

Universidade Federal da Paraíba
Centro de Ciências Exatas e da Natureza
Departamento de Informática
Programa de Pós-Graduação em Informática

Thyago Maia Tavares de Farias

**SISTEMA EMBARCADO PARA UM MONITOR
HOLTER QUE UTILIZA O MODELO PPM
NA COMPRESSÃO DE SINAIS ECG**

**João Pessoa – Paraíba
Março de 2010**

Thyago Maia Tavares de Farias

SISTEMA EMBARCADO PARA UM MONITOR HOLTER QUE UTILIZA O MODELO PPM NA COMPRESSÃO DE SINAIS ECG

Dissertação apresentada ao Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, como requisito parcial para a obtenção do Título de Mestre em Informática (Sistemas de Computação).

Área de concentração: Processamento de Sinais e Sistemas Gráficos

Sub-Área: Sistemas Embarcados

Orientador: José Antônio Gomes de Lima

João Pessoa – Paraíba
Março de 2010

F224s Farias, Thyago Maia Tavares de.
Sistema Embarcado para um Monitor Holter que Utiliza o
Modelo PPM na Compressão de Sinais ECG/ Thyago Maia
Tavares de Farias. – João Pessoa, 2010.
128ff. :il.

Orientador: José Antônio Gomes de Lima.
Dissertação (Mestrado) – UFPb - CCEN

1.Sistemas de Computação. 2. Sistemas Embarcados. 3. Com-
pressão de dados.

UFPb/BC

CDU: 004(043)

Ata da Sessão Pública de Defesa de Dissertação de
Mestrado de **Thyago Maia Tavares de Farias**,
candidato ao Título de Mestre em Informática na
Área de Sistemas de Computação, realizada em 04
de março de 2010.

Aos quatro dias do mês de março do ano dois mil e dez, às 10 horas, no Auditório do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Processamento de Sinais e Sistemas Gráficos*", o Sr. Thyago Maia Tavares de Farias. A comissão examinadora foi composta pelos professores doutores: José Antonio Gomes de Lima (DI-UFPB), Orientador e Presidente da Banca Examinadora, Lucidio dos Anjos Formiga Cabral (DI-UFPB) e Antonio Carlos Cavalcanti (DI-UFPB), examinador interno e Rômulo Pires Coelho Ferreira (Instituto Federal de Educação, Ciência e Tecnologia de Alagoas-IFAL), examinador externo. Dando início aos trabalhos, o professor José Antonio Gomes de Lima, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*Sistema Embarcado para um monitor Holter que utiliza o modelo PPM na Compressão de Sinais ECG*". Concluída a exposição, o candidato foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Tatiana Aires Tavares, Coordenadora deste Programa, servindo de secretária lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 04 de março de 2010.

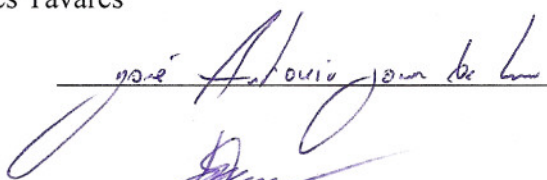
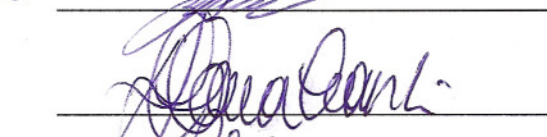
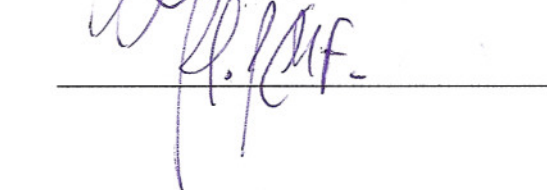

Tatiana Aires Tavares

Prof. Dr. José Antonio Gomes de Lima
Orientador (DI-UFPB)

Prof. Dr. Lucidio dos Anjos Formiga Cabral
Examinador Interno (DI-UFPB)

Prof. Dr. Antonio Carlos Cavalcanti
Examinador Interno (DI-UFPB)

Prof. Dr. Rômulo Pires Coelho Ferreira
Examinador Externo (IFAL)

DEDICATÓRIA

Dedico este trabalho a minha família, em especial ao meu pai Antonio Fernando Tavares de Farias, que sempre foi e sempre será um exemplo de honestidade, caráter, integridade, coragem e força, a minha mãe Iraneide Maia Tavares, minha melhor amiga, que sempre me apoiou em todas as minhas decisões e esteve ao meu lado em todos os momentos, e aos meus irmãos Afonso Maia e Fernanda Maia.

A minha noiva e futura esposa Suzienne Martins, que, com seu carinho, afeto e companheirismo, esteve sempre ao meu lado, ajudando-me nos momentos de dificuldade, aconselhando-me nos momentos de dúvida, reerguendo-me nas derrotas, glorificando-me nas vitórias e me amando ontem, hoje e sempre.

A todos os meus grandes amigos, em especial a Israel Cavalcanti Costa, Fernanda Brito, Dayse Rodrigues, Bruno Alberto, Diogo Max, Henrique Régis, Renato Sassi, Walter Travassos, Thatiana Souza, Victor Santos, Vitor de Figueiredo, Adalto Ribeiro, Pedro Viana, Wildson Lucena, Wamberto Farias, Marcílio Santos, Felipe Sette e Ana Paula Abrantes.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me conceber uma vida saudável e cheia de oportunidades.

Agradeço ao Professor Dr. Luis Carlos Carvalho, primeiro professor da Universidade Federal da Paraíba a me dar oportunidade para atuar em projetos de pesquisa. O mesmo me ensinou a ser mais disciplinado, dedicado e pró-ativo. Os trabalhos desenvolvidos no NETEB despertaram a vontade de seguir a carreira acadêmica.

Ao Professor Dr. José Antônio Gomes de Lima, pela orientação, confiança e pelo apoio financeiro para a participação do *IADIS Applied Computing 2009*, realizado em Roma – Itália.

Aos Professores Dr. Antônio Carlos Cavalcanti e Dr. Lucídio dos Anjos Formiga Cabral, pelas preciosas informações e sugestões para o desenvolvimento dos trabalhos práticos e para a produção desta dissertação.

Ao Professor Hamilton Soares da Silva, pela solicitude, confiança e pela oportunidade em poder lecionar a disciplina de Iniciação em Computação para alunos do Centro de Tecnologia da Universidade Federal da Paraíba em 2008.

À Empresa de Assistência Técnica e Extensão Rural da Paraíba – EMATER/PB, à Coordenadoria de Informática da EMATER/PB – CODIN e aos companheiros da ATYS Tecnologia, Renato Sassi e Walter Travassos, pelo apoio para que eu pudesse ingressar no mestrado e me dedicar integralmente ao mesmo.

A todos os componentes do Laboratório de Sistemas Digitais da Universidade Federal da Paraíba – LASID/UFPB, em especial a Yuri Gonzaga, Bruno Maia, Igor Gadelha, João Janduy, Jerry Lee Alves e Daniel Soares, pelas preciosas informações e pelo companheirismo.

A Francisco Neto, pela grande ajuda na disciplina de Compressão de Dados, integrante da grade de disciplinas optativas do Programa de Pós-Graduação em Informática.

À Coordenação do Programa de Pós-Graduação em Informática, pela constante assistência e pelo apoio financeiro para a participação do *IADIS Applied Computing 2009*, realizado em Roma – Itália.

Ao CNPq, através do Programa Nacional de Microeletrônica, CT-INFO/MCT/CNPq - Nº 13/2007, pelo apoio financeiro.

E a todos que direta ou indiretamente contribuíram com a realização desse trabalho.

“Às vezes, quando tudo dá errado, acontecem coisas tão maravilhosas, que jamais teriam acontecido se tudo tivesse dado certo”.

Autor desconhecido

RESUMO

Neste trabalho, é apresentado o desenvolvimento de um sistema embarcado com prototipagem em FPGA contendo instanciação do processador *soft-core* Nios II (SOPC – *System on a Programmable Chip*), para um monitor holter que implementa compressão de dados, utilizando o algoritmo PPM, e simula sinais ECG através da implementação das Séries de Fourier. Através de um monitor holter, cardiologistas podem obter sinais ECG, que servem de base para a percepção de sintomas e atividades em pacientes, captados e armazenados pelos monitores em períodos maiores ou iguais a 24 horas, requisitando grandes espaços de armazenamento, aumentando, assim, o custo deste monitor. Utilizando o PPM, o dispositivo desenvolvido poderá reduzir consideravelmente a quantidade de dados armazenados, reduzindo, portanto, o espaço de armazenamento e o custo do dispositivo, permitindo ainda a rápida transmissão dos dados. Integrando o simulador de sinais ECG ao dispositivo, possibilita-se a geração de amostras de sinais eletrocardiográficos através do sistema embarcado, economizando tempo e eliminando dificuldades na obtenção de sinais, em comparação com a captação real de sinais ECG através de métodos invasivos e não-invasivos. O mesmo permite a análise e o estudo de sinais ECG normais e anormais. Um sistema embarcado em chip programável (SOPC) foi prototipado com uma placa contendo periféricos e uma pastilha FPGA dotada de compatibilidade com o Nios II; a arquitetura do *soft-core* foi configurada em sistema operacional compacto e módulos de software foram exitosamente desenvolvidos, portados e validados sobre essa plataforma.

Palavras Chave: Sistemas Embarcados, FPGA, Monitor Holter, ECG, PPM, Compressão de Dados, Processador Nios II, Séries de Fourier, Simulação.

ABSTRACT

In this work, we present the development of an embedded system prototyping with soft-core Nios II and FPGA for a holter monitor that implements data compression, using the PPM Algorithm, and simulate ECG signals through the implementation of the Fourier series. Through a holter monitor, cardiologists can obtain ECG signals, serving as the basis for the perception of symptoms and activities of patients. These signals are captured and recorded by monitors in periods greater than or equal to 24 hours, requiring large storage size to store them, therefore increasing cost of the monitor. Using the PPM algorithm, a monitor holter can considerably reduce the size of the signals stored, thus reducing storage space and cost of device, addition to allow rapid transmission of the data. Integrating the ECG signal simulator to the device, is possible to generate samples of ECG via the embedded system, saving time and eliminating difficulties in obtaining signals, compared with the capture of real ECG signals by invasive and noninvasive methods. It enables the analysis and study of normal and abnormal ECGs. An embedded system on programmable chip (SOPC) was prototyped with a development kit containing peripherals and FPGA chip compatible with the Nios II. Architecture soft-core was set to compact operating system and software modules have been successfully developed, ported and validated on this platform.

Keywords: Embedded Systems, FPGA, Holter Monitor, ECG, PPM, Data Compression, Nios II Processor, Fourier Series, Simulation.

LISTA DE FIGURAS

Figura 3.1. Um sinal ECG típico	23
Figura 3.2. O Triângulo de Einthoven, os pontos BE (braço esquerdo), BD (braço direito), P (perna esquerda) e os pontos aVL, aVF e aVR.....	24
Figura 3.3. Formação do traçado QRS nas derivações precordiais.....	24
Figura 3.4. Ondas componentes de um ECG.....	25
Figura 3.5. Atuação de um monitor holter	27
Figura 3.6. Colocação dos eletrodos em 5 ou 7 derivações.....	27
Figura 3.7. Monitor holter que interage via sinais sonoros	28
Figura 3.8. Monitor holter com LCD.....	29
Figura 3.9. Monitor holter sem compressão BI9800TL+3, da Biomedical Instruments	31
Figura 3.10. Exemplo de hardware simulador de ECG	32
Figura 3.11. MP4 Player	33
Figura 3.12. A essência dos sistemas embarcados	34
Figura 3.13. Um FPGA da Altera® com 20.000 células	39
Figura 3.14. Estrutura de um FPGA	39
Figura 3.15. Modelo PPM para a palavra “abracadabra”	46
Figura 3.16. Exemplo de subdivisão de intervalos na Codificação Aritmética	48
Figura 4.1. Arquitetura do sistema embarcado.....	53
Figura 4.2. Detalhamento do sistema embarcado.....	54
Figura 4.3. Procedimentos para os cálculos das porções P, T e U, além das porções QRS, Q e S, respectivamente	56
Figura 4.4. Construção do Código Gray	59
Figura 4.5. Exemplo de uma decomposição em plano de bits	60
Figura 4.6. Razão de compressão em função de Kmax para cada combinação Gray / nº de planos de bit (Primo, 2009; Cavalcanti, 2009)	61
Figura 4.7. A placa de desenvolvimento utilizada	64

Figura 4.8. Recursos fornecidos pelo kit de desenvolvimento	65
Figura 4.9. Camada de hardware do sistema embarcado.....	66
Figura 4.10. Detalhamento do sistema de descompressão / visualização de sinais ECG	69
Figura 5.1. Três mil amostras do sinal ECG 100.dat, oriundo do MIT-BIH Arrhythmia Database	72
Figura 5.3. Três mil amostras do sinal original ECG 102.dat, canal 2.....	77
Figura 5.4. Três mil amostras do sinal ECG 102.dat, canal 2, resultante do processo de descompressão	77
Figura 5.5. Sinal ECG gerado a partir dos parâmetros listados na Tabela 5.7	78
Figura 5.6. Sinal ECG gerado a partir dos parâmetros listados na Tabela 5.8	79

LISTA DE TABELAS

Tabela 4.1. As Séries de Fourier	57
Tabela 4.2. Síntese do projeto de hardware.....	68
Tabela 5.1. Taxas de compressão em função de $K_{max} = 1$ obtidas para sinais utilizados em testes	74
Tabela 5.2. Taxas de compressão em função de $K_{max} = 2$ obtidas para sinais utilizados em testes	74
Tabela 5.3. Taxas de compressão em função de $K_{max} = 3$ obtidas para sinais utilizados em testes	74
Tabela 5.4. Taxas de compressão em função de $K_{max} = 4$ obtidas para sinais utilizados em testes	75
Tabela 5.5. Taxas de compressão em função de $K_{max} = 5$ obtidas para sinais utilizados em testes	75
Tabela 5.6. Médias das taxas de compressão obtidas em diferentes projetos	75
Tabela 5.7. Parâmetros utilizados para a simulação do sinal ECG 1	78
Tabela 5.8. Parâmetros utilizados para a simulação do sinal ECG 2.....	78
Tabela 5.9. Taxas de compressão obtidas para os sinais apresentados nas Figuras 5.5 e 5.6	80

LISTA DE ABREVIATURAS

ADC	Analog to Digital Converter
ASCII	American Standard Code for Information Interchange
aVF	heart/left leg (or foot)
aVL	heart/left arm
aVR	heart/right arm
BIH	Beth Israel Deaconess Medical Center
CAD	Computer-Aided Design
CD	Compact Disc
DAC	Digital to Analog Converter
DMA	Direct Memory Access
ECG	Electrocardiogram
EMI	Electromagnetic interference
EPROM	Erasable Programmable Read-Only Memory
EEPROM	Electrically-Erasable Programmable Read-Only Memory
FPGA	Field Programmable
GNU	GNU's Not Unix
HAL	Hardware Abstract Layer
IDE	Integrated Development Environment
I/O	Input/Output
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MAC	Media Access Control
MIT	Massachusetts Institute of Technology
PC	Personal Computer
PIO	Programmed Input/Output

PPM	Prediction by Partial Matching
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SD	Secure Digital
SOC	System on a Chip
SoCs	System on a Chip
SOPC	System On a Programmable Chip
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
ULPI	Universal Low Pin Interface
USB	Universal Serial Bus
UTMI	USB 2.0 Transceiver Macrocell Interface
V1 – V6	Ventrículos de 1 a 6
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit

SUMÁRIO

INTRODUÇÃO	17
1. Objetivos	20
1.1. Objetivos Gerais	20
1.2. Objetivos Específicos	20
2. Metodologia.....	21
3. Fundamentação Teórica	23
3.1. Sinais Eletrocardiográficos	23
3.2. Monitores Holter	26
3.2.1. Interação Com o Usuário.....	28
3.2.2. Especificações Técnicas	29
3.3. Simuladores de ECG	31
3.4. Sistemas Embarcados.....	32
3.4.1. System On a Chip.....	35
3.4.2. NIOS II.....	37
3.5. Computação Reconfigurável	38
3.5.1. FPGA's	38
3.6. Compressão de Dados	40
3.6.1. Informação e Entropia	42
3.6.2. Codificação da Entropia.....	43
3.6.3. Modelo de Compressão.....	44
3.6.4. PPM.....	44
3.6.5. Codificação Aritmética	47
3.7. Trabalhos Relacionados	48
4. Arquitetura do Sistema Embarcado.....	53
4.1. Módulos da Arquitetura.....	53

4.2. Descrição no Nível de Sistema.....	54
4.2.1. Módulo Simulador de Sinais ECG	55
4.2.2. Módulo de Compressão.....	58
4.3. Camada de Hardware.....	63
4.4. Sistema para Descompressão / Visualização de Sinais.....	69
5. Trabalhos Efetuados e Resultados Obtidos	71
5.1. Testes e Validação do Módulo Compressor.....	71
5.2. Testes e Validação do Módulo Simulador	77
5.3. Testes e Validação do Sistema de Compressão em Tempo Real	79
5.4. Tamanho de Código e Recursos Embarcados.....	80
Conclusões	82
Referências Bibliográficas.....	85
Glossário	90
APÊNDICE A - Artigo “ECG DATA COMPRESSION FOR AN EMBEDDED SYSTEM OF A HOLTER MONITOR USING PPM”	95
APÊNDICE B - Artigo “EMBEDDED SYSTEM THAT SIMULATES ECG WAVEFORMS”	98
APÊNDICE C - Diagrama UML de Classes do Módulo de Compressão	102
APÊNDICE D - Implementação em Linguagem C do Sistema Embarcado	103

INTRODUÇÃO

Monitores holter são dispositivos portáteis capazes de captar sinais eletrocardiográficos de pacientes durante o período de 24 horas ou mais de gravação. Tais gravações são úteis para cardiologistas, pois possibilitam a detecção de vários problemas do coração. Seu período estendido de gravação é muitas vezes útil para a observação de arritmias cardíacas ocasionais, que seriam difíceis de serem identificadas em um período menor de gravação, como em um exame de eletrocardiografia.

Os dispositivos antigos costumavam gravar os dados em fitas cassete. Os modelos mais atuais utilizam dispositivos de memória flash para a armazenagem de dados. Devido ao longo período de gravação deste tipo de dispositivo, é de suma importância o uso de memórias flash que tenham capacidades suficientes para armazenar grandes faixas de sinais ECG durante o período exigido de gravação (OTTLEY, 2007). Também se faz necessária a inserção de recursos adicionais que possam facilitar o processo de armazenamento, transmissão e análise de sinais, permitindo ao usuário do equipamento tratar sinais ECG de forma simples e rápida (OTTLEY, 2007; JIN et al., 2007; SOUZA et al., 2007).

Constatou-se, através de levantamento de recursos e características em monitores holter comerciais, que grande parte não aplica a compressão em sinais ECG captados, ou simplesmente não divulgam claramente tais informações ao consumidor (BRAEMAR, 2009; FUKUDA DENSHI, 2009; INNOMED MEDICAL, 2009). Na falta deste recurso, fabricantes projetam dispositivos que não aproveitam sua capacidade de armazenamento de forma inteligente, sendo obrigados a adotarem memórias flash de alta capacidade. Quanto maior a capacidade de armazenamento de uma memória flash, maior será seu consumo de energia e mais cara ela será, tornando o monitor holter mais dispendioso.

Também foi constatado que tais dispositivos utilizam interfaces RS232, USB, cabos Ethernet, Bluetooth ou Wireless na transmissão dos sinais ECG captados. Caso os dispositivos comerciais comprimissem os sinais gravados, tais sinais seriam transmitidos muito mais rapidamente, acelerando o acesso, análise e processamento dos mesmos.

Utilizando mecanismos de compressão neste tipo de monitor, os recursos de armazenamento e transmissão de dados serão usados com maior eficiência. Sinais ECG comprimidos requisitam menor espaço de armazenamento e podem ser submetidos mais rapidamente através da Internet ou em uma rede Wireless. Como consequência, o dispositivo poderá ser comercializado com preços bem mais baixos que os praticados atualmente, exigindo menos componentes, menor consumo de energia, tornando-o mais portátil, com possibilidade de agregação de novos recursos (SOUZA et al., 2007).

Quando a compressão de sinais não é aplicada, a redução de recursos de armazenamento pode necessitar da redução dos parâmetros de gravação, como das taxas de amostragem ou do tamanho das amostras, reduzindo dessa forma a quantidade de informação do sinal gravado (OTTLEY, 2007).

Também foi observado que vários projetos acadêmicos relacionados ao tema focam a compressão de sinais como um importante recurso para um projeto de monitor, mas utilizam algoritmos de compressão menos robustos que o PPM, como a Codificação Preditiva Linear, em parceria com a Técnica de Particionamento de Sinal (OTTLEY, 2007). Muitos utilizam algoritmos de compressão com perdas como o *Sample-Skipping Method with Area Error Limitation* (FRANCESCO et al., 1988), EZW (*Embedded Zerotree Wavelet*) (BOUKAACHE, 2006) e JPEG2000 (BILGIN et al., 2003); tais algoritmos, apesar de comprimir sinais ECG com grandes taxas de compressão, podem descartar várias amostras de um sinal ECG original, omitindo dados importantes que poderiam indicar algum sintoma, doença ou anomalia.

Motivados pelos benefícios que a compressão de dados trará para tornar um monitor holter mais eficiente, barato e robusto, visou-se a utilização do algoritmo PPM no sistema

embarcado. O mesmo é considerado um dos compressores sem perdas de propósito genérico mais eficazes da atualidade. Através do mesmo, podem-se obter taxas de compressão expressivas para arquivos com amostras de sinais ECG (SOUZA et al., 2007).

O trabalho está dividido em seis capítulos. O capítulo 1 apresenta os objetivos gerais e específicos da pesquisa. No capítulo 2 é apresentada a metodologia de desenvolvimento do referido trabalho. O capítulo 3 apresenta a fundamentação teórica para os trabalhos desenvolvidos, sendo introduzidos os assuntos e conceitos utilizados para a produção deste documento, com destaque para os conceitos de compressão de dados, sistemas embarcados, além da descrição de trabalhos relacionados. O capítulo 4 contém uma descrição detalhada da arquitetura do sistema embarcado em questão, apresentando os módulos de arquitetura desenvolvidos no nível de software, a camada de hardware utilizada na prototipação, além de apresentar o software desenvolvido para decodificar e apresentar na forma de gráficos os sinais codificados pelo sistema embarcado. No capítulo 5 são apresentados os trabalhos efetuados e os resultados obtidos para cada fase do projeto, desde o desenvolvimento e validação do módulo compressor, passando pelo desenvolvimento e validação do módulo de simulação de sinais ECG, até a integração dos módulos de simulação e compressão de sinais e validação do sistema autônomo para geração e compressão de sinais ECG. O capítulo 6 apresenta as considerações finais do trabalho desenvolvido, contendo uma discussão sobre trabalhos futuros, dificuldades encontradas e conclusões finais.

1. Objetivos

1.1. Objetivos Gerais

O objetivo geral deste trabalho é desenvolver um sistema embarcado para um monitor holter versátil, eficiente e robusto, com recursos de hardware e software que otimizem a captura, o armazenamento e a análise de sinais eletrocardiográficos, beneficiando desta forma pacientes que dependam dos resultados obtidos através de tais sinais na obtenção de diagnósticos.

1.2. Objetivos Específicos

Os principais objetivos específicos são:

- Implementação em linguagem C do software embarcado para compressão de sinais ECG;
- Implementação em linguagem C do software embarcado para simulação de sinais ECG, utilizando as Séries de Fourier;
- Configuração do *soft-core* Nios II e periféricos na placa de prototipação para atendimento das funcionalidades do projeto;
- Implementação em linguagem Java do sistema para visualização, análise e validação dos sinais captados, comprimidos e descomprimidos;
- Execução de testes para a arquitetura desenvolvida, a fim de validar a mesma;

2. Metodologia

A idéia do referido trabalho é projetar um monitor holter através do desenvolvimento de módulos de software embarcados, modelados utilizando a linguagem C, simulados através de ferramentas CAD e IDE's para a síntese de circuitos específicos em FPGA, utilizando o processador Nios II para validação.

O processador Nios II é *soft-core* RISC de 32 bits distribuído pela *Altera Corporation*[®]. Este trabalho envolve a descrição para a plataforma Nios II, incluindo periféricos internos e acesso a dispositivos externos, desenvolvimento de software através de GNU C/C++ no Eclipse IDE (*Integrated Development Environment*) e *debugging* provido por hardware. Uma série de periféricos padrão compõe a plataforma, existindo ainda a possibilidade de desenvolvimento de periféricos personalizados.

Para projetar o sistema embarcado objeto deste trabalho, foi utilizado o *Altera*[®] *Nios Development Board, Stratix Edition*. A referida placa de desenvolvimento provê uma plataforma de hardware para o desenvolvimento de sistemas embarcados em dispositivos *Altera*[®]. Os softwares utilizados na definição do projeto são o *Altera*[®] *Quartus II*, *SOPC Builder* e o *Nios II Embedded Design Suite*.

O *SOPC Builder* é utilizado na criação de sistemas baseados no processador Nios II. O referido software é mais que um construtor de sistemas Nios II; O mesmo é uma ferramenta de propósito geral para a criação de sistemas que possam ou não conter um processador, ou que incluam outro processador além do Nios II.

Através do Quartus II são desenvolvidos alguns módulos da arquitetura do sistema embarcado proposto, utilizando a linguagem VHDL. O Nios II EDS, que possui uma *suite* para desenvolvimento de software em C/C++, é usado na implementação dos módulos a serem executados pelo núcleo do Nios II. O software desenvolvido neste aplicativo é carregado e executado pela placa de desenvolvimento utilizada.

Para a visualização dos sinais captados e comprimidos pelo sistema embarcado, se fez necessário o desenvolvimento de software que possui a incumbência de obter os sinais ECG captados pelo dispositivo, descomprimí-los e exibi-los através de gráficos, em um sistema de processamento hospedeiro dedicado para a análise e processamento de tais sinais. Para o desenvolvimento de tal software, foi utilizada a linguagem Java, a biblioteca *open source* JFreeChart para a apresentação dos sinais ECG em gráficos e o Netbeans IDE na produção de código-fonte e no desenvolvimento de GUI.

3. Fundamentação Teórica

3.1. Sinais Eletrocardiográficos

O eletrocardiograma (ECG) é o registro dos fenômenos elétricos que se originam durante a atividade cardíaca. O eletrocardiógrafo é um galvanômetro (aparelho que mede a diferença de potencial entre dois pontos) que mede pequenas intensidades de corrente, recolhidas a partir de eletrodos (pequenas placas de metal conectadas a um fio condutor) dispostos em determinados pontos do corpo humano. Ele serve como auxiliar valioso no diagnóstico de grande número de cardiopatias e outras condições como, por exemplo, os distúrbios hidroeletrólíticos (RAMOS, 2007).

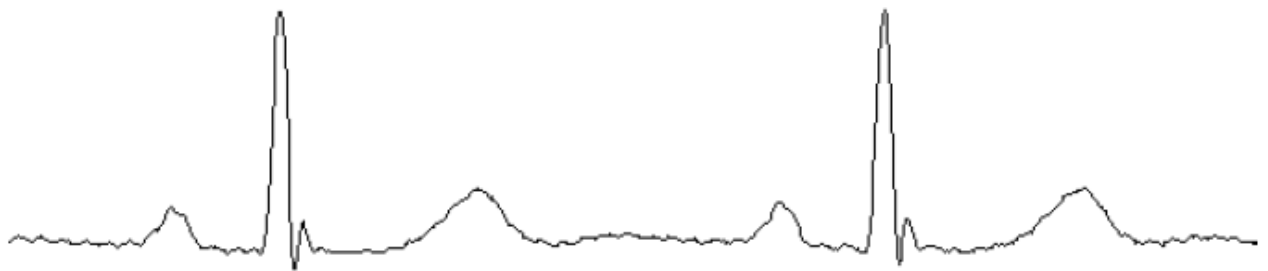


Figura 3.1. Um sinal ECG típico

A atividade elétrica do coração é medida através de eletrodos, que são colocados na pele. O dispositivo então obtém a atividade cardíaca e a armazena, usualmente em uma memória flash que incorpora a arquitetura de um monitor holter, durante o período de 24 horas ou mais.

Inicialmente, sensores eram colocados em pontos do braço esquerdo, braço direito e perna esquerda, pois são eletricamente equidistantes do coração. Cada par de pontos a

partir dos três são localizações padrão para medição. Desde então, nove pontos de medição padrão foram adicionados, utilizando um ponto central próximo ao coração como referência (conceito do Triângulo de Einthoven, formado pelos pontos do braço esquerdo, direito e perna esquerda. Vide Figuras 3.2 e 3.3), juntamente com as três derivações aVL, aVR, e aVF, bem como as seis derivações precordiais V1-V6. Cada derivação reflete diferentes porções do coração e dá pistas sobre qual parte do músculo cardíaco é afetado por alguma distorção. Os gráficos gerados a partir de cada condutor variam na aparência, devido a diferentes forças em pulsos elétricos e polaridades em cada ponto de medição.

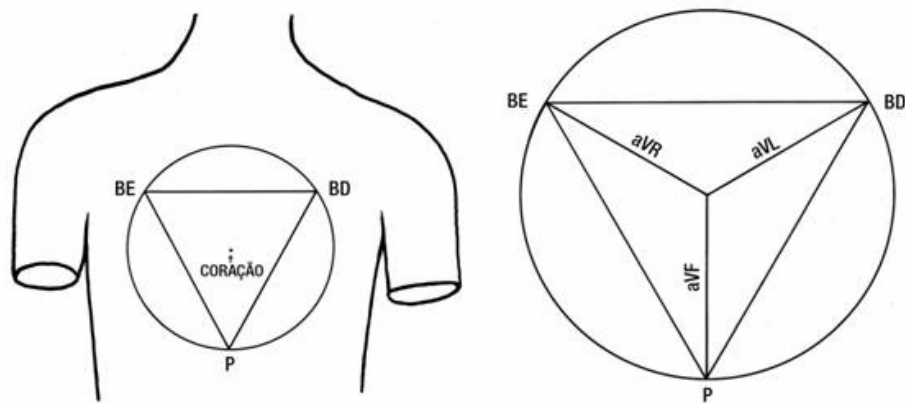


Figura 3.2. O Triângulo de Einthoven, os pontos BE (braço esquerdo), BD (braço direito), P (perna esquerda) e os pontos aVL, aVF e aVR

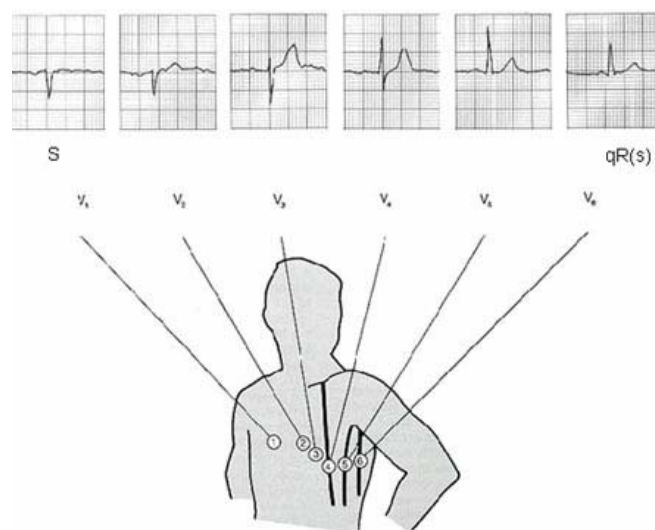


Figura 3.3. Formação do traçado QRS nas derivações precordiais

Uma frequência cardíaca de repouso típica varia em gênero, idade e condições físicas, mas, para adultos em geral, a mesma está entre 70-75 batimentos por minuto, com a faixa de 50-100 sendo considerada um intervalo normal.

É importante constatar que o coração, durante sua atividade, age como um gerador de correntes elétricas e que estas correntes, espalhando-se no meio condutor que é o coração, geram potenciais elétricos cuja evolução no tempo e no espaço podem ser aproximadamente previstas. Assim funciona o eletrocardiograma que nada mais é do que o registro das variações do potencial elétrico do meio extracelular decorrentes da atividade cardíaca. O ECG consiste de ondas características (P, Q, R, S, T e U) as quais correspondem a eventos elétricos da ativação do miocárdio. A onda P corresponde à despolarização atrial, o complexo QRS à despolarização ventricular, a onda T à repolarização dos ventrículos e a onda U à repolarização da fibra de Purkinje. A convenção internacional adotada para o ECG determina que as deflexões para cima são positivas e as deflexões para baixo são negativas (NOUAILHETAS et al, 2003).

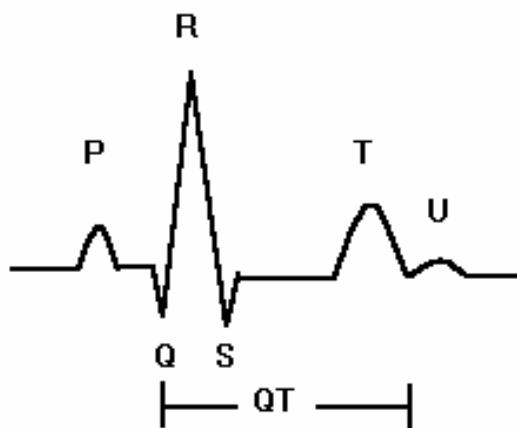


Figura 3.4. Ondas componentes de um ECG

O registro da atividade elétrica cardíaca na superfície do tórax é uma qualidade do método que não é obtida por outras técnicas e esse é, sem dúvidas, o procedimento mais utilizado para auxiliar o diagnóstico das doenças cardíacas. Este método é de simples

realização, seguro, reproduzível, podendo ser usado em grandes estudos devido ao seu baixo custo. A metodologia foi favorecida pelo desenvolvimento de aparelhos mais sofisticados, computadorizados e menores, o que facilitou a sua utilização nas situações de emergência e nas rotinas ambulatoriais e hospitalares (SBC/ECG, 2009).

A tecnologia dos computadores, inclusive os pessoais, trouxe poderosos sistemas de captação de sinais e de avaliação de algoritmos, aumentando a dimensão do uso do eletrocardiograma (SBC/ECG, 2009).

Pode-se afirmar, enfim, que o eletrocardiograma é ferramenta fundamental para os especialistas nas decisões clínicas, pois, ainda que possa ter sido interpretado em serviços de eletrografia, é parte integrante das discussões clínicas e à beira do leito (SOCIEDADE BRASILEIRA 100 LIMITE, 2009).

3.2. Monitores Holter

O monitor holter, inventado pelo Dr. Norman J. Holter em 1961, é um dispositivo móvel para monitoramento contínuo da atividade elétrica do coração durante o período de 24 horas ou mais. Seu período alongado de gravação é útil para a observação ocasional de arritmias cardíacas que seriam difíceis de identificar, em curto período de tempo. Para pacientes com mais sintomas transitórios, é usual o acompanhamento de eventos cardíacos no período de um mês ou mais.

O referido dispositivo acompanha os pacientes durante todo o período de gravação, preso a eles graças a pulseiras colocadas ao longo do ombro ou em torno da cintura. Dependendo da dimensão do monitor, é possível o transporte do equipamento no bolso, ou em cintos, graças a presilhas similares às de telefones celulares.

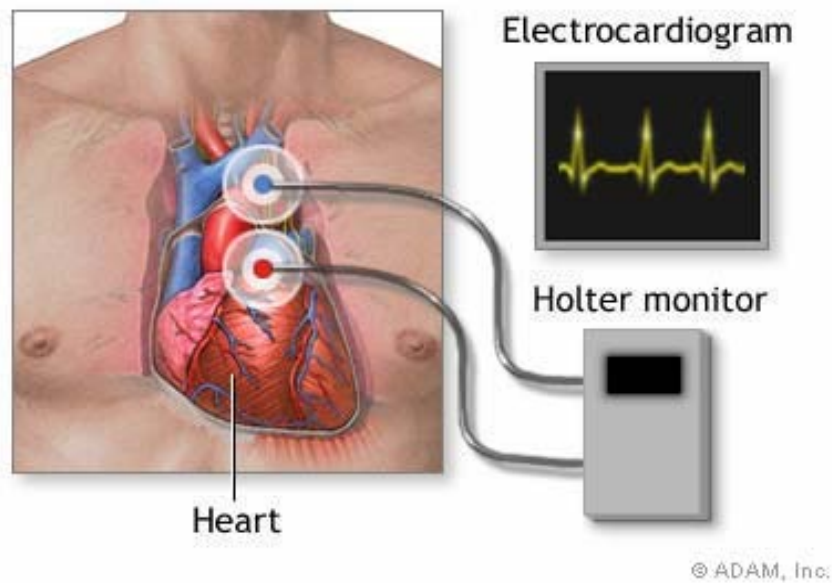


Figura 3.5. Atuação de um monitor holter

Ligados ao monitor, encontramos 5, 7 ou 10 condutores. Estes condutores possuem discos de metal chamados eletrodos, que são colados em pontos específicos no tórax do paciente graças a adesivos presentes ao redor dos discos, como é demonstrado na Figura 3.6. Por serem muito sensíveis, os discos de metal podem captar os impulsos elétricos do coração.

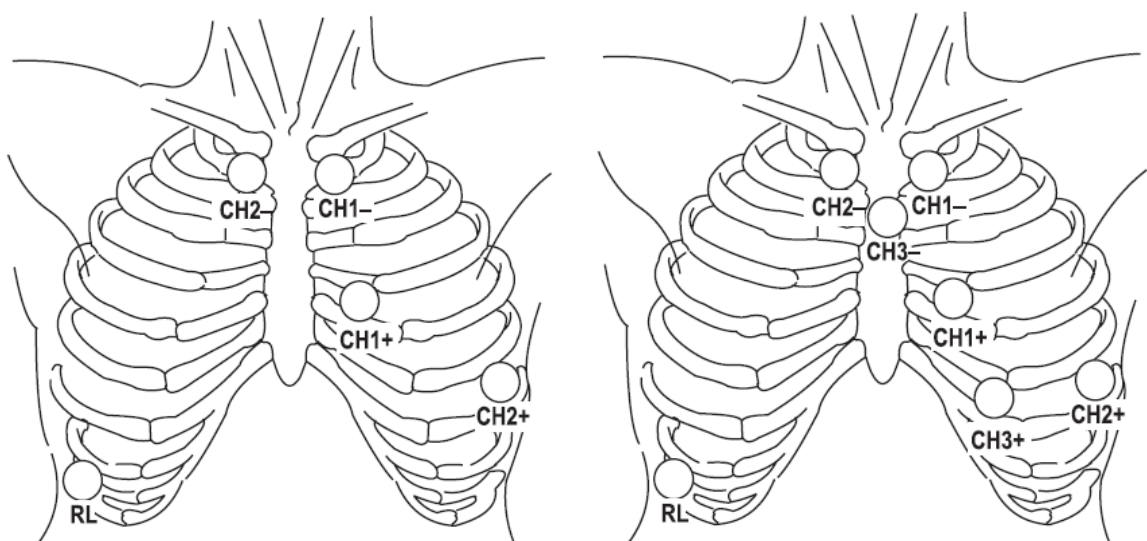


Figura 3.6. Colocação dos eletrodos em 5 ou 7 derivações

3.2.1. Interação Com o Usuário

Monitores holter comerciais e acadêmicos utilizam alternativas variadas para disponibilizar informações durante o período de gravação de sinais ECG. A presença de determinados dispositivos de I/O ou não em monitores holter irá depender de dimensões e recursos alçados pelos projetistas dos mesmos.

Em monitores de tamanho muito reduzido, é normal a utilização de LED's para a passagem de *feedback* do estado do sistema. Determinadas disposições de acendimento dos diodos podem representar mensagens referentes à gravação. Outro recurso utilizado é a transmissão de sinais sonoros, onde determinados sons emitidos pelo monitor identificam algum estado, como erro ou conclusão de gravação.



Figura 3.7. Monitor holter que interage via sinais sonoros

Monitores mais robustos também utilizam com frequência pequenas telas LCD para a apresentação de feedback ao paciente. Estas telas apresentam as mais diversas informações durante uma gravação, como relógios para averiguação de período de uso, gráficos de ECG exibido em tempo real, mensagens de erro na gravação, canais captados, frequência cardíaca, entre outros.



Figura 3.8. Monitor holter com LCD

Para o download, exibição, impressão, processamento e análise mais precisa dos sinais ECG captados, são utilizados softwares fornecidos pelos fabricantes. Os mesmos normalmente acompanham os monitores em CD's de instalação, gratuitamente, ou vendidos separadamente.

3.2.2. Especificações Técnicas

As especificações técnicas dos monitores holter são similares quanto ao método de gravação de dados, captação de sinais e alimentação. As diferenças são evidentes quanto aos recursos de I/O, dispositivos de armazenamento utilizados, quantidade de canais utilizados na captação, sistemas embarcados, interfaces para conexão com outros dispositivos, transmissão dos dados, dimensão e peso (BRAEMAR, 2009; FUKUDA DENSHI, 2009; INNOMED MEDICAL, 2009).

O método padrão para gravação de dados é o digital. Como o aparelho acompanha o paciente durante longos períodos de tempo, é necessário que o monitor gere arquivos de

amostras ECG e os armazene em dispositivos de memória que integram o equipamento (SOUZA, 2007).

A alimentação dos monitores holter é feita através de pilhas. Dependendo da dimensão e da quantidade de energia necessária para seu funcionamento, os equipamentos utilizam diversos tipos de pilha, desde a AA até as utilizadas em relógios de pulso.

A quantidade média de canais utilizados variam de 2 a 8 canais, com resolução de amostragem variando entre 8 e 16 bits. A frequência de amostragem varia entre 128 e 1024 amostras por segundo.

Os recursos de I/O presentes nos monitores são LED's, displays LCD, pequenas saídas de áudio e botões para controle de comandos. A presença e quantidade dos referidos recursos variam de monitor para monitor.

O dispositivo de memória utilizado pela maioria dos monitores holter é o Flash Card ou SD Card, com capacidades de armazenamento variando entre 16 MB a 2 GB. Utilização de interface USB é comum em alguns monitores comerciais e acadêmicos, porém são mais utilizados em conexões PC/Monitor, para a transferência dos arquivos gravados, do que para conexões com dispositivos de armazenamento auxiliar (BRAEMAR, 2009; FUKUDA DENSHI, 2009; INNOMED MEDICAL, 2009).

Sistemas de compressão de dados são embarcados em monitores holter para melhor aproveitamento do espaço de armazenamento disponível. Porém, este recurso é mais presente em projetos acadêmicos (SOUZA et al., 2007; OTTLEY, 2007; Bilgin et al., 2003). Grande parte dos monitores comerciais não oferece este recurso, ou não deixam clara sua utilização (BRAEMAR, 2009; FUKUDA DENSHI, 2009; INNOMED MEDICAL, 2009).



Figura 3.9. Monitor holter sem compressão BI9800TL+3, da Biomedical Instruments

As formas utilizadas para a transmissão de dados entre monitores holter e sistemas de processamento hospedeiros são variadas. São utilizados o Flash Card, cabo par trançado, USB, Infra-Vermelho e Wireless.

3.3. Simuladores de ECG

O objetivo dos simuladores de ECG é produzir formas de onda ECG típicas a partir de diferentes estímulos. O uso de simuladores de ECG tem diversas vantagens, como economizar tempo e eliminar dificuldades na obtenção de sinais, em comparação com a captação real de sinais ECG através de métodos invasivos e não-invasivos. Os mesmos permitem a análise e o estudo de sinais ECG normais e anormais (KARTHIK, 2006).

Existem dois tipos de simuladores de ECG no mercado: simuladores implementados em software e em hardware. Os softwares simuladores de ECG são desenvolvidos utilizando várias linguagens de programação, como Java e MATLAB Script, geram sinais a partir da implementação de funções periódicas e podem ser distribuídos gratuitamente, como os simuladores com menos recursos, ou pagos, para os mais elaborados. Já os dispositivos de simulação (hardware) são normalmente portáteis, contém um microprocessador que inclui todas as funções digitais e analógicas, geram sinais através da síntese digital de frequência e normalmente são aplicados no auxílio à manutenção corretiva e preventiva de vários tipos de equipamento de eletrocardiografia, como monitores

cardíacos, registradores de ECG e monitores Holter, no suporte na avaliação da real necessidade de manutenção de monitores Holter, na verificação periódica dos limites de operação de monitores cardíacos e equipamentos semelhantes, na avaliação e comparação de desempenhos confiáveis entre equipamentos de fabricantes diferentes, além de aplicados na avaliação, comparação e validação de programas de análise de ECG tais como centrais de Holter.



Figura 3.10. Exemplo de hardware simulador de ECG

3.4. Sistemas Embarcados

Muitos consumidores de eletrodomésticos e eletroeletrônicos não sabem, mas é corriqueiro que tais aparelhos possuam embutidos em sua estrutura computadores para processamento de tarefas específicas, como a obtenção e exibição da temperatura de um freezer em um display, ou a execução de uma contagem regressiva em microondas, ou até mesmo o controle de batimentos automáticos em batedeiras.

Vários outros equipamentos adotam esta tecnologia e fazem com que tais sistemas estejam cada vez mais presentes no cotidiano. Em carros, por exemplo, sistemas

embarcados estão presentes em computadores de bordo, injeção eletrônica e sensores; Em celulares, tal tecnologia está sempre presente; Telefones, roteadores, calculadoras, aviões, videogames, MP4 players (vide Figura 3.11) e sistemas de segurança são outros exemplos de aparelhos que aproveitam os benefícios da utilização de sistemas embarcados.



Figura 3.11. MP4 Player

Sistemas embarcados são pequenos sistemas de computação que são projetados e programados para controlar pequenas tarefas, ao contrário dos computadores de propósito geral. Sistemas embarcados normalmente possuem especificações fixas de hardware escolhidas para as tarefas que eles são projetados a realizar (OTTLEY, 2007).

A especialização dos sistemas embarcados é a característica principal que os diferenciam dos computadores de propósito geral. Os computadores de propósito geral são projetados para fazer diversas tarefas, além de possuírem especificações variadas. Através de um desktop (PC), exemplo claro de um computador de propósito geral, pode-se acessar a Internet, gravar um CD, digitar um documento, entre outros.

Os sistemas embarcados são compostos por hardware e software, de forma interdependente. Sendo sistemas, os mesmos requisitam alguma ação (uma entrada) para que possam efetuar algum tipo de processamento. O processamento de determinada(s) entrada(s) resultará em uma reação (uma saída) específica para o sistema. Ligação com dispositivos de interação com o usuário e com outros sistemas embarcados também são possíveis.

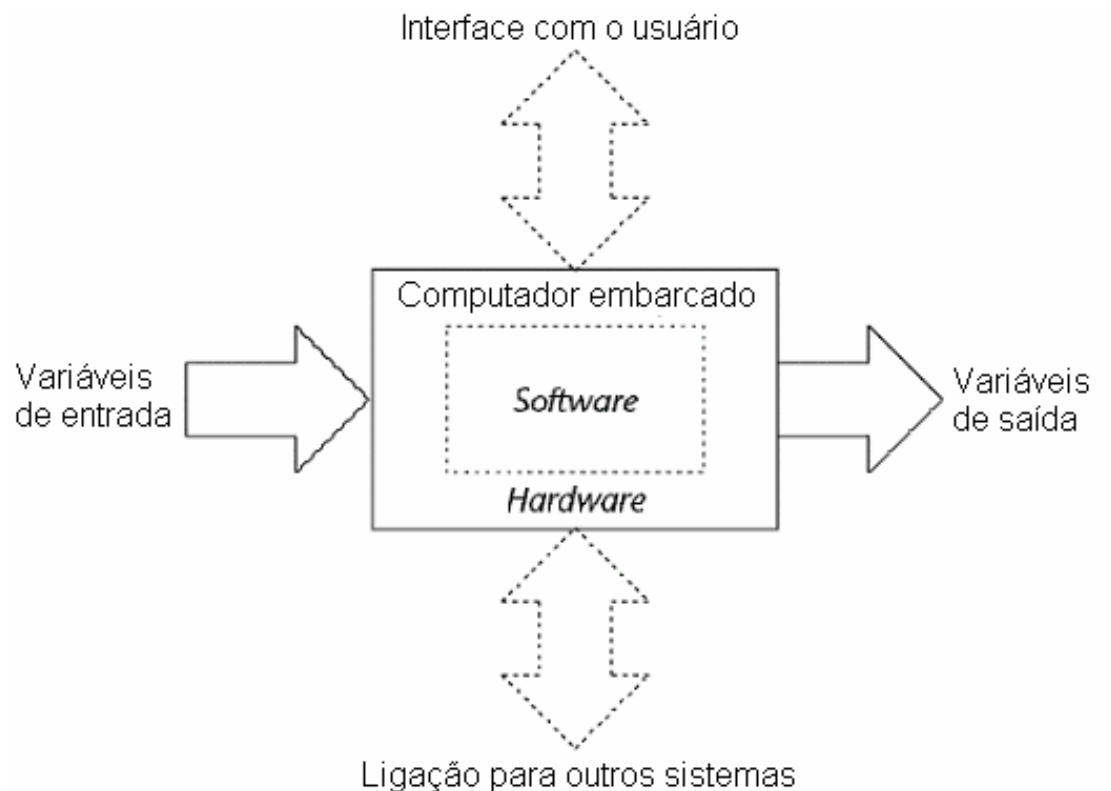


Figura 3.12. A essência dos sistemas embarcados

Um exemplo da aplicação deste esquema é o funcionamento do sistema embarcado para sensores de proximidade de carros. Se a chave estiver nas proximidades do veículo, as portas do automóvel abrem automaticamente, sendo desnecessário o pressionamento de qualquer botão ou inserir a chave na fechadura da porta. Portanto, neste exemplo, a entrada seria a chave e a saída seria a ação das portas; O sistema embarcado avaliaria a distância

da chave e, baseado em tal informação, abriria as portas do carro ou as manteriam fechadas.

3.4.1. System On a Chip

Uma tendência tecnológica é o desenvolvimento de sistemas embarcados, na sua totalidade, em um único chip. Tal tendência é conhecida por *System on a Chip*, SoC ou SOC.

System on a Chip refere-se à integração de todos os componentes de um computador ou outro sistema eletrônico em um único circuito integrado (chip com programação de conexões de hardware e software). O mesmo pode conter várias funções, como digitais e analógicas, todas em um. Uma aplicação típica está na área de sistemas embarcados.

A estrutura típica de um SOC consiste em um microcontrolador ou microprocessador; blocos de memória, incluindo ROM, RAM, EEPROM e Flash; fontes de *Timing*, incluindo osciladores e *phase-locked loops*; periféricos, incluindo *Timers* contadores, *Timers* de tempo real e geradores *power-on reset*; interfaces externas, incluindo padrões como USB, FireWire, Ethernet, USART e SPI; interfaces analógicas, incluindo ADCs e DACs; reguladores de voltagem e circuitos para gerenciamento de energia. Estes blocos são conectados através de um barramento padrão para indústria ou padrão proprietário. Controladores DMA (*Direct Memory Access*), que permitem certos subsistemas de hardware acessar sistemas de memória para leitura e/ou escrita independentemente da unidade central de processamento, incrementam a taxa de transferência de dados nos SOC's.

Em sua maioria, SOC's são criados a partir de componentes de hardware pré-validados, agrupados com a utilização de ferramentas CAD e de drivers que controlam sua operação, desenvolvidos em um ambiente de desenvolvimento de software (VARGAS et al, 2007).

O desenvolvimento de sistemas embarcados através de projetos SOC's é muito atrativo, principalmente graças aos baixos custos que esta metodologia proporciona. Como é necessária apenas a fabricação de um único dispositivo, os custos de produção de chips podem ser reduzidos significativamente. Isto ocorre devido ao desenvolvimento de protótipos, que descartam a necessidade da produção de circuitos impressos. Graças a computação reconfigurável presente nos SOC's, onde encontra-se componentes que podem ser reconfigurados para implementar uma funcionalidade específica, a produção de protótipos reconfiguráveis é viável. Em caso de erro em um determinado projeto, basta reconfigurar o hardware responsável pelo mesmo erro. Caso o erro fosse constatado após a produção de circuito impresso, o mesmo seria descartado, e outro circuito deveria ser produzido com a correção.

A possibilidade de tornar projetos de hardware modulares é outro grande benefício que esta abordagem oferece. Como os SOC's são criados a partir de componentes de hardware interligados, os mesmos componentes podem ser desenvolvidos e validados de forma independente, mesmo na ausência de componentes que possuam uma ligação direta. Conhecidas as entradas e saídas de cada componente (as interfaces), pode-se simular o funcionamento dos mesmos através de blocos de teste. Desta forma, os projetos são concluídos mais rapidamente, com alta confiabilidade e qualidade.

A modularização de projetos de hardware também beneficia o reaproveitamento de componentes desenvolvidos. Um módulo que funciona como um conversor analógico-digital, que compõe a arquitetura de um polígrafo digital, poderá ser reaproveitado na arquitetura de um monitor holter, por exemplo. Desta forma, trabalhos efetuados anteriormente não precisarão ser re-executados, diminuindo significativamente o tempo de conclusão dos projetos de hardware.

3.4.2. NIOS II

O Nios II é um processador RISC com recursos configuráveis e escaláveis de propósito geral que provê as seguintes funcionalidades: um conjunto completo de instruções, espaço de endereçamento e de dados de 32 bits; 32 registradores de propósito geral; 32 fontes de interrupções externas; instruções simples de 32 X 32, utilizados na multiplicação e divisão para geração de resultados de 32 bits; instruções dedicadas para a computação de produtos de multiplicação de 64 bits e 128 bits; instruções de ponto flutuante para operações de ponto flutuante com precisão simples; acesso a uma variedade de periféricos *on-chip*, e interfaces para memórias e periféricos *on-chip* (ALTERA CORPORATION, 2004).

O sistema do processador Nios II é equivalente a um microcontrolador ou “computador em um chip” que inclui um processador e uma combinação de periféricos e memória em um chip simples. O sistema Nios II consiste em um processador Nios II, uma série de periféricos *on-chip*, memória *on-chip*, e interfaces para a memória *on-chip*, todas implementadas sobre um único dispositivo da fabricante Altera[®]. Tal como um microcontrolador, todos os sistemas do processador Nios II utilizam um conjunto consistente de instruções e um modelo de programação (ALTERA CORPORATION, 2004).

Trata-se de um processador *soft-core*, que nada mais é do que um microprocessador integralmente descrito em software capaz de ser sintetizado em hardware programável (FPGA).

Similar ao Nios original, a arquitetura Nios II é implementada inteiramente na lógica programável e blocos de memória dos FPGA's Altera[®]. A natureza *soft-core* do processador Nios II permite ao projetista de um sistema especificar e gerar um núcleo Nios II próprio, sob medida para as exigências específicas da aplicação a ser desenvolvida. Projetistas de sistemas podem estender as funcionalidades básicas adicionando uma unidade de gerenciamento de memória predefinida, ou definindo instruções e periféricos próprios.

3.5. Computação Reconfigurável

A computação reconfigurável é um paradigma de computação no qual circuitos integrados em hardware programável são usados em conjunto com softwares na reconfiguração de circuitos integrados FPGA de forma dinâmica para produzir arquiteturas de computador sob demanda em tempo de execução (MARTINS, 2009). Este paradigma baseia-se em dispositivos lógico reprogramáveis que podem atingir um desempenho elevado e, ao mesmo tempo, fornecer a flexibilidade da programação a nível de portas lógicas (SKLIAROVA et al., 2003).

Trata-se de uma solução intermediária entre as soluções em hardware e software, com objetivos, metas e motivações relacionadas com a melhoria de desempenho, flexibilidade, generalidade, eficiência, custo e outros. E entre os motivos para o seu estudo devemos considerar e analisar a demanda computacional de algumas importantes aplicações atuais e futuras, a inadequação de alguns modelos e estilos de computação atuais em termos de desempenho e flexibilidade, a posição e interesse das principais empresas e universidades do mundo e também a evolução dos dispositivos computacionais em termos de arquitetura e tecnologia (MARTINS, 2009).

Os FPGA's são comumente utilizados na computação reconfigurável, devido a sua arquitetura flexível que permite o desenvolvimento de vários projetos de hardware. Entre estes projetos, pode-se listar a implementação de controladores de dispositivos, circuitos de codificação, lógica arbitrária, prototipagem e emulação de sistemas, etc. (SKLIAROVA et al., 2003).

3.5.1. FPGA's

FPGA (*Field Programmable Gate Array*) é um circuito integrado que contém um grande número (na ordem de milhões) de unidades lógicas. Tais unidades lógicas podem

ser vistas como componentes padrões que podem ser configurados independentemente e interconectados a partir de uma matriz programável de trilhas condutoras e chaves programáveis. Um arquivo binário é gerado para configuração do FPGA a partir de ferramentas de software, seguindo um determinado fluxo de projeto. Esse arquivo binário contém as informações necessárias para especificar a função de cada unidade lógica e para seletivamente fechar os *switches* da matriz de interconexão. Resumindo, o arranjo de unidades lógicas e a matriz de interconexão, que podem ser programados pelo usuário, formam a estrutura básica de um FPGA para a especificação de circuitos integrados complexos (CARDOSO, 2007).

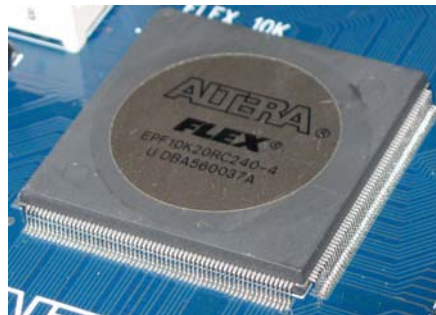


Figura 3.13. Um FPGA da Altera® com 20.000 células

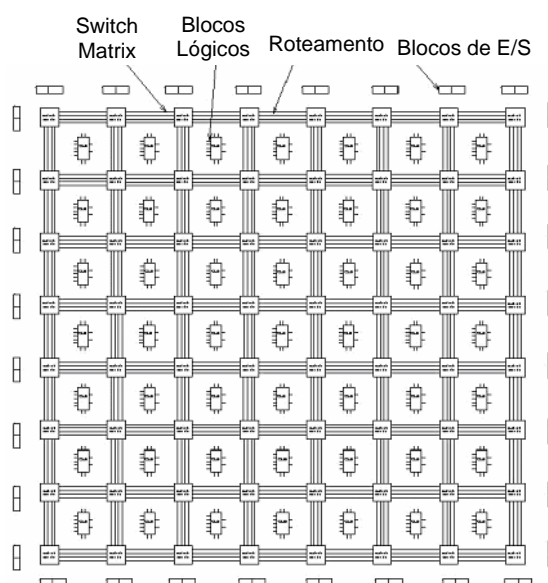


Figura 3.14. Estrutura de um FPGA

Um único FPGA pode simular desde um processador simples até outros tipos de circuito. Os modelos recentes incluem uma pequena quantidade de memória RAM e circuitos de apoio, de forma que você pode ter um sistema completo usando apenas um chip FPGA previamente programado, um chip de memória EPROM (ou memória flash) com o software, a placa de circuito com as trilhas e conectores e uma bateria ou outra fonte de energia.

3.6. Compressão de Dados

Compressão de dados é o processo de conversão de um fluxo de dados (*stream*) de entrada (os dados originais brutos) em outro fluxo de dados (a saída) que possui tamanho reduzido. Tal processo é popular por duas razões: (1) Pessoas gostam de acumular dados e odeiam jogá-los fora. Não importa quão grande seja o dispositivo de armazenamento, mais cedo ou mais tarde ele irá transbordar. Compressão de dados é útil porque o mesmo retarda esta inevitabilidade. (2) Pessoas odeiam esperar muito tempo por transferências de dados. Quando sentamos à frente do computador, esperando o carregamento de uma página Web ou a conclusão do *download* de um arquivo, nós naturalmente sentimos que algo mais longo que alguns segundos é um longo tempo de espera (SALOMON, 2004).

A idéia da compressão é remover a redundância sugerida pela lei geral da compressão de dados, que é “atribuir códigos curtos para eventos comuns (símbolos ou frases) e códigos longos para eventos raros”. Existem vários caminhos para implementar esta regra, e uma análise para qualquer método de compressão mostra que todas trabalham obedecendo tal regra (SALOMON, 2004).

A compressão de dados é possível porque dados são normalmente representados no computador em um formato maior que o necessário. A razão para que representações de dados ineficientes (longos) sejam usados é que os mesmos são mais fáceis de serem

processados, e processamento de dados é mais comum e mais importante que compressão (SALOMON, 2004).

Existem dois tipos de compressão: compressão de dados com perdas e compressão de dados sem perdas. Na compressão de dados com perdas, como o próprio nome sugere, dados são perdidos na compressão para que maiores taxas de compressão sejam atingidas (quanto maior a taxa de compressão, menores arquivos são gerados). Devido a tal característica, este tipo de compressão gera maior compressão do que a compressão sem perdas, porém o arquivo descomprimido não é idêntico ao arquivo original. O uso desta técnica é útil, por exemplo, para a compressão de imagens, vídeo e sons, já que a perda de informações sensoriais é quase imperceptível. Porém o uso de tal técnica em outros tipos de *streams*, como sinais eletrocardiográficos, poderá eliminar informações importantes nos arquivos decodificados (descomprimidos), dificultando ou até inviabilizando o uso, análise ou interpretação dos mesmos.

Já a compressão sem perdas permite a reconstrução perfeita de um *stream* original a partir de um *stream* codificado, isto é, o *stream* descomprimido será idêntico ao *stream* original, diferentemente da compressão com perdas. Para que isto seja viável, a codificação é feita removendo-se a redundância do *stream* de dados, sem eliminar informações. Graças a isso, taxas de compressão obtidas a partir desta técnica serão menores do que a partir da compressão com perdas.

Vale frisar também que implementações para a compressão sem perdas são bem mais complexas e lentas na codificação e decodificação do que para a compressão com perdas. O tipo de *stream* a ser comprimido é quem ditará a melhor técnica de compressão a ser aplicada, isto é, quem demonstrará o melhor custo / benefício para a compressão de dados será o contexto de aplicação.

3.6.1. Informação e Entropia

Nós intuitivamente sabemos o que é informação. Nós constantemente recebemos e enviamos informações na forma de texto, fala e imagens. Nós também sabemos que informação é uma grandeza não-matemática elusiva que não pode ser precisamente definida, capturada e mensurada. As definições padrão dos dicionários para informação são: (1) conhecimento procedente a partir de estudo, experiência ou instrução; (2) conhecimento de um evento ou situação específica; inteligência; (3) uma coleção de fatos ou dados; (4) A ação de informar ou a condição de ser informado; comunicação do conhecimento (SALOMON, 2004).

A importância da teoria da informação está na quantificação da informação (SALOMON, 2004). A quantificação da informação está baseada na quantidade de “surpresa” contida em uma determinada mensagem, isto é, quanto maior for o grau de incerteza ou imprevisibilidade de uma mensagem, maior a quantidade de informação. Para medir a quantidade de “surpresa” em uma mensagem, utilizamos a entropia da informação. Quanto maior a “surpresa” gerada por uma mensagem, maior será sua entropia. Se uma mensagem é previsível, menos “surpresa” será gerada, menos informação será transmitida, menor será sua entropia.

A entropia da informação está intimamente relacionada à compressão de dados. Uma mensagem com alta previsibilidade (baixa entropia) pode ser considerada como uma mensagem redundante. Tal redundância poderá ser gerenciada via compressão, através da lei geral de compressão de dados. Logo, pode-se afirmar que mensagens com baixa entropia poderão ser codificadas sob altas taxas de compressão, devido à redundância inerente a elas. Mensagens com alta entropia possuem maior quantidade de informação ou “surpresa”, sendo menos redundantes. Pouca redundância gera compressão com baixas taxas. Desta forma, é possível o uso da entropia para avaliar a eficiência de uma codificação em uma mensagem.

3.6.2. Codificação da Entropia

De acordo com Shannon, conhecido como “o pai da Teoria da Informação”, a comunicação pode ser considerada como um problema matemático rigorosamente embasado na estatística, dando aos engenheiros da comunicação um modo de determinar a capacidade de um canal de comunicação em termos de ocorrência de bits (SHANNON, 1948). Foi Shannon quem definiu a entropia para medir a quantidade de informação em uma mensagem. Para calcular o valor da entropia, Shannon adotou o cálculo de logaritmo, definido como:

$$H(X) = - \sum_{x \in X} p(x) \log_2 p(x)$$

onde \log é o logaritmo na base 2, que determina o grau de caoticidade (incerteza, imprevisibilidade) da distribuição de probabilidade p_i . O mesmo adotou a mensuração da informação via logaritmo (entropia) devido aos seguintes fatores: (1) é praticamente mais útil. Parâmetros de importância para a engenharia como tempo, largura de banda, etc., tendem a variar linearmente com o logaritmo; (2) é mais próximo do nosso sentimento intuitivo quanto à medida justa. Isto está intimamente relacionado com (1) desde que nós intuitivamente medimos entidades através da comparação linear com normas comuns; (3) é matematicamente mais adequado (SHANNON, 1948).

A escolha da base do logaritmo para o cálculo da entropia corresponde à escolha de uma unidade para a mensuração da informação. Se a base 2 é usada, a unidade de medida resultante chama-se dígitos binários, ou bits (SHANNON, 1948). Portanto, a partir da entropia, pode-se prever quantos bits poderão ser utilizados para codificar uma determinada mensagem.

A codificação da entropia cria e atribui um código de prefixo único para cada símbolo que ocorra na entrada de um fluxo de dados. A mesma comprime dados através da substituição de cada símbolo de entrada de um *stream*, de tamanho fixo, pelo código de

prefixo correspondente, de tamanho variado. O tamanho de cada palavra-código (a quantidade de bits) é aproximadamente proporcional ao valor negativo do logaritmo de uma probabilidade, isto é, a sua entropia. Logo, símbolos mais comuns usam códigos de prefixo curtos.

Duas das técnicas de codificação da entropia mais comuns são a codificação de Huffman (1952), técnica que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo, através do uso de árvores binárias completas, e a codificação aritmética, descrita em detalhes neste capítulo.

3.6.3. Modelo de Compressão

Um modelo é um método utilizado para prever (para atribuir probabilidades para) os dados a serem comprimidos. Um modelo simples poderá ser construído a partir da leitura completa de um *stream* de entrada, contando-se o número de vezes que cada símbolo aparece em uma mensagem (a frequência de ocorrência de cada símbolo), e computando-se a probabilidade de ocorrência de cada símbolo. O *stream* de dados é inserido novamente (como entrada), símbolo a símbolo, e é comprimido utilizando a informação no modelo de probabilidades (SALOMON, 2004).

Um modelo que captura mais precisamente as características reais da fonte poderá reduzir a entropia da mesma, aumentando as oportunidades de compressão.

3.6.4. PPM

O PPM é um método sofisticado, parte do estado da arte do método de compressão originalmente desenvolvido por J. Cleary e I. Witten (1984), com extensões e uma

implementação por A. Moffat (1990). O método é baseado em um codificador que mantém um modelo estatístico do texto.

PPM consiste em uma técnica de modelagem estatística de contexto finito. A idéia por trás do PPM é gerar a probabilidade condicional para o caractere atual armazenando o contexto dos últimos n caracteres. Modelos que condicionam suas predições em símbolos imediatamente anteriores são chamados modelos de contexto finito de ordem K , onde K é o número de símbolos precedentes usados na predição. PPM emprega um conjunto de modelos contextuais de ordem fixada com diferentes valores de k , limitado superiormente a um valor K_{max} , para predizer os caracteres subseqüentes (COUTINHO et al., 2005).

A maneira mais simples para construir o modelo é manter um dicionário para toda possível seqüência s de n caracteres, e para cada seqüência armazenar contadores para cada caractere x que segue s . O tamanho máximo do contexto é uma constante K_{max} . Probabilidades de predição para cada contexto no modelo são calculadas a partir dos contadores de freqüência. Estes últimos são atualizados adaptativamente. O símbolo que ocorre atualmente é codificado com a distribuição de probabilidades predita. A probabilidade condicional de x no contexto s , $P(x|s)$ é então estimada por $C(x|s) / C(s)$, onde $C(x|s)$ é o número de vezes que x segue s e $C(s)$ é o número de vezes que s é encontrada (COUTINHO et al., 2005). A Figura 3.15 exemplifica a construção de um modelo PPM para a palavra “abracadabra”.

Ordem k = 2				Ordem k = 1				Ordem k = 0				Ordem k = -1			
Predição		c	p	Predição		c	p	Predição		c	p	Predição		c	p
ab	→ r	2	$\frac{2}{3}$	a	→ b	2	$\frac{2}{7}$	→	a	5	$\frac{5}{16}$	→	A	1	$\frac{1}{ A }$
	→ Esc	1	$\frac{1}{3}$		→ c	1	$\frac{1}{7}$		b	2	$\frac{2}{16}$				
ac	→ a	1	$\frac{1}{2}$		→ d	1	$\frac{1}{7}$		→ c	1	$\frac{1}{16}$				
	→ Esc	1	$\frac{1}{2}$	→ Esc	3	$\frac{3}{7}$	→ d		1	$\frac{1}{16}$					
ad	→ a	1	$\frac{1}{2}$	b	→ r	2	$\frac{2}{3}$		→ r	2	$\frac{2}{16}$				
	→ Esc	1	$\frac{1}{2}$		→ Esc	1	$\frac{1}{3}$		→ Esc	5	$\frac{5}{16}$				
br	→ a	2	$\frac{2}{3}$	c	→ a	1	$\frac{1}{2}$								
	→ Esc	1	$\frac{1}{3}$		→ Esc	1	$\frac{1}{2}$								
ca	→ d	1	$\frac{1}{2}$	d	→ a	1	$\frac{1}{2}$								
	→ Esc	1	$\frac{1}{2}$		→ Esc	1	$\frac{1}{2}$								
da	→ b	1	$\frac{1}{2}$	r	→ a	2	$\frac{1}{3}$								
	→ Esc	1	$\frac{1}{2}$		→ Esc	1	$\frac{1}{3}$								
ra	→ c	1	$\frac{1}{2}$												
	→ Esc	1	$\frac{1}{2}$												

Figura 3.15. Modelo PPM para a palavra “abracadabra”

As distribuições de probabilidades são então usadas por um codificador aritmético (MOFFAT, 1990) para gerar a sequência de bits. A atual implementação do PPM mantém contextos de todos os tamanhos inteiros abaixo do Kmax, e efetivamente combina as diferentes distribuições, usando um mecanismo de *escape* (representação de todos os caracteres ainda não lidos através de um único caractere genérico). O modelo com o valor mais alto de K é, por padrão, o primeiro usado para a codificação. Se um novo caractere é encontrado no contexto, significa que o mesmo não pode ser usado para a codificação do caractere, então um símbolo de *escape* é transmitido como sinal para a saída, situação denominada evento de escape. Sendo assim, o algoritmo continua sua busca no contexto para o próximo valor inferior de K. Este processo é repetido para tamanhos cada vez menores de K até que se encontre o símbolo em questão. Neste caso, o caractere é

codificado com a distribuição de probabilidades daquele modelo. Na prática, uma única estrutura de dados é usada para armazenar todas as probabilidades nos diferentes modelos de contexto. Para assegurar que o processo termina, o PPM assume um modelo abaixo do menor nível de K , normalmente (-1) , contendo todos os caracteres do alfabeto codificado (COUTINHO et al., 2005).

3.6.5. Codificação Aritmética

A codificação aritmética é originária do problema de se atribuir códigos inteiros para símbolos individuais, pela atribuição de um código para um arquivo de entrada. O método inicia com um certo intervalo, lê símbolo por símbolo do arquivo de entrada, e usa a probabilidade de cada símbolo para reduzir o intervalo. Especificar um intervalo requer mais bits, mas o número construído pelo algoritmo cresce continuamente. Para conquistar a compressão, o algoritmo é desenvolvido para que um símbolo de alta probabilidade reduza o intervalo para que o mesmo seja menor que um símbolo de baixa probabilidade, com resultado que símbolos de alta probabilidade contribuam com menos bits para a saída (SALOMON, 2004).

Um intervalo pode ser especificado por limites superiores e inferiores, ou por um limite e tamanho. O intervalo $[0.1, 0.512]$ pode ser especificado pelos números 0.1 e 0.412. O intervalo muito estreito $[0.12575, 0.1257586]$ é especificado pelos números 0.12575 e 0.0000086.

A saída do codificador aritmético é interpretada como o número na faixa $[0,1)$. O código 9746509 será interpretado como 0.9746509, onde nem o 0 antes do ponto, nem o ponto, são incluídos no arquivo de saída.

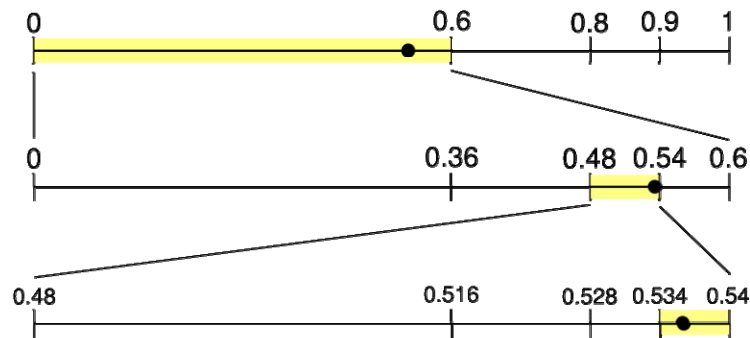


Figura 3.16. Exemplo de subdivisão de intervalos na Codificação Aritmética

Para cada símbolo processado, o intervalo corrente fica menor, mas serão necessários mais bits para expressá-lo. O ponto é que a saída final será um número simples formado pelo (0.) e a totalidade dos bits gerados, e não consistirá em códigos para símbolos individuais. O tamanho médio do código pode ser obtido dividindo o tamanho da saída (em bits) pelo tamanho da entrada (em símbolos). Há de se notar que as probabilidades usadas podem mudar a todo instante, já que as mesmas podem ser substituídas por um modelo probabilístico adaptativo.

O decodificador trabalha em um caminho oposto. O mesmo inicia inserindo os símbolos e suas faixas, reconstruindo a tabela de frequências e probabilidades, para então entrar com o resto do código. Exemplo: O primeiro dígito é 7, mas o decodificador imediatamente sabe que o código é um número na forma 0.7. Este número pertence a subfaixa $[0.5, 1)$ de S, mas o primeiro símbolo é S. O decodificador então elimina o efeito do símbolo S do código através da subtração do limite mais baixo 0.5 de S e dividindo pelo tamanho da subfaixa de S (0.5).

3.7. Trabalhos Relacionados

Pesquisas relacionadas à compressão de sinais eletrocardiográficos foram levantadas e analisadas, a fim de obter-se conhecimento sobre a real eficácia do referido

trabalho de pesquisa, através da comparação de abordagens, suposições, metodologia e resultados obtidos. Tal levantamento serve para expor o diferencial e a importância do trabalho em questão.

Francesco e Carlo (1988) recorreram ao uso da compressão com perdas para codificar sinais ECG. Foi utilizado o algoritmo intitulado *Sample-Skipping Method with Area Error Limitation*, técnica que memoriza apenas algumas amostras de sinal que permitem desenhar um polígono que não se desvie do traço original. O uso de tal técnica resultou em altas taxas de compressão, porém nunca foi possível reproduzir com exatidão o sinal original a partir de um sinal codificado, devido à eliminação de amostras no processo de codificação.

Ottley (2007), propôs o desenvolvimento de um sistema embarcado baseado em um algoritmo de codificação sem perdas capaz de alcançar taxas de compressão de pelo menos 2:1. Tal algoritmo é baseado na Codificação Preditiva Linear, em parceria com a Técnica de Particionamento de Sinal, para efetuar compressão em sinais ECG. A técnica divide os sinais em blocos seqüenciais, codificando-os independentemente. Após o particionamento, a Codificação Preditiva Linear aplica operações para remoção de correlação entre canais, bem como nos próprios canais. No final do processo de codificação é utilizado um modelo estatístico de codificação da entropia. *Buffers* são utilizados no armazenamento dos blocos de amostras já codificados. Tal trabalho de pesquisa obteve bons resultados, porém demonstrou-se menos robusto que o PPM, principalmente em sinais menos ruidosos ou normais, onde as taxas de compressão sofrem queda, já que o referido algoritmo não explora a natureza altamente periódica dos sinais ECG.

Souza (2008) propôs o desenvolvimento e implementação em FPGA de um sistema portátil para aquisição e compressão sem perdas de eletrocardiogramas, através da implementação em VHDL do PPM com um codificador aritmético, aplicando-se o código Gray. Em sua pesquisa, apesar de propor o mesmo algoritmo para compressão de sinais ECG utilizado neste trabalho, Souza implementou o compressor como um controlador, isto

é, um dispositivo de hardware que faz a interface entre o exterior e o seu funcionamento interno. A pesquisa descrita nesta dissertação adotou a implementação do algoritmo de compressão em software, como um módulo integrante da camada de software do sistema embarcado proposto, implementado em linguagem C. Devido às limitações impostas pela implementação em FPGA, a pesquisa de Souza atingiu a taxa média de compressão (TMC) igual a 2.48:1. Tal TMC é inferior ao conquistado neste trabalho e, para atingir a melhor compressão, utiliza valores de contexto maiores, resultando em tabelas de probabilidades maiores.

Boukaache (2006) recorreu ao algoritmo *Improved Embedded Zerotree Wavelet* (*Improved EZW*) para a compressão de sinais ECG. Trata-se de uma versão melhorada do algoritmo de compressão com perdas EZW, que transmite os coeficientes mais significantes antes de transmitir os coeficientes menos significantes. Isto é conquistado através de múltiplos passos sobre os coeficientes de onda, reduzindo o limiar por um fator de dois a cada passagem. Assim como o trabalho de Francesco e Carlo (1988), tal algoritmo elimina dados importantes de sinais ECG submetidos para o processo de codificação, ao passo que comprime sob altas taxas de compressão.

Bilgin, Marcellin e Altbach (2003) propuseram o desenvolvimento de *codecs* de hardware e software seguindo-se o padrão JPEG2000, normalmente utilizado na compressão de imagens. O uso de tais *codecs* pode eliminar a necessidade desenvolver-se hardware especializado. Tal padrão utiliza compressão com perdas e, assim como os trabalhos de Francesco e Carlo (1988), além de Boukaache (2006), alcança altas taxas de compressão através da eliminação de informação de sinais ECG para atingir compressão.

O software comercial WinRAR (2009), utilizado nos testes de compressão, adota um algoritmo de compressão baseado no LZW (WELCH, 1984) e no PPM, especificamente a implementação PPMd (SHKARIN, 2009). O LZW obtém um *string* de entrada e o processa utilizando um dicionário; Inicialmente, o dicionário contém todos os caracteres possíveis com suas codificações correspondentes; A entrada é lida e acumulada em uma cadeia de

caracteres S; Sempre que a sequência contida em S não estiver presente no dicionário, é emitido um código correspondente à versão anterior de S (sem o último caractere), sendo adicionado ao dicionário; S volta a ser inicializado com o último caractere lido (o último caractere) e o processo se repete até que não haja mais caracteres de entrada. O foco desta pesquisa foi implementar o PPM com um modelo binário (que reconhecesse o alfabeto binário, composto pelos dígitos 0 e 1), específico para a codificação de amostras binárias de sinais ECG, diferentemente do PPMd de Shkarin, que possui um modelo universal de compressão, sendo considerado um compressor de propósito geral (o mesmo é compatível para a compressão de imagens, áudio, vídeo e texto). Portanto, apesar de tal implementação gerar compressão com taxas equivalentes às obtidas por esta pesquisa (se aplicada na compressão de amostras binárias de ECG), o uso de um compressor universal foge do escopo deste trabalho.

O compressor ZIP nativo do Windows XP, também utilizado nos testes, utiliza o algoritmo de compressão de dados sem perdas Deflate (NETWORK WORKING GROUP, 1996). Tal algoritmo utiliza uma combinação entre o algoritmos LZ77 e a codificação de Huffman. O LZ77 (ZIV, 1977; LEMPEL, 1977) se baseia na utilização das partes que já foram lidas em um arquivo como um dicionário, substituindo as próximas ocorrências das mesmas seqüências de caracteres pela posição da sua última ocorrência. Para limitar o espaço de busca e de endereçamento necessário, as ocorrências anteriores são limitadas por uma “janela deslizante” (*sliding window*) que tem tamanho fixo e “desliza” sobre o arquivo, delimitando o início e o fim da área onde são buscadas as ocorrências anteriores. Já a codificação de Huffman é um algoritmo de codificação da entropia, que usa um método específico para selecionar a representação de cada símbolo, resultando em um código de prefixo que expressa os caracteres mais comuns utilizando *strings* de bits curtos que são usados para fontes de símbolos menos comuns (WINZIP, 2009). Tal algoritmo também implementa um compressor de propósito geral, gerando compressão com taxas próximas às

obtidas por este trabalho. Como esta pesquisa não objetivou a implementação de um compressor universal, o uso do Deflate também foge do escopo deste trabalho.

Os trabalhos descritos confirmam a importância do uso do PPM no sistema embarcado em questão. Diferentemente dos algoritmos utilizados por Francesco e Carlo (1988), Boukaache (2006) e Bilgin, Marcellin e Altbach (2003), que adotaram algoritmos de compressão com perdas em suas pesquisas, o PPM faz uma análise extensiva dos dados para garantir que nenhuma informação seja perdida ao final do processo de compressão / descompressão. Mantendo-se as características originais das amostras de sinais ECG, preservam-se informações que poderão ser de suma importância para a detecção de possíveis anomalias. Mesmo se comparado com outros algoritmos de compressão sem perdas, como o adotado por Ottley (2007) em suas pesquisas, o PPM apresenta um melhor desempenho, atingindo taxas de compressão mais expressivas.

4. Arquitetura do Sistema Embarcado

O projeto para o desenvolvimento do sistema embarcado para monitor holter envolve a execução de trabalhos em hardware e software. Em hardware, são definidos os módulos utilizados na prototipagem. Em software, são implementados os módulos para simulação e compressão de sinais ECG.

Os tópicos a seguir demonstram em detalhes a arquitetura do referido monitor, com destaque para a descrição no nível de software, além das especificações da camada de hardware.

4.1. Módulos da Arquitetura

Os módulos da arquitetura do sistema embarcado proposto são apresentados de forma simplificada na Figura 4.1. A arquitetura contém um módulo de simulação, que gera sinais ECG a partir das séries de Fourier, além de um módulo de compressão que possui uma implementação do modelo PPM binário mais um codificador aritmético, que utiliza a codificação Gray e decomposição em plano de bits.

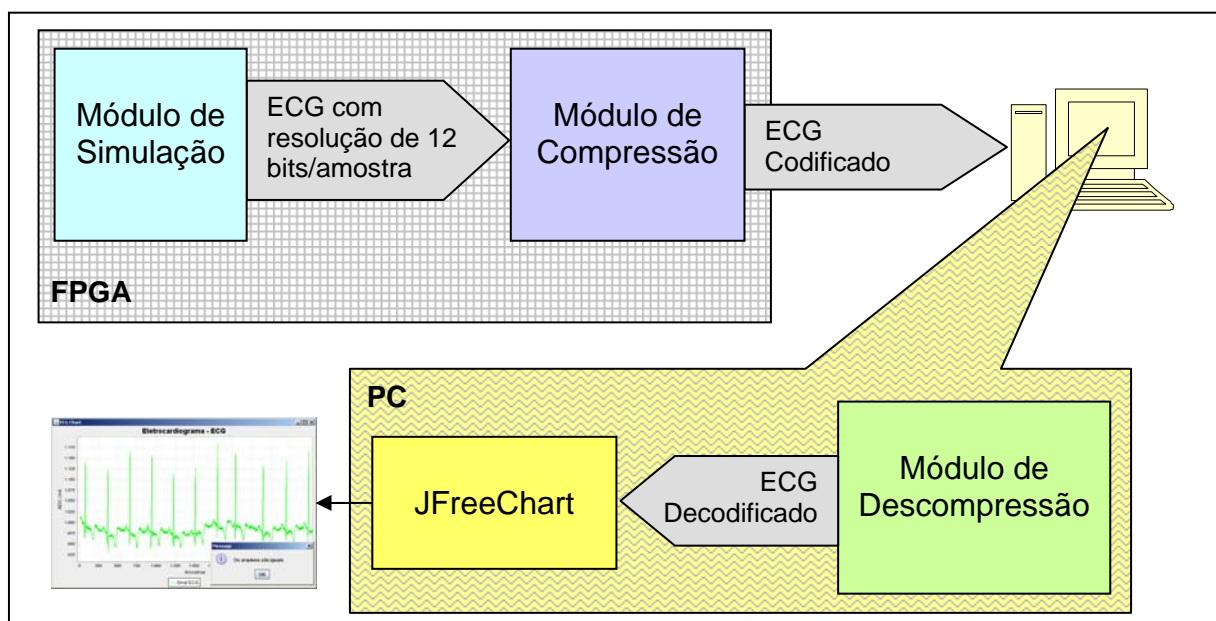


Figura 4.1. Arquitetura do sistema embarcado

4.2. Descrição no Nível de Sistema

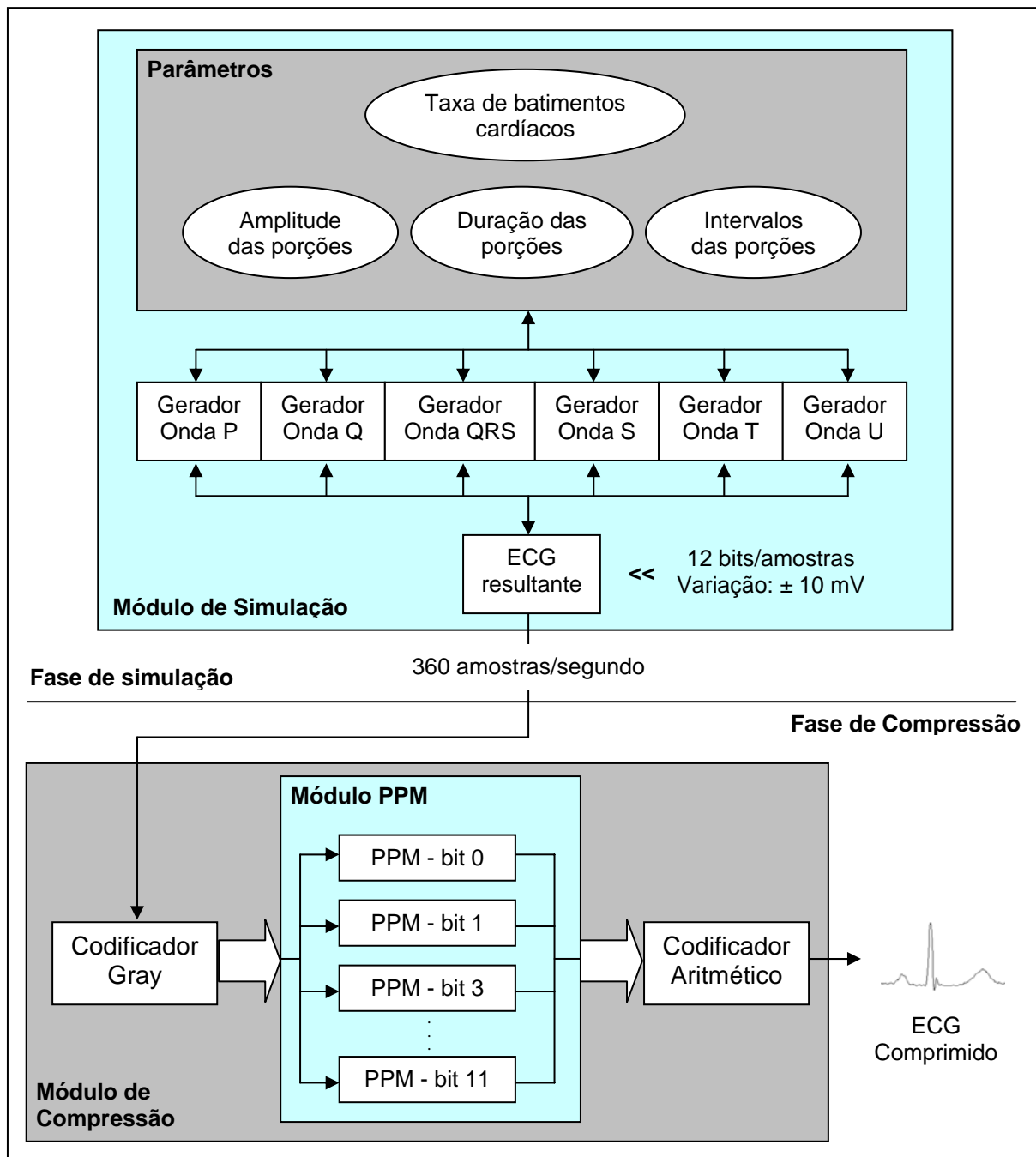


Figura 4.2. Detalhamento do sistema embarcado

O sistema embarcado é dividido em dois módulos: módulo de simulação e módulo de compressão de sinais ECG. A Figura 4.2 esquematiza o funcionamento dos módulos no nível de sistema, e os tópicos posteriores descrevem cada módulo em detalhes.

No módulo de simulação, amostras binárias de sinais ECG com resolução de 12 bits (representando amostras entre ± 10 mV) são geradas a partir da implementação das Séries de Fourier. Parâmetros utilizados pela simulação como batimentos cardíacos, além de amplitude, duração e intervalos das ondas QRS, P, Q, S, T e U podem ser definidos pelo usuário, ou tais parâmetros podem ser definidos pelo próprio sistema, com o uso de parâmetros padrão.

O módulo de compressão é responsável pela compressão de sinais ECG gerados na fase de simulação de sinais. Tal módulo de compressão é composto por três sub-módulos: o Codificador Gray, o módulo PPM e o Codificador Aritmético. A Figura 4.2 demonstra a organização do referido módulo.

Para a viabilização da execução dos módulos de simulação e compressão em paralelo, foi utilizada a biblioteca MicroC/OS-II, integrante do Nios II IDE. O MicroC/OS-II é um kernel de sistema operacional multitarefa de baixo custo baseado em tempo-real desenvolvido para microprocessadores, escrito em linguagem de programação C. O mesmo é destinado principalmente para sistemas embarcados, e é livre para uso não-comercial ou educacional. Tal biblioteca define vários procedimentos para criação, execução, controle, gerenciamento e encerramento de *threads* a serem usados em projetos desenvolvidos em linguagem C/C++ no Nios II IDE. Graças à mesma, é possível a implementação dos módulos de simulação e compressão como *threads* componentes do sistema embarcado em questão.

4.2.1. Módulo Simulador de Sinais ECG

A implementação do referido simulador foi baseada no algoritmo desenvolvido em MATLAB Script por Karthik (2006). A partir da definição de parâmetros de sinal, como taxa de batimentos cardíacos, além de amplitude, duração e intervalos das porções P, Q, QRS, S, T e U, o algoritmo calcula separadamente tais porções para um sinal ECG típico. As referidas porções são ilustradas na Figura 3.4. O cálculo de cada porção é baseado nas

Séries de Fourier (1826), descritas na Tabela 4.1. Cada característica significativa de um sinal ECG é gerada a partir da soma de cada uma destas porções de onda. O esquema de funcionamento de tal módulo é apresentado na Figura 4.2. Os algoritmos utilizados nos cálculos das porções de sinais ECG são apresentados na Figura 4.3.

```

procedure p_t_u(amp, dur, hbr, int)
{
    x = 0.01:0.01:600;

    x = x - int;

    li = 30/hbr;

    b = (2*li)/dur;

    u1 = 1/l;

    n = <TOTAL_NUMBER_OF_SAMPLES>;

    for i = 1 to n
        harm =
            (((sin((PI/(2*b))*(b-(2*i))))/(b-
                (2*i)))+(sin((PI/(2*b))*(b+(2*i))))/(b+(2*i)))*(2/PI))*cos((i*PI*x)/l);

        u2 = u2 + harm;
    end

    wave_1 = u1 + u2;

    final_wave = wave_1 * amplitude;
}

procedure qrs_q_s(amp, dur, hbr)
{
    x = 0.01:0.01:600;

    li = 30/hbr;

    b = (2*li)/dur;

    wave_1 = (amp/(2*b))*(2-b);

    n = <TOTAL_NUMBER_OF_SAMPLES>;

    for i = 1 to n
        harm = (((2*b*amp)/((i*i)*(PI*PI)))*(1
            cos((i*PI)/b))*cos((i*PI*x)/l);

        wave_2 = wave_2 + harm;
    end

    final_wave = wave_1 + wave_2;
}

```

Figura 4.3. Procedimentos para os cálculos das porções P, T e U, além das porções QRS, Q e S, respectivamente

De acordo com Dirichlet (1829), qualquer função periódica que satisfaça condições limite em que uma variável de estado permanece constante ao longo do tempo, pode ser representada por uma série de grandezas escalares em termos de senos e co-senos para frequências que ocorram como um múltiplo da frequência fundamental. Karthik (2006) também afirma que sinais ECG são periódicos, com frequência fundamental determinada pelos batimentos cardíacos, e a função que as geram, dado um intervalo de amostras, satisfaz as seguintes condições de Dirichlet:

- A função toma valores únicos e finitos de amostras, em um intervalo dado;
- A função é absolutamente integrável;
- A função possui número finito de máximos e mínimos entre intervalos finitos;
- A função tem um número finito de descontinuidades;

Portanto, Séries de Fourier podem representar sinais ECG. As Séries de Fourier são descritas na Tabela 4.1.

$f(x) =$	$(a_0/2) + \sum_{n=1}^{\infty} a_n \cos (n\pi x / l) + \sum_{n=1}^{\infty} b_n \sin (n\pi x / l)$	(1a)
$a_0 =$	$(1/l) \int_T f(x) dx, T = 2l$	(1b)
$a_n =$	$(1/l) \int_T f(x) \cos (n\pi x / l) dx, n = 1, 2, 3 \dots$	(1c)
$b_n =$	$(1/l) \int_T f(x) \sin (n\pi x / l) dx, n = 1, 2, 3 \dots$	(1d)

Tabela 4.1. As Séries de Fourier

Um único período de um sinal ECG é uma mistura de formas de onda triangular e senoidal. Cada característica do sinal pode ser representada por versões deslocadas e escalares de uma das seguintes formas de onda:

- As porções QRS, Q e S de um sinal ECG podem ser representadas por formas de onda triangulares;
- As porções P, T e U podem ser representadas por formas de onda senoidais;

Uma vez que é gerada cada uma destas porções, pode-se juntá-las em um único sinal, obtendo-se o sinal ECG resultante.

Desenvolvendo-se um módulo de software baseado nas Séries de Fourier para o sistema embarcado proposto, permite-se a prototipagem em hardware de um dispositivo simulador que ajudará pesquisadores na análise de sinais eletrocardiográficos, eliminando-se a dependência de métodos invasivos e não-invasivos de captação de sinais ECG. Sinais ECG normais e anormais, além de vários tipos de arritmias, poderão ser gerados a partir do referido protótipo.

4.2.2. Módulo de Compressão

O módulo de compressão recebe amostras de sinais ECG com resolução de 12 bits/amostras. Tais amostras são codificadas com código Gray, e a codificação gerada é decomposta em um plano de bits. Cada plano de bits é então codificado separadamente através do módulo PPM, adaptado para o alfabeto binário.

O codificador Gray, primeiro sub-módulo a ser executado no processo de compressão, gera um sistema de numeração binário onde dois valores sucessivos diferem em apenas um bit, através de reflexão de código. O fato de um só bit ser alterado por vez faz com que os módulos de entrada de circuitos eletrônicos com controladores programáveis leiam mais facilmente dados de entrada, além de identificarem com maior exatidão posições inválidas e/ou ambíguas, minimizando o efeito de erro na conversão de sinais analógicos para digitais.

Em sinais digitais extraídos de sistemas físicos como eletrocardiogramas, os valores de duas amostras adjacentes não variam muito devido à natureza contínua dos mesmos (SOUZA, 2008). O referido módulo aumenta a redundância das amostras, graças à redução acentuada na variação da representação binária entre amostras sucessivas, diminuindo a entropia do sinal e ampliando a capacidade de compressão do sistema como um todo. A Figura 4.4 exemplifica a construção do código Gray.

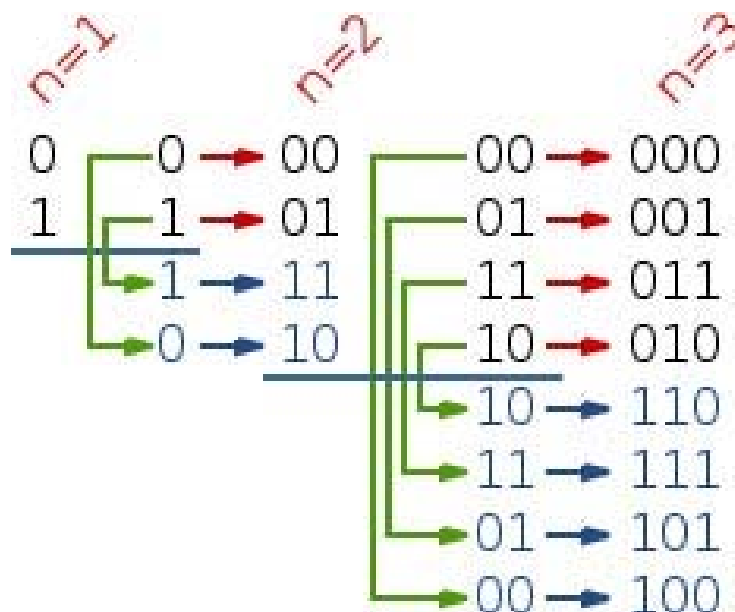


Figura 4.4. Construção do Código Gray

A codificação Gray de cada amostra é então decompostas em um plano de bits. Tal decomposição permite que o sinal de 12 bits/amostra seja dividido em várias seqüências binárias, onde cada seqüência gera um sinal à parte. Como a referida implementação trabalha com amostras de 12 bits, 12 sinais são gerados. O primeiro bit do sinal 0, por exemplo, é formado pelo último bit da primeira amostra, o segundo bit do mesmo sinal é formado pelo segundo bit da segunda amostra, e assim sucessivamente para cada sinal gerado a partir das seqüências binárias obtidas. A decomposição em plano de bits é exemplificada na Figura 4.5.

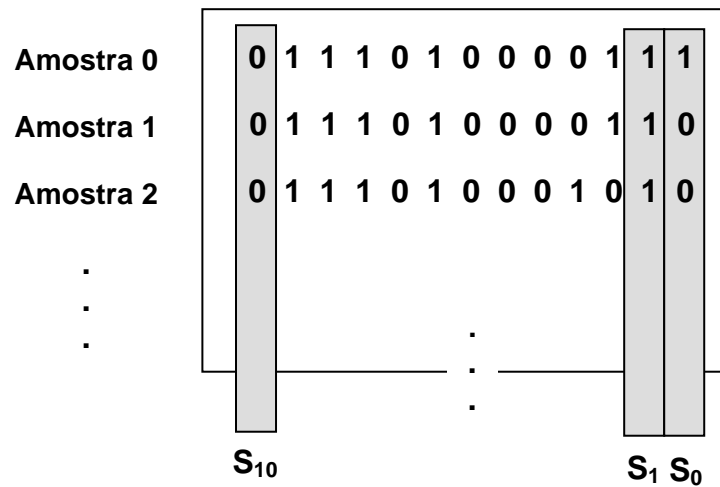


Figura 4.5. Exemplo de uma decomposição em plano de bits

A eficácia do uso da codificação Gray e da decomposição em plano de bits, em parceria com o PPM no aumento das taxas de compressão, foi comprovada por Primo e Cavalcanti (2009) através de testes de compressão em amostras digitalizadas do MIT-BIH Database. Nos testes, foram executadas codificações através das seguintes combinações: PPM com plano de bits e com codificação Gray, PPM com plano de bits e sem codificação Gray, PPM sem plano de bits e com codificação Gray e PPM sem plano de bits e sem codificação Gray. Tais testes comprovaram que a melhor combinação obtida foi a codificação Gray em conjunto com a divisão das amostras em plano de bits. A Figura 4.6 exibe algumas taxas de compressão obtidas no referido trabalho para cada combinação testada.

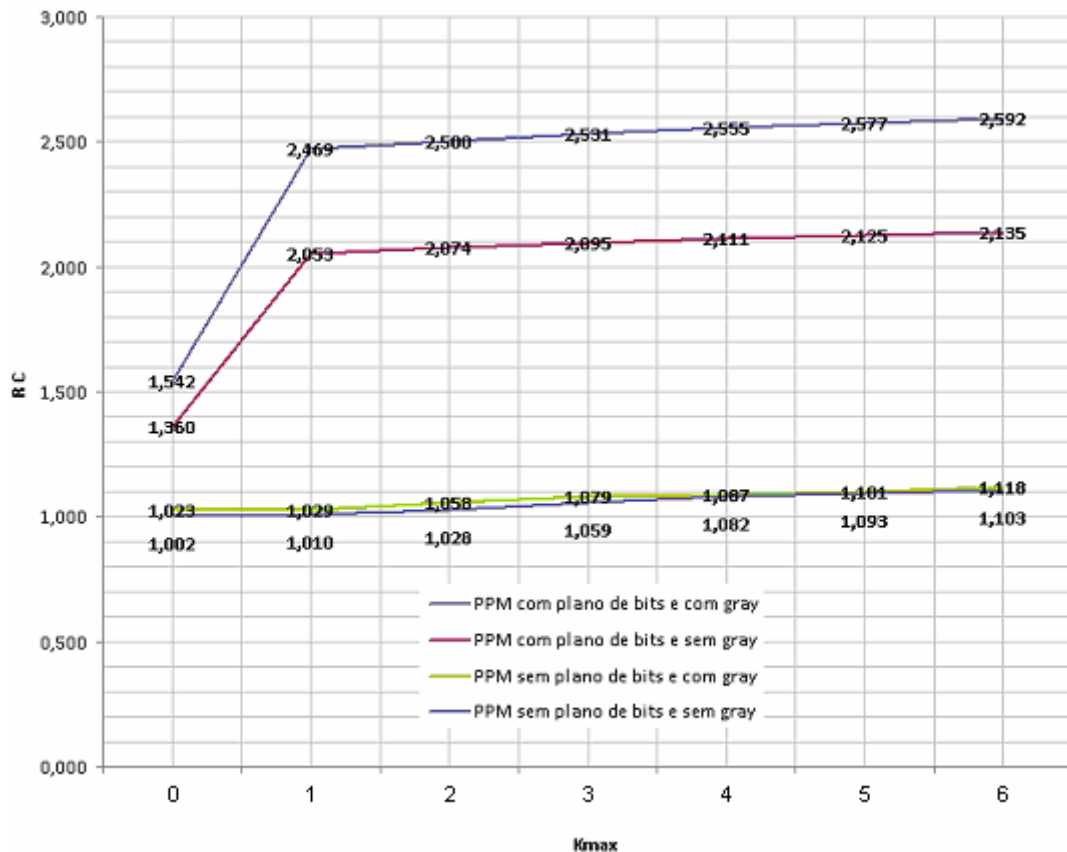


Figura 4.6. Razão de compressão em função de Kmax para cada combinação Gray / nº de planos de bit (Primo, 2009; Cavalcanti, 2009)

Cada plano de bits decomposto alimenta um modelo PPM binário, que constrói um modelo estatístico a partir dos mesmos e os envia para o codificador aritmético. O codificador aritmético envia um *stream* de saída que compõe o sinal comprimido.

O codificador PPM muda para um contexto curto quando outro contexto já tenha resultado em probabilidade 0. O PPM, portanto, inicia com um contexto de ordem N ($K_{max} = N$). Tal codificador busca estas estruturas de dados para uma ocorrência prévia do contexto corrente C seguido pelo próximo símbolo S. Se ele não encontra ocorrências (isto é, se a probabilidade deste C particular para este S é 0), o mesmo muda para uma ordem N-1 e tenta a mesma coisa. C' será um string consistindo nos N-1 símbolos mais a direita de C. O codificador PPM busca esta estrutura de dados para uma ocorrência prévia do contexto

corrente C' seguido do símbolo S . O PPM, portanto, tenta usar o menor e as menores partes do contexto C , o qual é a razão para seu nome.

O codificador lê o próximo símbolo S a partir de um stream de entrada, olha para contexto corrente C de ordem N (os últimos N símbolos lidos), e se baseia nos dados de entrada que foram vistos no passado, determina a probabilidade P que S irá aparecer seguindo um contexto particular C . O codificador então invoca um algoritmo de codificação aritmética adaptativa para codificar símbolos S com probabilidade P .

O problema da probabilidade zero é resolvido no PPM através da troca para um contexto menor. O codificador PPM chama a si mesmo. Se o próximo símbolo S nunca for visto antes seguindo o contexto de ordem 2, o PPM muda para o contexto de ordem 1, e assim sucessivamente. Se o símbolo S nunca foi visto antes (uma situação comum no início de qualquer processo de compressão), o codificador PPM muda para um modo chamado contexto de ordem -1, onde S receberá a probabilidade fixa $1/(\text{tamanho do alfabeto})$.

O Codificador Aritmético elimina a associação entre símbolos individuais e palavras-código de comprimento inteiro e, com isto, é capaz de praticamente igualar a entropia da fonte em todos os casos. Desta forma, símbolos de alta probabilidade irão contribuir com menos bits para a saída, aumentando a compressão. O primeiro passo seguido pelo codificador aritmético é calcular, ou pelo menos estimar, as frequências de ocorrência de cada símbolo. Para melhores resultados, as frequências exatas são calculadas na leitura de um *stream* de entrada no primeiro ou segundo passo de compressão.

Seguem abaixo os principais passos da codificação aritmética:

- 1- Inicia-se definindo um “intervalo corrente” como $[0,1)$.
- 2- Repetem-se os dois passos seguintes para cada símbolo s no *stream* de entrada:
 - 2.1- Divide-se o intervalo corrente em subintervalos, onde seus tamanhos são proporcionais às probabilidades dos símbolos.

2.2- Seleciona-se o subintervalo para s , sendo definido como o novo intervalo corrente.

3- Quando o *stream* de entrada puder ser processado, a saída poderá ser qualquer número que identifique unicamente o intervalo corrente (isto é, qualquer número que pertença ao intervalo).

Para cada símbolo processado, o intervalo corrente fica menor, mas é codificado com mais bits. Porém, a saída final é um único número formado por todos os bits que foram acrescentados à medida que se criavam intervalos menores, ao invés de códigos para símbolos individuais.

4.3. Camada de Hardware

O kit de desenvolvimento utilizado na representação da camada de hardware e na prototipação da arquitetura do sistema embarcado foi o *Nios Development Board - Stratix Edition*. O mesmo fornece uma plataforma de hardware para o desenvolvimento de sistemas embarcados baseados em dispositivos Altera[®]. O kit possui o dispositivo Stratix EP1S10F780C6, com 10.570 elementos lógicos e 920.448 bits de memória on-chip. A Figura 4.7 apresenta o referido kit de desenvolvimento.

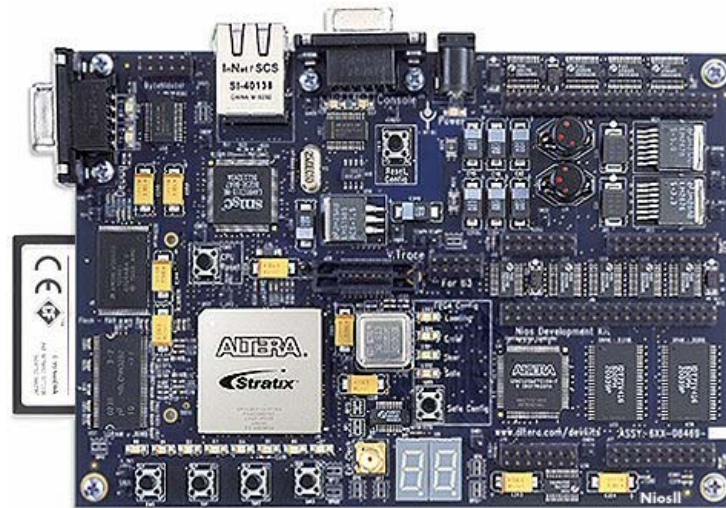


Figura 4.7. A placa de desenvolvimento utilizada

Outros recursos fornecidos pelo referido kit são:

- Memória flash de 8 Mbytes;
- 1 Mbytes para memória RAM estática;
- 16 Mbytes para memória SDRAM;
- Lógica *On-Board* para a configuração do dispositivo Stratix a partir da memória flash;
- Dispositivo *On-Board* Ethernet MAC/PHY;
- Conector para cartões de memória tipo I;
- Conector Mictor para *debug* em hardware e software;
- Duas portas seriais RS-232 DB9;
- Quatro botões alternadores;
- Oito LED's;
- Dois displays de 7 segmentos;

- Conectores JTAG para dispositivos Altera®;
- Oscilador de 50 MHz;

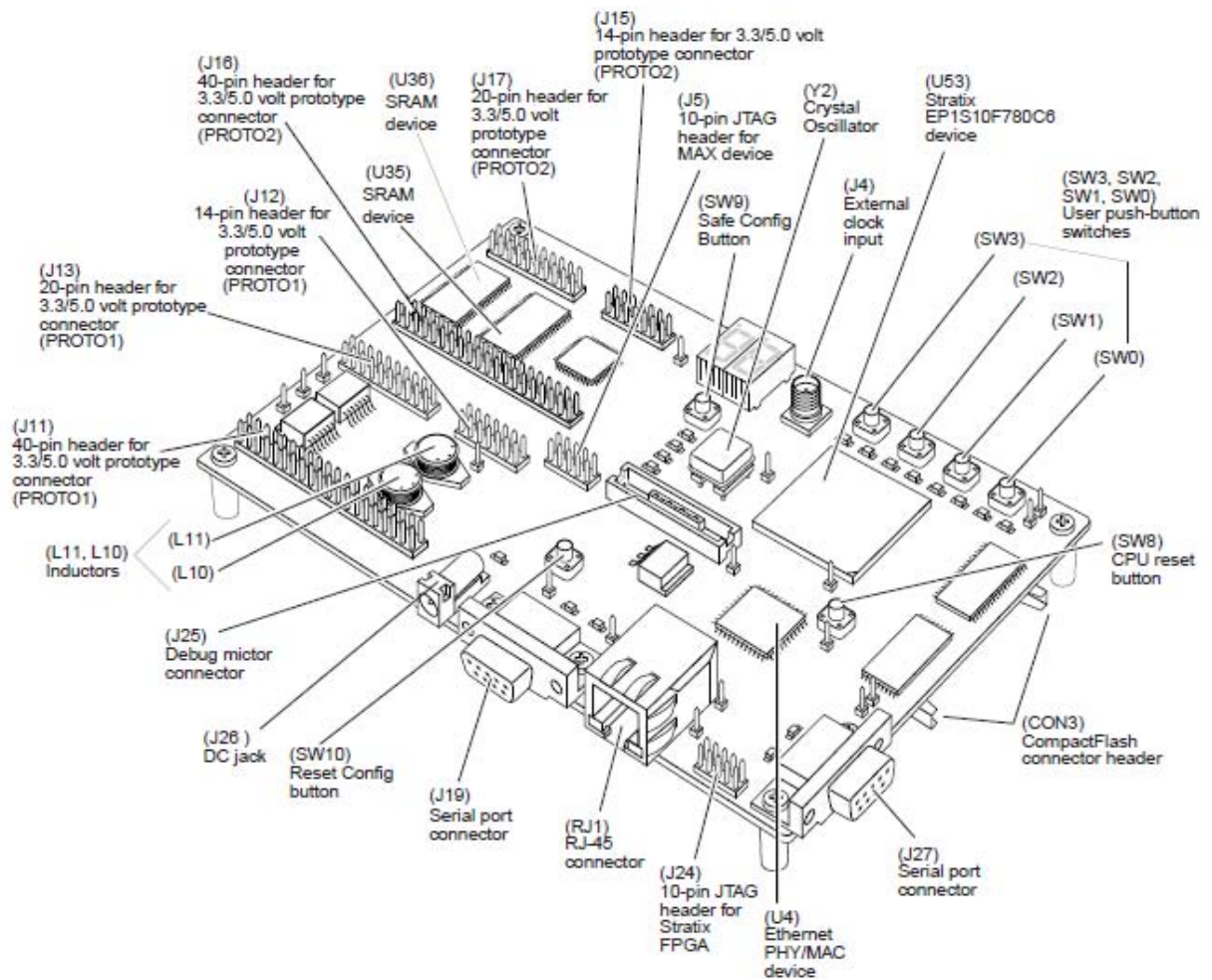


Figura 4.8. Recursos fornecidos pelo kit de desenvolvimento

A camada de hardware foi configurada com os seguintes componentes:

- Núcleo Nios II;
- Memória e Controlador SDRAM;
- Memória Flash;
- Interface UART;
- Núcleo JTAG UART;

- Barramento Avalon;
- Display LCD;

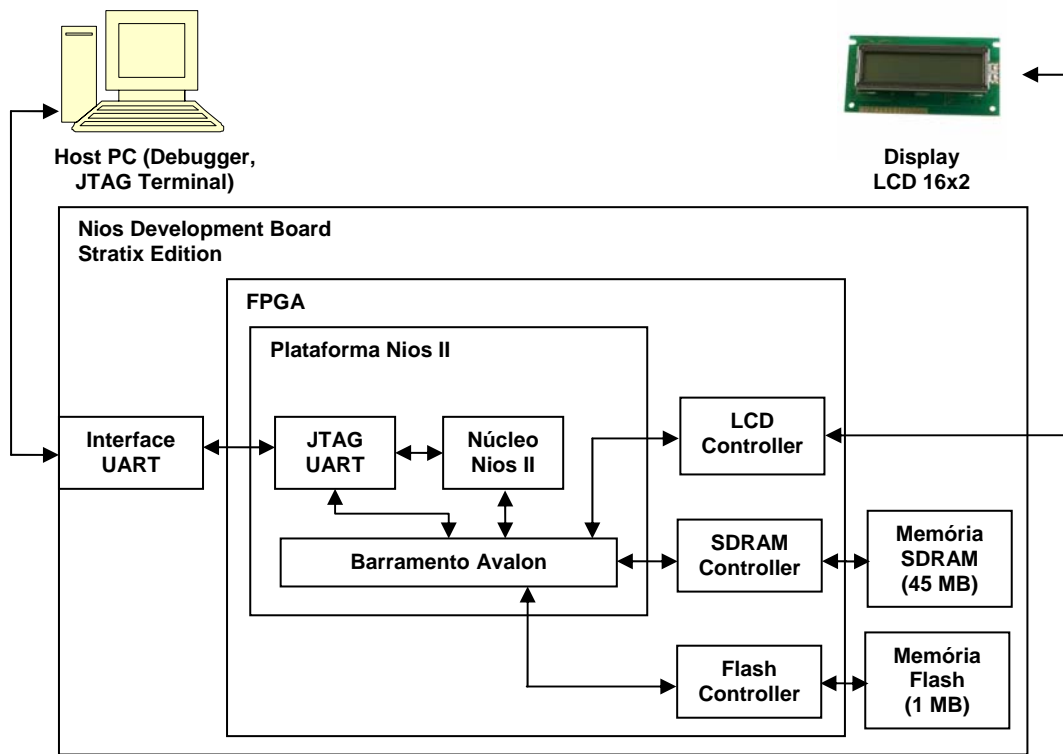


Figura 4.9. Camada de hardware do sistema embarcado

O uso de tais componentes é definido através do software SOPC Builder, que automatiza a integração de componentes de hardware. Utilizando métodos tradicionais de projeto, o programador escreveria manualmente módulos HDL, para então os unir para um sistema proposto. Utilizando o *SOPC Builder*, o programador pode, então, especificar os componentes de sistema através de uma interface gráfica, gerando automaticamente interconexões lógicas. O *SOPC Builder* gera arquivos HDL que definem todos os componentes de um sistema, além de suas interconexões. O referido software gera arquivos na extensão VHDL ou Verilog HDL.

A Figura 4.9 demonstra os componentes utilizados para a arquitetura do sistema embarcado e suas interligações. Em seguida, são fornecidas informações sobre cada componente integrante da camada de hardware:

Núcleo Nios II - O núcleo do processador Nios II executará os módulos pertencentes à camada de software, previamente armazenados na memória SDRAM ou na memória Flash, integrantes da arquitetura do dispositivo. Maiores informações sobre o processador Nios II são apresentadas no capítulo 3, tópico 3.4.2.

Barramento Avalon - O Avalon é um barramento especial que prioriza a velocidade de comunicação de dados, permitindo conexões em paralelo, sendo responsável pela integração entre o núcleo de processamento e os outros dispositivos.

Memória Flash - A Memória Flash é um dispositivo de 8 MB AMD AM29LV065D, usado como memória de propósito-geral para leitura e armazenamento não-volátil. O referido dispositivo compartilha endereços e dados de conexão com chips SRAM. Na camada de hardware descrita, a referida memória é utilizada no armazenamento de projetos de software. Desta forma, softwares desenvolvidos para a placa de desenvolvimento em questão podem ser carregados e executados imediatamente após o ligamento da placa. O desligamento da placa não excluirá a codificação armazenada, viabilizando um sistema dedicado.

Memória SDRAM - O dispositivo SDRAM é um Micron MT48LC4M32B2 de 128 MB, com funcionalidade PC100 e modo de auto-atualização. A SDRAM é totalmente sincronizada com os sinais positivos registrados na borda do relógio do sistema. Os pinos do dispositivo SDRAM são conectados ao dispositivo Stratix. Um controlador SDRAM é incluído no kit de desenvolvimento Nios II, permitindo ao processador Nios II enxergar o dispositivo SDRAM como uma memória larga e linearmente endereçável.

Display LCD - O LCD, com um display de 2 linhas X 16 caracteres, exibe informações de status úteis, mensagens de progresso e de alerta. Tempo decorrido para a aquisição de sinais ECG e frequência cardíaca em bpm são exemplos de saída que podem ser exibidos pelo LCD.

Núcleo JTAG-UART - O núcleo JTAG-UART utiliza o JTAG para FPGA's Altera[®], e provê acesso a sistemas de processamento hospedeiro via pinos JTAG no FPGA. O sistema de processamento hospedeiro pode se conectar ao FPGA através de qualquer cabo de download JTAG da Altera[®], como o cabo USB-Blaster. Suporte de software para o núcleo JTAG-UART é provido pela Altera[®]. Para o processador Nios II, drivers de dispositivos são providos em uma biblioteca de sistema HAL, permitindo que software acesse o núcleo utilizando rotinas da biblioteca stdio.h da linguagem C ANSI padrão.

O sistema definido e gerado no SOPC Builder é integrado em um projeto editado no software Quartus II, que atribui localização de pinos, requisitos de tempo e outros limites de projeto. O Quartus II gera um arquivo SRAM *Object* após a compilação do projeto, que será utilizado na configuração do FPGA. Tal compilação também fornece um relatório com dados sobre a síntese do projeto de hardware em questão. O relatório gerado a partir da compilação do projeto para o sistema embarcado é apresentado na Tabela 4.2.

Dispositivo - Stratix EP1S10F780C6	
Total – Elementos Lógicos	9,409 / 10,570 (89%)
Total – Pinos	228 / 427 (53%)
Total – Pinos Virtuais	0
Total – Bits de Memória	692,480 / 920,448 (75%)
Blocos DSP	8 / 48 (17%)
Total – PLLs (Phase Locked Loop)	2 / 6 (33 %)
Total – DLLs	0 / 2 (0 %)

Tabela 4.2. Síntese do projeto de hardware

4.4. Sistema para Descompressão / Visualização de Sinais

A descompressão dos sinais ECG captados pelo sistema embarcado proposto é efetuada através de software desenvolvido em linguagem de alto nível. O mesmo estará presente em um sistema de processamento hospedeiro que receberá os sinais armazenados e previamente comprimidos pelo sistema embarcado. Além de decodificar os sinais, graças ao módulo de descompressão de sinais implementado em Java, o software exibe em forma de gráficos os sinais resultantes, graças a utilização da biblioteca JFreeChart.

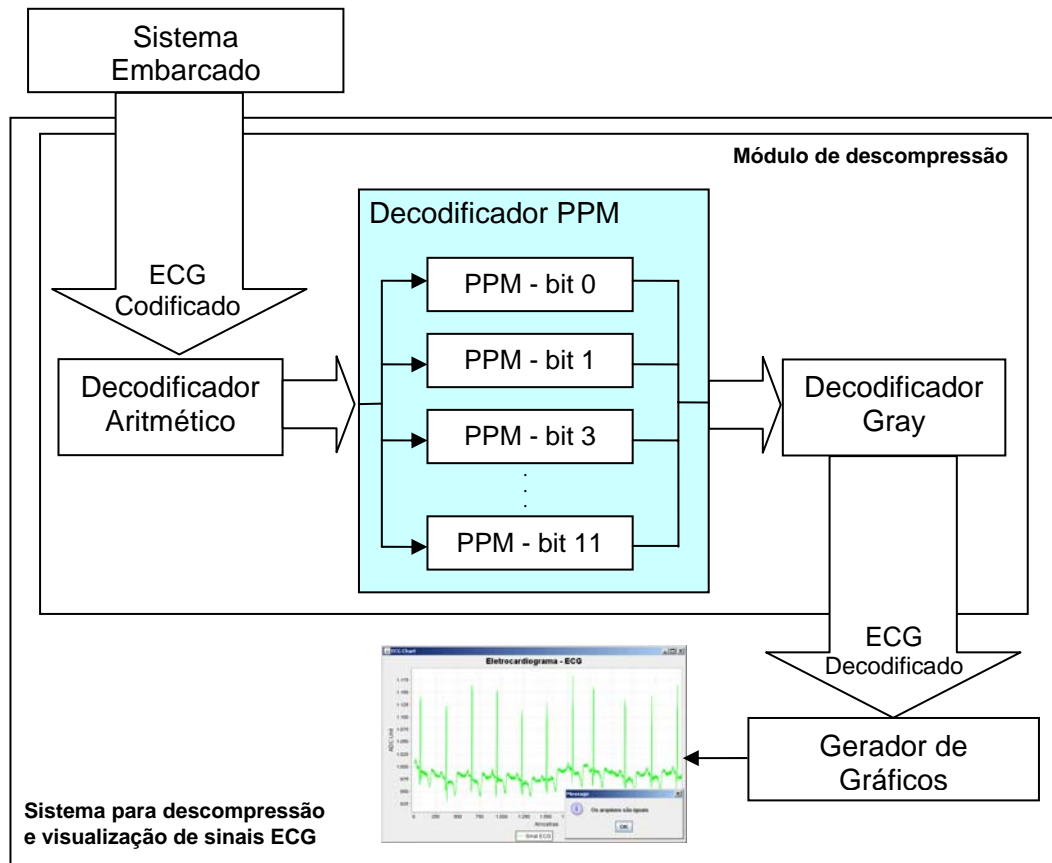


Figura 4.10. Detalhamento do sistema de descompressão / visualização de sinais ECG

O módulo PPM, integrado a camada de software do sistema embarcado, sempre visualiza o próximo símbolo e baseia sua próxima etapa para descobrir o que este símbolo é. O mesmo muda para um contexto menor baseado no próximo símbolo. O decodificador PPM não pode imitar isso, já que não conhece qual o próximo símbolo.

O recurso usado pelo decodificador PPM é reservar um símbolo do alfabeto como um símbolo de *escape*. Quando o módulo PPM decide mudar para um contexto menor, o decodificador PPM escreve o símbolo de *escape* (codificado aritmeticamente) no *stream* de saída. Tal módulo pode decodificar o símbolo de *escape*, já que o mesmo é codificado no presente contexto. Depois de decodificar um *escape*, o módulo de descompressão também muda para um contexto menor.

Tanto o decodificador aritmético quanto o decodificador Gray trabalham em caminhos opostos aos seus respectivos codificadores.

5. Trabalhos Efetuados e Resultados Obtidos

Inicialmente visou-se o desenvolvimento do módulo de compressão de sinais ECG, componente da camada de software para a arquitetura do sistema embarcado, utilizado na compressão dos sinais ECG através da implementação do codificador Gray, PPM e do codificador aritmético. Também foi objetivado nesta fase o desenvolvimento de software que teria como função decodificar os sinais ECG armazenados no sistema embarcado, exibir em forma de gráficos os sinais utilizados para os testes de compressão e os sinais decodificados, além de efetuar comparações entre os sinais originais e os sinais decodificados, a fim de validar o módulo de compressão.

Após a validação e conclusão do módulo descrito anteriormente, foi desenvolvido o módulo de simulação de sinais ECG, que gera sinais eletrocardiográficos a partir da implementação das Séries de Fourier.

Finalmente, tendo os módulos de compressão e simulação de sinais em mãos, foi desenvolvido o sistema que integra tais módulos. Em tempo real, o mesmo gera amostras de sinais ECG, simula a captação de amostras de acordo com uma frequência de amostragem pré-definida e as comprime em paralelo.

5.1. Testes e Validação do Módulo Compressor

Os sinais ECG utilizados para validação e testes do módulo de compressão do sistema embarcado foram oriundos do banco de sinais ECG da MIT-BIH (*Massachusetts Institute of Technology* em parceria com a *Beth Israel Deaconess Medical Center*) disponibilizados pela Internet através do PhysioNet.org (2009) (serviço público para a pesquisa de recursos para sinais fisiológicos complexos). Este banco de dados é o principal conjunto de materiais de testes padrão para avaliação de detectores de arritmia.

O referido banco de dados contém 48 gravações ambulatoriais de ECG de dois canais, obtidos a partir de 47 indivíduos estudados pelo *BIH Arrhythmia Laboratory* entre 1975 e 1979. Vinte e três gravações foram escolhidas aleatoriamente em um conjunto de 4000 gravações ambulatoriais de ECG, captadas no período de 24 horas de duração cada, coletadas de uma população mista de pacientes internados (cerca de 60%) e ambulatoriais (cerca de 40%) do *Boston's Beth Israel Hospital*. As 25 gravações restantes foram selecionadas a partir do mesmo conjunto para incluir arritmias não tão comuns, mas clinicamente significantes, que não seriam bem representados em uma pequena amostra aleatória (PHYSIONET, 2009).

As gravações foram digitalizadas a 360 amostras por segundo e por canal, com resolução de 11 bits e faixa de 10 mV. A Figura 5.1 apresenta o programa desenvolvido para descompressão e/ou exibição em execução, exibindo o sinal 100, utilizado nos testes.

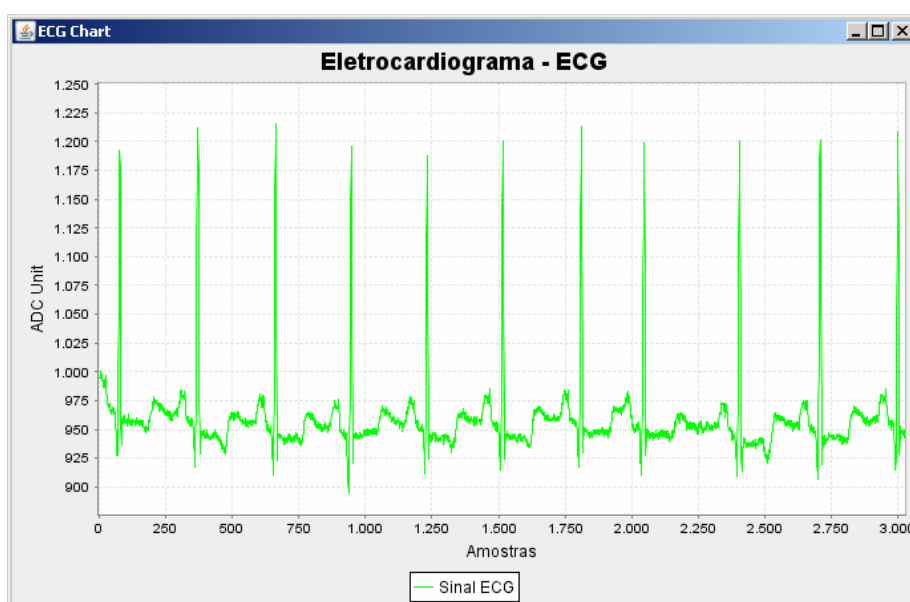


Figura 5.1. Três mil amostras do sinal ECG 100.dat, oriundo do MIT-BIH Arrhythmia Database

Para o desenvolvimento de testes do módulo em questão, foi utilizado o componente *Altera® Host Based File System*, presente no Nios II IDE. O referido componente permite

que programas executados em uma placa de desenvolvimento alvo leiam e escrevam arquivos armazenados em um PC (sistema de processamento hospedeiro). Graças a este recurso, podemos simular a captação de um sinal ECG por parte do sistema embarcado em questão, armazenando os sinais a serem comprimidos pelo dispositivo em um sistema de processamento hospedeiro que esteja conectado à placa de desenvolvimento utilizada (através do cabo de download USB Blaster). A placa obtém os arquivos com as amostras de ECG a partir do sistema de processamento hospedeiro, comprime-os (escrevendo um novo arquivo com as amostras comprimidas) e os armazena no sistema hospedeiro.

Uma das unidades adotadas pelo MIT-BIH, utilizadas nas amostras contidas nos arquivos de sinal ECG pertencentes ao referido banco de dados, é o *ADC Unit* (acrônimo para *Analog to Digital Converter Unit*), que consiste em valores inteiros sem sinal de 11 bits. Este trabalho adota por padrão amostras binárias de 12 bits, onde cada amostra é separada por caractere vazio. Devido à diferença de padrões de arquivo utilizados, houve a necessidade de desenvolver um aplicativo que formatasse os arquivos obtidos do banco de dados MIT-BIH para a formatação de arquivo utilizado pelo sistema embarcado em questão. Para tal, foi utilizada a linguagem orientada a objetos Java. A IDE utilizada no desenvolvimento de tal aplicativo foi o Netbeans IDE. Tal aplicativo simplesmente aumenta a resolução das amostras de 11 para 12 bits, adicionando 0 ao bit mais significativo, além de um caractere vazio, que representa a delimitação entre amostras.

De posse dos arquivos comprimidos obtidos a partir dos sinais ECG originais em questão, foi possível aferir a eficiência do módulo compressor desenvolvido. Para tal, foi adotado o cálculo da taxa de compressão para cada sinal, obtido através das seguintes fórmulas: $TC = 1 - (\text{Tamanho do arquivo comprimido} / \text{Tamanho do arquivo original})$ e $TC = (\text{Tamanho do arquivo original} / \text{Tamanho do arquivo comprimido}) : 1$. Adotou-se os contextos Kmax iguais a 1, 2, 3, 4 e 5 nos testes de compressão. Valores para Kmax menores ou iguais a 5 fazem com que o módulo compressor codifique com boas taxas, sem exceder a capacidade de memória do kit de desenvolvimento na preparação das tabelas de

probabilidade. Contextos maiores que 5 geram taxas de compressão muito próximas, resultando em uso excessivo de memória sem aumento significativo na compressão. Tal fato é comprovado nos resultados da pesquisa de Primo e Cavalcanti (2009), demonstrados em forma de gráfico na Figura 4.6. As taxas de compressão obtidas para os contextos Kmax iguais a 1, 2, 3, 4 e 5 são demonstradas nas Tabelas 5.1, 5.2, 5.3, 5.4 e 5.5 respectivamente.

ECG (MIT-BIH)	Tamanho original (I)	Tamanho comprimido (II)	$TC = 1 - (II/I)$ (K = 1)	$TC = (I/II) : 1$ (K = 1)
100	1905 KB	703 KB	0,63	2,71:1
101	1905 KB	664 KB	0,65	2,87:1
102	1905 KB	836 KB	0,56	2,28:1
103	1905 KB	760 KB	0,60	2,51:1
104	1905 KB	754 KB	0,60	2,53:1
105	1905 KB	763 KB	0,60	2,50:1

Tabela 5.1. Taxas de compressão em função de Kmax = 1 obtidas para sinais utilizados em testes

ECG (MIT-BIH)	Tamanho original (I)	Tamanho comprimido (II)	$TC = 1 - (II/I)$ (K = 2)	$TC = (I/II) : 1$ (K = 2)
100	1905 KB	692 KB	0,64	2,75:1
101	1905 KB	658 KB	0,65	2,90:1
102	1905 KB	827 KB	0,57	2,30:1
103	1905 KB	746 KB	0,61	2,55:1
104	1905 KB	746 KB	0,61	2,55:1
105	1905 KB	757 KB	0,60	2,52:1

Tabela 5.2. Taxas de compressão em função de Kmax = 2 obtidas para sinais utilizados em testes

ECG (MIT-BIH)	Tamanho original (I)	Tamanho comprimido (II)	$TC = 1 - (II/I)$ (K = 3)	$TC = (I/II) : 1$ (K = 3)
100	1905 KB	682 KB	0,64	2,79:1
101	1905 KB	654 KB	0,66	2,91:1
102	1905 KB	813 KB	0,57	2,34:1
103	1905 KB	737 KB	0,61	2,58:1
104	1905 KB	740 KB	0,61	2,57:1
105	1905 KB	752 KB	0,61	2,53:1

Tabela 5.3. Taxas de compressão em função de Kmax = 3 obtidas para sinais utilizados em testes

ECG (MIT-BIH)	Tamanho original (I)	Tamanho comprimido (II)	$TC = 1 - (II/I)$ (K = 4)	$TC = (I/II) : 1$ (K = 4)
100	1905 KB	671 KB	0,65	2,84:1
101	1905 KB	647 KB	0,66	2,94:1
102	1905 KB	799 KB	0,58	2,38:1
103	1905 KB	730 KB	0,62	2,61:1
104	1905 KB	735 KB	0,61	2,59:1
105	1905 KB	745 KB	0,61	2,58:1

Tabela 5.4. Taxas de compressão em função de Kmax = 4 obtidas para sinais utilizados em testes

ECG (MIT-BIH)	Tamanho original (I)	Tamanho comprimido (II)	$TC = 1 - (II/I)$ (K = 5)	$TC = (I/II) : 1$ (K = 5)
100	1905 KB	659 KB	0,65	2,89:1
101	1905 KB	635 KB	0,66	3:1
102	1905 KB	785 KB	0,59	2,42:1
103	1905 KB	720 KB	0,62	2,65:1
104	1905 KB	727 KB	0,61	2,62:1
105	1905 KB	788 KB	0,58	2,42:1

Tabela 5.5. Taxas de compressão em função de Kmax = 5 obtidas para sinais utilizados em testes

Na comparação dos resultados de compressão obtidos com outros projetos de monitores e com softwares de compactação comerciais, foi adotada a seguinte fórmula: $TC = (\text{Tamanho do arquivo original} / \text{Tamanho do arquivo comprimido}) : 1$. Adotou-se a média das taxas de compressão obtidas por cada projeto e software. Também foram utilizados arquivos de sinais ECG oriundos do MIT-BIH, com resolução de 12 bits/amostra, formatados através do aplicativo Java descrito anteriormente, para aferimento das taxas de compressão. As médias obtidas são apresentadas na Tabela 5.6.

	Este trabalho	(Primo, 2009; Cavalcanti, 2009)	(Ottley, 2007)	(Souza, 2008)	WinRAR 3.80	Win XP Zip
TCs	2,67:1	2,59:1	2,55:1	2,48:1	2,69:1	2,68:1

Tabela 5.6. Médias das taxas de compressão obtidas em diferentes projetos

De acordo com as taxas de compressão obtidas, apresentadas nas Tabelas 5.1, 5.2, 5.3, 5.4 e 5.5, foram constatadas a eficiência e robustez do módulo em questão. Com taxas de até 66%, o módulo de compressão gera sinais comprimidos de tamanho bastante reduzido, fazendo com que o dispositivo não necessite de memórias com grande capacidade de armazenamento, tornando-o menos dispendioso. Também podemos constatar a superioridade de desempenho na compressão de sinais ECG, em comparação a outros projetos de monitor já que, na média, possui a maior taxa média de compressão. Também é importante destacar que o referido módulo alcançou taxas de compressão muito próximas a taxas obtidas em softwares de compressão comerciais consagrados.

Concluída com sucesso a fase de compressão, foram iniciados os trabalhos de desenvolvimento de software para a fase de descompressão, com a implementação de um aplicativo que receberia os arquivos comprimidos a partir do sistema embarcado, descomprimiria-os e os exibiria em gráficos. Além disto, foi necessário o desenvolvimento de um módulo que ficaria responsável pela comparação de arquivos descomprimidos com arquivos originais, a fim de validar a descompressão. Para tal, foi utilizada a linguagem orientada a objetos Java, utilizando em conjunto o componente JFreeChart para a criação de gráficos. A IDE utilizada no desenvolvimento e testes foi o Netbeans IDE.

Os testes foram efetuados em todos os sinais comprimidos pelo módulo de compressão presente na camada de software do sistema embarcado, obtendo-se sinais descomprimidos com amostras idênticas às amostras originais, constatados visualmente através do software desenvolvido, através de gráficos ou de *feedbacks* de comparação. As Figuras 5.3 e 5.4 demonstram um dos testes efetuados, aplicado sobre o 2º canal do sinal 102 do MIT-BIH. É possível constatar visualmente a similaridade do sinal original com o sinal resultante após descompressão.

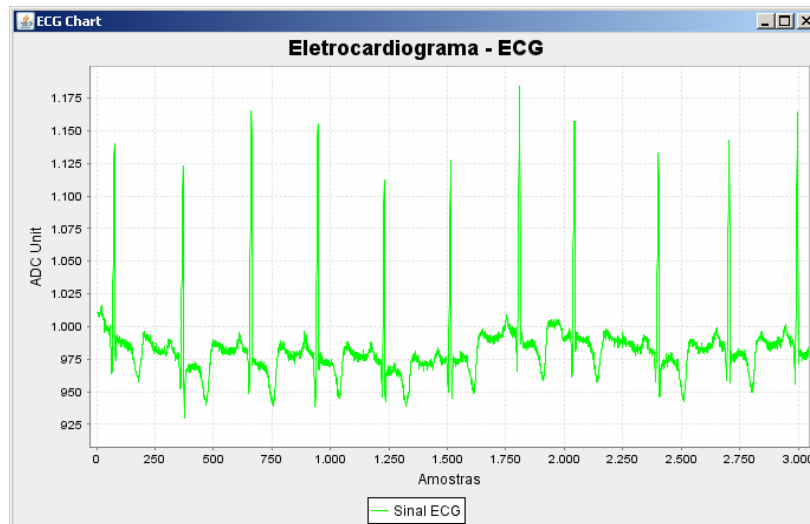


Figura 5.3. Três mil amostras do sinal original ECG 102.dat, canal 2

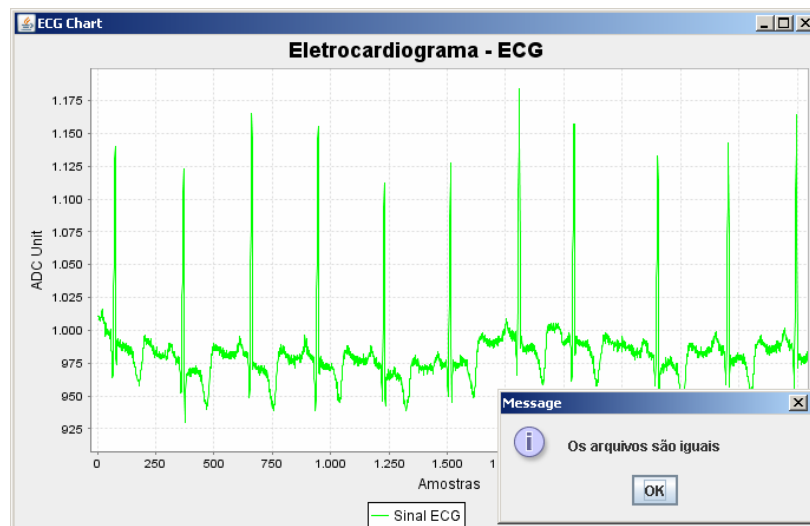


Figura 5.4. Três mil amostras do sinal ECG 102.dat, canal 2, resultante do processo de descompressão

5.2. Testes e Validação do Módulo Simulador

Sinais ECG com resolução de 12 bits foram gerados de acordo com os parâmetros listados na Tabelas 5.7 e 5.8. Os sinais gerados a partir de tais parâmetros são apresentados nas Figuras 5.5 e 5.6, respectivamente. O módulo em questão gera as amostras de sinal para cada conjunto de parâmetros e as armazena em arquivos .txt. Tais

arquivos de sinal foram comparados com os arquivos de amostra gerados pelo script de Karthik (2006) a partir dos mesmos conjuntos de parâmetros.

Batimentos Cardíacos	Amplitude Onda P	Amplitude Onda R	Amplitude Onda Q	Amplitude Onda T	Duração Intervalo P-R	Duração Intervalo S-T	Duração Intervalo P	Duração Intervalo QRS
72	25 mV	1.60 mV	0.025 mV	0.35 mV	0.16 s	0.18 s	0.09 s	0.11 s

Tabela 5.7. Parâmetros utilizados para a simulação do sinal ECG 1

Batimentos Cardíacos	Amplitude Onda P	Amplitude Onda R	Amplitude Onda Q	Amplitude Onda T	Duração Intervalo P-R	Duração Intervalo S-T	Duração Intervalo P	Duração Intervalo QRS
90	30 mV	1.80 mV	0.020 mV	0.50 mV	0.13 s	0.40 s	0.09 s	0.08 s

Tabela 5.8. Parâmetros utilizados para a simulação do sinal ECG 2

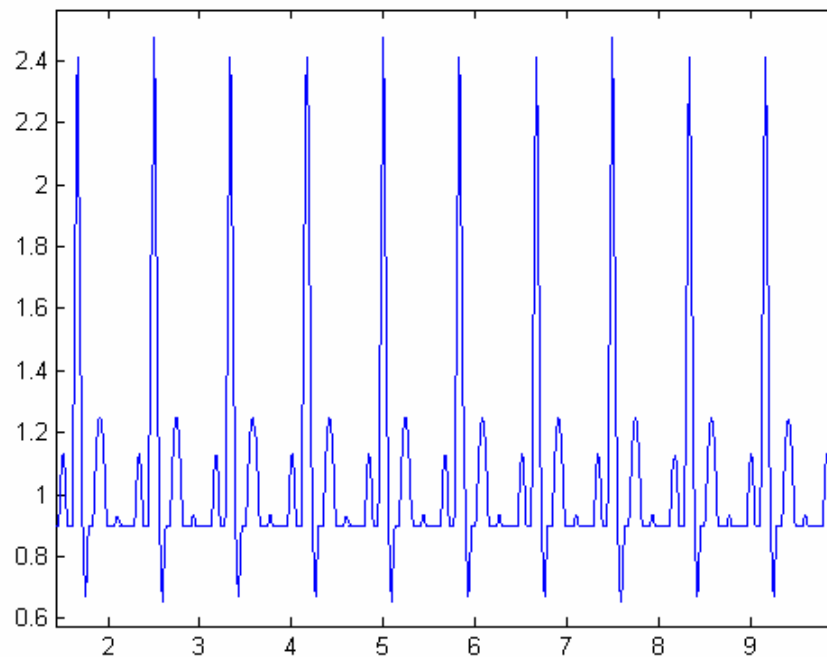


Figura 5.5. Sinal ECG gerado a partir dos parâmetros listados na Tabela 5.7

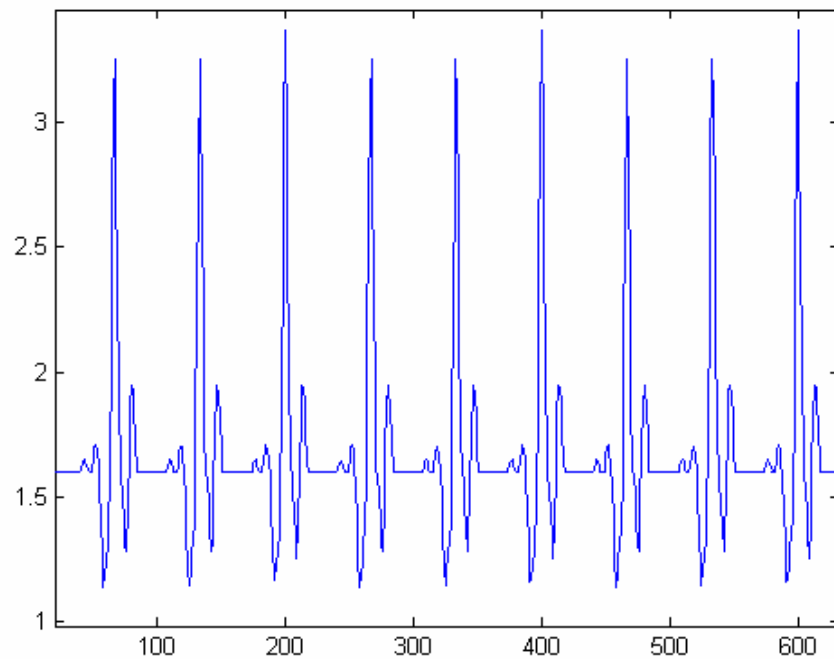


Figura 5.6. Sinal ECG gerado a partir dos parâmetros listados na Tabela 5.8

Tal comparação confirmou que o módulo simulador desenvolvido para a camada de software do sistema embarcado em questão simulou os dois sinais ECG corretamente, gerando sinais idênticos aos sinais gerados pelo script de Karthik (2006).

Finalizada a validação do referido módulo, foi possível a integração de todos os módulos pertencentes à camada de software do sistema embarcado, gerando desta forma o sistema de compressão de sinais ECG em tempo real, utilizando sinais gerados a partir das Séries de Fourier para a simulação de um monitor holter em processo de captação de sinais.

5.3. Testes e Validação do Sistema de Compressão em Tempo Real

O sistema embarcado definitivo gera amostras de sinais ECG com resolução de 12 bits por amostra a partir do módulo de simulação. Em paralelo, o módulo de simulação transmite 360 amostras por segundo para o módulo de compressão, que as codifica.

Amostras de sinais ECG foram geradas de acordo com os parâmetros listados anteriormente nas Tabelas 5.7 e 5.8. Os sinais gerados a partir de tais parâmetros são apresentados nas Figuras 5.5 e 5.6. Ao todo, foram geradas 31.200.000 amostras para cada sinal, 360 amostras por segundo (limite máximo de amostras/segundo definido no simulador), totalizando 24 horas de simulação para cada sinal. Desta forma, pôde-se testar o sistema embarcado em um período de uso equivalente a de um monitor holter.

Foi adotado o contexto $K_{max} = 5$ nos testes de compressão em tempo real. A aplicação de tal valor de contexto resultará em compressão com taxas máximas, de acordo com os resultados obtidos nos testes para o módulo de compressão, descritos anteriormente no tópico 5.2. As taxas de compressão resultantes são demonstradas na Tabela 5.9.

ECG	Tamanho original (I)	Tamanho comprimido (II)	$TC = 1 - (II / I)$ ($K = 5$)	%	$TC = (I / II) : 1$ ($K = 5$)
Figura 5.5	30.509 KB	18.620 KB	0,39	39 %	1,64 : 1
Figura 5.6	30.008 KB	13.545 KB	0,55	55 %	2,22 : 1

Tabela 5.9. Taxas de compressão obtidas para os sinais apresentados nas Figuras 5.5 e 5.6

De acordo com os resultados obtidos, foi constatada uma redução de até 55% no tamanho dos arquivos de sinal gerados pelo módulo de simulação, mantendo-se a eficiência e robustez do módulo de compressão, mesmo em processos de compressão em tempo real aplicados em sinais simulados.

5.4. Tamanho de Código e Recursos Embarcados

Uma aplicação Nios II C/C++ consiste em uma coleção de código fonte combinado em um único arquivo executável (extensão .elf). Tal arquivo é gerado após a compilação

bem sucedida de um aplicativo desenvolvido no Nios II IDE, podendo ser executado em um processador Nios II.

Além do executável, também é gerado um arquivo utilizado na programação da memória flash integrante do kit de desenvolvimento (extensão .flash). A partir deste, pode-se programar a memória flash com o código fonte implementado no Nios II IDE, fazendo com que o kit de desenvolvimento execute a aplicação desenvolvida de forma dedicada. Sua utilização é opcional, sendo independente do arquivo executável.

Os tamanhos obtidos para o código e recursos embarcados são os seguintes:

- Tamanho do arquivo executável (.elf):
 - Tamanho Total: 794 KB;
 - Módulo de Compressão: 547 KB;
 - Módulo de Simulação: 247 KB;

- Tamanho do arquivo de programação da memória flash (.flash):
 - Tamanho Total: 403 KB;
 - Módulo de Compressão: 207 KB;
 - Módulo de Simulação: 196 KB;

Conclusões

Os trabalhos de pesquisa produziram resultados relevantes e bastante satisfatórios, que serviram de base na validação de todo o trabalho efetuado e confirmaram sua viabilidade.

Graças à aplicação do modelo PPM na camada de software do dispositivo proposto, foram conquistadas taxas de compressão excelentes, o que viabiliza o projeto de um monitor holter eficiente e robusto, com o recurso de compressão de sinais, carente em projetos comerciais deste tipo de dispositivo. Ainda graças ao tamanho reduzido dos arquivos de ECG gerados, a submissão de sinais através da Internet e o armazenamento dos mesmos em dispositivos móveis de armazenamento, como cartões de memória, serão efetuados de forma otimizada.

Tal sistema atingiu taxas de compressão superiores a outros projetos de monitores holter que utilizam compressão sem perda de dados, além de taxas de compressão similares a softwares comerciais de compressão de propósito geral consagrados. O protótipo, mesmo sendo superado por outros projetos que utilizam compressão com perdas no que se refere a taxas de compressão, mantém os dados originais no processo de codificação/decodificação, gerando sinais decodificados que poderão ser analisados com a precisão que os sistemas com perdas não poderão aferir.

Gerando-se arquivos de sinais ECG com tamanhos reduzidos, os recursos de armazenamento e transmissão de dados serão usados com maior eficiência. Sinais ECG comprimidos requisitam menor espaço de armazenamento, e podem ser submetidos mais rapidamente através da Internet ou em uma rede Wireless. Como consequência, o dispositivo poderá ser comercializado com preços bem mais baixos que os utilizados atualmente, exigindo menos componentes, menor consumo de energia, tornando-o mais portátil, com possibilidade de agregação de novos recursos.

O sistema embarcado mostrou-se útil, não apenas como um protótipo para um monitor holter, mas também como um sistema de simulação de sinais eletrocardiográficos. Tal sistema poderá ser utilizado no aferimento, calibração, desenvolvimento, teste, validação, pesquisa e produção de outros monitores de ECG. O mesmo também poderá ser utilizado no treinamento e pesquisas em eletrocardiografia. Com o simulador de sinais ECG integrado ao protótipo, o sistema embarcado poderá se autocalibrar, sendo desnecessário o uso de dispositivos adicionais para este trabalho.

Devido à limitação do kit de desenvolvimento utilizado, não foi possível a gravação dos sinais ECG codificados em cartões de memória. Apesar do kit utilizado possuir interface para este dispositivo de armazenamento móvel, a mesma só pode ser utilizada em procedimentos de leitura de dados. Tal limitação poderá ser sanada naturalmente através da adoção de kits de desenvolvimento que forneçam interfaces menos limitadas, ou através da implementação de drivers. No referido trabalho, sinais comprimidos foram armazenados em um sistema de processamento hospedeiro, conectado ao kit de desenvolvimento, através do uso do componente *Altera[®] Host Based File System*, presente no Nios II IDE.

Sendo este um projeto de sistema embarcado, e dada sua característica reconfigurável, o mesmo poderá servir de base para outros projetos de hardware e software, acadêmicos e comerciais, que por ventura venham a utilizar compressão e simulação de sinais ECG.

A referida pesquisa deu origem a duas publicações internacionais, intituladas *ECG DATA COMPRESSION FOR AN EMBEDDED SYSTEM OF A HOLTER MONITOR USING PPM* (2009) e *EMBEDDED SYSTEM THAT SIMULATES ECG WAVEFORMS* (2010). O primeiro artigo foi apresentado e publicado nos anais da conferência *IADIS Applied Computing 2009*, realizada no período de 19 a 21 de novembro na cidade de Roma, na Itália. O segundo artigo foi aceito para apresentação e publicação nos anais da conferência *VI Southern Programmable Logic Conference*, a ser realizado no período de 24 a 26 de

março, no município de Ipojuca, em Pernambuco. Os artigos são apresentados na íntegra nos Apêndices A e B, respectivamente.

Portanto, pode-se afirmar que a pesquisa em questão atendeu as finalidades objetivadas.

Em trabalhos futuros, são sugeridos o desenvolvimento de um controlador para aquisição analógica / digital e a integração de um conversor A/D à camada de hardware do sistema embarcado. Através dos mesmos, pode-se viabilizar a digitalização de sinais ECG através de métodos invasivos e não-invasivos. Também é sugerido o desenvolvimento de um novo sistema embarcado, desta vez objetivando a calibração de monitores de ECG, utilizando-se o módulo simulador de sinais ECG desenvolvido neste trabalho como ponto de partida.

Referências Bibliográficas

- [1] ALTERA CORPORATION. **Nios II Development Kit – Getting Started User Guide**. San Jose, CA, USA, 2007.
- [2] ALTERA CORPORATION. **Nios Development Board Reference Manual, Stratix Edition**. San Jose, CA, EUA, 2004.
- [3] ALTERA CORPORATION. **Nios II Hardware Development Tutorial**. San Jose, CA, USA, 2007.
- [4] AMORE, R. d'. **VHDL: descrição e síntese de circuitos digitais**. Rio de Janeiro: LCT, 2005.
- [5] ANALOG DEVICES. **Complete 12-Bit A/D Converter – AD574A Manual**. Norwood, MA, EUA, 1995.
- [6] BRASILEIRO, J. J. P. ; CAVALCANTI, A. C. **Compressão Sem Perdas De Sinais Biológicos (Aplicação Ao Ecg) Utilizando A Codificação Gray E Versão Binária Do Algoritmo Prediction By Partial Matching (PPM)**. 2009. Universidade Federal da Paraíba, João Pessoa, XVII Encontro de Iniciação Científica (Apresentação de Trabalho/Seminário).
- [7] BILGIN, A.; MARCELLIN, M. W.; ALTBACH, M. I. **Compression of electrocardiogram signals using JPEG2000**. IEEE Xplore digital library. 2003. Disponível em: <<http://www.ieee.org>>. Acesso em: 13 de maio de 2009.
- [8] BOUKAACHE, A. **An Improvement Embeeded Zerotree Wavelet Algorithm for ECG Data Compression**. 2006. Disponível em: <<http://www.eurasip.org/Proceedings/Ext/ISCCSP2006/defevent/papers/cr1284.pdf>>. Acesso em: 13 de maio de 2009.
- [9] BRAEMAR, Inc. **DL900 Digital Holter Monitor**. EUA, 2009. Disponível em: <www.advancedbiosensor.com/downloads/DL900-2009.pdf>. Acesso em: 30 de Julho de 2009.

- [10] CARDOSO, F.; FERNANDES, M.; ARANTES, D. **FPGA e Fluxo de Projeto**. Setembro de 2007. Disponível em: <http://www.decom.fee.unicamp.br/~cardoso/ie344b/Introducao_FPGA_Fluxo_de_Projeto.pdf>. Acesso em: 5 de maio de 2009.
- [11] CLEARY, J. G.; WITTEN, I. H. **Data Compression Using Adaptive Coding and Partial String Matching**. IEEE Transactions on Communications COM-32(4):396–402, Abril de 1984.
- [12] COUTINHO, B. C. et al. **Atribuição de Autoria usando PPM**. In: XXV Congresso da Sociedade Brasileira de Computação: A universidade da Computação: um agente de inovação e conhecimento. Unisinos, São Leopoldo/RS. Disponível em: <http://www.unisinos.br/_diversos/congresso/sbc2005/_dados/anais/pdf/arq0115.pdf>. Acesso em: 17 de dezembro de 2009.
- [13] DIRICHLET, G. L. **Sur la convergence des séries trigonométriques qui servent à représenter une fonction arbitraire entre des limites-données**, 1829. Journal für die reine und angewandte Mathematik. v. 1829(4). p. 157-169.
- [14] FARIAS, T. M. T.; LIMA, J. A. G. **Ecg Data Compression for an Embedded System of a Holter Monitor Using PPM**, 2009, Roma. Proceedings of the IADIS INTERNATIONAL CONFERENCE APPLIED COMPUTING 2009. LISBOA : IADIS PRESS, 2009. v. II. p. 340-342.
- [15] FARIAS, T. M. T.; LIMA, J. A. G. **Embedded System That Simulates Ecg Waveforms**, 2010, Ipojuca, Pernambuco. Proceedings of the IV SOUTHERN PROGRAMMABLE LOGIC CONFERENCE 2010.
- [16] FUKUDA DENSHI. **Digital Holter ECG Recorder – Digital Walk FM-150**. Tokyo, Japão, 2009. Disponível em: <http://www.fukuda.co.jp/english/products/holter/pdf/fm_150.pdf>. Acesso em: 30 de Julho de 2009.
- [17] FOURIER, J. **Théorie du mouvement de la chaleur dans les corps solides (suite)**, 1826. Mémoires de l'Académie royale des sciences de l'Institut de France. p. 153-246.
- [18] GRUPO DE ESTUDOS DE ELETROCARDIOGRAFIA. **SBC/ECG – Consensos/Diretrizes**. Disponível em: <<http://departamentos.cardiol.br/eletroc/publicacoes/diretrizes/diretriz11/introducao.asp>>. Acesso em: 18 de maio de 2009.
- [19] HUFFMAN, D. A. **A Method for the Construction of Maximum of Minimum Redundancy Codes**, Proc. IRE, 1098-1101, 1952. 4.2, 9.3.3.

- [20] INNOMED MEDICAL. **ArguSys – Holter Monitoring System**. Budapeste, Hungria, 2009. Disponível em: <www.festta.hr/include/docs/argusys_en%5B1%5D.pdf>. Acesso em: 30 de Julho de 2009.
- [21] JFRECHART. Disponível em: <<http://www.jfree.org/jfreechart/>>. Acesso em: 1 de abril de 2009.
- [22] JIN, H.; MIAO, B. 2007. **Design of holter ECG System Based on MSP430 and USB Technology**. IEEE Conference eXpress.
- [23] KARTHIK, R. 2006. **ECG Simulation Using MATLAB – Principle of Fourier Series**. Disponível em: <<http://www.mathworks.com/matlabcentral/fileexchange/10858>>. Acesso em: 13 de Agosto de 2009.
- [24] MARTINS, C. A. P. S. et al. **Computação Reconfigurável: conceitos, tendências e aplicações**. Disponível em: <<http://www.inf.ucp.br/nsb/downloads/arquivos/ComputacaoReconfiguravel.pdf>>. Acesso em: 28 de setembro de 2009.
- [25] MOFFAT, A. **Implementing the PPM Data Compression Scheme**. IEEE Transactions on Communications, vol. COM-38, Novembro de 1990.
- [26] NANCY, C. S. et al. **An ECG Compression Algorithm for Full Disclosure in a Solid-State Real-Time Holter Monitor**. Hewlett-Packard Company, McMinnville, Oregon, USA. 1989.
- [27] NETBEANS IDE. Disponível em: <<http://www.netbeans.org>>. Acesso em: 1 de abril de 2009.
- [28] NETWORK WORKING GROUP. **DEFLATE Compressed Data Format Specification version 1.3**. Alemanha, 1996. Disponível em: <<http://tools.ietf.org/html/rfc1951>>. Acesso em: 30 de Julho de 2009.
- [29] NOUAILHETAS, V. L. A. et al. **Eletrocardiograma**. Disponível em: <<http://www.virtual.epm.br/material/tis/curr-bio/trab2003/g5/menu.html>>. Acesso em: 28 de setembro de 2009.
- [30] OLIVEIRA, P. A. de. **Normas para elaboração de dissertação de mestrado e tese de doutorado**. Disponível em: <www.icb.ufmg.br/big/pg-gen/arquivos/resolucoes/norma_dissertacao_mest_dout.pdf>. Acesso em: 3 de junho de 2009.

- [31] OLIVEIRA, P. A. de. **Normas para elaboração de referências bibliográficas**. Disponível em: <http://www.fmr.edu.br/publicacoes/pub_24.pdf>. Acesso em: 3 de junho de 2009.
- [32] OTTLEY, A. C. **ECG Compression for holter Monitoring**. Dissertação de Mestrado, University of Saskatchewan Library Electronic Theses & Dissertations, University of Saskatchewan , 2007.
- [33] PHYSIONET. **MIT-BIH Arrhythmia Database**. 2009. Disponível em: <<http://www.physionet.org/physiobank/database/mitdb/>>. Acesso em: 5 de janeiro de 2009.
- [34] PINCIROLI, F.; COMBI, C. **Compression of ECG Signals in Holter Monitoring**. Disponível em: <<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245206/>>. Acesso em: 13 de maio de 2009.
- [35] PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA CIVIL - UFB. **Normas de Formatação e Redação – Dissertação de Mestrado**. Uberlândia, 2003.
- [36] RAMOS, A. P.; SOUSA, B. S. **Eletrocardiograma: princípios, conceitos e aplicações**. 2007. Disponível em: <<http://www.centrodeestudos.org.br/pdfs/ecg.pdf>>. Acesso em: 13 de maio de 2009.
- [37] SALOMON, D. **Data Compression**. Springer, New York, USA. 2004.
- [38] SHANNON, C. E. (1948) **A Mathematical Theory of Communication**. Bell Syst. Tech. J., v. 27, p.379-423.
- [39] SHKARIN, D. **PPMd Compressor**. Disponível em: <<http://compression.ru/ds/>>. Acesso em: 23 de julho de 2009.
- [40] SKLIAROVA, I.; FERRARI, A. B. **Introdução à Computação Reconfigurável**. Revista do DETUA, volume 2, nº 6, 2003. Disponível em: <http://www.ieeta.pt/~iouliia/Papers/2003/1_SF_ETSet2003.pdf>. Acesso em: 8 de dezembro de 2009.
- [41] SMITH, N. C.; PLATT, J. S. **An Ecg Compression Algorithm For Full Disclosure In A Solid-State Real-Time Holter Monitor**. Hewlett-Packard Company, McMinnville, Oregon, USA, 1989.

- [42] SOCIEDADE BRASILEIRA 100 LIMITE. **Diretriz de interpretação de eletrocardiograma de repouso**, Arq. Bras. Cardiol. volume 80, (suplemento II), 2003. Disponível em: <<http://publicacoes.cardiol.br/consenso/2003/8002/repouso.pdf>>. Acesso em: 18 de maio de 2009.
- [43] SOUZA, A. R. C. et al. **Compressão Contextual sem Perdas de Eletrocardiogramas utilizando Recursos Reduzidos de Memória**. In: IV Latin American Congress on Biomedical Engineering 2007, Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil. 2007.
- [44] SOUZA, A. R. C. **Desenvolvimento e Implementação em FPGA de um sistema portátil para aquisição e compressão sem perdas de eletrocardiogramas**. Dissertação de Mestrado, Programa de Pós-Graduação em Informática, Universidade Federal da Paraíba, João Pessoa - PB, 2008.
- [45] SPIEGEL, J. V. **VHDL Tutorial**. 2001. Disponível em: <http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html#_Toc526061341>. Acesso em: 9 de dezembro de 2009.
- [46] VARGAS, R. et al. **Sistemas Embarcados: Acoplamento do Soft-Core Plasma ao Barramento OPB de um PowerPC 405**. Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil. 2007.
- [47] WAGNER, B.; BARR, M. **Introduction to Digital Filters, Embedded Systems Programming**, Dezembro de 2002, pp. 47-48.
- [48] WELCH, T. A. **A Technique for High-Performance Data Compression**. Computer, Junho de 1984, Vol. 17, pp. 8-19.
- [49] WIKIPEDIA. Disponível em: <<http://pt.wikipedia.com>>. Acesso em: 1 de junho de 2009.
- [50] WinRAR. Disponível em: <<http://www.win-rar.com/homepage.html>>. Acesso em: 23 de julho de 2009.
- [51] WinZip. **Additional Compression Methods Specification**. Disponível em: <http://www.winzip.com/comp_info.htm>. Acesso em: 23 de julho de 2009.
- [52] ZIV, J.; LEMPEL, A. **A Universal Algorithm for Sequential Data Compression**, IEEE Transactions on Information Theory, 23(3), pp. 337-343, maio de 1977. Disponível em: <http://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempe1_1977_universa1_algorithm.pdf>. Acesso em: 23 de julho de 2009.

Glossário

A

Altera Corporation

É uma empresa fabricante de dispositivos lógicos programáveis.

ASCII

É um código padrão americano utilizado no intercâmbio de informações. Trata-se de um código numérico usado para representar letras, números e caracteres especiais, e entendido por quase todos os computadores, impressoras e programas de edição de texto.

B

Buffer

Região de memória temporária utilizada para leitura e escrita de dados. Os mesmos podem ser implementados em software ou hardware.

C

CAD

Computer-Aided Design, ou desenho auxiliado por computador. Softwares CAD permitem a criação e edição de desenhos gráficos aplicados em engenharia, arquitetura, design, publicidade, entre outros.

D

Debugging

Depuração. Processo que ocorre em consequência de um teste, para determinar a natureza e a localização do erro no código a fim de corrigi-lo.

DMA

Permite que certos dispositivos de hardware em um computador acessem a memória do sistema para leitura e escrita independentemente da CPU.

E

EPROM

É um tipo de chip de memória que mantém seus dados quando a alimentação é desligada (não-volátil). A mesma é programada por um dispositivo eletrônico que dá voltagens maiores do que os usados normalmente em circuitos elétricos. Uma vez programada, uma EPROM pode ser apagada apenas por exposição a uma fonte de luz ultravioleta.

EEPROM

Memória que pode ser programada e apagada várias vezes, eletricamente. Pode ser lida em um número limitado de vezes, mas só pode ser apagada e programada em um número limitado de vezes, que variam entre 100.000 e 1 milhão.

F

FireWire

Interface serial para computadores pessoais e aparelhos digitais de áudio e vídeo que fornece comunicações de alta velocidade e serviços de dados em tempo real. Trata-se de uma tecnologia sucessora da quase obsoleta SCSI.

G

GNU

Acrônimo recursivo que significa *GNU is not Unix* (GNU não é Unix). Sistema totalmente livre, que qualquer pessoa tem o direito de usar, modificar e redistribuir, junto ao seu código-fonte.

H

HAL

Hardware Abstract Layer, ou camada de abstração de hardware. É um software de código aberto que permite que outros softwares acessem funções e obtenham informações do sistema operacional e de hardwares com muito menos dificuldade.

HDL

Hardware Language Description, ou linguagem de descrição de hardware. Permite a descrição de um projeto eletrônico.

Host

É qualquer computador conectado a uma rede.

Huffman (Algoritmo de Codificação)

Trata-se de uma técnica de compressão sem perda, que atribui códigos de tamanho variado para uma série de valores conhecidos. Valores que ocorrem com maior frequência serão atribuídos a códigos curtos.

N

Nios II

Arquitetura para um processador embarcado de 32 bits para FPGA's da família Altera.

N-Grama

Subseqüência de n itens de uma dada seqüência (digramas, trigramas, etc.).

O

Open Source

Significa Código Aberto, isto é, código que respeita quatro liberdades (ver GNU).

P

Plug and Play

Significa, ao pé da letra, plugar e usar. Designa dispositivos que bastam ser conectados ao computador para que funcionem com eficácia.

R

RAM

Memória de acesso aleatório. É um tipo de memória que permite a leitura e escrita, utilizada como memória primária em sistemas eletrônicos digitais. Trata-se de uma memória volátil, isto é, seu conteúdo é perdido quando sua alimentação é desligada.

RISC

Reduced Instruction Set Computer ou Computador com Conjunto Reduzido de Instruções. É uma linha de arquitetura de computadores que favorece um conjunto simples e pequeno de instruções que levam aproximadamente a mesma quantidade de tempo para serem executadas.

ROM

Memória apenas de leitura. É um tipo de memória que permite apenas a leitura, ou seja, suas informações são gravadas pelo fabricante uma única vez e após isso não podem ser alteradas ou apagadas, somente acessadas. São memórias cujo conteúdo é gravado permanentemente (memória não-volátil).

RTL

Register Transfer Level, ou transferência a nível de registrador. É um tipo de programação em VHDL.

S

SPI

Protocolo que permite a comunicação de um microcontrolador com diversos outros componentes, formando uma rede.

Stream

Fluxo de dados em um sistema computacional.

U

USART

É um formato padrão para comunicação de dados de forma serial. Os microcomputadores costumam ter até 4 portas USART, nomeadas de COM1 a COM4. Entre os dispositivos ligados em protocolo USART estão o modem e o mouse.

V

Verilog

É uma linguagem de descrição de hardware usada para modelar sistemas eletrônicos. A mesma suporta o design, verificação e implementação de projetos analógicos, digitais e

circuitos híbridos em vários níveis de abstração. Com placas especiais, é possível descarregar o código gerado nessa linguagem em FPGA's.

VHDL

VHSIC Hardware Description Language. É uma linguagem de programação que tem por objetivo facilitar o desenvolvimento de projetos de hardware. A partir de um algoritmo produzido nesta linguagem é possível desenvolver um circuito, documentar, descrever, efetuar síntese, simular, testar, efetuar verificação formal e ainda compilar software.

APÊNDICE A - Artigo “ECG DATA COMPRESSION FOR AN EMBEDDED SYSTEM OF A HOLTER MONITOR USING PPM”

ECG DATA COMPRESSION FOR AN EMBEDDED SYSTEM OF A HOLTER MONITOR USING PPM

Thyago Maia Tavares de Farias

*Programa de Pós-Graduação em Informática – Departamento de Informática – Universidade Federal da Paraíba
Cidade Universitária - João Pessoa - PB - Brazil – CEP: 58059-900*

José Antônio Gomes de Lima

*Programa de Pós-Graduação em Informática – Departamento de Informática – Universidade Federal da Paraíba
Cidade Universitária - João Pessoa - PB - Brazil – CEP: 58059-900*

ABSTRACT

This work describe the proposal for an embedded system of a holter monitor that implements data compression, using the PPM Algorithm. Through a holter monitor, cardiologists can obtain ECG signals, serving as the basis for the perception of symptoms and activities of patients. These signals are captured and recorded by holter monitors in periods greater than or equal to 24 hours, requiring large storage size. Using the PPM Algorithm, a holter monitor can considerably reduce the size of the signals stored, thus reducing storage space.

KEYWORDS

Holter Monitor, PPM, ECG, Data Compression, Embedded System, FPGA.

INTRODUCTION

The holter monitor is a device that aims to track patients through the captation of ECG signals. An ECG is a graph that show the electrical activity of the heart muscle [Ottley, 2007]. The electrical activity of the heart is captured by means of electrodes connected to the holter monitor, which are placed on the skin of the patient. The device then stores the cardiac activity, usually in a flash memory that incorporates the monitor holter structure, during 24 hours or more. Due to this event, is important the use of flash memories that have sufficient capacity to store large strips of ECG signals in this period of time. Is also necessary to insert additional resources which can facilitate the process of storage signals and allow the user to transport ECG signals easily and quickly.

Was found through surveys of resources and characteristics of commercial holter monitors that most of them don't have the compression processing of ECGs, or simply don't disclose or otherwise provide such information. It was also observed that many academic projects related to the theme focusing on the compression as an important resource for a holter monitor, but use less robust algorithms than the PPM. Using the PPM algorithm in the embedded system proposed, features of storing and transferring data will be used more efficiently. Compressed ECG signals require less space for storage, and they can be sent more quickly over the Internet or a Wireless Network.

METHODOLOGY

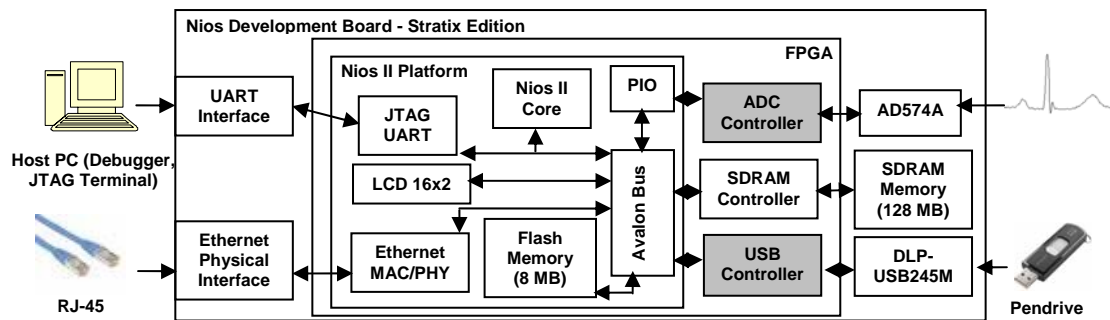
Its proposal involves description of the platform Nios II, including internal peripheral and access to external devices, software development with GNU C/C++ in Eclipse IDE and debugging aided by hardware. Nios II is

treated as a reconfigurable processor soft-core. A set of standard peripherals follow the platform and there is the possibility of development of personalized peripherals. The development kit used to design the holter monitor was the Altera Nios Development Board, Stratix Edition. This board provides a hardware platform for developing embedded systems based on Altera Stratix devices.

HARDWARE LAYER

Figure 1 shows the hardware structure of the holter monitor proposed in this article.

Figure 1. Hardware layer of the holter monitor proposed



The Nios II Core executes the modules of the software layer, previously stored in SDRAM memory. The Avalon is a special bus that prioritizes the speed of data communication, allowing connections in parallel. The PIO module offers input and output ways, establishing a communication between the Nios II platform and blocks used. The Flash Memory device is an 8 Mbyte AMD AM29LV065D, used as general-purpose readable memory and non-volatile storage. The LCD, with a two-line x 16-character, displays useful status, progress messages and warning messages. The JTAG UART core uses the JTAG circuitry built in to Altera FPGA's, and provides host access via the JTAG pins on the FPGA. The Ethernet interface communication is guaranteed by the implementation of the MAC layer inside of the Nios II platform that interact externally with the physical layer [Altera Corporation, 2004].

The DLP-USB245M is a USB communication interface that provides a method of data transfer to/from peripherals and hosts up to 8 bits per second [DLP Design, 2002]. This component will be controlled through a USB controller developed in VHDL.

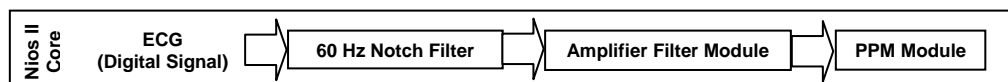
The AD574A is an analog-to-digital converter that has a reference voltage of 10V and maximum frequency of 25 kHz [Analog Devices, 1995]. The ADC Controller, also developed in VHDL, digitizes and sends the ECG signals for the Nios II Core, with a resolution of 11 bits, working with 8 channels and variable sampling frequency.

The software used to define and generate the system proposed are the Altera Quartus II and SOPC Builder.

SOFTWARE LAYER

Figure 2 shows the software structure of the holter monitor proposed in this article.

Figure 2. Software layer of the holter monitor proposed



The 60 HZ Notch Filter Module is used to remove interference of captured signals, through a FIR Filter implementation. The Amplifier Filter Module has the task of increasing the amplitude of captured ECG signals. The PPM Module is responsible for compression of ECG signals received by the holter monitor. The PPM (Prediction by Partial Matching) is a technique for entropy coding based on statistical modeling and prediction for adaptive context. This model is considered one of the most efficient compressors for general purpose today.

The software used for the development of modules that comprise this layer was the Nios II Embedded Design Suite. This IDE carries the developed modules for the SDRAM memory, to be performed by Nios II Core.

PRELIMINARY RESULTS

Compression tests were run on signal files from the MIT/BIH Database at 360 samples per second per channel with 11-bit resolution over a 10 mV range [PhysioNet, 2009].

The criterion used for evaluation was the analysis of compression rates (CR). This is calculated as: $1 - (\text{compressed response size} / \text{original response size})$.

The signals used in the tests were compressed using the context $K = 5$. Table 1 shows the average compression ratio obtained for the signals 100, 101 and 102.

Table 1. Signals used and average compression ratio obtained in tests

ECG's (MIT-IBH)	Original Size (Avg.)	Compressed Size (Avg.)	CR, $K = 5$ (Avg.)	Percentage (Avg.)
100, 100 and 102	1.905 KB	693 KB	0,63	63%

The following formula was used to compare compression rates obtained through other projects and compression programs: $TC = (\text{original file size} / \text{compressed file size})$: 1. Was adopted the average compression rates obtained by each project and software. The averages obtained are shown in Table 2.

Table 2. Average compression rates obtained in different projects

	This Paper	[Ottley, 2007]	WinRAR 3.80	Win XP Zip
Compression Rates	2.77:1	2.55:1	2.77:1	2.77:1

CONCLUSION

The results obtained show that the PPM Module got excellent compression rates. With compression rates greater than or equal to academic holter monitors and commercial software compression, PPM module was very feasible and efficient for ECG data compression. Concluded this stage, this work will be continued with the development of other resources that directly or indirectly will benefit by the same module.

ACKNOWLEDGEMENT

We would like to thank CNPq of Brazil for financial support. We would also like to thank LASID-DI-UFPB by providing the Altera's development kit, used in this proposal.

REFERENCES

- Altera Corporation, 2004. *Nios Development Board Reference Manual, Stratix Edition*. Altera Corp., San Jose, CA, USA.
- Analog Devices, 1995. *Complete 12-Bit A/D Converter – AD574A Manual*. Analog Devices, Norwood, MA, USA.
- DLP Design, Inc., 2008. *DLP-USB245M User Manual*. DLP Design, Roma Lane Allen, TX 75013, USA.
- Ottley, A. C., 2007. *ECG Compression for holter Monitoring*. Master Thesis. Saskatoon, SK, Canada: University of Saskatchewan.
- PhysioNet, 2009. *MIT-BIH Arrhythmia Database*. [Online] Available at: www.physionet.org [accessed 5 January 2009].
- Salomon, D., 2004. *Data Compression*. Springer, New York, USA.

APÊNDICE B - Artigo “EMBEDDED SYSTEM THAT SIMULATES ECG WAVEFORMS”

EMBEDDED SYSTEM THAT SIMULATES ECG WAVEFORMS

Thyago Maia Tavares de Farias

*Programa de Pós-Graduação em Informática
Universidade Federal da Paraíba
Cidade Universitária - João Pessoa - PB – Brasil – CEP: 58059-900
email: contato@thyagomaia.com*

José Antônio Gomes de Lima

*Programa de Pós-Graduação em Informática
Universidade Federal da Paraíba
Cidade Universitária - João Pessoa - PB – Brasil – CEP: 58059-900
email: jose@di.ufpb.br*

ABSTRACT

This paper describes a embedded system developed for simulation of electrocardiographic signals, also known as ECG signals. The objective of this system is generate several examples of ECG waveforms for analysis and reviews in short time periods, eliminating the difficulties of obtaining real ECG signals through the invasive and noninvasive methods. One can simulate any given ECG waveform using this embedded system. This simulator was developed through the Altera's Nios® II development kits and Altera's CAD software, for definition of hardware layer, beyond the use of Fourier series and Karthik's algorithm implemented in language C through the Altera's Nios® II IDE, for implementation of software layer.

1. INTRODUCTION

According to Karthik [1], any periodic functions which satisfies Dirichlet's condition, boundary condition in which a state variable remains constant over time [2], can be expressed as a series of scaled magnitudes of sine and cosine terms of frequencies which occur as a multiple of fundamental frequency. Karthik [1] also states that ECG signals are periodics, with frequency determined by heart beats, and satisfy Dirichlet's condition. Therefore, Fourier series can represent ECG signals. Fourier series are described in (1a).

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(n\pi x / l) + \sum_{n=1}^{\infty} b_n \sin(n\pi x / l) \quad (1a)$$

$$a_0 = \frac{1}{l} \int_T f(x) dx, \quad T = 2l \quad (1b)$$

$$a_n = \frac{1}{l} \int_T f(x) \cos(n\pi x / l) dx, \quad n = 1, 2, 3, \dots \quad (1c)$$

$$b_n = \frac{1}{l} \int_T f(x) \sin(n\pi x / l) dx, \quad n = 1, 2, 3, \dots \quad (1d)$$

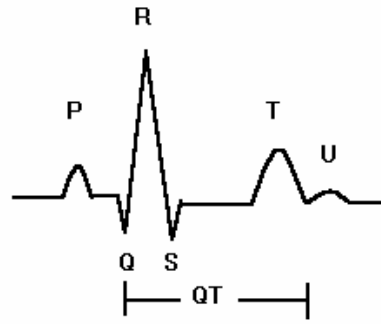


Fig. 1. Typical ECG signal.

This work is inspired in the algorithm implemented in MATLAB® script language by Karthik [1]. From the definition of signal parameters as heart beats frequency, amplitude and duration, the algorithm calculates separately the portions P, T, U, Q, S and QRS of a typical ECG signal. These portions are illustrated in Fig. 1. The calculation of each portion is based in Fourier series described in (1a). Every significant feature of an ECG signal is generated from the sum of each of these waveforms.

Developing an embedded system based on Karthik's algorithm [1] enables the prototyping of hardware that will help researchers in the analysis and reviews of electrocardiographic signals. Normal and abnormal ECG signals, and various types of arrhythmias, can be generated with this prototype.

2. METHODOLOGY

The work involves description of the platform Nios® II, including internal peripheral and access to external devices, software development with GNU C/C++ in Eclipse® IDE and debugging aided by hardware. Nios® II is treated as a reconfigurable processor soft-core. A set of standard peripherals follows the platform and there is the possibility of development of personalized peripherals. The development kit used to design the holter monitor was the Altera's® Nios Development Board, Stratix Edition. This board provides a hardware platform for developing embedded systems based on Altera® Stratix devices.

3. HARDWARE LAYER

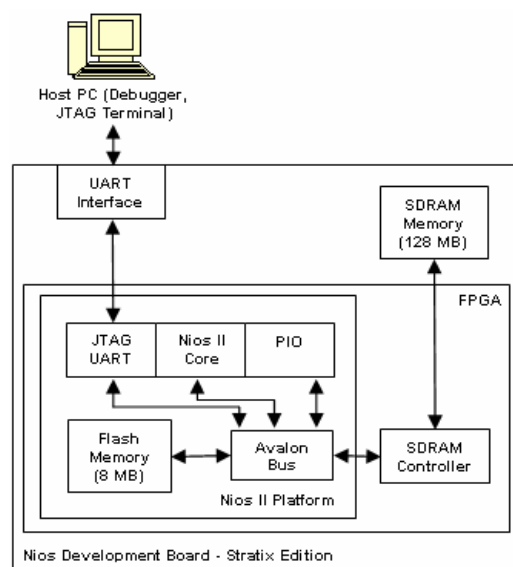


Fig. 2. Hardware Layer.

Fig. 2 shows the hardware structure of the simulator. The Nios II Core executes the module of the software layer, previously stored in SDRAM memory. The Avalon® is a special bus that prioritizes speed data-communication, allowing connections in parallel. The PIO module offers input and output ways, establishing a communication between the Nios II platform and blocks used. The Flash Memory device is a 8 Mbyte AMD AM29LV065D, used as general-purpose readable memory and non-volatile storage. The JTAG UART core uses the JTAG circuitry built in to Altera® FPGA's, and provides host access via JTAG pins on the FPGA [3].

The software used to define and generate the system are the Quartus® II and SOPC Builder.

4. SOFTWARE LAYER

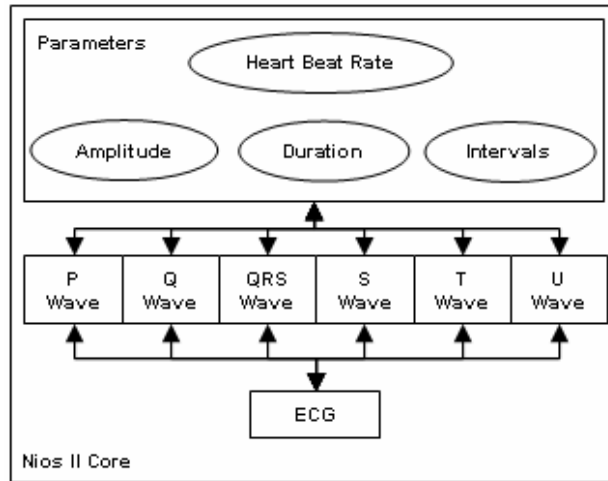


Fig. 3. Software layer.

```

procedure qrs_q_s(amp, dur, hbr)
{
    x = 0.01:0.01:600;

    li = 30/hbr;

    b = (2*li)/dur;

    wave_1 = (amp/(2*b))*(2-b);

    n = <TOTAL_NUMBER_OF_SAMPLES>;

    for i = 1 to n
        harm =
            (((2*b*amp)/((i*i)*(PI*PI)))*(1
            cos((i*PI)/b)))*cos((i*PI*x)/1);

    wave_2 = wave_2 + harm;

    end

    final_wave = wave_1 + wave_2;
}

```

Fig. 4. Algorithm for the calculation of QRS, Q and S portions.

```

procedure p_t_u(amp, dur, hbr, int)
{
    x = 0.01:0.01:600;

    x = x - int;

    li = 30/hbr;

    b = (2*li)/dur;

    u1 = 1/1;

    n = <TOTAL_NUMBER_OF_SAMPLES>;

    for i = 1 to n
        harm =
            (((sin((PI/(2*b))*(b-(2*i))))/(b-
            (2*i)))+(sin((PI/(2*b))*(b+(2*i))))
            )/(b+(2*i)))*(2/PI))*cos((i*PI*x)
            /1);

    u2 = u2 + harm;

    end

    wave_1 = u1 + u2;

    final_wave = wave_1 * amplitude;
}

```

Fig. 5. Algorithm for the calculation of P, T and U portions.

Fig. 3 shows the software structure of the simulator. The parameters of amplitude, duration, heart beat rate and intervals (P-R and S-T) are used to calculate the P, Q, R, S, and T portions. Each sample portion is generated from 2 procedures, where one will be responsible for calculating the samples of QRS, Q and S portions, since these parts can be represented by triangular waveforms [1], and other will be responsible for calculating the samples of P, T and U portions, since these parts can be represented by sinusoidal waveforms [1]. Fig. 4 shows the algorithm for the calculation of the QRS, Q and S portions, and Fig. 5 shows the algorithm for the calculation of the P, T and U portions.

The main procedure is responsible to order the necessary parameters for the calculation of wave portions in auxiliary procedures and merge the same calculated portions into a single wave, the ECG signal resulting. Samples of the resulting signal are written to a text file, to be opened in any software that generates graphics, such as MATLAB® and Excel®.

The software used for the development of this layer was the Nios® II Embedded Design Suite, and the language C was used in the implementation. This IDE carries the developed software for the SDRAM memory of the Altera's development kit, to be executed by Nios® II Core.

5. RESULTS

Table 1. Input data used in tests.

Heart beat	72
Amplitude - P wave	25 mV
Amplitude - R wave	1.60 mV
Amplitude - Q wave	0.025 mV
Amplitude - T wave	0.35 mV
Duration - P-R interval	0.16s
Duration - S-T interval	0.18s
Duration - P interval	0.09s
Duration - QRS interval	0.11s

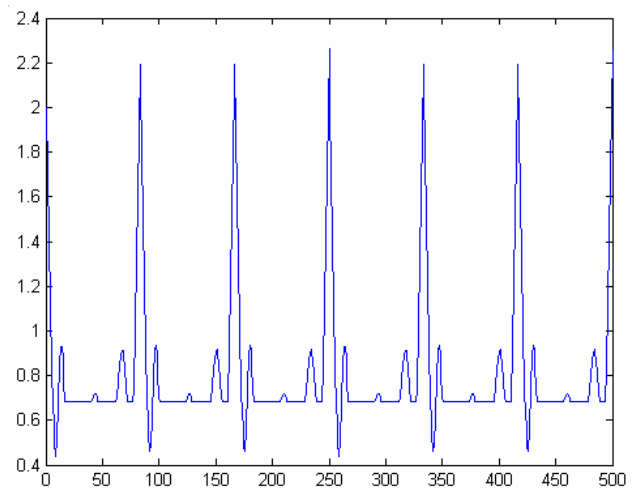


Fig. 6. Obtained ECG signal after tests.

Table 1 shows the input data used in tests for the generation of an ECG signal by the embedded system developed. These values are used with default values by the simulator. Other values can be specified for the generation of ECG signals with distinct features. Fig. 5 shows the resulting ECG signal.

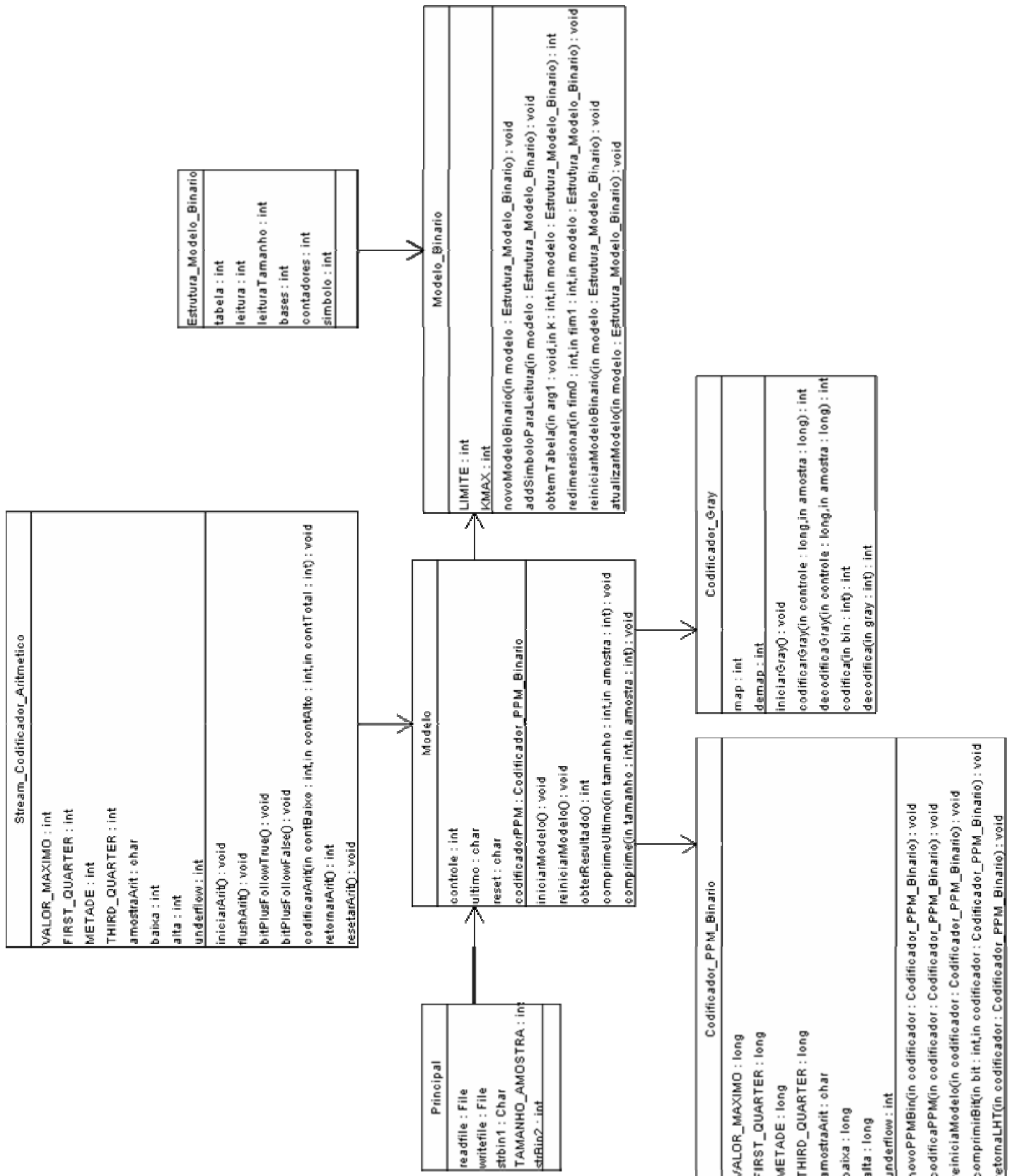
7. CONCLUSION

The results obtained show that the developed embedded system had success in simulate an ECG signal through the Fourier series and the Karthik's algorithm [1]. This simulator can simulate any given ECG waveform without using the ECG machine, removing the difficulties of taking real ECG signals with invasive and noninvasive methods.

7. REFERENCES

- [1] R. Karthik. (2009, Aug 13). ECG Simulation Using MATLAB – Principle of Fourier Series. [Online]. Available: <http://www.mathworks.com/matlabcentral/fileexchange/10858>
- [2] E-Geo. (2009, Aug 13). *Léxico de Termos Hidrogeológicos*. [Online]. Available: <http://e-geo.ineti.pt>
- [3] Altera Corporation. (2009, Aug 13). *Nios Development Board Reference Manual, Stratix Edition*. [Online]. Available: http://www.altera.com/literature/manual/mnl_nios2_board_stratix_1s10.pdf

APÊNDICE C - Diagrama UML de Classes do Módulo de Compressão



APÊNDICE D - Implementação em Linguagem C do Sistema Embarcado

Código Principal (principal.c)

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "includes.h"
#include "Model.h"
#include "alt_types.h"
#include "system.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"

/* Definição das pilhas de tarefas */
#define TASK_STACKSIZE_1 524288
#define TASK_STACKSIZE_2 524288
OS_STK task1_stk[TASK_STACKSIZE_1];
OS_STK task2_stk[TASK_STACKSIZE_2];

#define AMOSTRA_TAMANHO 12
#define TOTAL_AMOSTRAS 3600
float PI = 3.14159265;

/* Definição das prioridades de tarefas */

#define TASK1_PRIORITY 1
#define TASK2_PRIORITY 2

int cont, cont2, cont_buffer, j, cont_compress, k;
int buffer[TOTAL_AMOSTRAS];

int retorno = 0;
int porcentagem = 0;
int count = 0;

float a_pwav = 0.25;
float d_pwav = 0.09;
float t_pwav = 0.16;

float a_qwav = 0.025;
float d_qwav = 0.066;
float t_qwav = 0.166;

float a_qrswav = 1.6;
float d_qrswav = 0.11;

float a_swav = 0.25;
float d_swav = 0.066;
float t_swav = 0.09;

float a_twav = 0.35;
float d_twav = 0.142;
float t_twav = 0.2;
```



```

float a_uwav = 0.035;
float d_uwav = 0.0476;
float t_uwav = 0.433;

float taxa = 0.0;
float li = 0.416667;

float ecg[TOTAL_AMOSTRAS];
float aux = 0.00;

int d;
int i;

float pwav[TOTAL_AMOSTRAS];
float qwav[TOTAL_AMOSTRAS];
float qrswav[TOTAL_AMOSTRAS];
float swav[TOTAL_AMOSTRAS];
float twav[TOTAL_AMOSTRAS];
float uwav[TOTAL_AMOSTRAS];

float x[TOTAL_AMOSTRAS];

char bin[TOTAL_AMOSTRAS];

// Transforma de decimal para binario
char* intToBin(int d)
{
    int a=11;

    int i;

    for(i=0;i<=11;i++)
        bin[i]='0';

    do {
        if((d % 2) == 0)
            bin[a] = '0';
        else
            bin[a] = '1';
        a--;
        if(d==2)
            bin[a] = '1';
        d = d/2;
    } while(d>=1);

    return bin;
}

// Transforma de milivolt para inteiro
int mVtoADC(float mV)
{
    int valor;

    valor = (mV * (4095*0.153) + 2048)/2;

    return valor;
}

// Cálculo da onda P
void p_wav(float a_pwav, float d_pwav, float t_pwav, float li)
{
    float l = li;

```

```

float a = a_pwav;

int i,j,k;

float harm1[TOTAL_AMOSTRAS];
float p2[TOTAL_AMOSTRAS];
float pwav1[TOTAL_AMOSTRAS];
float xi[TOTAL_AMOSTRAS];

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    harm1[i] = 0.0;
    p2[i] = 0.0;
    pwav1[i] = 0.0;
    xi[i] = x[i];
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    xi[i] = xi[i] + t_pwav;
}

float b = (2*l)/d_pwav;

int n = 100;

float p1 = 1/l;

for(i = 1; i <= n; i++)
{
    for(j = 0; j < TOTAL_AMOSTRAS; j++)
    {
        harm1[j] = (((sin((PI/(2*b))*(b-(2*i))))/(b-(2*i)))+(sin((PI/(2*b))*(b+(2*i))))/(b+(2*i)))*(2/PI))*cos((i*PI*xi[j])/l);
    }

    for(k = 0; k < TOTAL_AMOSTRAS; k++)
    {
        p2[k] = p2[k] + harm1[k];
    }
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
    pwav1[i] = p1 + p2[i];

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    pwav[i] = a * pwav1[i];
}

// Cálculo da onda Q
void q_wav(float a_qwav, float d_qwav, float t_qwav, float li)
{
    float l = li;
    float a = a_qwav;

    int i,j,k;

    double harm5[TOTAL_AMOSTRAS];
    float q2[TOTAL_AMOSTRAS];

```

```

float xi[TOTAL_AMOSTRAS];

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    harm5[i] = 0.0;
    q2[i] = 0.0;
    xi[i] = x[i];
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    xi[i] = xi[i] + t_qwav;
}

float b = (2*l)/d_qwav;

int n = 100;

float q1 = (a/(2*b))*(2-b);

for(i = 1; i <= n; i++)
{
    for(j = 0; j < TOTAL_AMOSTRAS; j++)
    {
        harm5[j] = (((2*b*a)/((i*i)*(PI*PI)))*(1-
cos((i*PI)/b)))*cos((i*PI*xi[j])/l);
    }

    for(k = 0; k < TOTAL_AMOSTRAS; k++)
    {
        q2[k] = q2[k] + harm5[k];
    }
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    qwav[i] = (-1) * (q1 + q2[i]);
}
}

// Cálculo da onda QRS
void qrs_wav(float a_qrswav, float d_qrswav, float li)
{
    float l = li;
    float a = a_qrswav;

    int i,j,k;

    float harm[TOTAL_AMOSTRAS];
    float qrs2[TOTAL_AMOSTRAS];
    float xi[TOTAL_AMOSTRAS];

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        harm[i] = 0.0;
        qrs2[i] = 0.0;
        xi[i] = x[i];
    }

    float b = (2*l)/d_qrswav;

```

```

int n = 100;

float qrs1 = (a/(2*b))*(2-b);

for(i = 1; i <= n; i++)
{
    for(j = 0; j < TOTAL_AMOSTRAS; j++)
    {
        harm[j] = (((2*b*a)/((i*i)*(PI*PI)))*(1-
cos((i*PI)/b)))*cos((i*PI*xi[j])/l);
    }

    for(k = 0; k < TOTAL_AMOSTRAS; k++)
    {
        qrs2[k] = qrs2[k] + harm[k];
    }
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    qrswav[i] = qrs1 + qrs2[i];
}
}

// Cálculo da onda S
void s_wav(float a_swav, float d_swav, float t_swav, float li)
{
    float l = li;
    float a = a_swav;

    int i,j,k;

    float harm3[TOTAL_AMOSTRAS];
    float s2[TOTAL_AMOSTRAS];
    float xi[TOTAL_AMOSTRAS];

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        harm3[i] = 0.0;
        s2[i] = 0.0;
        xi[i] = x[i];
    }

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        xi[i] = xi[i] - t_swav;
    }

    float b = (2*l)/d_swav;

    int n = 100;

    float s1 = (a/(2*b))*(2-b);

    for(i = 1; i <= n; i++)
    {
        for(j = 0; j < TOTAL_AMOSTRAS; j++)
        {
            harm3[j] = (((2*b*a)/((i*i)*(PI*PI)))*(1-
cos((i*PI)/b)))*cos((i*PI*xi[j])/l);
        }
    }
}

```

```

        for(k = 0; k < TOTAL_AMOSTRAS; k++)
        {
            s2[k] = s2[k] + harm3[k];
        }
    }

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        swav[i] = -1 * (s1 + s2[i]);
    }
}

// Cálculo da onda T
void t_wav(float a_twav, float d_twav, float t_twav, float li)
{
    float l = li;
    float a = a_twav;

    int i,j,k;

    float harm2[TOTAL_AMOSTRAS];
    float t2[TOTAL_AMOSTRAS];
    float twav1[TOTAL_AMOSTRAS];
    float xi[TOTAL_AMOSTRAS];

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        harm2[i] = 0.0;
        t2[i] = 0.0;
        twav1[i] = 0.0;
        xi[i] = x[i];
    }

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        xi[i] = xi[i] - t_twav - 0.045;
    }

    float b = (2*l)/d_twav;

    int n = 100;

    float t1 = 1/l;

    for(i = 1; i <= n; i++)
    {
        for(j = 0; j < TOTAL_AMOSTRAS; j++)
        {
            harm2[j] = (((sin((PI/(2*b))*(b-(2*i))))/(b-
(2*i)))+(sin((PI/(2*b))*(b+(2*i))))/(b+(2*i)))*(2/PI))*cos((i*PI*xi[j])/l);
        }

        for(k = 0; k < TOTAL_AMOSTRAS; k++)
        {
            t2[k] = t2[k] + harm2[k];
        }
    }

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
        twav1[i] = t1 + t2[i];
}

```

```

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        twav[i] = a * twavl[i];
    }
}

// Cálculo da onda U
void u_wav(float a_uwav, float d_uwav, float t_uwav, float li)
{
    float l = li;
    float a = a_uwav;

    int i,j,k;

    float harm4[TOTAL_AMOSTRAS];
    float u2[TOTAL_AMOSTRAS];
    float uwavl[TOTAL_AMOSTRAS];
    float xi[TOTAL_AMOSTRAS];

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        harm4[i] = 0.0;
        u2[i] = 0.0;
        uwav[i] = 0.0;
        xi[i] = x[i];
    }

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        xi[i] = xi[i] - t_uwav;
    }

    float b = (2*l)/d_uwav;

    int n = 100;

    float u1 = 1/l;

    for(i = 1; i <= n; i++)
    {
        for(j = 0; j < TOTAL_AMOSTRAS; j++)
        {
            harm4[j] = (((sin((PI/(2*b))*(b-(2*i))))/(b-(2*i)))+(sin((PI/(2*b))*(b+(2*i))))/(b+(2*i)))*(2/PI))*cos((i*PI*xi[j])/l);
        }

        for(k = 0; k < TOTAL_AMOSTRAS; k++)
        {
            u2[k] = u2[k] + harm4[k];
        }
    }

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
        uwavl[i] = u1 + u2[i];

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        uwav[i] = a * uwavl[i];
    }
}

```

```

// Controle para os displays de 7 segmentos, integrantes do kit
static void sevenseg_set_hex(int hex)
{
    static alt_u8 segments[16] = {
        0x81, 0xCF, 0x92, 0x86, 0xCC, 0xA4, 0xA0, 0x8F, 0x80, 0x84,
        0x88, 0xE0, 0xF2, 0xC2, 0xB0, 0xB8 };

    unsigned int data = segments[hex & 15] | (segments[(hex >> 4) & 15] <<
8);

    IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE, data);
}

// Thread 1 - Simula a captação de amostra de sinais ECG, dada uma
// frequência de amostragem
void task1(void* pdata)
{
    INT8U err;
    int i;
    int k=0;

    while(1)
    {
        OSTimeDlyHMSM(0, 0, 1, 0);

        if(cont_buffer < TOTAL_AMOSTRAS)
        {
            for(i = 0; i < 180; i++)
            {
                buffer[cont_buffer] = mVtoADC(ecg[cont_buffer]);
                buffer[cont_buffer+1] = mVtoADC(ecg[cont_buffer+1]);
                cont_buffer++;cont_buffer++;
            }
        }
        else
        {
            sevenseg_set_hex(170);
            err = OSTaskSuspend(1);
        }
    }
}

// Thread 2 - Comprime as amostras geradas pela Thread 1
void task2(void* pdata)
{
    INT8U err;

    FILE *writefile;
    writefile = fopen ("/mnt/host/comprimido.txt", "w+");

    if (writefile == NULL) {
        printf ("Erro ao abrir arquivo..\n");
    }

    init();

    while (1)
    {

```

```

        OSTimeDlyHMSM(0, 0, 1, 0);

        if ((k+2) > TOTAL_AMOSTRAS) {
            compress (AMOSTRA_TAMANHO, strBin2Int (intToBin(buffer[k]),
AMOSTRA_TAMANHO));

            retorno = getResult();
            while (retorno != -1) {
                retorno = retorno & 255;
                putc(retorno,writefile);
                retorno = getResult();
            }

            compressLast (AMOSTRA_TAMANHO, strBin2Int
(intToBin(buffer[k+1]), AMOSTRA_TAMANHO));
            //printf("%d\n",buffer[k+1]);

            retorno = getResult();
            while (retorno != -1) {
                retorno = retorno & 255;
                putc(retorno,writefile);
                retorno = getResult();
            }

            sevenseg_set_hex(2);

            fclose(writefile);

            err = OSTaskSuspend(2);

        } else {
            compress (AMOSTRA_TAMANHO, strBin2Int (intToBin(buffer[k]),
AMOSTRA_TAMANHO));

            retorno = getResult();
            while (retorno != -1) {
                retorno = retorno & 255;
                putc(retorno,writefile);
                retorno = getResult();
            }

            compress (AMOSTRA_TAMANHO, strBin2Int (intToBin(buffer[k+1]),
AMOSTRA_TAMANHO));

            retorno = getResult();
            while (retorno != -1) {
                retorno = retorno & 255;
                putc(retorno,writefile);
                retorno = getResult();
            }
        }

        k++;k++;
    }
}

```



```

/* Recebe os valores definidos pelo usuário para a geração de amostras de
sinais ECG, calcula o sinal ECG resultante e inicia o processo de simulação
e compressão em paralelo */
int main(void)
{
    int i, escolha;

    sevenseg_set_hex(0);

    cont = 0;
    cont2 = 0;
    cont_buffer = 0;
    j = 0;
    k = 0;

    for(i = 0; i < TOTAL_AMOSTRAS; i++)
    {
        ecg[i] = 0.0;
        pwav[i] = 0.0;
        qwav[i] = 0.0;
        qrswav[i] = 0.0;
        swav[i] = 0.0;
        twav[i] = 0.0;
        uwav[i] = 0.0;
    }

    sevenseg_set_hex(1);

    printf("Digite 1 para os valores padrão de ECG ou pressione 2: ");
    scanf("%d",&escolha);

    if(escolha == 2)
    {
        printf("Digite a taxa de batimentos cardíacos: ");
        scanf("%f", &taxa);

        li = 30/taxa;

        // Especificações da onda P
        printf("Especificações da onda P - ");
        printf("Digite 1 para especificações padrão ou digite 2: ");
        scanf("%d", &d);

        if(d == 1)
        {
            a_pwav=0.25;
            d_pwav=0.09;
            t_pwav=0.16;
        }
        else
        {
            printf("Amplitude: ");
            scanf("%f", &a_pwav);
            printf("Duração: ");
            scanf("%f", &d_pwav);
            printf("Intervalo P-R: ");
            scanf("%f", &t_pwav);
            d=0;
        }

        // Especificações da onda Q

```

```
printf("Especificações da onda Q - ");
printf("Digite 1 para especificações padrão ou digite 2: ");
scanf("%d", &d);

if(d == 1)
{
    a_qwav=0.025;
    d_qwav=0.066;
    t_qwav=0.166;
}
else
{
    printf("Amplitude: ");
    scanf("%f", &a_qwav);
    printf("Duração: ");
    scanf("%f", &d_qwav);
    t_qwav = 0.166;
    d=0;
}

// Especificações da onda QRS
printf("Especificações da onda QRS - ");
printf("Digite 1 para especificações padrão ou digite 2: ");
scanf("%d", &d);

if(d == 1)
{
    a_qrswav=1.6;
    d_qrswav=0.11;
}
else
{
    printf("Amplitude: ");
    scanf("%f", &a_qrswav);
    printf("Duração: ");
    scanf("%f", &d_qrswav);
    d=0;
}

// Especificações da onda S
printf("Especificações da onda S - ");
printf("Digite 1 para especificações padrão ou digite 2: ");
scanf("%d", &d);

if(d == 1)
{
    a_qwav=0.25;
    d_qwav=0.066;
    t_qwav=0.09;
}
else
{
    printf("Amplitude: ");
    scanf("%f", &a_swav);
    printf("Duração: ");
    scanf("%f", &d_swav);
    t_qwav = 0.09;
    d=0;
}

// Especificações da onda T
```

```

printf("Especificações da onda T - ");
printf("Digite 1 para especificações padrão ou digite 2: ");
scanf("%d", &d);

if(d == 1)
{
    a_twav=0.35;
    d_twav=0.142;
    t_twav=0.2;
}
else
{
    printf("Amplitude: ");
    scanf("%f", &a_qwav);
    printf("Duração: ");
    scanf("%f", &d_qwav);
    printf("Intervalo S-T: ");
    scanf("%f", &t_twav);
    d=0;
}

// Especificações da onda U
printf("Especificações da onda U - ");
printf("Digite 1 para especificações padrão ou digite 2: ");
scanf("%d", &d);

if(d == 1)
{
    a_uwav=0.035;
    d_uwav=0.0476;
    t_uwav=0.433;
}
else
{
    printf("Amplitude: ");
    scanf("%f", &a_uwav);
    printf("Duração: ");
    scanf("%f", &d_uwav);
    t_uwav=0.433;
    d=0;
}
}

for(i=0;i<TOTAL_AMOSTRAS;i++)
{
    aux = aux + 0.01;
    x[i] = aux;
}

sevensseg_set_hex(2);

p_wav(a_pwav,d_pwav,t_pwav,li);
sevensseg_set_hex(3);
q_wav(a_qwav,d_qwav,t_qwav,li);
sevensseg_set_hex(4);
qrs_wav(a_qrswav,d_qrswav,li);
sevensseg_set_hex(5);
s_wav(a_swav,d_swav,t_swav,li);
sevensseg_set_hex(6);
t_wav(a_swav,d_swav,t_swav,li);
sevensseg_set_hex(7);

```

```

u_wav(a_uwav,d_uwav,t_uwav,li);
sevensseg_set_hex(8);

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    ecg[i] = pwav[i] + qwav[i] + qrswav[i] + swav[i] + twav[i] +
uwav[i];
}

sevensseg_set_hex(9);

printf("%d - Amostras geradas com sucesso\n\n", TOTAL_AMOSTRAS);

FILE *original;
original = fopen ("/mnt/host/original.txt", "w+");

if (original == NULL) {
    printf ("Erro ao abrir arquivo..\n");
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    fprintf (original, "%f ", ecg[i]);
}

fclose(original);

FILE *originalint;
originalint = fopen ("/mnt/host/originalint.txt", "w+");

if (originalint == NULL) {
    printf ("Erro ao abrir arquivo..\n");
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    fprintf (originalint, "%d ", mVtoADC(ecg[i]));
}

fclose(originalint);

FILE *originalbin;
originalbin = fopen ("/mnt/host/originalbin.txt", "w+");

if (originalbin == NULL) {
    printf ("Erro ao abrir arquivo..\n");
}

for(i = 0; i < TOTAL_AMOSTRAS; i++)
{
    fprintf (originalbin, "%s ", intToBin(mVtoADC(ecg[i])));
}

fclose(originalbin);

sevensseg_set_hex(10);

OSTaskCreateExt(task1,
                NULL,
                (void *)&task1_stk[TASK_STACKSIZE_1],

```

```
        TASK1_PRIORITY,  
        TASK1_PRIORITY,  
        task1_stk,  
        TASK_STACKSIZE_1,  
        NULL,  
        0);  
  
    OSTaskCreateExt(task2,  
        NULL,  
        (void *)&task2_stk[TASK_STACKSIZE_2],  
        TASK2_PRIORITY,  
        TASK2_PRIORITY,  
        task2_stk,  
        TASK_STACKSIZE_2,  
        NULL,  
        0);  
  
    printf("Processo de simulação de frequência de amostragem e compressão  
iniciado\n\n");  
  
    OSStart();  
  
    return 0;  
}
```

Model.h

```
#include "ArithEncoderStream.h"  
#include "BinaryModel.h"  
#include "PPMBinEncoder.h"  
#include "GrayEncoder.h"  
  
#ifndef _MODEL  
    #define _MODEL  
  
    extern char last, reset;  
  
    extern tPPMBinEncoder *ppmEncoder;  
  
    extern void init ();  
    extern void compress (int size, int sample);  
    extern void compressLast (int size, int sample);  
    extern int getResult ();  
    extern void reset ();  
#endif
```

Model.c

```
#include <stdio.h>  
#include <stdlib.h>  
#include "Model.h"  
  
int control;  
  
void compress (int size, int sample) {
```

```

    int low, high, total;
    unsigned long lht;
    int i;

    if (reset || last) {
        control = size & 255;
        reset = last = 0;
        _resetArith();

        free (ppmEncoder);

        ppmEncoder = malloc (control * sizeof(tppmbinEncoder));

        _initArith();
        for (i = 0; i < control; i++)
            _NewPPMBin (&ppmEncoder[i]);
    }

    sample = encodeGray ( (long)control, (long)sample);

    for (i = 0; i < control; i++) {
        _CompressBit ((sample >> i) & 1, &ppmEncoder[i]);

        lht = _ReturnLHT (&ppmEncoder[i]);

        total = lht & 511;
        high = (lht >> 9) & 511;
        low = (lht >> 18) & 255;

        _encodeArith (low, high, total);
    }
}

void compressLast (int size, int sample) {
    int low, high, total;
    unsigned long lht;
    int i;

    if (reset || last) {
        control = size & 255;
        reset = last = 0;
        _resetArith();

        free (ppmEncoder);

        ppmEncoder = malloc (control * sizeof(tppmbinEncoder));

        _initArith();
        for (i = 0; i < control; i++)
            _NewPPMBin (&ppmEncoder[i]);
    }

    sample = encodeGray ( (long)control, (long)sample);

    for (i = 0; i < control; i++) {
        _CompressBit ((sample >> i) & 1, &ppmEncoder[i]);

        lht = _ReturnLHT (&ppmEncoder[i]);
    }
}

```

```

        total = lht & 511;
        high = (lht >> 9) & 511;
        low = (lht >> 18) & 255;

        _encodeArith (low, high, total);
    }
    last = 1;
    _flushArith();
}

int getResult () {
    return _returnArith();
}

void init () {
    reset = last = 1;
    initGray();
}

void reset () {
    int i;

    reset = 1;

    for(i = 0; i < control; i++) {
        _ResetModel (&ppmEncoder[i]);
    }

    _resetArith ();
}

```

ArithEncoderStream.h

```

#ifndef _ARITH_ENCODER
#define _ARITH_ENCODER
extern void _initArith ();
extern void _flushArith();
extern void _bitPlusFollowTrue();
extern void _bitPlusFollowFalse();
extern void _encodeArith(int lowCount, int highCount, int
totalCount);
extern int _returnArith();
extern void _resetArith();
#endif

```

ArithEncoderStream.c

```

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "ArithEncoderStream.h"
#include "utils.h"

unsigned long long TOP_VALUE;
unsigned long long FIRST_QUARTER;
unsigned long long HALF;
unsigned long long THIRD_QUARTER;

```

```

char ArithSample[100000];
unsigned long long low;
unsigned long long high;
int underflow;

void _initArith () {
    TOP_VALUE = 2147483647L;
    FIRST_QUARTER = 536870912L;
    HALF = 1073741824L;
    THIRD_QUARTER = 1610612736L;

    low = 0;
    high = TOP_VALUE;
    underflow = 0;
}

void _flushArith() {
    ++underflow;
    if (low < FIRST_QUARTER)
        _bitPlusFollowFalse();
    else
        _bitPlusFollowTrue();

    int rest = 8 - (strlen(ArithSample) % 8);

    rest = rest == 8 ? 0 : rest;

    int i;
    for(i = 0; i < rest; i++) {
        strcat(ArithSample, "0");
    }
}

void _bitPlusFollowTrue() {
    strcat(ArithSample, "1");
    while(underflow > 0) {
        strcat(ArithSample, "0");
        underflow--;
    }
}

void _bitPlusFollowFalse() {
    strcat(ArithSample, "0");
    while(underflow > 0) {
        strcat(ArithSample, "1");
        underflow--;
    }
}

void _encodeArith(int lowCount, int highCount, int totalCount) {
    unsigned long long range = high - low + 1;
    high = low + (range * highCount) / totalCount - 1;
    low = low + (range * lowCount) / totalCount;

    while(1) {
        if (high < HALF) {
            _bitPlusFollowFalse();
        }
        else if (low >= HALF) {
            _bitPlusFollowTrue();
        }
    }
}

```



```

        low -= HALF;
        high -= HALF;
    }
    else if (low >= FIRST_QUARTER && high < THIRD_QUARTER) {
        underflow++;
        low -= FIRST_QUARTER;
        high -= FIRST_QUARTER;
    }
    else {
        return;
    }

    low <= 1;
    high = (high << 1) + 1;
}

}

int _returnArith() {
    if(strlen(ArithSample) < 8) {
        return -1;
    }
    else {
        unsigned char retorno = strBin2Int(ArithSample,8);
        delete(ArithSample);
        return retorno;
    }
}

void _resetArith() {
    low = 0;
    high = TOP_VALUE;
    underflow = 0;
    ArithSample[0] = '\\0';
}

```

ArithEncoderStreamModel.h

```

#ifndef _MODELO_ARITH
#define _MODELO_ARITH

    char last, reset;

    extern void initArith ();
    extern void encodeArith(int low, int high, int total);
    extern void encodeLastArith(int low, int high, int total);
    extern void resetAirhtModel();
    extern int returnArith ();

#endif

```

ArithEncoderStreamModel.c

```

#include <stdio.h>
#include "ArithEncoderStreamModel.h"
#include "ArithEncoderStream.h"

void initArith () {
    _initArith();
}

```

```

void encodeArith(int low, int high, int total) {
    if(reset || last) {
        _resetArith();
        reset = last = 0;
    }

    _encodeArith(low, high, total);
}

void encodeLastArith(int low, int high, int total) {
    if(reset || last) {
        _resetArith();
        reset = last = 0;
    }
    _encodeArith(low, high, total);
    _flushArith();
    last = 1;
}

void resetAirhtModel() {
    reset = 1;

    _resetArith();
}

int returnArith () {
    return _returnArith();
}

```

BinaryModel.h

```

#ifndef _BINARY_MODEL
#define _BINARY_MODEL
#define LIMIT 255
#define KMAX 5

typedef struct {
    int table[3];
    int read;
    int readLength;
    int bases[KMAX + 1];
    unsigned short int counters[(1 << (KMAX + 2)) - 2]; //todos comecam
com 1
    int simbol; //0 ou 1
} tBinaryModel;

extern void _NewBinaryModel(tBinaryModel *model);
extern void _AddSimbToRead(tBinaryModel *model);
extern int* _GetTable(int k, tBinaryModel *model);
extern void _Resize(int end0, int end1, tBinaryModel *model);
extern void _ResetBinaryModel(tBinaryModel *model);
extern void _UpdateModel(tBinaryModel *model);

#endif

```

BinaryModel.c

```

#include <stdio.h>
#include <stdlib.h>

```

```

#include "BinaryModel.h"

void _NewBinaryModel (tBinaryModel *model) {
    int i;
    model->read = model->readLength = 0;

    for (i = 0; i <= KMAX; i++)
        model->bases[i] = (1 << (i+1)) - 2;

    for (i = 0; i < ((1 << (KMAX+2)) - 2); i++)
        model->counters[i] = 1;
}

void _AddSimbToRead(tBinaryModel *model) {
    model->read = (model->read << 1) | model->simbol;

    model->readLength++;

    if (model->readLength > KMAX) {
        model->readLength = KMAX;
    }
}

void _UpdateModel(tBinaryModel *model) {
    int k = KMAX;

    if(k < 0) return;

    if(k > model->readLength)
        k = model->readLength;

    int and = (1 << k) - 1;

    int level = model->read & and;

    int end0 = model->bases[k] + (level << 1);
    int end1 = end0+1;
    int endS = end0+model->simbol;

    model->counters[endS]++;

    _Resize(end0,end1,model);
}

int *_GetTable(int k, tBinaryModel *model) {
    if (k > model->readLength)
        return NULL;

    int and = (1 << k) - 1;

    int level = model->read & and;

    int end0 = model->bases[k] + (level << 1);
    int end1 = end0 + 1;

    int cont0 = model->counters[end0];
    int cont1 = model->counters[end1];

    model->table[0] = 0;
    model->table[1] = cont0;
    model->table[2] = cont0 + cont1;
}

```

```

        return model->table;
    }

    void _Resize(int end0, int end1, tBinaryModel *model) {
        if( (model->counters[end0] == LIMIT) || (model->counters[end1] ==
LIMIT) ) {
            if(model->counters[end0] > 1)
                model->counters[end0] >>= 1; //divide por 2 se > 1

            if(model->counters[end1] > 1)
                model->counters[end1] >>= 1; //divide por 2 se > 1
        }
    }

    void _ResetBinaryModel(tBinaryModel *model) {
        int i;
        for (i = 0; i < ((1 << (KMAX+2)) - 2); i++)
            model->counters[i] = 1;
    }
}

```

GrayEncoder.h

```

#ifndef _GRAY_H_
#define _GRAY_H_
unsigned int map[65536];
unsigned int demap[65536];

extern void initGray();
extern unsigned int encodeGray(long control, long sample);
extern unsigned int decodeGray(long control, long sample);
extern unsigned int encode(unsigned int bin);
extern unsigned int decode(unsigned int gray);
#endif

```

GrayEncoder.c

```

#include "GrayEncoder.h"

void initGray() {
    unsigned int i;
    for (i = 0; i < 65536; i++) {
        map[i] = encode(i);
        demap[i] = decode(i);
    }
}

unsigned int encodeGray(long control, long sample) {
    long cont = sample >> control;
    sample = sample - (cont << control);
    return map[sample];
}

unsigned int decodeGray(long control, long sample) {
    long cont = sample >> control;
    sample = sample - (cont << control);

    return demap[sample];
}

```

```

unsigned encode(unsigned int bin) {
    long bins[16];
    long i;
    for (i = 0; i < 16L; i++) {
        bins[i] = bin >> i;
    }

    long grays[16];

    for (i = 0L; i < 15L; i++) {
        grays[i] = bins[i] ^ bins[i + 1];
    }
    grays[15] = bins[15];
    long gray = 0;
    for (i = 15L; i >= 0L; i--) {
        gray = (gray << 1) | grays[i];
    }
    return (unsigned int) gray;
}

unsigned int decode(unsigned int gray) {
    long grays[16];
    long i;
    for (i = 0L; i < 16L; i++) {
        grays[i] = gray >> i;
    }
    long bins[16];
    bins[15] = grays[15];
    for (i = 14L; i >= 0L; i--) {
        bins[i] = grays[i] ^ bins[i + 1];
    }
    int bin = 0;
    for (i = 15L; i >= 0L; i--) {
        bin = (bin << 1) | bins[i];
    }
    return (unsigned int) bin;
}

```

PPMBinEncoder.h

```

#include "BinaryModel.h"

#ifndef _PPMBIN_ENCODER
#define _PPMBIN_ENCODER

typedef struct {
    int lht[3];
    tBinaryModel model;
} tPPMBinEncoder;

extern void _NewPPMBin (tPPMBinEncoder *encoder);
extern void _EncodePpm(tPPMBinEncoder *encoder);
extern void _ResetModel(tPPMBinEncoder *encoder);
extern void _CompressBit(int bit, tPPMBinEncoder *encoder);
extern int _ReturnLHT(tPPMBinEncoder *encoder);

#endif

```

PPMBinEncoder.c

```

#include <stdio.h>
#include <stdlib.h>
#include "BinaryModel.h"
#include "PPMBinEncoder.h"

void _NewPPMBin (tPPMBinEncoder *encoder) {
    _NewBinaryModel (&(encoder->model));
}

void _EncodePpm(tPPMBinEncoder *encoder) {
    int k = KMAX;

    while(1) {
        if(k > encoder->model.readLength) {
            k = encoder->model.readLength;
        }

        if(k < 0) {
            int simb = encoder->model.simbol;
            encoder->lht[0] = simb;
            encoder->lht[1] = simb+1;
            encoder->lht[2] = 2;
        }
        else {
            int *tab = _GetTable(k, &(encoder->model));

            if(tab == NULL) {
                k--;
                continue;
            }
            else {
                int simb = encoder->model.simbol;
                encoder->lht[0] = tab[simb];
                encoder->lht[1] = tab[simb+1];
                encoder->lht[2] = tab[2];
            }
        }
        break;
    }

    _UpdateModel(&(encoder->model));
    _AddSimbToRead(&(encoder->model));
}

void _CompressBit(int bit, tPPMBinEncoder *encoder) {
    encoder->model.simbol = bit & 1;
    _EncodePpm(encoder);
}

int _ReturnLHT(tPPMBinEncoder *encoder) {
    int output = 0;

    output = (output << 8) | encoder->lht[0];
    output = (output << 9) | encoder->lht[1];
    output = (output << 9) | encoder->lht[2];

    return output;
}

void _ResetModel(tPPMBinEncoder *encoder) {
    _ResetBinaryModel(&(encoder->model));
}

```

```
}
```

PPMModel.h

```
#include "PPMBinEncoder.h"
#include "BinaryModel.h"

#ifndef _MODELO_PPM
#define _MODELO_PPM

    int gSample, gControl, gCont;
    char gLast, gReset;

    tPPMBinEncoder *ppmEncoder;

    extern void InitPPM ();
    extern void EncodePPM (int size, int data);
    extern void EncodeLastPPM (int size, int data);
    extern int ReturnLHT ();
    extern void ResetPPM ();
    extern void FinilizePPM ();

#endif
```

PPMModel.c

```
#include <stdio.h>
#include <stdlib.h>
#include "PPMModel.h"

void InitPPM () {
    gReset = gLast = 1;
    gCont = 0;
}

void EncodePPM (int size, int data) {
    int i, bit;
    gSample = data;

    if(gReset || gLast) {
        gControl = size & 15; //garante que controle possua apenas 4
bits;

        gReset = gLast = 0;
        free (ppmEncoder);
        gCont = 0;

        ppmEncoder = malloc (gControl * sizeof(tPPMBinEncoder));

        for (i = 0; i < gControl; i++) {
            _NewPPMBin(&ppmEncoder[i]);
        }

        for (bit = 0; bit < gControl; bit++)
        {
            _CompressBit((gSample >> bit) & 1, &ppmEncoder[bit]);
        }
    }
}
```

```
void EncodeLastPPM (int size, int data) {
    int i, bit;
    gSample = data;

    if(gReset || gLast) {
        gControl = size & 15; //garante que controle possua apenas 4
bits;
        gReset = gLast = 0;
        free (ppmEncoder);
        gCont = 0;

        ppmEncoder = malloc (gControl * sizeof(tPPMBinEncoder));

        for (i = 0; i < gControl; i++) {
            _NewPPMBin(&ppmEncoder[i]);
        }

        for (bit = 0; bit < gControl; bit++)
        {
            _CompressBit((gSample >> bit) & 1, &ppmEncoder[bit]);
        }

        gLast = 1;
    }

    void ResetPPM() {
        int i;
        gReset = 1;

        for(i = 0; i < gControl; i++) {
            _ResetModel(&ppmEncoder[i]);
        }
    }

    void FinilizePPM () {
        free (ppmEncoder);
    }

    int ReturnLHT () {
        int i = gCont;
        gCont = (gCont + 1)%gControl;
        return _ReturnLHT (&ppmEncoder[i]);
    }
}
```