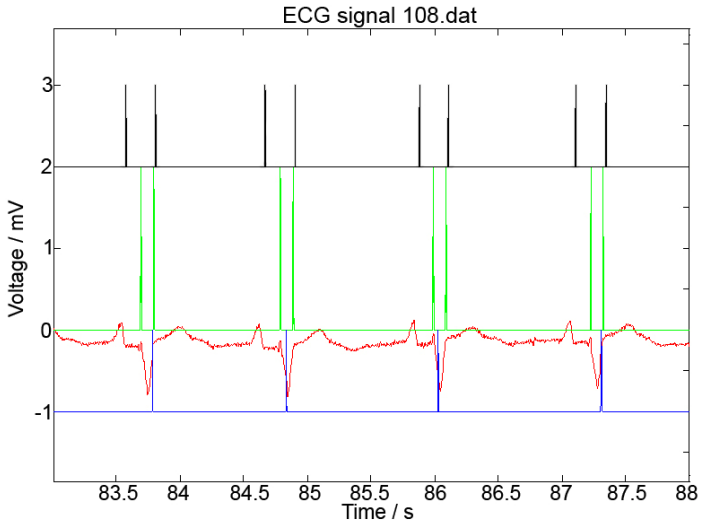

Figura 4.13: Ondas P detectadas como sendo complexo QRS pelo algoritmo da TW de Haar

Observa-se na Figura 4.13 que o resultado da Transformada *Wavelet* de Haar detectou as ondas P como sendo complexos QRS. Veja na Figura 4.14 o gráfico referente à função ao quadrado. O *threshold* está abaixo da onda de maior amplitude referente à onda P. Assim, o final da onda P é detectada como sendo o complexo QRS e em virtude do período refratário, o verdadeiro complexo QRS é detectado apenas na segunda onda (onda maior) e o resultado se apresenta fora do intervalo como um falso-positivo.

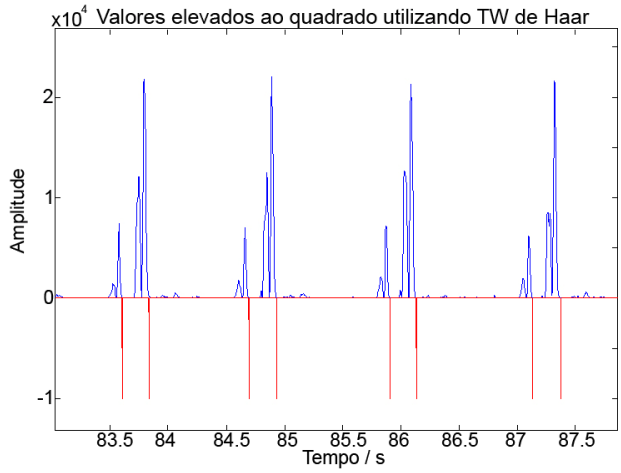

Figura 4.14: Último estágio de pré-processamento da TW de Haar mostrando as detecções das ondas P

A derivação V1, utilizada como análise do arquivo 108, apresenta pontos de ruídos de alta frequência e em virtude da amplitude considerável, acaba interferindo no resultado final. Observa-se na Figura 4.15 alguns falsos-positivos nos dois algoritmos, sendo mais expressivo no algoritmo de Haar.

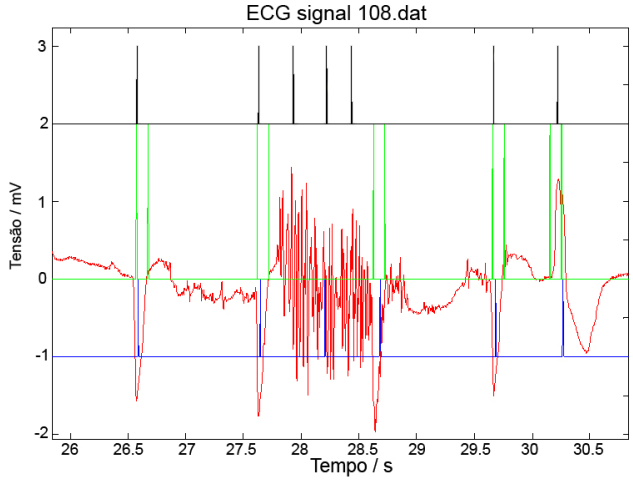

Figura 4.15: Trecho do arquivo 108 com alto grau de ruído

O bloco referente à função integrativa do sinal se mostra bastante eficiente para este tipo de ruído, uma vez que amortiza o sinal e ressalta apenas o complexo QRS. Já o algoritmo de Haar exibe de forma relevante estas energias indesejáveis, conforme ilustrado na Figura 4.16.

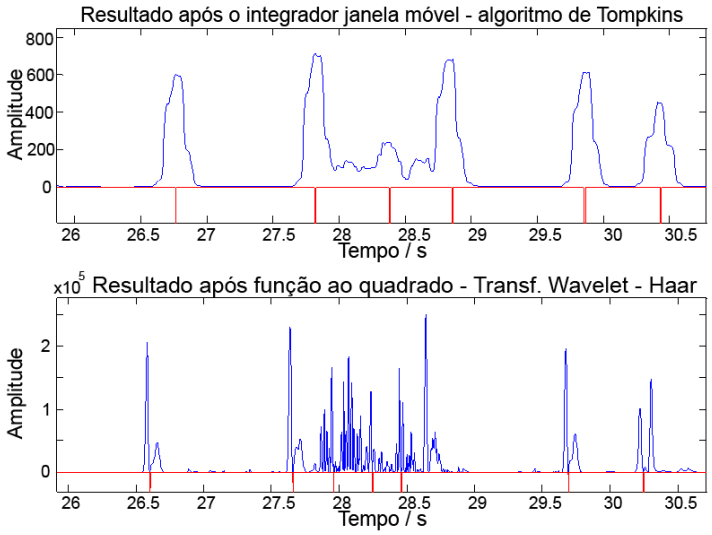

Figura 4.16: Última etapa de pré-processamento ilustrando o trecho do sinal 108 com expressivo ruído

Observou-se que para leves alterações na linha de base, os dois algoritmos responderam bem às detecções, conforme mostra a Figura 4.17.

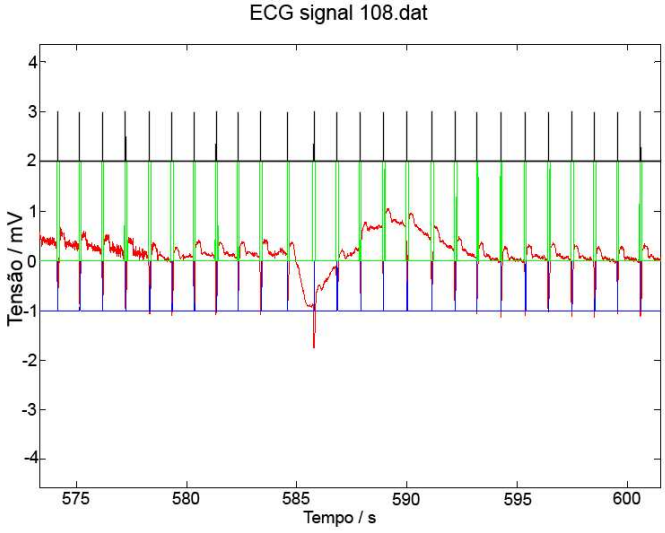

Figura 4.17: Leve alteração da linha de base de um trecho do arquivo 108

No entanto, para alterações mais significativas na linha de base, o algoritmo de Pan & Tompkins se mostrou mais robusto, conforme ilustra a Figura 4.18 com quatro falsos-negativos apresentados pelo algoritmo da TW de Haar contra apenas um apresentado pelo algoritmo de Pan & Tompkins.

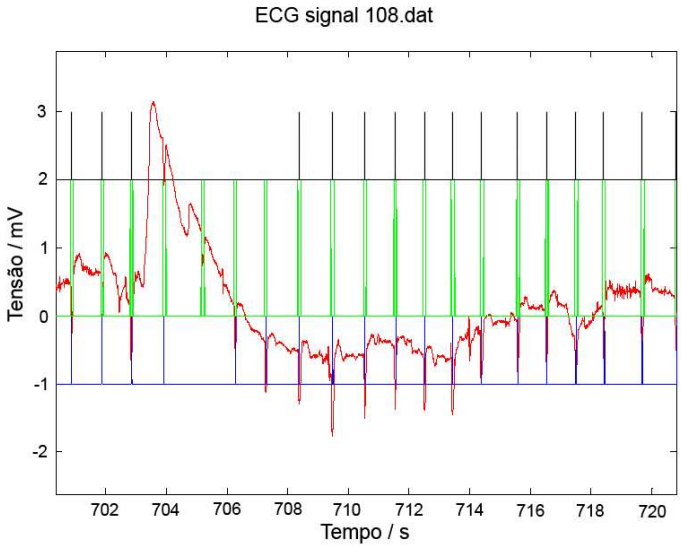

Figura 4.18: Alteração brusca da linha de base de um trecho do arquivo 108

O arquivo 203 apresenta na derivação MLII alterações na morfologia do QRS, ruído muscular e alterações na linha de base. É considerado pelos especialistas como um sinal de difícil análise. Segue abaixo na Figura 4.19 um trecho do sinal e as detecções dos algoritmos.

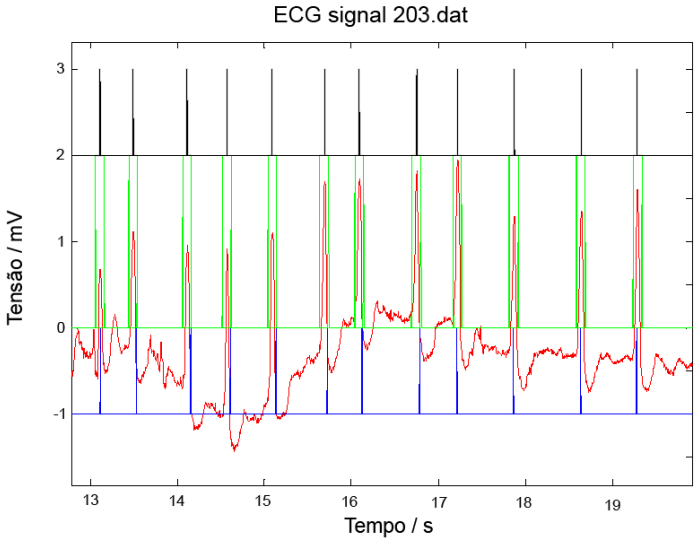

Figura 4.19: Trecho do sinal 203 e as localizações dos complexos QRS pelos dois algoritmos

Observa-se na Figura 4.20 uma grande quantidade de falsos-negativos apresentados pelos dois algoritmos. Isso ocorreu, pois mesmo sendo adaptativos os *thresholds* dos dois algoritmos, as variações bruscas do sinal resultavam em baixas energias em determinados instantes, acarretando grande quantidade de falsos-negativos.

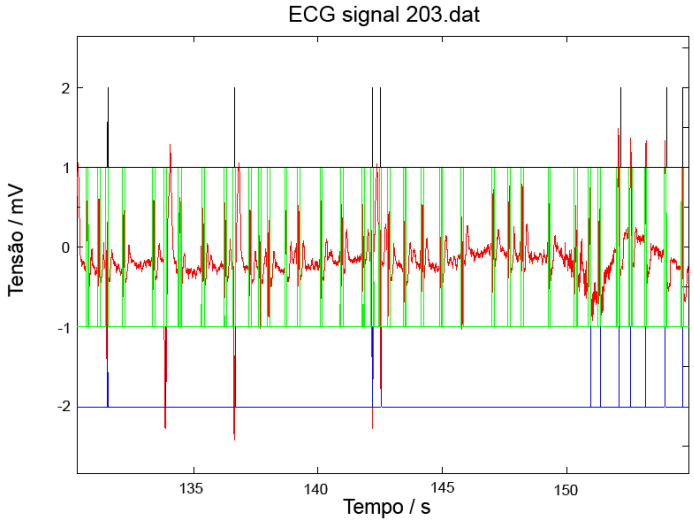

Figura 4.20: Trecho do arquivo 203 com presença de falsos-negativos

O arquivo 222 apresenta em vários pontos ruídos de alta frequência e alguns artefatos de movimento. Ambos os algoritmos apresentaram bons resultados, sendo que o algoritmo de Tompkins apresentou resultado melhor. Segue abaixo um trecho do sinal com verdadeiros-positivos pelos dois algoritmos.

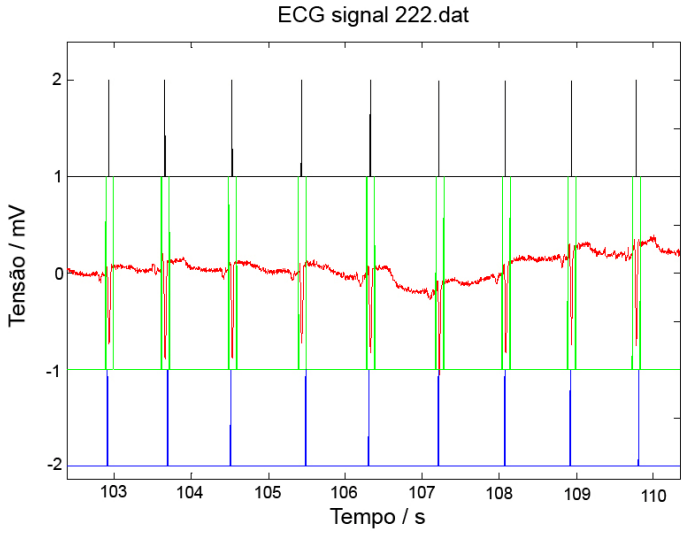

Figura 4.21: Trecho do arquivo 222 com presença de verdadeiros-positivos

Segue abaixo um trecho onde o algoritmo da TW de Haar acusou dois falsos-negativos no instante da diminuição da amplitude do complexo QRS.

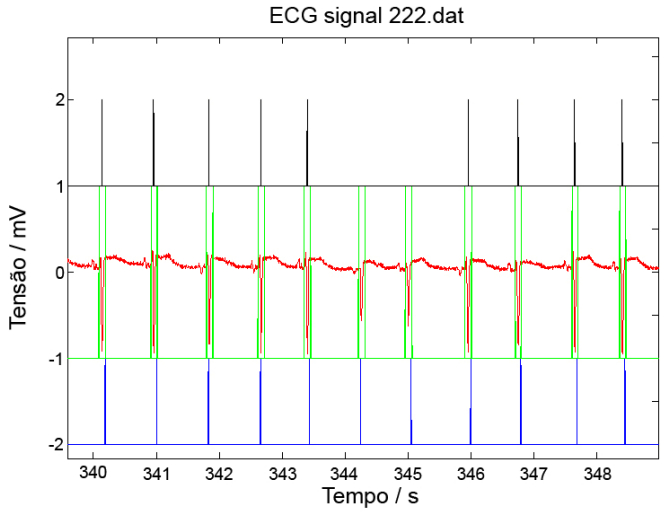

Figura 4.22: Falsos-negativos apresentados pelo algoritmo da TW de Haar

Segue abaixo outro trecho do arquivo 222 que apresentou artefato de movimento com grande variação da linha de base e falso-negativo pelos dois algoritmos.

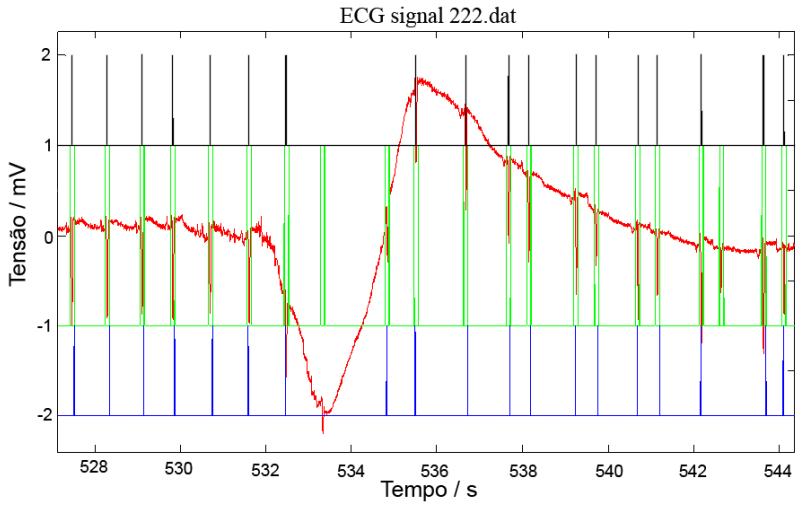

Figura 4.23: Trecho do arquivo 222 com grande variação da linha de base

Capítulo 5

5 Conclusões

5.1 Considerações deste trabalho

Este trabalho teve o objetivo de investigar dois algoritmos para o processamento de sinais cardíacos e detecção do complexo QRS com relação à eficácia dos mesmos e tempos de processamento utilizando duas metodologias diferentes. A utilização de FPGA para o processamento dos sinais cardíacos foi motivada pela flexibilidade que esta ferramenta possui em se trabalhar com diversos blocos independentes e conseqüentemente um baixo tempo de processamento, podendo ser muito útil em equipamentos que exigem respostas rápidas.

É importante salientar que este trabalho possibilitou a investigação de processamento digital de sinais e projeto de arquiteturas de *hardware* configuráveis. Na área de processamento digital de sinais, foi necessário fazer uma revisão da literatura e um estudo das técnicas existentes para detecção do complexo QRS, onde não se observou a existência de um algoritmo com 100% de eficácia. Com relação aos dispositivos lógicos programáveis, confirmou-se a importância cada vez maior dessa área no projeto de sistemas atuais e uma das grandes dificuldades foi resolver o problema da restrição de *hardware*, uma vez que quem trabalha apenas com programação de *software* não encontra essa dificuldade.

Outro obstáculo foi o limite do dispositivo utilizado, uma vez que para os filtros implementados em VHDL, o projeto ultrapassou a capacidade do *hardware*, havendo a necessidade de se aplicar algumas manipulações de programação para contornar o problema, como os coeficientes dos filtros da TWE que possuem 4 casas decimais (0,7071). Neste caso, conseguiu-se economizar multiplicadores do dispositivo transformando este número em uma

série de números fracionários com os denominadores na potência de dois, realizando a divisão através do deslocamento do registrador.

Vale ressaltar que o tempo de processamento foi o único a ser levado em conta, não se preocupando com os tempos para a transmissão e recepção dos dados.

Os resultados obtidos demonstram que o algoritmo utilizando a Transformada *Wavelet* apresenta bons resultados e diferencia do algoritmo de Pan & Tompkins em alguns comportamentos específicos do sinal de ECG. Para os cinco arquivos utilizados neste trabalho, o algoritmo utilizando a TWE foi mais satisfatório para o arquivo 105, que apresenta ruídos provenientes de artefatos de movimento, sendo que para os arquivos 108, 203 e 222, o algoritmo proposto por Pan & Tompkins obteve melhores resultados. O estudo de outras *wavelets* mãe pode trazer resultados mais satisfatórios com relação à eficácia do algoritmo, como as Cubic e Quadratic Spline e Db3 (DINH, KUMAR, PAH, BURTON, 2001).

Os complexos QRS geralmente têm amplitudes maiores do que as amplitudes das demais ondas do ECG. Desta forma, muitos trabalhos científicos e produtos comerciais de aplicações reais utilizam algoritmos de detecção dos complexos QRS para obter referências no sinal de ECG a partir das quais as técnicas de análise e de classificação dos segmentos de ECG são aplicadas. Portanto, a detecção dos complexos QRS é a etapa mais importante no processamento de sinais de ECG porque a qualidade de todas as demais etapas de análise e de classificação dos sinais de ECG depende desta.

Foi possível entender a enorme dimensão que envolve o projeto de um detector digital de complexos QRS, considerando a complexidade das diversas técnicas de processamento digital de sinais, bem como as ferramentas necessárias para se trabalhar com os sistemas digitais.

É possível afirmar que os resultados obtidos através do estudo nos ramos da medicina, matemática e computação atendem as metas estabelecidas no início do trabalho.

5.2 Trabalhos futuros

Como sugestões de trabalhos futuros, poder-se-ia dar continuidade a essa linha de pesquisa com a investigação de outras famílias de funções *wavelets*, bem como a utilização de outros arquivos de sinais de eletrocardiograma e o refinamento das técnicas de limiares adotados.

Além da detecção do complexo QRS, outras ondas do ECG podem ser detectadas, assim como os diversos intervalos e segmentos. Assim, este trabalho pode ser uma referência para trabalhos futuros nesta área, avançando ainda mais para análise de determinados grupos de cardiopatias e tomadas de decisão.

A flexibilidade que o FPGA dá para a implementação de diversos blocos independentes abre espaço para a implementação de vários algoritmos em um mesmo *chip* e a decisão da ocorrência ou não do complexo QRS através de técnicas de inteligência artificial.

6 Referências Bibliográficas

ABBAS, R. Prediction of Ventricular Tachyarrhythmia in Electrocardiograph Signal using Neuro-Wavelet approach. **National Conference on Emerging Technologies**, pp 82-87, 2004.

ACHARYA, U.R.; SURI, J.S.; SPAAN, J.A.E.; KRISHNAN S.M. **Advances in Cardiac Signal Processing.** Berlin: Springer, 2007.

ADDISON, P.D. Wavelet Transforms and the ECG: A Review. **Physiological Measurements,** vol.25, pp 155-199, 2005.

AFONSO, V.X.; TOMPKINS, W.J.; NGUYEN, T.Q.; LUO, S. ECG Beat Detection Using Filter Banks. **IEEE Transactions on Biomedical Engineering**, vol.46, N.2, pp 192-202, 1999.

AKAY, M. **Time Frequency and Wavelets in Biomedical Signal Processing.** New York: IEEE Press, 1998.

ALTERA Co.; Altera Home Page. Disponível em <http://www.altera.com> . Acesso em Outubro de 2009.

ATANASOV, N. Isolation of Systolic Heart Murmurs Using Wavelet Transform and Energy Index. **Congress on Image and Signal Processing**, vol.1, pp 216-220, 2008.

BRONZINO, J.D. **The Biomedical Engineering HandBook**. CRC Press, 2000.

BROWN, S.; VRANESIC, Z. **Fundamentals of Digital Logic with VHDL Design.** Second Edition. New York: McGraw-Hill, 2005.

BUENO, N.M. **Classificação automática de cardiopatias baseada em eletrocardiograma.** 2006, 98p. Dissertação (Mestrado) – Faculdade de Engenharia Elétrica – Universidade Federal de Uberlândia, Uberlândia.

CARVALHO, J.L.A. **Ferramenta para análise tempo-frequêncial da variabilidade da freqüência cardíaca.** 2003, 97p. Dissertação (Mestrado) – Faculdade de Tecnologia – Universidade de Brasília, Brasília.

CHAN, P.K.; MOURAD, S. **Digital Design using Field Programmable Gate Arrays**. Prentice Hall, 1994.

CINTRA, E.R.F. **Diagnóstico de cardiopatias baseado no reconhecimento de padrões pelo método de correlação**. 2006. Dissertação (Mestrado) – Universidade Federal de Itajubá, Itajubá.

CLIFFORD, Gari D. **Signal processing methods for heart rate variability.** 2002, 244p. Tese (Doutorado) – University of Oxford.

DALE, D. **Interpretação rápida do ECG.** Tradução de Ismar Chaves da Silveira. Publicações científicas, 2004.

DINH, H.A.N.; KUMAR, D.K.; PAH, N.D.; BURTON, P. Wavelets for QRS Detection. **Proceedings – 23rd Annual Conference – IEEE/EMBS** - Istanbul, pp 25-28, 2008.

FRIESEN, G.M.; JANNETT, T.C.; JADALLAH, M.A.; YATES, S.L.; QUINT, S.R.; NAGLE, H.T. A Comparison of the Noise Sensitivity of Nine QRS Detection Algorithms. **IEEE Transactions on Biomedical Engineering**, vol.37, N.1, pp 85-98, 1990.

GARCIA, Euler V. **Processamento de sinais usando wavelets para caracterização da repolarização ventricular durante hipoglicemia.** 2005, 205p. Tese (Doutorado) – Universidade Federal de Santa Catarina, Santa Catarina.

GONSALES, Alex D. **Projeto de uma nova arquitetura de FPGA para aplicações BIST e DSP.** 2002, 108p. Dissertação (Mestrado) – Instituto de Informática – Universidade Federal do Rio Grande do Sul, Porto Alegre.

GUTIÉRREZ, A.; HERNÁNDEZ, P.R.; LARA, M.; PÉREZ, S.J. A QRS Detection Algorithm Based on Haar Wavelet. **Computers in Cardiology.** Vol.25, 1998

HAMILTON P.; TOMPKINS W. Quantitative Investigation of QRS Detection Rules Using the MIT/BIH Arrythmia Database, **IEEE Transactions on Biomedical Engineering**, vol. BME-33, N.12, pp 1157-1165, 1986.

HAYES, M.H. **Digital Signal Processing.** New York: McGraw-Hill, 1999.

HAYKIN, S.; VEEN, B.V. **Sinais e Sistemas.** Tradução – José Carlos Barbosa dos Santos. Porto Alegre: Bookman, 2001.

HSU H.P. **Análise de Fourier**. Rio de Janeiro: Livros Técnicos e Científicos, 1973.

IEONG, C.I.; VAI, M.I.; MAK, P.U. QRS Recognition with Programmable *Hardware*. **Bioinformatics and Biomedical Engineering**, pp 2028-2031, 2008.

JENNINGS, D; FLINT, A.; TURTON, B.C.H.; NOKES, L.D.M. **Introduction to Medical Electronics Applications**. London: Edward Arnold, 1995.

KADAMBE, S.; MURRAY, R.; BOUDREAUX-BARTELS, F. Wavelet Transform-Based QRS Complex Detector. **IEEE Transactions on Biomedical Engineering**, vol. 46, N.7, pp 838-848, 1999.

LENGYEL, L. **Eletrocardiografia Clínica**. São Paulo: Sarvier, 1974.

LI, C.; ZHENG, C.; TAI, C. Detection of ECG Characteristic Points Using Wavelet Transforms. **IEEE Transactions on Biomedical Engineering**, vol. 42, N.1, pp 21-28, 1995.

LIN, C.H. Adaptive Wavelet Network for Multiple Cardiac Arrhythmias Recognition. **Expert Systems with Applications**, pp 2601-2611, 2008.

MALLAT, S. **A wavelet tour of signal processing.** New York: Academic Press, 1999.

MANZAN, William A. **Utilização das transformadas wavelets na extração de características e no reconhecimento de padrão em um sinal de ECG.** 2006, 96p. Dissertação (Mestrado) – Universidade Federal de Uberlândia, Uberlândia.

MAX, C.M. **The Design Warrior's Guide to FPGAs.** Burlington: Newnes, 2004.

MELCO, Tito C. **Estudo do eletrocardiograma sob uma abordagem matemática.** 2006, 100p. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo, São Paulo.

MIT/BIH Arrhythmia database. Disponível em: HTTP://www.physionet.org/physiobank/database/mitdb. Acesso em Novembro de 2009.

MOHRMAN, D.E.; HELLER, L.J. **Cardiovascular Physiology**. McGraw-Hill, 2006.

MORETTIN, P.A. **Ondas e Ondaletas.** São Paulo: EdUSP, 1999.

MORRIS, F.; EDHOUSE, J.; BRADY, W.; CAMM, J. **ABC of Clinical Electrocardiography**. London: BMJ Books, 2003.

MORRIS, F.; EDHOUSE, J.; BRADY, W.; CAMM, J. **ABC of clinical electrocardiography.** London: BMJ Books, 2003. 80p.

PADELETTI, L. Digital Technology in Cardiac Pacing. Methods for Morphology Analysis of Sensed Endocavitary Signals. **Journal of Interventional Cardiac Electrophysiology**, pp 9-16, 2005.

PAN, J.; TOMPKINS, W.J. A Real-Time QRS Detection Algorithm. **IEEE Transaction on Biomedical Engineering,** vol.32, pp 230-236, 1985.

PEDRONI, V.A. **Circuit Design with VHDL.** London: MIT Press, 2004. 376p.

PETRONE, J. **Adaptive filter architectures for FPGA implementation.** 2004, 96p. Dissertação (Mestrado) – College of Engineering – The Florida State University.

PORTAL DO CORAÇÃO. **Portal do Coração Home Page**. Disponível em: <http://portaldocoracao.uol.com.br/angioplastia-coronariana.php?id=2738>. Acesso em Agosto de 2009.

QIAN, S. **Introduction to Time-Frequency and Wavelet Transforms.** New Jersey: Prentice Hall, 2002.

RIBEIRO, A.A.L. **Reconfigurabilidade dinâmica e remota de FPGAs.** 2002, 151p. Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, São Carlos.

RICCIOTTI, A.C.D. **Utilização de wavelets no processamento de sinais EMG.** 2006, 106p. Dissertação (Mestrado) – Faculdade de Engenharia Elétrica - Universidade Federal de Uberlândia, Uberlândia.

SEMMLOW, J.L. **Biosignal and Biomedical Image Processing.** New Jersey: Marcel Dekker, 2004. 443p.

SHUKLA, A.; MACCHIARULO, L. A Fast and Accurate FPGA based QRS detection System. **30th Annual International IEEE EMBS Conference**, pp 4828-4831, 2008.

SILVA, Murilo. **Implementação de um localizador de faltas híbrido para linhas de transmissão com três terminais baseado na transformada wavelet.** 2008, 237p. Tese (Doutorado) – Escola de Engenharia de São Carlos – Universidade de São Paulo, São Carlos.

STOCO, M.S.; ANDREÃO, R.V.; SEGATTO, M.V. Detecção Automática de Batimentos Cardíacos Utilizando Transformada Wavelet. **VI Workshop de Informática Médica**, v.1,PP 1-4, 2006.

TOMPKINS J.W. **Biomedical Digital Signal Processing**. Prentice Hall, 1993.

XILINX INC.; Xilinx Home Page. Disponível em <http://www.xilinx.com> . Acesso em Novembro de 2009.

Anexo A – Fluxo de projeto

A programação do dispositivo FPGA pode ser feita através de linguagem de descrição de *hardware*, tais como VHDL, Verilog, SystemC, etc, ou mesmo através de modelagem de sistemas (*System Genarator*).

O fluxo tradicional de projeto pode ser dividido em quatro fases distintas: especificação, verificação, implementação e *debug* do sistema, conforme ilustra a Figura A.1.

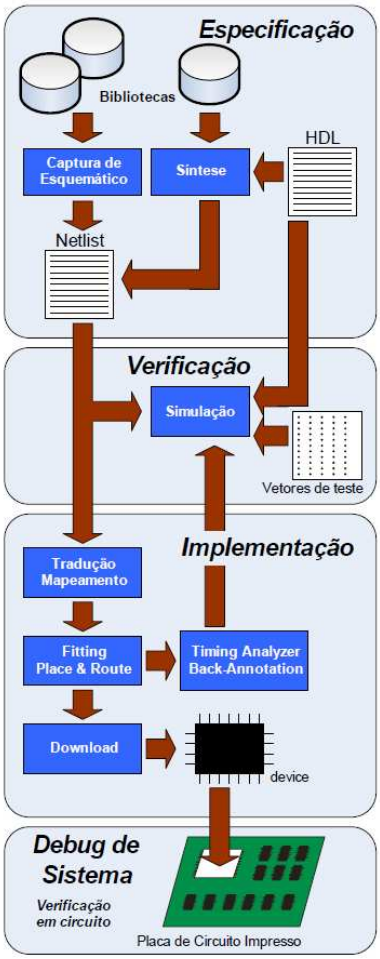

Figura A.1: Fluxo de projeto de um FPGA

Durante a etapa de especificação, o projetista faz um estudo e levanta todos os requisitos e características do sistema e define seu funcionamento. Esta fase é muito importante, pois permite o correto entendimento do sistema, o que evita a ocorrência de erros futuros. A verificação proporciona a validação do sistema, que é feita através de simulações do sistema ou partes dele. Durante a implementação, o modelo descrito será convertido para estruturas de dados representando as conexões, blocos, componentes e portas lógicas. Esta etapa é automática e dependente da ferramenta de *software* utilizada. Ainda durante esta etapa, os requisitos do sistema são pré-avaliados a fim de indicar se o circuito irá atendê-los adequadamente, sendo que o projetista pouco influencia neste processo, apenas podendo especificar parâmetros de otimização desejados.

Uma etapa importante do projeto consiste na especificação ou geração do *netlist*, que é uma descrição compacta, ou mesmo textual, do circuito para as ferramentas de verificação e de implementação de circuitos. O *netlist* é basicamente uma listagem de componentes do circuito e de como estes componentes estão interconectados, incluindo ainda os nomes dos pinos de I/O utilizados pelo circuito. A descrição do circuito realizada pelo *netlist* é dependente do fabricante e da família do dispositivo empregado, uma vez que os componentes utilizados na descrição são provenientes de bibliotecas específicas deste fabricante.

A geração do *netlist* pode ser feita através de captura de esquemático ou de síntese de código HDL (*Hardware Description Language*), como o VHDL ou Verilog. Neste caso, deve ser disponibilizada uma ferramenta de síntese que interprete o código HDL e gere um *netlist* otimizado em área ou velocidade, a partir de bibliotecas específicas de componentes de um determinado fabricante.

Uma vez especificado o *netlist*, pode-se entrar na fase de implementação. Isto é necessário porque o *netlist* descreve apenas os componentes e como os mesmos são interconectados. Em linhas gerais, na implementação, faz-se necessário mapear tais componentes para células lógicas, que podem ser configuradas como *look-up tables* (LUTs) ou mesmo RAM. Em seguida, é necessário definir o posicionamento dos componentes no dispositivo de tal forma que as interconexões (roteamentos) entre os mesmos atendam as restrições de tempo especificadas. Tipicamente, estas restrições podem ser geradas automaticamente a partir das informações dos

clocks praticados no circuito. O processo de mapeamento ainda pode ser otimizado visando minimização da área ocupada do FPGA ou a maximização da velocidade de operação do circuito. No fluxo de projeto da Xilinx Inc. existe ainda uma etapa anterior (tradução) que traduz o *netlist* de componentes lógicos para um *netlist* de primitivas da Xilinx Inc.. O objetivo é facilitar a etapa de mapeamento.

Finalizadas as etapas de tradução, mapeamento, posicionamento e roteamento (*place and route*), obtém-se um arquivo binário que pode ser baixado diretamente no dispositvo, através de uma interface *JTAG*, para a sua configuração. As demais etapas do fluxo de projeto consistem basicamente em simulações (do *HDL* e do *netlist*) e no *debug* do dispositivo por meio de um analisador lógico. Vale destacar que as simulações podem refletir o funcionamento do circuito bastante realista através da realimentação de informações do *timing* pelo *Timing Analizer* geradas após a etapa de *Place & Route*.

As linguagens de descrição de *hardware* proporcionam uma interface comum entre as equipes de desenvolvimento de sistemas e entre ferramentas de desenvolvimento, permitindo uma forma de intercâmbio de informações comuns em todos os níveis de desenvolvimento do projeto.

A linguagem utilizada foi a VHDL, que é uma linguagem padronizada para descrever componentes digitais, permitindo a transferência de componentes ou projetos para qualquer tecnologia em construção de *hardware* existente. O surgimento da VHDL se fez necessário devido ao rápido avanço tecnológico alcançado pelas indústrias de circuito integrado. Algumas vantagens da utilização desta linguagem estão na redução do tempo e custo de desenvolvimento; maior nível de abstração; projetos independentes da tecnologia e na facilidade de atualização dos projetos. Além disso, a VHDL foi adotada como uma linguagem padrão IEEE (*Institute of Electrical and Electronics Enginners*), facilitando a migração de código entre diversas ferramentas comerciais de simulação e assegurando o sucesso da linguagem.

A ferramenta utilizada para o projeto dos filtros foi a ferramenta de integração de componentes de software embarcados – a EDK (*Embedded Development Kit*). Esta ferramenta possibilita a implementação de propriedades intelectuais, tais como microcontroladores,

controladores de periféricos, barramentos, etc. A Figura A.2 ilustra diversos blocos possíveis de ser implementados em um FPGA da Xilinx Inc..

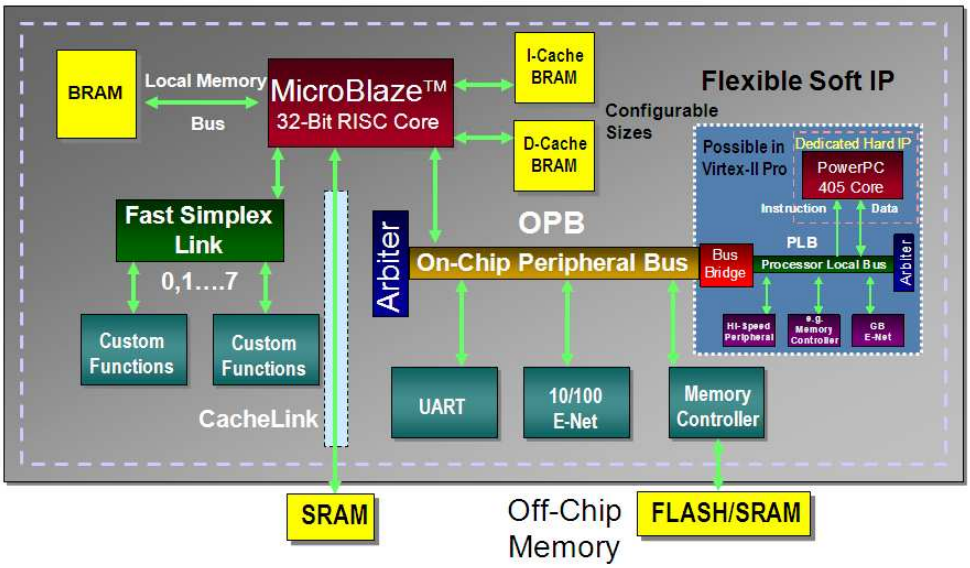

Figura A.2: Ilustração dos blocos implementados em um FPGA

O software *EDK* possibilita o desenvolvimento de um sistema embarcado programável completo, incluindo diversas ferramentas de suporte, documentação e *IP cores*, tais como o *soft processor core* MicroBlaze utilizado neste trabalho. Um poderoso recurso é a integração de componentes de *hardware* e *software*, possibilitando uma grande flexibilidade no desenvolvimento de um sistema embarcado, conforme ilustra a Figura A.3.

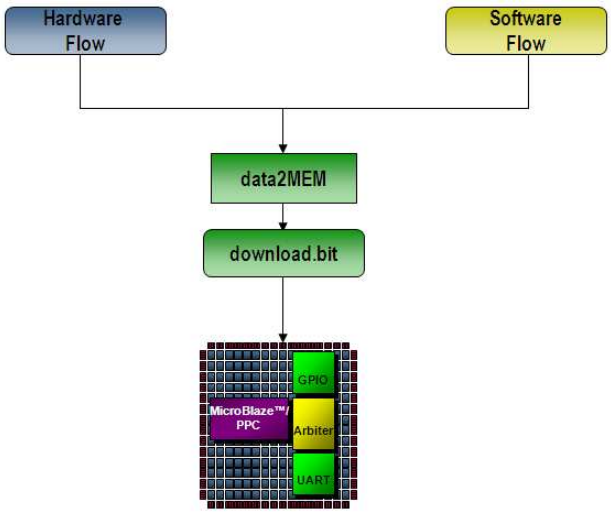

Figura A.3: Integração de programação de hardware e software

Xilinx Platform Studio (XPS) é um ambiente gráfico de projeto que incorpora as ferramentas de sistema embarcado para a criação de componentes de *hardware* e *software*. Através dela pode-se incorporar os diversos dispositivos e periféricos necessários para o desenvolvimento do projeto, possibilitando inclusive a seleção de kits de desenvolvimento fornecidos por empresas associadas a Xilinx Inc.. Segue abaixo algumas figuras ilustrando o ambiente de desenvolvimento *XPS*.

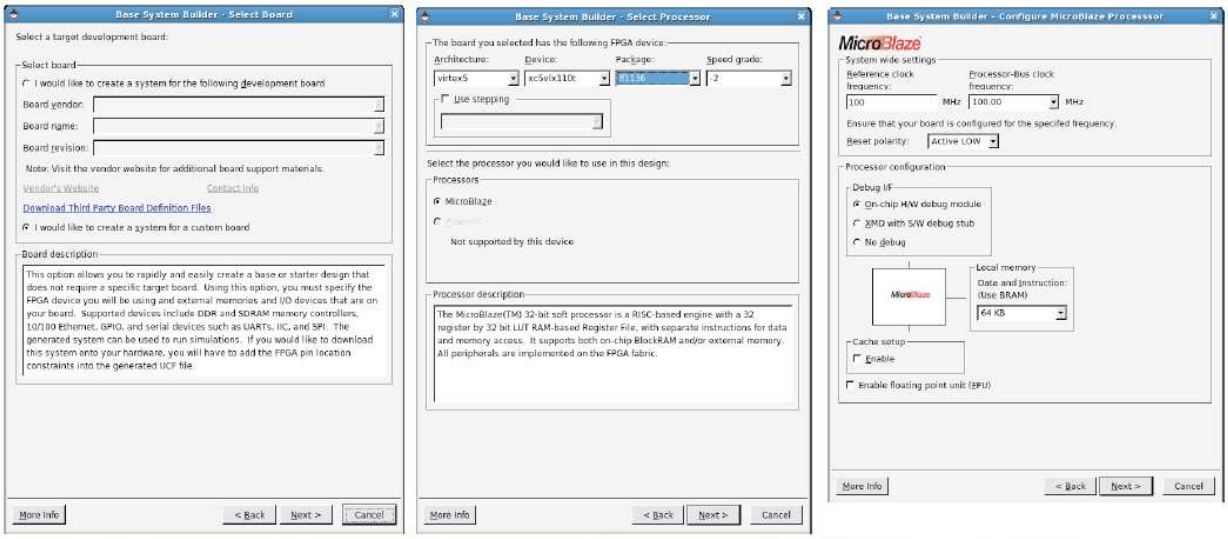

Figura A.4: Etapa de seleção do dispositivo e microcontrolador

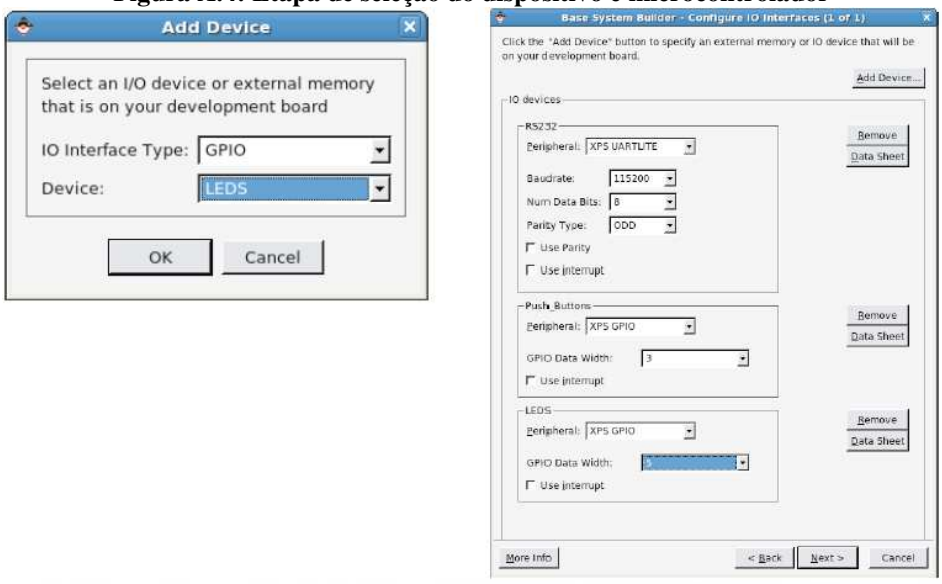

Figura A.5: Etapa de adição e configuração dos dispositivos

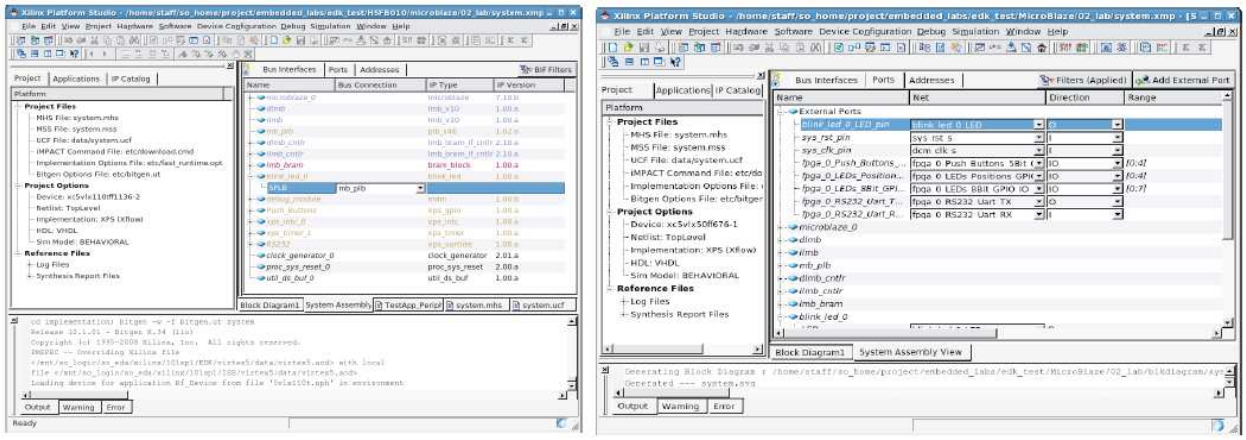

Figura A.6: Etapa de configuração dos *cores*

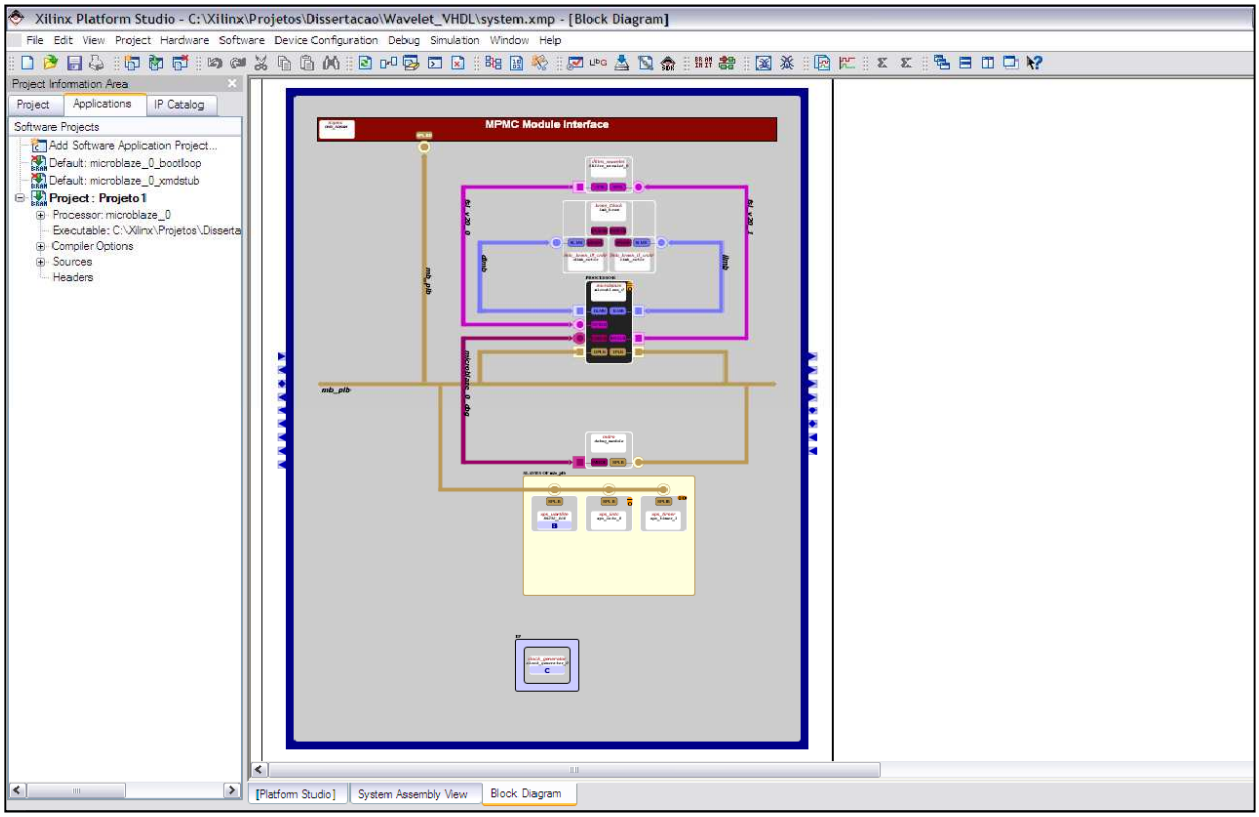

Figura A.7: Visão geral dos dispositivos selecionados

Apêndice A – Interação FPGA – microcomputador

Para o processamento do sinal cardíaco pelo dispositivo FPGA, foram criados arquivos de texto através do Matlab contendo os sinais de eletrocardiograma. Durante o início da comunicação do dispositivo com o microcomputador, o FPGA envia a frase "**Entre com dado:**", informando que este está pronto para receber todo o sinal e armazená-lo na memória DRAM do kit, conforme ilustra a Figura A.8.

Figura A.8: FPGA aguardando o envio dos dados

Os dados são selecionados através do arquivo texto e transferidos para o FPGA através do microcontrolador MicroBlaze, que por sua vez armazena os dados na memória DRAM do kit. A Figura A.9 ilustra o processo de transmissão dos dados.

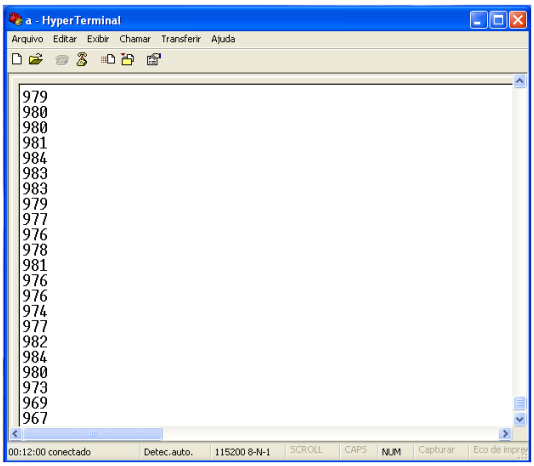

Figura A.9: Transmissão dos dados para o FPGA

Após a finalização dos dados, é dado um comando pressionando a letra "p" do teclado e então se inicia o processamento. Primeiramente o microcontrolador busca o primeiro dado na memória externa, processa e o envia de volta para o mesmo endereço de memória. Este procedimento é feito até o último dado armazenado. Após o processamento, o dispositivo envia um sinal via RS-232 para o *hyper-terminal*, comunicando o término do processamento e o tempo gasto para realizar todo o processo, conforme ilustra a Figura A.10.

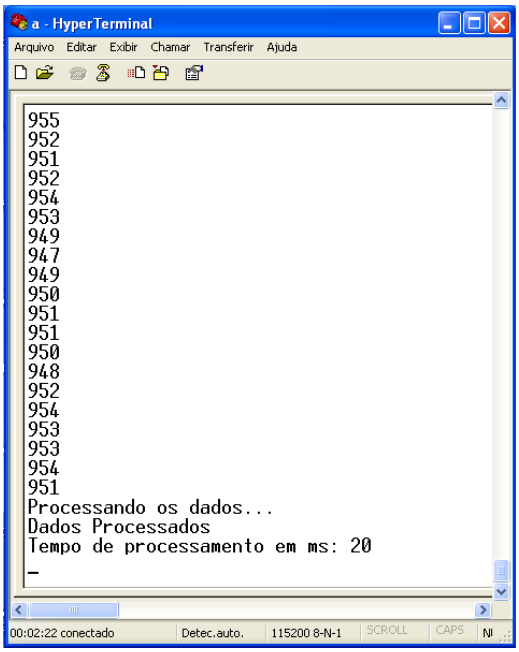

Figura A.10: Processamento dos dados e a informação do tempo de processamento

Com um sinal de recepção dos dados através do teclado do microcomputador, o MicroBlaze busca o primeiro valor na memória e o envia via RS-232 para o *hyper-terminal*, sendo possível a captura destes dados. A Figura A.11 ilustra o instante em que ocorre um complexo QRS, representado pelo número "1" e a Figura A.12 ilustra o fim da transferência.

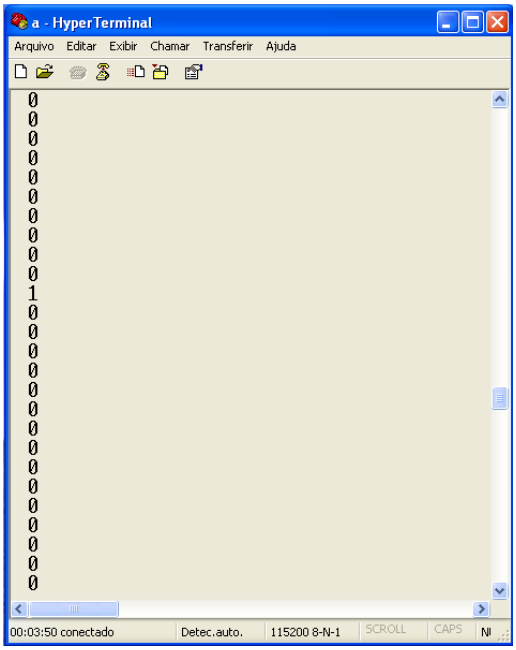

Figura A.11: Instante da transmissão do número 1, representando a ocorrência de um complexo QRS

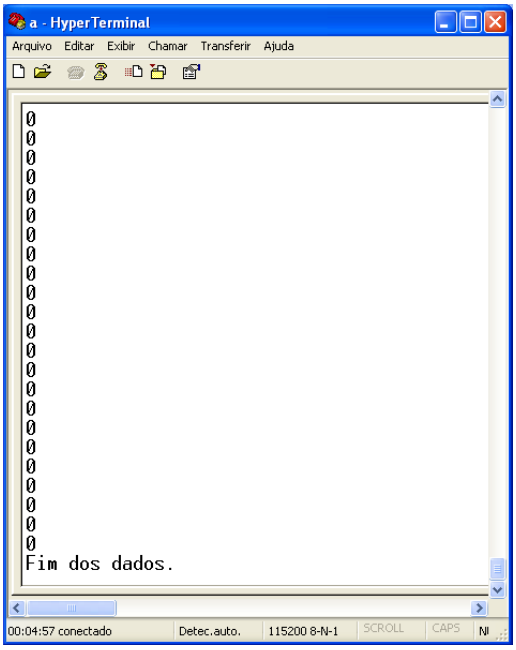

Figura A.12: Término da transmissão dos dados processados para o hyper-terminal

Apêndice B – Códigos fontes

B.1. Código do processamento em linguagem C do algoritmo de Pan & Tompkins

```
/***************************** Include Files *********************************/
#include "xbasic_types.h"
#include "xparameters.h"
#include "xuartlite.h"
#include "xstatus.h"
#include "xmpmc.h"
#include "serial.h"
#include "Xtmrctr_l.h"

// Rotina de interrupção para a contagem do tempo de processamento
Xuint32 millisecs;

void timer_int_handler(void * baseaddr_p)
{
  Xuint32 csr;
  csr = XTmrCtr_mGetControlStatusReg(baseaddr_p, XPAR_XPS_TIMER_1_BASEADDR);
  millisecs += 85899;
  XTmrCtr_mSetControlStatusReg(baseaddr_p, XPAR_XPS_TIMER_1_BASEADDR, csr);
}

int main ()
{
/* DDR SDRAM */
int Status;
XMpmc Mpmc;
XMpmc_Config *SdramCfgPtr;
SdramCfgPtr = XMpmc_LookupConfig(XPAR_DDR_SDRAM_DEVICE_ID);
Status = XMpmc_CfgInitialize(&Mpmc, SdramCfgPtr, SdramCfgPtr->BaseAddress);

//Xuint8  data[10], back;
            unsigned int data[10], back, decimal[10], nlp=12, nhp, nd=4, nq = 20, kk, jm;
            int valor = 0, le_mem, off_write = 0x8c000000, off_write_2 = 0x8c000000, off_read = 0x8c000000;
            int valor_mem[26], lp0 = 0, lp1 = 0, lp2 = 0, lp3 = 0, lowpass[33], hp0 = 0, hp1 = 0, hp2 = 0, highpass[10];
            int peak, count = 0, mpeak = 2000, peaks[8], shift = 0, aux_peaks = 0, th1 = 0, th2 = 0, aux1 = 0, thresh = 0;
            int loc_peak, derivada, quad[46];
            int i = 1, j = 0, k, m, menos = 0, zero = 0, f = 0;

  decimal[0] = 1;
            decimal[1] = 10;
            decimal[2] = 100;
            decimal[3] = 1000;
            decimal[4] = 10000;
            decimal[5] = 100000;
            decimal[6] = 1000000;
            decimal[7] = 10000000;
            decimal[8] = 100000000;

//limpar screen
  xil_printf("%c[2J",27);

  Status = UartInit();
```

```
if (Status != XST_SUCCESS)
{
	print(" -- UART Initialisation Error -- \n\r");
	return(1);
}
xil_printf("Entre com dado:");

	while (i==1){

		data[j] = recv_char();
		if (data[j] == 0x2D){		// Hexa de menos
			menos = 1;
			data[j] = recv_char();
		}
		if (data[0] == 0x30){
			data[0] = recv_char();
		}
		if (data[j] == 0x2E){		// Hexa de ponto
			data[j] = recv_char();
		}
		if (data[j] == 0x66){		// Hexa de f
			i = 0;
		}
		back = data[j];
		data[j] = data[j] - 0x30;
		k = j;
		if (back == 0x0D){				// Hexa de retorno de carro
			for (m = 0; m < k; m++){
				valor = valor + decimal[m] * data[(k-1)-m];
			}
			if (menos == 1){
				menos = 0;
				valor = - valor;
			}
			xil_printf("\r\n%d", valor);
			j = -1;
			XMpmc_mWriteReg(off_write, 0, valor);
			off_write = off_write + 0x00000004;
			valor = 0;
		}
		j++;
	}

	i = 0;
	while (i == 0) {
		data[0] = recv_char();
		if (data[0] == 0x70){		// Hexa de p
			i = 1;
		}
	}
	xil_printf("\r\nProcessando os dados...");

	// Inicia o timer
	Xuint32 start_time, end_time;
	millisecs = 0;
	XTmrCtr_mSetControlStatusReg(XPAR_XPS_TIMER_1_BASEADDR, 0,
	XTC_CSR_AUTO_RELOAD_MASK + XTC_CSR_ENABLE_INT_MASK);
	XTmrCtr_mEnable(XPAR_XPS_TIMER_1_BASEADDR, 0);
	start_time = XTmrCtr_mGetTimerCounterReg(XPAR_XPS_TIMER_1_BASEADDR, 0);
	microblaze_register_handler(timer_int_handler, XPAR_XPS_TIMER_1_BASEADDR);
	microblaze_enable_interrupts();


	i = 0;
	for (i = 0; i<26; i++){
		valor_mem[i] = 0;
	}
	for (i = 0; i<66; i++){
		lowpass[i] = 0;
	}
```

```
for (i = 0; i<10; i++){
		highpass[i] = 0;
}
for (i = 0; i<42; i++){
		quad[i] = 0;
}
i = 0;

th1 = 0.25*mpeak;

while (i == 0){
		valor_mem[nlp] = valor_mem[nlp + 13] = XMpmc_mReadReg(off_read, 0);
		off_read = off_read + 0x00000004;
		/* Filtro passa baixa */
		lp0=(2*lp1-lp2+valor_mem[nlp+0]-2*valor_mem[nlp+6]+valor_mem[nlp+12]);
		lp2=lp1;
		lp1=lp0;
		if (nlp < 1){
			nlp = 13;
		}
		nlp--;

		lowpass[0] = lp0 >> 5;

		/* Filtro passa alta */
		hp0=(hp1+lowpass[0]-lowpass[32]);
		hp1=hp0;
		for (nhp = 32; nhp > 0; nhp --){
			lowpass[nhp] = lowpass[nhp-1];
		}
		highpass[nd]=highpass[nd+5]=(lowpass[17]-(hp0 >> 5));

		/* Derivada */
		derivada=(2*highpass[nd+0]+highpass[nd+1]-highpass[nd+3]-2*highpass[nd+4]);
		if (nd < 1){
			nd = 5;
		}
		nd--;
		derivada = derivada >> 3;

		/* Valor ao quadrado */
		quad[nq+0]=quad[nq+21]=(derivada*derivada);

		if(nq < 1){
			nq = 21;
		}
		nq--;

		/* Janela móvel */
		jm = 0;
		for(kk=nq;kk<=nq+20;kk++){
jm=jm+quad[kk];
		}
		jm=jm/20;

		/* Threshold */
		thresh = 0;
		if (jm > th1 && aux1 == 0){
			peak = jm;
			loc_peak = off_write_2;
			thresh = 1;
			th2 = 0.5 * peak;
			aux1 = 1;
		}

		if (jm > peak && aux1 == 1){
			peak = jm;
			XMpmc_mWriteReg(loc_peak, 0, 0);
			loc_peak = off_write_2;
			thresh = 1;
```

```
                        th2 = 0.5 * peak;
                }

                if (aux1 == 1){
                        if (jm < th2) {
                                if (peak > (2*mpeak)){
                                        peaks[shift] = 2*mpeak;
                                } else {
                                        peaks[shift] = peak;
                                }
                                shift += 1;
                                if (shift == 8){
                                        shift = 0;
                                }
                                aux_peaks += 1;
                                if (aux_peaks == 1) {
                        mpeak = 2000;
                        th1=0.25 * mpeak;
                peaks[shift-1] = 0;
        }
        if (aux_peaks == 2) {
                        mpeak = (peaks[0] + peaks[1]) / 2;
                        th1=0.25 * mpeak;
        }
                                if (aux_peaks == 3) {
                        mpeak = (peaks[0] + peaks[1] + peaks[2]) / 3;
                        th1=0.25 * mpeak;
        }
                                if (aux_peaks == 4) {
                        mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3]) / 4;
                        th1=0.25 * mpeak;
        }
                                if (aux_peaks == 5) {
                        mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4]) / 5;
                        th1=0.25 * mpeak;
        }
                                if (aux_peaks == 6) {
                        mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4] + peaks[5]) / 6;
                        th1=0.25 * mpeak;
        }
                                if (aux_peaks == 7) {
                        mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4] + peaks[5] + peaks[6]) / 7;
                        th1=0.25 * mpeak;
        }
                                if (aux_peaks > 7) {
                        mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4] + peaks[5] + peaks[6] + peaks[7]) / 8;
                        th1=0.25 * mpeak;
        }
                aux1 = 2;
                                }
                }

                if (aux1 == 2){
                        count += 1;
                        if (count == 40){
                                aux1 = 0;
                                count = 0;
                        }
                }

                /* Fim do algoritmo */
                XMpmc_mWriteReg(off_write_2, 0, thresh);
                off_write_2 = off_write_2 + 0x00000004;
                if (off_read >= off_write){
                        i = 1;
                }
        }
        xil_printf("\r\nDados Processados");
// Obtendo o tempo de processamento
  end_time = XTmrCtr_mGetTimerCounterReg(XPAR_XPS_TIMER_1_BASEADDR, 0);
```

```
 microblaze_disable_interrupts();
 millisecs += (end_time-start_time)/50000;
 xil_printf("Tempo de processamento em ms: %d \n\r", millisecs);


		i = 0;
		while (i == 0) {
			data[0] = recv_char();
			if (data[0] == 0x2D){		// Hexa de menos
				i = 1;
			}
		}

		i = 0;
		xil_printf("\r\nDados da Memoria.");
		off_read = 0x8c000000;
		while (i == 0){
			le_mem = XMpmc_mReadReg(off_read, 0);
			xil_printf("\r\n%d", le_mem);
			off_read = off_read + 0x00000004;
			if (off_read >= off_write_2){
				i = 1;
			}
		}

		xil_printf("\r\nFim dos dados.");
return (0);
}
```

B.2. Código do processamento em linguagem C do algoritmo utilizando TW de Haar

```
/***************************** Include Files *********************************/
#include "xbasic_types.h"
#include "xparameters.h"
#include "xuartlite.h"
#include "xstatus.h"
#include "xmpmc.h"
#include "serial.h"
#include "Xtmrctr_l.h"

Xuint32 millisecs;
// Função de interrupção para adquirir o tempo de processamento
void timer_int_handler(void * baseaddr_p)
{
 Xuint32 csr;
 csr = XTmrCtr_mGetControlStatusReg(baseaddr_p, XPAR_XPS_TIMER_1_BASEADDR);
 millisecs += 85899;
 XTmrCtr_mSetControlStatusReg(baseaddr_p, XPAR_XPS_TIMER_1_BASEADDR, csr);
}

int main ()
{
/* DDR SDRAM */
int Status;
XMpmc Mpmc;
XMpmc_Config *SdramCfgPtr;
SdramCfgPtr = XMpmc_LookupConfig(XPAR_DDR_SDRAM_DEVICE_ID);
Status = XMpmc_CfgInitialize(&Mpmc, SdramCfgPtr, SdramCfgPtr->BaseAddress);

//Xuint8  data[10], back;
		unsigned int data[10], back, decimal[10], nlp1=1, nlp2 = 2, nlp3 = 4, nhp4 = 8;
		long int valor = 0, off_write = 0x8c000000, off_write_2 = 0x8c000000, off_read = 0x8c000000;
		int valor_mem[4], lp1 = 0, lp2 = 0, lp3 = 0, lowpass1[6], lowpass2[10], lowpass3[18], hp4 = 0, highpass4, quad, quad_anterior;
		int peak, count = 0, mpeak = 10000, peaks[8], shift = 0, aux_peaks = 0, th1 = 0, th2 = 0, th3 = 0, aux1 = 0, thresh = 0;
		long int loc_peak;

//		double valor_final = 0;
		int i = 1, j = 0, k, m, menos = 0, zero = 0, f = 0;
```

```
//Xuint32 test = 0x41424344;
  decimal[0] = 1;
		decimal[1] = 10;
		decimal[2] = 100;
		decimal[3] = 1000;
		decimal[4] = 10000;
		decimal[5] = 100000;
		decimal[6] = 1000000;
		decimal[7] = 10000000;
		decimal[8] = 100000000;

//limpar screen
  xil_printf("%c[2J",27);

  Status = UartInit();
  if (Status != XST_SUCCESS)
  {
		print(" -- UART Initialisation Error -- \n\r");
		return(1);
  }
  xil_printf("Entre com dado:");

		while (i==1){

			data[j] = recv_char();
			if (data[j] == 0x2D){			// Hexa de menos
				menos = 1;
				data[j] = recv_char();
			}
			if (data[0] == 0x30){
				data[0] = recv_char();
			}
			if (data[j] == 0x2E){			// Hexa de ponto
				data[j] = recv_char();
			}
			if (data[j] == 0x66){			// Hexa de f
				i = 0;
			}
			back = data[j];
			data[j] = data[j] - 0x30;
			k = j;
			if (back == 0x0D){					// Hexa de retorno de carro
				for (m = 0; m < k; m++){
					valor = valor + decimal[m] * data[(k-1)-m];
				}
				if (menos == 1){
					menos = 0;
					valor = - valor;
				}
				xil_printf("\r\n%d", valor);
				j = -1;
				XMpmc_mWriteReg(off_write, 0, valor);
				off_write = off_write + 0x00000004;
				valor = 0;
			}
			j++;
		}

		i = 0;
		while (i == 0) {
			data[0] = recv_char();
			if (data[0] == 0x70){			// Hexa de p
				i = 1;
			}
		}
		xil_printf("\r\nProcessando os dados...");
		// Inicia o timer e a habilita a interrupção
		Xuint32 start_time, end_time;
		millisecs = 0;
		XTmrCtr_mSetControlStatusReg(XPAR_XPS_TIMER_1_BASEADDR, 0,
```

```
	XTC_CSR_AUTO_RELOAD_MASK + XTC_CSR_ENABLE_INT_MASK);
	XTmrCtr_mEnable(XPAR_XPS_TIMER_1_BASEADDR, 0);
	start_time = XTmrCtr_mGetTimerCounterReg(XPAR_XPS_TIMER_1_BASEADDR, 0);
	microblaze_register_handler(timer_int_handler, XPAR_XPS_TIMER_1_BASEADDR);
	microblaze_enable_interrupts();

	i = 0;
	for (i = 0; i<4; i++){
		valor_mem[i] = 0;
	}
	for (i = 0; i<6; i++){
		lowpass1[i] = 0;
	}
	for (i = 0; i<10; i++){
		lowpass2[i] = 0;
	}
	for (i = 0; i<18; i++){
		lowpass3[i] = 0;
	}
	i = 0;
	th1 = 0.5*mpeak;
	while (i == 0){
		valor_mem[nlp1] = valor_mem[nlp1 + 2] = XMpmc_mReadReg(off_read, 0);
		off_read = off_read + 0x00000004;
		/* Filtro passa baixas LP1 */
		lp1=((valor_mem[nlp1+0] >> 1) + (valor_mem[nlp1+0] >> 3) + (valor_mem[nlp1+0] >> 4) + (valor_mem[nlp1+0] >> 6) +
(valor_mem[nlp1+0] >> 8)) + ((valor_mem[nlp1+1] >> 1) + (valor_mem[nlp1+1] >> 3) + (valor_mem[nlp1+1] >> 4) + (valor_mem[nlp1+1] >>
6) + (valor_mem[nlp1+1] >> 8));
		if (nlp1 < 1){
			nlp1 = 1;
		}
		nlp1--;

		lowpass1[nlp2] = lowpass1[nlp2 + 3] = lp1;

		/* Filtro passa baixas LP2 */
		lp2=((lowpass1[nlp2+0] >> 1) + (lowpass1[nlp2+0] >> 3) + (lowpass1[nlp2+0] >> 4) + (lowpass1[nlp2+0] >> 6) +
(lowpass1[nlp2+0] >> 8)) + ((lowpass1[nlp2+2] >> 1) + (lowpass1[nlp2+2] >> 3) + (lowpass1[nlp2+2] >> 4) + (lowpass1[nlp2+2] >> 6) +
(lowpass1[nlp2+2] >> 8));

		//lp2 = 0.7071*lowpass1[nlp2+0]+0.7071*lowpass1[nlp2+2];
		if (nlp2 < 1){
			nlp2 = 3;
		}
		nlp2--;

		lowpass2[nlp3] = lowpass2[nlp3+5] = lp2;

		/* Filtro passa baixas LP3 */
		lp3=((lowpass2[nlp3+0] >> 1) + (lowpass2[nlp3+0] >> 3) + (lowpass2[nlp3+0] >> 4) + (lowpass2[nlp3+0] >> 6) +
(lowpass2[nlp3+0] >> 8)) + ((lowpass2[nlp3+4] >> 1) + (lowpass2[nlp3+4] >> 3) + (lowpass2[nlp3+4] >> 4) + (lowpass2[nlp3+4] >> 6) +
(lowpass2[nlp3+4] >> 8));

		//lp3 = 0.7071*lowpass2[nlp3+0]+0.7071*lowpass2[nlp3+4];
		if (nlp3 < 1){
			nlp3 = 5;
		}
		nlp3--;
		lowpass3[nhp4] = lowpass3[nhp4+9] = lp3;

		/* Filtro passa alta HP4 */
		hp4= ((-lowpass3[nhp4+0] >> 1) - (lowpass3[nhp4+0] >> 3) - (lowpass3[nhp4+0] >> 4) - (lowpass3[nhp4+0] >> 6) -
(lowpass3[nhp4+0] >> 8)) + ((lowpass3[nhp4+8] >> 1) + (lowpass3[nhp4+8] >> 3) + (lowpass3[nhp4+8] >> 4) + (lowpass3[nhp4+8] >> 6) +
(lowpass3[nhp4+8] >> 8));

		//hp4 = -0.7071*lowpass3[nhp4+0]+0.7071*lowpass3[nhp4+8];
		if (nhp4 < 1){
			nhp4 = 9;
		}
		nhp4--;
```

```
		highpass4 = hp4;

		/* Valores elevado ao quadrado */
		quad = highpass4*highpass4;

	/* Threshold */
		thresh = 0;
		if (quad > th1 && aux1 == 0){
			peak = quad;
			loc_peak = off_write_2;
			//thresh = 1;
			th2 = 0.5 * peak;
			aux1 = 1;
		}

		if (quad > peak && aux1 == 1){
			peak = quad;
			XMpmc_mWriteReg(loc_peak, 0, 0);
			loc_peak = off_write_2;
			th2 = 0.5 * peak;
		}
		if (aux1 == 1){
			if (quad < th2) {
				aux1 = 2;
			}
		}
		if (aux1 == 2){
			if (quad > quad_anterior) {
				thresh = 1;
				aux1 = 3;
			}
		}
		quad_anterior = quad;

		if (aux1 == 3){
			if (peak > (2*mpeak)){
				peaks[shift] = 2*mpeak;
			} else {
				peaks[shift] = peak;
			}

			if (shift == 8){
				shift = 0;
			}
			shift += 1;
			aux_peaks += 1;
			if (aux_peaks == 1) {
		mpeak = 10000;
		th1=0.5 * mpeak;
	peaks[shift-1] = 0;
}
			if (aux_peaks == 2) {
			mpeak = (peaks[0] + peaks[1]) / 2;
			th1=0.5 * mpeak;
}
			if (aux_peaks == 3) {
					mpeak = (peaks[0] + peaks[1] + peaks[2]) / 3;
			th1=0.5 * mpeak;
}
			if (aux_peaks == 4) {
			mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3]) / 4;
			th1=0.5 * mpeak;
}
			if (aux_peaks == 5) {
			mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4]) / 5;
			th1=0.5 * mpeak;
}
			if (aux_peaks == 6) {
			mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4] + peaks[5]) / 6;
			th1=0.5 * mpeak;
```

```
	}
				if (aux_peaks == 7) {
				mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4] + peaks[5] + peaks[6]) / 7;
				th1=0.5 * mpeak;
	}
				if (aux_peaks > 7) {
				mpeak = (peaks[0] + peaks[1] + peaks[2] + peaks[3] + peaks[4] + peaks[5] + peaks[6] + peaks[7]) / 8;
				th1=0.5 * mpeak;
	}
				aux1 = 4;
			}
			if (aux1 == 4){
				count += 1;
				if (count == 40){
					aux1 = 0;
					count = 0;
				}
			}
			/* Fim do algoritmo */
			XMpmc_mWriteReg(off_write_2, 0, thresh);
			off_write_2 = off_write_2 + 0x00000004;
			if (off_read >= off_write){
				i = 1;
			}
		}
		xil_printf("\r\nDados Processados\r\n");

		// Obtendo o tempo de processamento
  end_time = XTmrCtr_mGetTimerCounterReg(XPAR_XPS_TIMER_1_BASEADDR, 0);
  microblaze_disable_interrupts();
  millisecs += (end_time-start_time)/50000;
  xil_printf("Tempo de processamento em ms: %d \n\r", millisecs);
		i = 0;
		while (i == 0) {
			data[0] = recv_char();
			if (data[0] == 0x2D){		// Hexa de menos
				i = 1;
			}
		}
		i = 0;
		xil_printf("\r\nDados da Memoria.");
		off_read = 0x8c000000;
		while (i == 0){
			valor_mem[0] = XMpmc_mReadReg(off_read, 0);
			xil_printf("\r\n%d", valor_mem[0]);
			off_read = off_read + 0x00000004;
			if (off_read >= off_write_2){
				i = 1;
			}
		}

		xil_printf("\r\nFim dos dados.");
return (0);
}
```

B.3. Código VHDL dos filtros do algoritmo de Pan & Tompkins

```
----------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;

---------------------------------------------------------------------------------------
--
--
-- Definition of Ports
```

```
-- FSL_Clk            : Synchronous clock
-- FSL_Rst            : System reset, should always come from FSL bus
-- FSL_S_Clk       : Slave asynchronous clock
-- FSL_S_Read      : Read signal, requiring next available input to be read
-- FSL_S_Data      : Input data
-- FSL_S_CONTROL   : Control Bit, indicating the input data are control word
-- FSL_S_Exists    : Data Exist Bit, indicating data exist in the input FSL bus
-- FSL_M_Clk       : Master asynchronous clock
-- FSL_M_Write     : Write signal, enabling writing to output FSL bus
-- FSL_M_Data      : Output data
-- FSL_M_Control   : Control Bit, indicating the output data are contol word
-- FSL_M_Full      : Full Bit, indicating output FSL bus is full
--
------------------------------------------------------------------------------

------------------------------------------------------------------------------
-- Entity Section
------------------------------------------------------------------------------

entity filtro_tompkins is
        GENERIC (lp_nx: INTEGER :=3; lp_ny: INTEGER :=2; hp_nx: INTEGER := 2; hp_ny: INTEGER :=1; der_n: INTEGER :=5; jm_n:
INTEGER := 29; m: INTEGER :=32);
        port
        (
                -- DO NOT EDIT BELOW THIS LINE ----------------------
                -- Bus protocol ports, do not add or delete.
                FSL_Clk    : in           std_logic;
                FSL_Rst    : in           std_logic;
                FSL_S_Clk              : out          std_logic;
                FSL_S_Read             : out          std_logic;
                FSL_S_Data             : in           signed (31 downto 0);
                FSL_S_Control          : in           std_logic;
                FSL_S_Exists           : in           std_logic;
                FSL_M_Clk              : out          std_logic;
                FSL_M_Write            : out          std_logic;
                FSL_M_Data             : out          signed (31 downto 0);
                FSL_M_Control          : out          std_logic;
                FSL_M_Full             : in           std_logic
                -- DO NOT EDIT ABOVE THIS LINE ----------------------
        );

attribute SIGIS : string;
attribute SIGIS of FSL_Clk : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";

end filtro_tompkins;


architecture Behavioral of filtro_tompkins is


  -- Total number of input data.
  constant NUMBER_OF_INPUT_WORDS  : natural := 1;

  -- Total number of output data
  constant NUMBER_OF_OUTPUT_WORDS : natural := 1;

  type STATE_TYPE is (Idle, Read_Inputs, Write_Outputs);

  signal state       : STATE_TYPE;

  -- Accumulator to hold sum of inputs read at any point in time
  signal sum         : std_logic_vector(0 to 31);
  -- Counters to store the number inputs read & outputs written
  signal nr_of_reads  : natural range 0 to NUMBER_OF_INPUT_WORDS - 1;
  signal nr_of_writes : natural range 0 to NUMBER_OF_OUTPUT_WORDS - 1;


        -- Sinais genéricos para implementação dos filtros
```

```
        signal habilita : std_logic:='0';

        -- Sinais para implementação do filtro passa-baixa
        signal lp_y          : signed(31 downto 0):= (others => '0');
        TYPE lp_registersx IS ARRAY (lp_nx+8 downto 0) OF
                                                                SIGNED(m-1 downto 0);
        TYPE lp_registersy IS ARRAY (lp_ny-1 downto 0) OF
                                                                SIGNED(m-1 downto 0);
        TYPE lp_coefficientsx IS ARRAY (lp_nx-1 downto 0) OF
                                                                SIGNED(m-29 downto 0);
        TYPE lp_coefficientsy IS ARRAY (lp_ny-1 downto 0) OF
                                                                SIGNED(m-29 downto 0);
        SIGNAL lp_regx: lp_registersx;
        SIGNAL lp_regy: lp_registersy;
        CONSTANT lp_coefx: lp_coefficientsx := ("0001", "1110", "0001");
        CONSTANT lp_coefy: lp_coefficientsy := ("1111", "0010");

        -- Sinais para implementação do filtro passa-alta
        signal hp_y          : signed(31 downto 0):= (others => '0');
        TYPE hp_registersx IS ARRAY (hp_nx+29 downto 0) OF
                                                                SIGNED(m-1 downto 0);
        TYPE hp_registersy IS ARRAY (hp_ny-1 downto 0) OF
                                                                SIGNED(m-1 downto 0);
        TYPE hp_coefficientsx IS ARRAY (hp_nx-1 downto 0) OF
                                                                SIGNED(m-29 downto 0);
        SIGNAL hp_regx: hp_registersx;
        SIGNAL hp_regy: hp_registersy;
        CONSTANT hp_coefx: hp_coefficientsx := ("1111", "0001");
        CONSTANT hp_coefy: signed(m-29 downto 0) := "0001";

        -- Sinais para implementação do filtro derivativo
        signal der_y         : signed(31 downto 0):= (others => '0');
        TYPE der_registers IS ARRAY (der_n-2 downto 0) OF
                                                                SIGNED(m-1 downto 0);
        TYPE der_coefficients IS ARRAY (der_n-1 downto 0) OF
                                                                SIGNED(m-29 downto 0);
        SIGNAL der_reg: der_registers;
        CONSTANT der_coef: der_coefficients := ("1110", "1111", "0000", "0001", "0010");

        -- Sinais para implementação da função ao quadrado
        SIGNAL square_y : signed(31 downto 0):= (OTHERS => '0');

        -- Sinais para implementação da janela móvel
        signal jm_y : signed(31 downto 0):= (OTHERS => '0');
        TYPE jm_registers IS ARRAY (jm_n downto 0) OF
                                                                SIGNED(m-1 downto 0);
        SIGNAL jm_reg: jm_registers;
        CONSTANT jm_coef: signed(m-29 downto 0) := "0001";

        -- Sinais para implementação do threshold
        signal detectado: signed(1 downto 0):= (OTHERS => '0');

begin

        -- CAUTION:
  -- The sequence in which data are read in and written out should be
  -- consistent with the sequence they are written and read in the
  -- driver's filtro.c file

  FSL_S_Read  <= FSL_S_Exists   when state = Read_Inputs   else '0';
  FSL_M_Write <= not FSL_M_Full when state = Write_Outputs else '0';
 FSL_M_Data <= "00000000000000000000000000000000" & detectado;


  The_SW_accelerator : process (FSL_Clk) is
  begin  -- process The_SW_accelerator
   if FSL_Clk'event and FSL_Clk = '1' then     -- Rising clock edge
     if FSL_Rst = '1' then               -- Synchronous reset (active high)
       -- CAUTION: make sure your reset polarity is consistent with the
       -- system reset polarity
```

```
    state       <= Idle;
    nr_of_reads  <= 0;
    nr_of_writes <= 0;
    --sum         <= (others => '0');
   else
    case state is
     when Idle =>
      if (FSL_S_Exists = '1') then
        state       <= Read_Inputs;
        nr_of_reads <= NUMBER_OF_INPUT_WORDS - 1;
        --sum         <= (others => '0');
      end if;

     when Read_Inputs =>
      if (FSL_S_Exists = '1') then
       -- Coprocessor function (Adding) happens here
       habilita <= '1';
                                              --sum        <= std_logic_vector(unsigned(sum) + 3*unsigned(FSL_S_Data));
                                              if (nr_of_reads = 0) then
         state       <= Write_Outputs;
         nr_of_writes <= NUMBER_OF_OUTPUT_WORDS - 1;
       else
         nr_of_reads <= nr_of_reads - 1;
       end if;
      end if;

     when Write_Outputs =>
      habilita <= '0';
                                            if (nr_of_writes = 0) then
       state <= Idle;
      else
       if (FSL_M_Full = '0') then
         nr_of_writes <= nr_of_writes - 1;
       end if;
      end if;
    end case;
   end if;
 end if;
end process The_SW_accelerator;

          lowpass: process(FSL_Clk, FSL_Rst) is
                                VARIABLE acc, prod1, prod2:
                                                        SIGNED (m-1 downto 0) := (OTHERS=>'0');
                                VARIABLE sign: STD_LOGIC;
          begin
                                if (FSL_Rst = '1') then
                                        lp_regx(0) <= (OTHERS => '0');
                                        lp_regx(1) <= (OTHERS => '0');
                                        lp_regx(2) <= (OTHERS => '0');
                                        lp_regx(3) <= (OTHERS => '0');
                                        lp_regx(4) <= (OTHERS => '0');
                                        lp_regx(5) <= (OTHERS => '0');
                                        lp_regx(6) <= (OTHERS => '0');
                                        lp_regx(7) <= (OTHERS => '0');
                                        lp_regx(8) <= (OTHERS => '0');
                                        lp_regx(9) <= (OTHERS => '0');
                                        lp_regx(10) <= (OTHERS => '0');
                                        lp_regx(11) <= (OTHERS => '0');
                                        lp_regy(0) <= (OTHERS => '0');
                                        lp_regy(1) <= (OTHERS => '0');
                                elsif FSL_Clk'event and FSL_Clk = '1' then
                                        if (habilita = '1') then
                                                acc := lp_coefx(0)*FSL_S_Data;
                                                FOR i IN 1 TO lp_nx-1 LOOP
                                                        sign := acc(m-1);
                                                        prod1 := lp_coefx(i)*lp_regx((lp_nx+9)-(6*i));
                                                        prod2 := lp_coefy(i-1)*lp_regy(lp_ny-i);
                                                        acc := acc + prod1 + prod2;
                                                END LOOP;
                                                lp_regx <= FSL_S_Data & lp_regx(lp_nx+8 DOWNTO 1);
```

```
                                        lp_regy <= acc & lp_regy(lp_ny-1 DOWNTO 1);
                                end if;
                        end if;
                        lp_y <= "00000" & acc(m-1 downto 5);
        end process lowpass;

        highpass: process(FSL_Clk, FSL_Rst) is
                        VARIABLE acc, prod1, prod2:
                                                SIGNED (m-1 downto 0) := (OTHERS=>'0');
                        VARIABLE sign: STD_LOGIC;

                BEGIN
                        IF (FSL_Rst = '1') THEN
                                hp_regx(0) <= (OTHERS => '0');
                                hp_regx(1) <= (OTHERS => '0');
                                hp_regx(2) <= (OTHERS => '0');
                                hp_regx(3) <= (OTHERS => '0');
                                hp_regx(4) <= (OTHERS => '0');
                                hp_regx(5) <= (OTHERS => '0');
                                hp_regx(6) <= (OTHERS => '0');
                                hp_regx(7) <= (OTHERS => '0');
                                hp_regx(8) <= (OTHERS => '0');
                                hp_regx(9) <= (OTHERS => '0');
                                hp_regx(10) <= (OTHERS => '0');
                                hp_regx(11) <= (OTHERS => '0');
                                hp_regx(12) <= (OTHERS => '0');
                                hp_regx(13) <= (OTHERS => '0');
                                hp_regx(14) <= (OTHERS => '0');
                                hp_regx(15) <= (OTHERS => '0');
                                hp_regx(16) <= (OTHERS => '0');
                                hp_regx(17) <= (OTHERS => '0');
                                hp_regx(18) <= (OTHERS => '0');
                                hp_regx(19) <= (OTHERS => '0');
                                hp_regx(20) <= (OTHERS => '0');
                                hp_regx(21) <= (OTHERS => '0');
                                hp_regx(22) <= (OTHERS => '0');
                                hp_regx(23) <= (OTHERS => '0');
                                hp_regx(24) <= (OTHERS => '0');
                                hp_regx(25) <= (OTHERS => '0');
                                hp_regx(26) <= (OTHERS => '0');
                                hp_regx(27) <= (OTHERS => '0');
                                hp_regx(28) <= (OTHERS => '0');
                                hp_regx(29) <= (OTHERS => '0');
                                hp_regx(30) <= (OTHERS => '0');
                                hp_regx(31) <= (OTHERS => '0');
                        ELSIF (FSL_Clk'event and FSL_Clk = '1') THEN
                                if (habilita = '1') then
                                        acc := hp_coefx(0)*lp_y;
                                        FOR i IN 1 TO hp_nx-1 LOOP
                                                sign := acc(m-1);
                                                prod1 := hp_coefx(i)*hp_regx((hp_nx+30)-(32*i));
                                                prod2 := hp_coefy * hp_regy(hp_ny-i);
                                                acc := acc + prod1 + prod2;
                                        END LOOP;
                                        hp_regx <= lp_y & hp_regx(hp_nx+29 DOWNTO 1);
                                        hp_regy <= acc & hp_regy(hp_ny-1 DOWNTO 1);
                                end if;
                        END IF;
                        hp_y <= hp_regx(hp_nx+14) - ("00000" & acc(m-1 downto 5)); -- passa-tudo menos passa-baixa dividido por
32.
        end process highpass;

        derivative: process(FSL_Clk, FSL_Rst) is
                        VARIABLE acc, prod:
                                                SIGNED (m-1 downto 0) := (OTHERS=>'0');
                        VARIABLE sign: STD_LOGIC;
                BEGIN
                        IF (FSL_Rst = '1') THEN
                                FOR i IN der_n-2 DOWNTO 0 LOOP
                                        FOR j IN m-1 DOWNTO 0 LOOP
```

```
                                        der_reg(i)(j) <= '0';
                                END LOOP;
                        END LOOP;
                ELSIF (FSL_Clk'event and FSL_Clk = '1') THEN
                        if (habilita = '1') then
                                acc := der_coef(0)*hp_y;
                                FOR i IN 1 TO der_n-1 LOOP
                                        sign := acc(m-1);
                                        prod := der_coef(i)*der_reg((der_n-1)-(i));
                                        acc := acc + prod;
                                END LOOP;
                                der_reg <= hp_y & der_reg(der_n-2 DOWNTO 1);
                                if ( acc < 0) then
                                        acc := "111" & acc(m-1 downto 3);
                                else
                                        acc := "000" & acc(m-1 downto 3);
                                end if;
                        end if;
                END IF;
                der_y <= acc;
end process derivative;

squaring: process(FSL_Clk, FSL_Rst) is
                VARIABLE prod:
                                        SIGNED (m-1 downto 0) := (OTHERS=>'0');
                VARIABLE sign: STD_LOGIC;

        BEGIN
                IF (FSL_Rst = '1') THEN

                ELSIF (FSL_Clk'event and FSL_Clk = '1') THEN
                        if (habilita = '1') then
                                prod:= der_y * der_y;
                        end if;
                END IF;
                square_y <= prod;
end process squaring;

movingwindow: process(FSL_Clk, FSL_Rst) is
                VARIABLE acc, total:
                                        SIGNED (m-1 downto 0) := (OTHERS=>'0');
                VARIABLE sign: STD_LOGIC;

        BEGIN
                IF (FSL_Rst = '1') THEN
                        FOR i IN jm_n DOWNTO 0 LOOP
                                FOR j IN m-1 DOWNTO 0 LOOP
                                        jm_reg(i)(j) <= '0';
                                END LOOP;
                        END LOOP;
                ELSIF (FSL_Clk'event and FSL_Clk = '1') THEN
                        if (habilita = '1') then
                                acc := jm_coef * square_y;
                                total := total + acc - jm_reg(0);
                                jm_reg <= square_y & jm_reg(jm_n DOWNTO 1);
                        end if;
                END IF;
                jm_y <= "00000" & total(m-1 downto 5); -- Valor dividido por 32. Valor mais próximo de 30.
end process movingwindow;

threshold: process(FSL_Clk, FSL_Rst) is
                TYPE last_peaks IS ARRAY (7 downto 0) OF SIGNED(31 downto 0);
                VARIABLE peak, th1, th2: SIGNED (31 downto 0);
                VARIABLE count: INTEGER := 0;
                VARIABLE shift: INTEGER range 0 to 8 := 0;
                VARIABLE aux1: INTEGER range 0 to 2;
                VARIABLE thresh: SIGNED (1 downto 0):="00";
                VARIABLE mpeak: SIGNED (31 downto 0) := "00000000000000000000011111010000";
                VARIABLE aux_mpeak: SIGNED (31 downto 0);
                VARIABLE peaks: last_peaks;
```

```
                BEGIN
                        IF (FSL_Rst = '1') THEN
                                peaks(1):= (OTHERS=>'0');
                                peaks(2):= (OTHERS=>'0');
                                peaks(3):= (OTHERS=>'0');
                                peaks(4):= (OTHERS=>'0');
                                peaks(5):= (OTHERS=>'0');
                                peaks(6):= (OTHERS=>'0');
                                peaks(7):= (OTHERS=>'0');
                                aux1:= 0;

                        ELSIF (FSL_Clk'event and FSL_Clk = '1') THEN
                                if (habilita = '1') then
                                        thresh := "00";
                                        if ( jm_y > th1 and aux1 = 0) then
                                                peak := jm_y;
                                                thresh := "01";
                                                th2 := "0" & peak(31 downto 1); -- 50% do valor de pico
                                                aux1 := 1;
                                        end if;

                                        if ( jm_y > peak and aux1 = 1) then
                                                peak := jm_y;
                                                thresh := "01";
                                                th2 := "0" & peak(31 downto 1); -- 50% do valor de pico
                                        end if;

                                        if ( aux1 = 1) then
                                                if ( jm_y < th2) then
                                                                peaks(shift) := peak;
                                                        thresh := "00";
                                                        shift := shift + 1;
                                                        if (shift = 8) then
                                                                shift := 0;
                                                        end if;
                                                        aux_mpeak := peaks(0) + peaks(1) + peaks(2) + peaks(3) +
peaks(4) + peaks(5) + peaks(6) + peaks(7);
                                                        mpeak := "000" & aux_mpeak(31 downto 3);
                                                        th1 := "00" & mpeak(31 downto 2); -- 25% do valor de mpeak
                                                        aux1 := 2;
                                                end if;
                                        end if;
                                        if (aux1 = 2) then
                                                count := count + 1;
                                                if (count = 40) then
                                                        count := 0;
                                                        aux1 := 0;
                                                end if;
                                        end if;
                                end if;
                        END IF;
                        detectado <= thresh;
        end process threshold;
end architecture Behavioral;
```

B.4. Código VHDL dos filtros do algoritmo da transformada *wavelet* de Haar

```
----------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
----------------------------------------------------------------------------------
-- Definition of Ports
-- FSL_Clk            : Synchronous clock
-- FSL_Rst            : System reset, should always come from FSL bus
-- FSL_S_Clk       : Slave asynchronous clock
-- FSL_S_Read      : Read signal, requiring next available input to be read
-- FSL_S_Data      : Input data
-- FSL_S_CONTROL   : Control Bit, indicating the input data are control word
```

```
-- FSL_S_Exists    : Data Exist Bit, indicating data exist in the input FSL bus
-- FSL_M_Clk       : Master asynchronous clock
-- FSL_M_Write     : Write signal, enabling writing to output FSL bus
-- FSL_M_Data      : Output data
-- FSL_M_Control   : Control Bit, indicating the output data are contol word
-- FSL_M_Full      : Full Bit, indicating output FSL bus is full
--
------------------------------------------------------------------------------

------------------------------------------------------------------------------
-- Entity Section
------------------------------------------------------------------------------

entity filtro_wavelet is
        port
        (
                -- DO NOT EDIT BELOW THIS LINE ----------------------
                -- Bus protocol ports, do not add or delete.
                FSL_Clk   : in          std_logic;
                FSL_Rst   : in          std_logic;
                FSL_S_Clk               : out         std_logic;
                FSL_S_Read              : out         std_logic;
                FSL_S_Data              : in          signed (31 downto 0);
                FSL_S_Control           : in          std_logic;
                FSL_S_Exists            : in          std_logic;
                FSL_M_Clk               : out         std_logic;
                FSL_M_Write             : out         std_logic;
                FSL_M_Data              : out         signed (31 downto 0);
                FSL_M_Control           : out         std_logic;
                FSL_M_Full              : in          std_logic
                -- DO NOT EDIT ABOVE THIS LINE ----------------------
        );

attribute SIGIS : string;
attribute SIGIS of FSL_Clk : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";
end filtro_wavelet;
------------------------------------------------------------------------------
architecture Behavioral of filtro_wavelet is
  -- Total number of input data.
  constant NUMBER_OF_INPUT_WORDS  : natural := 1;
  -- Total number of output data
  constant NUMBER_OF_OUTPUT_WORDS : natural := 1;
  type STATE_TYPE is (Idle, Read_Inputs, Write_Outputs);
  signal state       : STATE_TYPE;
  -- Accumulator to hold sum of inputs read at any point in time
  signal sum         : std_logic_vector(0 to 31);
  -- Counters to store the number inputs read & outputs written
  signal nr_of_reads  : natural range 0 to NUMBER_OF_INPUT_WORDS - 1;
  signal nr_of_writes : natural range 0 to NUMBER_OF_OUTPUT_WORDS - 1;
        -- Sinais genéricos para implementação dos filtros
        signal habilita : std_logic:='0';
        -- Sinais para implementação dos filtros passa-baixa
        signal lp1_y         : signed(23 downto 0):= (others => '0');
        signal lp2_y         : signed(23 downto 0):= (others => '0');
        signal lp3_y         : signed(23 downto 0):= (others => '0');
        TYPE lp2_registersx IS ARRAY (1 downto 0) OF
                                                                        SIGNED(23 downto 0);
        TYPE lp3_registersx IS ARRAY (3 downto 0) OF
                                                                        SIGNED(23 downto 0);
        SIGNAL lp1_regx: SIGNED(23 downto 0);
        SIGNAL lp2_regx: lp2_registersx;
        SIGNAL lp3_regx: lp3_registersx;
        CONSTANT lp_coefx: SIGNED(6 downto 0) := "0110101";   -- valor 53 devido a manobra algébrica
        -- Sinais para implementação do filtro passa-alta
        signal hp4_y         : signed(23 downto 0):= (others => '0');
        TYPE hp4_registersx IS ARRAY (7 downto 0) OF
                                                                        SIGNED(23 downto 0);
        TYPE hp_coefficientsx IS ARRAY (1 downto 0) OF
```

```
                                                                                SIGNED(6 downto 0);
        SIGNAL hp4_regx: hp4_registersx;
        CONSTANT hp_coefx: hp_coefficientsx := ("1001011","0110101"); -- 0,7071 ~= valor/2 + 53.valor/256. O valor "x,x" é -53,53
devido a manobra matemática
        SIGNAL square_y : signed(23 downto 0):= (OTHERS => '0');
        signal detectado: signed(1 downto 0):= (OTHERS => '0');
        -- Sinal de entrada de 24 bits
        SIGNAL FSL_S_Data_24bits : SIGNED(23 downto 0);
begin
        -- CAUTION:
  -- The sequence in which data are read in and written out should be
  -- consistent with the sequence they are written and read in the
  -- driver's filtro.c file
  FSL_S_Read  <= FSL_S_Exists   when state = Read_Inputs   else '0';
  FSL_M_Write <= not FSL_M_Full when state = Write_Outputs else '0';
        FSL_S_Data_24bits <= FSL_S_Data(23 downto 0);
        FSL_M_Data <= "00000000000000000000000000000000" & detectado;
  The_SW_accelerator : process (FSL_Clk) is
  begin  -- process The_SW_accelerator
   if FSL_Clk'event and FSL_Clk = '1' then     -- Rising clock edge
    if FSL_Rst = '1' then               -- Synchronous reset (active high)
      -- CAUTION: make sure your reset polarity is consistent with the
      -- system reset polarity
      state        <= Idle;
      nr_of_reads  <= 0;
      nr_of_writes <= 0;
    else
      case state is
        when Idle =>
          if (FSL_S_Exists = '1') then
            state       <= Read_Inputs;
            nr_of_reads <= NUMBER_OF_INPUT_WORDS - 1;
            --sum         <= (others => '0');
          end if;

        when Read_Inputs =>
          if (FSL_S_Exists = '1') then
            -- Coprocessor function (Adding) happens here
            habilita <= '1';
                                                --sum         <= std_logic_vector(unsigned(sum) + 3*unsigned(FSL_S_Data));
                                                if (nr_of_reads = 0) then
              state        <= Write_Outputs;
              nr_of_writes <= NUMBER_OF_OUTPUT_WORDS - 1;
            else
              nr_of_reads <= nr_of_reads - 1;
            end if;
          end if;

        when Write_Outputs =>
          habilita <= '0';
                                              if (nr_of_writes = 0) then
            state <= Idle;
          else
            if (FSL_M_Full = '0') then
              nr_of_writes <= nr_of_writes - 1;
            end if;
          end if;
      end case;
    end if;
   end if;
  end process The_SW_accelerator;

        lowpass1: process(FSL_Clk, FSL_Rst) is
                                VARIABLE acc,
prod1,aux_acc,aux2_acc,aux3_acc,aux4_acc,aux5_acc,aux_prod1,aux2_prod1,aux3_prod1,aux4_prod1,aux5_prod1:
                                                                SIGNED (23 downto 0) := (OTHERS=>'0');
        begin
                                if (FSL_Rst = '1') then
                                        lp1_regx <= (OTHERS => '0');
```

```
                elsif FSL_Clk'event and FSL_Clk = '1' then
                        if (habilita = '1') then
                                aux_acc := "0" & FSL_S_Data_24bits(23 downto 1);
                                aux2_acc := "000" & FSL_S_Data_24bits(23 downto 3);
                                aux3_acc := "0000" & FSL_S_Data_24bits(23 downto 4);
                                aux4_acc := "000000" & FSL_S_Data_24bits(23 downto 6);
                                aux5_acc := "00000000" & FSL_S_Data_24bits(23 downto 8);
                                acc := aux_acc + aux2_acc + aux3_acc + aux4_acc + aux5_acc;
                                aux_prod1 := "0" & lp1_regx(23 downto 1);
                                aux2_prod1 := "000" & lp1_regx(23 downto 3);
                                aux3_prod1 := "0000" & lp1_regx(23 downto 4);
                                aux4_prod1 := "000000" & lp1_regx(23 downto 6);
                                aux5_prod1 := "00000000" & lp1_regx(23 downto 8);
                                prod1 := aux_prod1 + aux2_prod1 + aux3_prod1 + aux4_prod1 + aux5_prod1;
                                acc := acc + prod1;
                                lp1_regx <= FSL_S_Data_24bits;
                        end if;
                end if;
                lp1_y <= acc;
        end process lowpass1;

        lowpass2: process(FSL_Clk, FSL_Rst) is
                VARIABLE acc,
prod1,aux_acc,aux2_acc,aux3_acc,aux4_acc,aux5_acc,aux_prod1,aux2_prod1,aux3_prod1,aux4_prod1,aux5_prod1:
                                                        SIGNED (23 downto 0) := (OTHERS=>'0');

        begin
                if (FSL_Rst = '1') then
                        lp2_regx(0) <= (OTHERS => '0');
                        lp2_regx(1) <= (OTHERS => '0');


                elsif FSL_Clk'event and FSL_Clk = '1' then
                        if (habilita = '1') then
                                aux_acc := "0" & lp1_y(23 downto 1);
                                aux2_acc := "000" & lp1_y(23 downto 3);
                                aux3_acc := "0000" & lp1_y(23 downto 4);
                                aux4_acc := "000000" & lp1_y(23 downto 6);
                                aux5_acc := "00000000" & lp1_y(23 downto 8);
                                acc := aux_acc + aux2_acc + aux3_acc + aux4_acc + aux5_acc;
                                aux_prod1 := "0" & lp2_regx(0)(23 downto 1);
                                aux2_prod1 := "000" & lp2_regx(0)(23 downto 3);
                                aux3_prod1 := "0000" & lp2_regx(0)(23 downto 4);
                                aux4_prod1 := "000000" & lp2_regx(0)(23 downto 6);
                                aux5_prod1 := "00000000" & lp2_regx(0)(23 downto 8);
                                prod1 := aux_prod1 + aux2_prod1 + aux3_prod1 + aux4_prod1 + aux5_prod1;
                                acc := acc + prod1;
                                lp2_regx <= lp1_y & lp2_regx(1 downto 1);
                        end if;
                end if;
                lp2_y <= acc;
        end process lowpass2;

        lowpass3: process(FSL_Clk, FSL_Rst) is
                VARIABLE acc,
prod1,aux_acc,aux2_acc,aux3_acc,aux4_acc,aux5_acc,aux_prod1,aux2_prod1,aux3_prod1,aux4_prod1,aux5_prod1:
                                                        SIGNED (23 downto 0) := (OTHERS=>'0');
        begin
                if (FSL_Rst = '1') then
                        lp3_regx(0) <= (OTHERS => '0');
                        lp3_regx(1) <= (OTHERS => '0');
                        lp3_regx(2) <= (OTHERS => '0');
                        lp3_regx(3) <= (OTHERS => '0');
                elsif FSL_Clk'event and FSL_Clk = '1' then
                        if (habilita = '1') then
                                aux_acc := "0" & lp2_y(23 downto 1);
                                aux2_acc := "000" & lp2_y(23 downto 3);
                                aux3_acc := "0000" & lp2_y(23 downto 4);
                                aux4_acc := "000000" & lp2_y(23 downto 6);
                                aux5_acc := "00000000" & lp2_y(23 downto 8);
```

```
                                        acc := aux_acc + aux2_acc + aux3_acc + aux4_acc + aux5_acc;
                                        aux_prod1 := "0" & lp3_regx(0)(23 downto 1);
                                        aux2_prod1 := "000" & lp3_regx(0)(23 downto 3);
                                        aux3_prod1 := "0000" & lp3_regx(0)(23 downto 4);
                                        aux4_prod1 := "000000" & lp3_regx(0)(23 downto 6);
                                        aux5_prod1 := "00000000" & lp3_regx(0)(23 downto 8);
                                        prod1 := aux_prod1 + aux2_prod1 + aux3_prod1 + aux4_prod1 + aux5_prod1;
                                        acc := acc + prod1;
                                        lp3_regx <= lp2_y & lp3_regx(3 downto 1);
                                end if;
                        end if;
                        lp3_y <= acc;
        end process lowpass3;

        highpass4: process(FSL_Clk, FSL_Rst) is
                        VARIABLE acc,
prod1,aux_acc,aux2_acc,aux3_acc,aux4_acc,aux5_acc,aux_prod1,aux2_prod1,aux3_prod1,aux4_prod1,aux5_prod1:
                                                SIGNED (23 downto 0) := (OTHERS=>'0');
        begin
                        if (FSL_Rst = '1') then
                                hp4_regx(0) <= (OTHERS => '0');
                                hp4_regx(1) <= (OTHERS => '0');
                                hp4_regx(2) <= (OTHERS => '0');
                                hp4_regx(3) <= (OTHERS => '0');
                                hp4_regx(4) <= (OTHERS => '0');
                                hp4_regx(5) <= (OTHERS => '0');
                                hp4_regx(6) <= (OTHERS => '0');
                                hp4_regx(7) <= (OTHERS => '0');

                        elsif FSL_Clk'event and FSL_Clk = '1' then
                                if (habilita = '1') then
                                        aux_acc := -("0" & lp3_y(23 downto 1));
                                        aux2_acc := -("000" & lp3_y(23 downto 3));
                                        aux3_acc := -("0000" & lp3_y(23 downto 4));
                                        aux4_acc := -("000000" & lp3_y(23 downto 6));
                                        aux5_acc := -("00000000" & lp3_y(23 downto 8));
                                        acc := aux_acc + aux2_acc + aux3_acc + aux4_acc + aux5_acc;
                                        if (hp4_regx(0) < 0 ) then
                                                aux_prod1 := "1" & hp4_regx(0)(23 downto 1);
                                                aux2_prod1 := "111" & hp4_regx(0)(23 downto 3);
                                                aux3_prod1 := "1111" & hp4_regx(0)(23 downto 4);
                                                aux4_prod1 := "111111" & hp4_regx(0)(23 downto 6);
                                                aux5_prod1 := "11111111" & hp4_regx(0)(23 downto 8);
                                        else
                                                aux_prod1 := "0" & hp4_regx(0)(23 downto 1);
                                                aux2_prod1 := "000" & hp4_regx(0)(23 downto 3);
                                                aux3_prod1 := "0000" & hp4_regx(0)(23 downto 4);
                                                aux4_prod1 := "000000" & hp4_regx(0)(23 downto 6);
                                                aux5_prod1 := "00000000" & hp4_regx(0)(23 downto 8);
                                        end if;
                                        prod1 := aux_prod1 + aux2_prod1 + aux3_prod1 + aux4_prod1 + aux5_prod1;
                                        acc := acc + prod1;
                                        hp4_regx <= lp3_y & hp4_regx(7 downto 1);
                                end if;
                        end if;
                        hp4_y <= acc;
        end process highpass4;

        squaring: process(FSL_Clk, FSL_Rst) is
                        VARIABLE prod:
                                                SIGNED (23 downto 0) := (OTHERS=>'0');
                        VARIABLE sign: STD_LOGIC;

                BEGIN
                        IF (FSL_Rst = '1') THEN

                        ELSIF (FSL_Clk'event and FSL_Clk = '1') THEN
                                if (habilita = '1') then
                                        prod:= hp4_y * hp4_y;
                                end if;
```

```
                END IF;
                square_y <= prod;
end process squaring;

threshold: process(FSL_Clk, FSL_Rst) is
                TYPE last_peaks IS ARRAY (7 downto 0) OF SIGNED(23 downto 0);
                VARIABLE peak, th1, th2: SIGNED (23 downto 0);
                VARIABLE count: INTEGER := 0;
                VARIABLE aux_peaks: INTEGER := 0;
                VARIABLE shift: INTEGER range 0 to 8 := 0;
                VARIABLE aux1: INTEGER range 0 to 4;
                VARIABLE thresh: SIGNED (1 downto 0):="00";
                VARIABLE mpeak: SIGNED (23 downto 0) := "000000000010011100010000"; -- Valor 10000
                VARIABLE dois_mpeak: SIGNED (23 downto 0) := (OTHERS=>'0');
                VARIABLE aux_mpeak: SIGNED (23 downto 0);
                VARIABLE peaks: last_peaks;
                VARIABLE square_y_anterior : signed(23 downto 0):= (OTHERS => '0');

        BEGIN
                IF (FSL_Rst = '1') THEN
                        peaks(1):= (OTHERS=>'0');
                        peaks(2):= (OTHERS=>'0');
                        peaks(3):= (OTHERS=>'0');
                        peaks(4):= (OTHERS=>'0');
                        peaks(5):= (OTHERS=>'0');
                        peaks(6):= (OTHERS=>'0');
                        peaks(7):= (OTHERS=>'0');
                        aux1:= 2; -- Serve para eliminar o primeiro valor de pico que nao corresponde ao QRS
                        th1:= "000000000010011100010000";

                ELSIF (FSL_Clk'event and FSL_Clk = '1') THEN
                        if (habilita = '1') then
                                thresh := "00";
                                if ( square_y > th1 and aux1 = 0) then
                                        peak := square_y;
                                        thresh := "01";
                                        --th2 := 0.33 * peak;
                                        th2 := "0" & peak(23 downto 1); -- 50% do valor de pico
                                        aux1 := 1;
                                end if;

                                if ( square_y > peak and aux1 = 1) then
                                        peak := square_y;
                                        thresh := "01";
                                        --th2 := 0.33 * peak;
                                        th2 := "0" & peak(23 downto 1); -- 50% do valor de pico
                                end if;

                                if ( aux1 = 1) then
                                        if ( square_y < th2) then
                                                aux1 := 2;
                                        end if;
                                end if;

                                if (aux1 = 2) then
                                        if (square_y > square_y_anterior) then
                                                thresh := "01";
                                                aux1 := 3;
                                        end if;
                                end if;
                                square_y_anterior := square_y;

                                if (aux1 = 3) then
                                                        peaks(shift) := dois_mpeak;
                                                        peaks(shift) := peak;
                                        thresh := "00";
                                        if (shift = 8) then
                                                shift := 0;
                                        end if;
                                        shift := shift + 1;
```

```
                                                aux_mpeak := peaks(0) + peaks(1) + peaks(2) + peaks(3) +
peaks(4) + peaks(5) + peaks(6) + peaks(7);
                                                mpeak := "000" & aux_mpeak(23 downto 3);
                                                th1 := "0" & mpeak(23 downto 1); -- 50% do valor de mpeak
                                                aux1 := 4;
                                        end if;
                                        if (aux1 = 4) then
                                                count := count + 1;
                                                if (count = 40) then
                                                        count := 0;
                                                        aux1 := 0;
                                                end if;
                                        end if;
                                end if;
                        END IF;
                        detectado <= thresh;
        end process threshold;
end architecture Behavioral;
```