Dissertação de Mestrado

Sistema Embarcado Reconfigurável para Aquisição de Sinais de Eletrocardiograma

Marcel Seiji Kay

Orientação:Prof. Dr. Fábio Faione

Área de Concentração: Sistemas de Computação

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
22 de Agosto de 2014.

Pesquisa desenvolvida com auxílio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) através de bolsa de mestrado.

Sistema Embarcado Reconfigurável para
Aquisição de Sinais de
Eletrocardiograma

Banca Examinadora:

Orientador: Prof. Dr. Fábio Faione

Prof. Dr. Ivens Gervasio Sene Junior

Prof. Dr. Ricardo Ribeiro dos Santos

Campo Grande, 22 de Agosto de 2014.

Agradecimentos

Agradeço a Deus por tudo o que ele tem me proporcionado e, em especial, por ter colocado em minha vida pessoas maravilhosas. Obrigado pai, mãe e irmã, sou muito grato por ter vocês em minha família, pelo apoio incondicional e por serem a minha base. Ao meu orientador, Prof. Fábio Faione, principalmente por ter me ajudado no início do mestrado, pela confiança, pela paciência e pelos ensinamentos transmitidos. À minha namorada Camila, obrigado pelos incentivos nos momentos mais difíceis, você é uma pessoa grandiosa e tenho orgulho de tê-la como namorada. Ao meu "tio-padrinho" Takeshi, que sempre me incentivou e que fico muito contente de tê-lo perto, mesmo estando longe. Ao meu amigo de longa data Ted, te considero muito desde a época da catequese. À equipe Gracie Barra Pantanal, obrigado pela higiene mental e pelos treinos. Por fim, muito obrigado a todos que direta ou indiretamente me ajudaram.

Conteúdo

Lista de Abreviaturas

***Dalvik* VM** *Dalvik Virtual Machine*. 29

m-Health *Mobile Health*. 1, 4, 5

ACL *Asynchronous Connection-Less*. 25

ADC *Analog-to-Digital Converter*. x, 2, 5, 12, 13, 16, 20, 21, 23, 24, 32, 38–40, 44, 71

AFH *Adaptive Frequency Hopping*. 24

AHB *Anadigm Hex File*. 19

API *Application Programming Interface*. 29, 71, 72

CAB *Configurable Analog Block*. xi, 14–17, 56

CAM *Configurable Analog Modules*. x, xi, xiii, 17–19, 33–35, 42, 54, 57, 71

CMRR Taxa de Rejeição em Modo Comum. xi, xiv, 51, 70, 73

CPU Unidade Central de Processamento. 22

CRC *Cyclic Redundancy Check*. 25, 35, 36, 39

DAC *Digital-to-Analog Converter*. 20

dpASP *dynamically programmed Analog Signal Processors*. 15

E/S Entrada/Saída. 16, 21, 38, 39

ECG Eletrocardiograma. v, ix–xi, xiii, xiv, 2–6, 8, 10–12, 29–31, 33–35, 39, 40, 42–45, 53, 55, 56, 68, 70–73, 79

EDR *Enhanced Data Rate*. 25

EEG Eletroencefalograma. 5

EEPROM *Electrically Erasable Programmable Read-Only Memory*. 17

EMG Eletromiograma. 5

FPAA *Field Programmable Analog Array*. v, ix–xi, xiii, xiv, 2–6, 14–19, 29–36, 39–45, 48, 49, 53–57, 62–66, 69–73

FPGA *Field Programmable Gate Array*. 2, 5, 6, 14, 53

FPS Frames por Segundo. 72

GPL *General Public License*. 35

GPS *Global Positioning System*. 4

GPU *Graphics Processing Unit*. 71

IA Inteligência Artificial. 73

IBM-PC *IBM Personal Computer*. ix, 19, 22, 35, 72

IDE *Integrated Development Environment*. 19, 40

ISM *Industrial, Scientific, Medical*. xiii, 1, 25

NCADpv Leitura do Conversor Analógico-Digital. xiii, 47, 52, 79–82

NSB *Least Significant Bit*. 22

NUT *Look-up-table*. 14, 16, 17

MCU *Microcontroller Unit*. ix, xiii, 17, 19–21, 23, 29, 31, 35, 69

NFC *Near Field Communication*. 1

OHA *Open Handset Alliance*. 27

ONU Organização das Nações Unidas. 2

op-amps Amplificadores Operacionais. 12, 14, 16, 21, 56

OpenGL *Open Graphics Library*. 71

OpenGL ES *OpenGL for Embedded Systems*. 71

PGA *Programmable-gain amplifier*. 20

PID *Proportional-Integral-Derivative*. 18, 73

RF Radiofrequência. 1, 24

RFID *Radio-frequency Identification*. 1

SAR *Successive Approximation Register*. 16, 23, 56

SCO *Synchronous Connection-Oriented*. 25

SDCC *Small Device C Compiler*. 35

SDK *Software Development Kit*. 40

SERAS-ECG Sistema Embarcado Reconfigurável para Aquisição de Sinais de Eletrocardiograma. x, xiii, 3, 30, 33, 36, 41, 44, 45, 51, 69, 70, 73

SFR *Special Function Register*. 23

SIG *Special Interest Group*. 23

SMS *Short Message Service*. 4

SO Sistema Operacional. 1, 3, 27, 73

SPI *Serial Peripheral Interface*. 17, 21, 30

SPP *Serial Port Protocol*. 26

SRAM *Static Random Access Memory*. 16, 17, 39

SYSCLK *System Clock*. 22

UART *Universal Asynchronous Receiver/Transmitter*. ix, 22, 26, 32, 38, 39

VMR *Voltage Main Reference*. 16, 17

WHO *World Health Organization*. 2

Lista de Figuras

Lista de Tabelas

Resumo

Os *Smartphones* apresentam atualmente recursos que possibilitam o desenvolvimento de sistemas móveis para utilização na área médica, originando uma área chamada *m-Health* (*mobile-health*). Um dos exames médicos mais comuns é o eletrocardiograma (ECG), o qual é efetuado de forma não invasiva e permite o diagnóstico de diferentes doenças cardíacas, possibilitando um tratamento prévio para evitar problemas mais graves. O objetivo deste trabalho foi desenvolver um sistema embarcado reconfigurável para aquisição de sinais de ECG e um aplicativo para visualização e armazenamento destes sinais obtidos em um *smartphone* com Android. O sistema embarcado é constituído por uma FPAA (*Field Programmable Analog Array*) na qual implementa-se, de forma reconfigurável, todo circuito eletrônico necessário para o condicionamento do sinal de ECG. O sinal amplificado e filtrado pela FPAA é digitalizado por um microcontrolador que transmite o sinal para o *smartphone* através de um módulo *Bluetooth*. Além da visualização e armazenamento dos sinais, o aplicativo possibilita a reconfiguração parcial do circuito de ECG (ganho ajustável) ou total (alteração total do circuito de condicionamento). Os parâmetros medidos nos testes do circuito de ECG (resposta em frequência, imunidade ao potencial de meia célula, CMRR, impedância de entrada e consumo) e a qualidade dos sinais de ECG registrados em um indivíduo foram satisfatórios. Pôde-se verificar a flexibilidade proporcionada pela FPAA através da reconfiguração total, onde as respostas em frequência medidas nos circuitos genéricos, corresponderam com fidelidade às respostas teóricas desejadas. Através dessa dissertação, conclui-se que a FPAA é um dispositivo recomendado para uso em sistemas de aquisição de sinais bioelétricos, assim como outros, principalmente pela facilidade da reconfiguração. Espera-se que o sistema desenvolvido sirva como base para o desenvolvimento de trabalhos futuros na área de *m-Health* e também no desenvolvimento de interfaces homem-máquina usando sinais bioelétricos.

Palavras-chave: M-health, Smartphone, Eletrocardiograma, Sistemas Embarcados, Field Programmable Analog Array, Microcontrolador.

Abstract

Smartphones today have features that enable the development of mobile systems for use in the medical field, creating an area called m-Health (mobile health). One of the most common medical tests is the electrocardiogram (ECG), which is non-invasive and allows the diagnosis of various heart diseases, enabling prior treatment to prevent more serious problems. The objective of this work was to develop a reconfigurable embedded system for acquiring ECG signals and an Android application for viewing and storing these signals in a Smartphone. The embedded system comprises an FPAA (Field Programmable Analog Array) in which it implements, at reconfigurable manner, all necessary electronic circuitry for conditioning the ECG signal. The signal amplified and filtered by the FPAA is digitalized by a microcontroller which transmits the signal to the smartphone through a Bluetooth module. Besides signal display and storage, the application enables partial reconfiguration of the ECG circuit (adjustable gain) or full reconfiguration (total change of the conditioning circuit). The parameters measured in the tests of ECG circuit (frequency response, half-cell potential immune, CMRR, input impedance and consumption) and the quality of ECG signals recorded from one patient were satisfactory. It can also be noted the flexibility offered by the FPAA through full reconfiguration, where the frequency responses measured in generic circuits, corresponded faithfully the theoretical responses. Through this dissertation, it is concluded that the FPAA is recommended for use in bioelectric signals acquisition system, as well as others, particularly the ease of reconfiguration. It is expected that the system will serve as a basis for the development of future work in the field of m-Health, and also in the development of human-machine interfaces using bioelectrical signals.

Keywords: M-health, Smartphone, Electrocardiography, Embedded systems, Field Programmable Analog Array, Microcontroller.

Capítulo 1

Introdução

A popularização dos telefones móveis (celulares) tornou-se um grande incentivo para diversos fabricantes aumentarem suas participações neste lucrativo mercado. No ano de 2012, as vendas mundiais totalizaram 59,5 milhões de celulares, sendo deste total, 16 milhões de *Smartphones* [11]. Estes números incentivam os diversos fabricantes deste segmento a criarem inovações quanto a capacidade de processamento, armazenamento, visualização e comunicação. A presença de um Sistema Operacional (SO) nestes dispositivos móveis proporciona a sensação de se estar usando um computador, existindo no mercado diversos SOs, tais como: Android, iOS, Bada, *BlackBerry* OS, *Windows Phone*, entre outros. O objetivo destes aparelhos é englobar diversas funcionalidades em um único dispositivo, tais como: câmera fotográfica, reprodutor de músicas e vídeos, navegador GPS (*Global Positioning System*), computador portátil, acelerômetro, barômetro e leitor de *tags Radio-frequency Identification* (RFID) (através do *Near Field Communication* (NFC)).

Uma tecnologia de comunicação amplamente presente nestes aparelhos é o *Bluetooth*. A tecnologia *Bluetooth* - baseada em ondas de Radiofrequência (RF) - surgiu para promover uma solução de comunicação sem fio de curto alcance, operando na faixa não licenciada ISM de 2,4 GHz. Devido a alta compatibilidade entre os dispositivos, e ao baixo custo e consumo de energia, várias aplicações adotaram o *Bluetooth* nos mais diferentes cenários, por exemplo: biotelemetria (monitoramento sem fio das funções vitais do corpo humano) [12], localização de robôs em ambientes fechados [13] e medidor de energia elétrica consumida em residências [14].

Estes recursos dos *Smartphones* - aliado ao aumento da expectativa de vida da população, que requer maiores cuidados de saúde - possibilitaram o desenvolvimento de sistemas móveis para utilização na área médica, originando uma área chamada *Mobile Health (m-Health)*. Existem diversos trabalhos nessa área, tais como: uso de um *smartphone* com um adaptador acoplado para exames oftalmológicos, englobando miopia, hipermetropia, astigmatismo e catarata [15, 16], desenvolvimento de um oxímetro de dedo de baixo custo (utilizado para medir a quantidade de oxigênio presente no sangue) [17], utilização de sensores para o monitoramento e visualização da pulsação cardíaca no celular [18], estetoscópio fazendo o uso do microfone do *smartphone* [19], ultrassonografia (ecografia), além de um microscópio e um espectrômetro que utilizam a câmera embutida do dispositivo móvel [20].

Um dos exames médicos mais realizados é o Eletrocardiograma (ECG), o qual é efetuado de forma não invasiva e permite o diagnóstico de diferentes doenças cardíacas, possibilitando um tratamento prévio para evitar problemas mais graves. Através de eletrodos fixados no indivíduo, os sinais bioelétricos gerados pela atividade eletroquímica das células cardíacas são captados, permitindo ao médico a análise posterior de certas informações do coração do indivíduo. Os sistemas eletrônicos que realizam exames de ECG consistem basicamente de um estágio analógico (amplificação e filtragem do sinal) e um estágio digital (digitalização do sinal) [3].

Muitos sistemas eletrônicos digitais necessitam que circuitos analógicos sejam utilizados para realizar a interface do sistema com o mundo real. Nessa área, no final da década de 90, surgiu a *Field Programmable Analog Array* (FPAA). A FPAA é um dispositivo empregado na implementação de sistemas analógicos, composto por diversos circuitos analógicos configuráveis - filtros, multiplexadores, amplificadores operacionais, entre outros -, e que proporciona os seguintes benefícios: facilidade na prototipação, redução do custo e das dimensões da placa de circuito impresso, aumento da confiabilidade do sistema e alta precisão [22].

Existem na literatura poucos trabalhos utilizando FPAA para a aquisição de sinais de ECG. Dentre eles, o trabalho de [21] visa explorar os recursos oferecidos pela FPAA no processamento de sinais de ECG em um laboratório de ensino, avaliando os resultados através de um osciloscópio e do *software* fornecido pela fabricante da FPAA. O trabalho de [10] desenvolveu um módulo de baixo consumo que permite a aquisição de sinais de ECG para serem visualizados e armazenados em um computador. As pesquisas realizadas em [22] e [23] desenvolveram um sistema utilizando tecnologias reconfiguráveis (FPAA e *Field Programmable Gate Array* (FPGA)) e um conversor analógico-digital (ADC) para captar sinais de ECG em indivíduos, sendo posteriormente transmitidos para um computador.

1.1 Justificativa

Em 2008, segundo estudos da *World Health Organization* (WHO) - agência especializada em saúde pública da Organização das Nações Unidas (ONU) - cerca de 17,3 milhões de pessoas tiveram mortes causadas por doenças cardiovasculares. Estima-se que até o ano de 2030, mais de 23 milhões de pessoas morrerão por ano devido a problemas no coração [24]. Visando evitar tais ocorrências, a eletrocardiografia vem ampliando o mercado de sistemas eletrônicos portáteis para tal fim, permitindo a realização de exames em qualquer local (mobilidade), de forma mais confortável e fácil. Com isso, diversos benefícios são proporcionados, tais como:

- Medicina preventiva;
- Inclusão social: a mobilidade permite a realização de exames em países subdesenvolvidos e em pacientes que moram em locais com poucos recursos médicos; e
- Redução dos custos e do risco de infecções: pois evita o deslocamento de pacientes até os hospitais.

Assim, o desenvolvimento de um sistema para a aquisição de sinais de ECG torna-se necessário. Diversos sistemas móveis transmitem os sinais captados para serem visualizados

em um computador, comprometendo em parte a mobilidade do sistema (restringe-se apenas a telemedicina). A utilização de um *smartphone* para a visualização e o armazenamento dos sinais permite contornar esta limitação. Atualmente, o sistema operacional Android está em cerca de 900 milhões de dispositivos ativos, sendo uma escolha interessante devido a sua grande popularidade [25, 26].

O desempenho e a viabilidade de utilização da FPAA foram comprovados em outros trabalhos nessa área [10, 22, 23], entretanto, nenhum trabalho encontrado utiliza um aplicativo em um *smartphone* que permita reconfigurar facilmente a FPAA, alterando diferentes parâmetros do circuito de aquisição (frequências de corte de filtros, ganhos, tipos de filtros, entre outros). Cabe observar que essa flexibilidade permite que o sistema seja usado também para outros sinais bioelétricos.

1.2 Objetivo

O objetivo deste trabalho é desenvolver um Sistema Embarcado Reconfigurável para Aquisição de Sinais de ECG (SERAS-ECG), e um *software* para visualização e armazenamento destes sinais em um *smartphone* com sistema operacional Android. O *software* para o SO Android também permite reconfigurar a FPAA sem a necessidade da utilização de um computador, através de comandos enviados pelo *Bluetooth* para o módulo SERAS-ECG. Portanto, o presente trabalho busca explorar as vantagens fornecidas pela FPAA, motivado pelos poucos trabalhos encontrados na literatura. Além do desenvolvimento do *hardware*, *firmware* e *software* associados, busca-se também a criação de rotinas em C para o microcontrolador configurar a FPAA, e a elaboração de um material bibliográfico que possa auxiliar em futuros trabalhos utilizando FPAAs.

1.3 Organização do Trabalho

Os capítulos seguintes desta dissertação encontram-se organizados da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica sobre os principais conceitos abordados no trabalho, sendo eles: *m-Health*, Eletrocardiograma, *Field Programmable Analog Array*, microcontroladores *Mixed-Signal* e *Bluetooth*, além de relatar os trabalhos correlatos encontrados (estado da arte);
- O Capítulo 3 apresenta os materiais e métodos utilizados no trabalho;
- O Capítulo 4 apresenta os resultados obtidos e a discussão pertinente; e
- O Capítulo 5 apresenta as conclusões sobre o trabalho.

Capítulo 2

Fundamentação Teórica

Este capítulo abordará toda a fundamentação teórica utilizada no desenvolvimento do trabalho. Na Seção 2.1 são apresentados as definições referentes à área *M-Health* e alguns trabalhos correlatos desenvolvidos para a aquisição de sinais de ECG usando FPAA. Na Seção 2.2 são abordados os principais conceitos sobre o eletrocardiograma e a teoria referente à aquisição de sinais de ECG. Na Seção 2.3 é realizada uma introdução sobre a FPAA, a qual é utilizada na implementação de sistemas analógicos. Na Seção 2.4 encontram-se informações referentes aos microcontroladores *Mixed-Signal* que integram partes analógicas e digitais em um mesmo *chip*. Na Seção 2.5 apresenta-se a tecnologia de comunicação *Bluetooth* e o sistema operacional Android, os quais podem ser utilizados em conjunto para se usufruir dos benefícios da transmissão de dados sem fio. Finalmente, na Seção 4.3 são apresentadas algumas considerações finais deste capítulo.

2.1 *Mobile Health*

Mobile Health ou *M-Health* é o termo empregado para definir a utilização de dispositivos móveis na área médica, através de atividades como, prevenção, diagnóstico e terapia [28]. As inovações nesta área proporcionam a igualdade no acesso à informação médica e ao atendimento especializado, superando os limites existentes para os usuários geograficamente distantes [27, 29].

Os países em desenvolvimento apresentam desafios quanto à saúde, principalmente devido à escassez de recursos médicos, o atual envelhecimento da população, o crescimento populacional e a capacidade limitada dos hospitais. Com isso, vários sistemas foram desenvolvidos para a detecção e prevenção de doenças. As principais características destes sistemas são a portabilidade, facilidade de uso, tamanho e peso relativamente minimizados, e baixo consumo de energia, que normalmente é fornecida por uma bateria de tamanho reduzido.

Atualmente, existem no mundo cerca de um bilhão de *smartphones*, os quais são amplamente utilizados na *M-Health* [28]. Entre os recursos mais usados estão as mensagens *Short Message Service* (SMS), a conexão com redes de dados para acesso à Internet, o navegador *Global Positioning System* (GPS) e a câmera fotográfica. Vale ressaltar que o acesso à Internet é essencial, pois permite o tratamento e diagnóstico remoto de pacientes.

A seguir são descritos alguns trabalhos que tiveram como objetivo o desenvolvimento de sistemas *M-Health* aplicados à aquisição de sinais de ECG.

Em [21] realizou-se uma pesquisa para incentivar os alunos no desenvolvimento de um detector do complexo QRS (representa a despolarização ventricular) em *hardware*, com o objetivo de detectar arritmias. Para isso, foram utilizados uma FPAA AN221E04, um osciloscópio e um computador. Através da FPAA, as seguintes funções analógicas foram implementadas: filtro passa-faixa, retificador e comparador. A partir destas funções, os alunos avaliaram sinais de ECG (confeccionados previamente em um *software*) no osciloscópio conectado ao computador. A taxa de sucesso obtida nos testes para deteção de arritmias foi de 85%.

O trabalho [10] desenvolveu um sistema embarcado de baixo custo e consumo, para a aquisição de sinais elétricos do coração. O sistema é composto por uma FPAA e um microcontrolador que realiza a transmissão dos sinais de ECG para o computador. O *software* permite a visualização e o armazenamento de sinais de ECG, os quais podem ser transmitidos pela Internet para um médico avaliar remotamente. Através de testes realizados, constatou-se que o sistema desenvolvido apresenta sinais de ECG com características de forma, amplitude e período similares ao eletrocardiógrafo Siemens E350 - foram aplicados sinais cardíacos gerados pelo simulador *Lionheart 3* da Bio-Tek -, validando portanto, o correto funcionamento do sistema. Cabe observar que a alimentação do sistema é realizada por 4 pilhas alcalinas AA, que proporciona uma autonomia de 2 horas.

O projeto desenvolvido em [30] utilizou uma plataforma adaptável - composta por uma FPGA, uma FPAA e um ADC - para adquirir e processar os sinais de ECG, Eletromiograma (EMG) e Eletroencefalograma (EEG). Através desta plataforma, o sistema pôde lidar com as diferentes características dos sinais bioelétricos, sem modificações de *hardware*. A FPGA foi utilizada para configurar a FPAA e transmitir os dados processados, através da porta serial EIA/TIA-232, para serem visualizados em tempo real no computador. O teste de ECG foi realizado através do simulador Metron PS-420, o teste de EEG pelo aparelho Class Technologies EEGSIM e o teste de EMG foi o único realizado com sinais adquiridos a partir de um ser humano. Os três testes realizados mostraram sucesso na aquisição destes sinais. Vale ressaltar que um algoritmo para detecção de QRS e um para o cálculo da taxa de batimentos foram implementados na FPGA. O trabalho [31] foi uma extensão do trabalho [30] (descrito anteriormente). As principais melhorias realizadas no projeto foram a inserção da interface VGA e de uma bateria, possibilitando maior independência e mobilidade do sistema, sendo que a bateria utilizada suportou 2,4 horas de uso contínuo, enquanto que sem o LCD ela pôde trabalhar por cerca de 7 horas.

Em [32], o objetivo da pesquisa foi desenvolver um sistema para telemedicina utilizando duas FPAAs e uma FPGA. O sistema é capaz de trabalhar com sinais de ECG e EMG, onde a primeira FPAA (IspPac10 da Lattice Semiconductor Corporation) implementou um amplificador de instrumentação que pode ser ajustado para os dois sinais bioelétricos, e a segunda FPAA (AN231E04 da Anadigm, Inc.) permite a filtragem dos sinais. Nos testes com ECG, os sinais foram simulados pelo simulador Metron PS-420, enquanto que os testes com sinais de EMG foram realizados diretamente em um ser humano. Os experimentos realizados obtiveram sucesso, tornando o projeto adequado para ser empregado em sistemas de telemedicina.

Os trabalhos [22,23] apresentam um sistema para aquisição de eletrocardiograma, baseado nos seguintes dispositivos reconfiguráveis: FPAA e FPGA. A FPAA é responsável pela aquisição e condicionamento do sinal analógico de ECG, enquanto que a FPGA permite o processamento de sinais digitais, além de realizar a configuração dos blocos analógicos da FPAA. Nos testes realizados, a plataforma reconfigurável apresentou desempenho considerável com diferentes tipos de eletrodos. Vale ressaltar que a reconfigurabilidade do sistema permitiu a realização da extração do eletrocardiograma fetal de forma não invasiva, o qual é eficaz para monitorar a oxigenação do coração e do cérebro do feto, sendo este, desconhecido pela maioria das gestantes.

Portanto, pode-se observar que os trabalhos correlatos encontrados possuem os seguintes pontos em comum:

- Utilizam um computador para visualizar os sinais de ECG em tempo real e armazená-los para posterior análise;
- A maioria dos trabalhos utilizam uma FPGA e uma FPAA, contidas em suas respectivas placas de desenvolvimento (possuem alto custo);
- As configurações da FPAA são geradas previamente em um computador e armazenadas na FPGA ou em uma memória auxiliar; e
- Os testes de ECG são realizados através de simuladores.

2.2 Eletrocardiograma

O eletrocardiograma (ECG) baseia-se no registro do sinal elétrico gerado pelas atividades eletroquímicas do tecido cardíaco (células excitáveis), adquirido através de eletrodos fixados em posições padronizadas do corpo humano. Trata-se de uma ferramenta de diagnóstico não invasiva e amplamente utilizada para avaliar a condição do coração, possibilitando diagnosticar possíveis cardiopatias (arritmia, ataque coronário, entre outras) [10,21,22,33,34].

Existem três principais tipos de exames de ECG, os quais não necessitam de preparação especial e não possuem quaisquer complicações [3,35]. São eles:

1. ECG de repouso: é o exame mais rápido e simples, o qual capta em um curto espaço de tempo os doze diferentes sinais elétricos na superfície do corpo, denominados derivações. O ECG de repouso visa uma avaliação cardíaca geral do indivíduo e o controle evolutivo de uma doença cardíaca já confirmada por outros métodos de diagnóstico;
2. ECG de esforço: serve para avaliar o coração sob uma condição de estresse, sendo obtido durante a execução de uma atividade física (caminhada ou corrida). Geralmente revela alterações não reveladas no ECG de repouso; e
3. ECG dinâmico ou Holter: realizado por uma longa duração (cerca de 24 horas) para detectar problemas durante atividades do dia a dia da pessoa.

2.2.1 Anatomia e Fisiologia do Coração

O coração funciona como uma bomba de quatro câmaras (dois átrios e dois ventrículos) e quatro válvulas que evitam o refluxo do sangue (pulmonar, entre a artéria pulmonar e o ventrículo direito, tricúspide, entre o átrio direito e o ventrículo direito, mitral, entre o átrio esquerdo e o ventrículo esquerdo, e aórtica, entre a artéria aorta e o ventrículo esquerdo). A fase de contração é denominada sístole, enquanto que a fase de relaxamento é denominada diástole. Assim, o processo de bombeamento funciona da seguinte forma: a veia cava leva o sangue não-oxigenado ao átrio direito. O sangue é então bombeado para o ventrículo direito, passando pela válvula tricúspide. O ventrículo direito bombeia o sangue aos pulmões (passando antes pela válvula pulmonar) para a realização das trocas gasosas. Com isso, o sangue oxigenado entra pelo átrio esquerdo, onde é bombeado para o ventrículo esquerdo, passando antes pela válvula mitral. A partir disso, o ventrículo esquerdo bombeia o sangue de volta para a circulação, passando pela válvula aórtica [1, 36]. A Figura 2.1 ilustra a anatomia do coração.

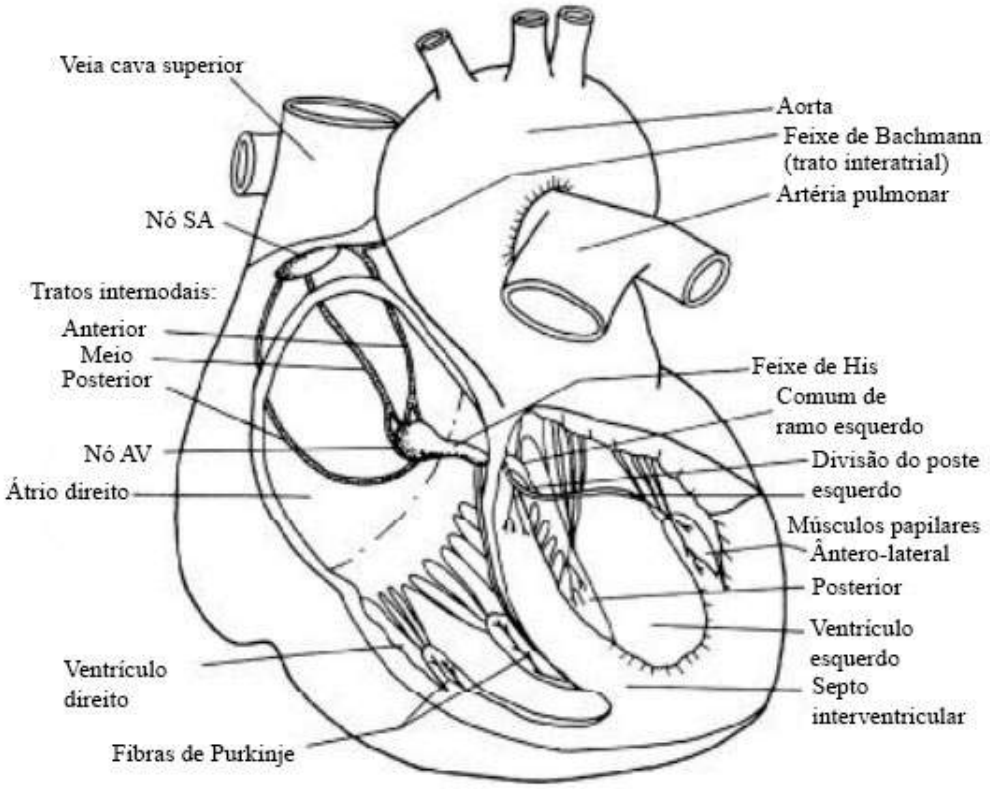

Figura 2.1: Representação do coração humano (modificado de [1]).

A contração rítmica dos átrios e ventrículos precisa de uma ativação elétrica que respeite uma sequência determinada na sua geração e propagação, visto que a contração do músculo ocorre por meio desta ativação elétrica. O ritmo cardíaco (contração coordenada dos átrios e ventrículos) se inicia nas células do nodo sinoatrial (células auto-excitáveis), as quais geram pulsos elétricos que são propagados pelos átrios, ocorrendo a sístole. Vale ressaltar que o feixe de Bachmann (trato interatrial) sai do trato internodal anterior conduzindo o pulso para

o átrio esquerdo, portanto, primeiro é ativado o átrio direito e depois o esquerdo. O nodo atrioventricular (única conexão elétrica entre átrios e ventrículos) leva os pulsos elétricos aos ventrículos. Para finalizar o batimento, os feixes de His e as fibras de Purkinje distribuem os pulsos para os ventrículos [1, 3].

A próxima subseção descreverá como são representados os potenciais elétricos de todos os pontos do coração, considerando o coração como uma fonte bioelétrica.

2.2.2 Os Sinais do Eletrocardiograma

Conforme descrito anteriormente, os potenciais elétricos do coração (sinais de ECG) são gerados pelos batimentos cardíacos e podem ser detectados por eletrodos (invasivos ou não invasivos) fixados geralmente nos braços, pernas e tórax [1, 37].

O sinal de ECG apresenta várias regiões, com base nas quais se descreve a atividade de um ciclo cardíaco. Estas regiões consistem nas ondas P, Q, R, S e T, que em conjunto são conhecidas por complexo PQRST (Figura 2.2). A onda P é gerada quando o átrio contrai para bombear sangue para dentro dos ventrículos (despolarização do átrio). O complexo QRS é formado quando os ventrículos estão contraindo para bombear o sangue para fora do coração (despolarização dos ventrículos). Já o segmento ST indica o fim da despolarização e o início da repolarização dos ventrículos (inatividade elétrica). A onda T indica a repolarização dos ventrículos (quando as fibras dos ventrículos começam a relaxar). Vale ressaltar que a repolarização atrial está escondida na despolarização ventricular, não aparecendo em condições normais devido à magnitude maior do complexo QRS [10, 22, 37].

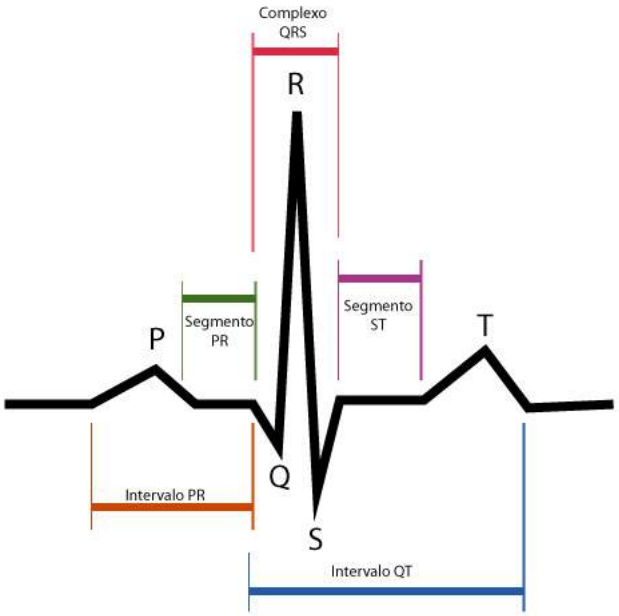

Figura 2.2: Regiões de um ciclo cardíaco, denominado complexo PQRST (modificado de [2]).

As irregularidades na morfologia do complexo PQRST indicam anormalidades do músculo cardíaco, enquanto que as irregularidades na temporização das formas de onda em um ou vários complexos indicam anormalidades no processo de condução do sinal pelo coração [22].

A atividade elétrica do coração pode ser representada através de um vetor dipolo localizado no tórax (representado por M). O dipolo representa a atividade elétrica do coração em um instante de tempo específico. À medida que um ciclo cardíaco progride, a magnitude e a direção de M variam. A Figura 2.3 mostra um desenho do vetor dipolo.

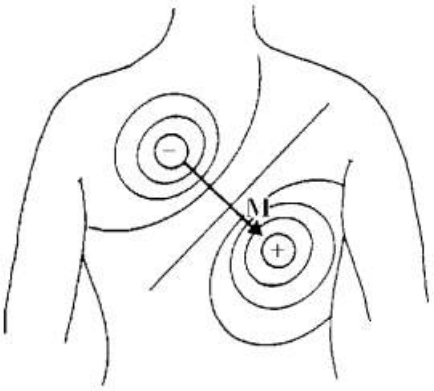

Figura 2.3: Representação de um dipolo através de um vetor dirigido da carga negativa para a carga positiva (M) (retirado de [1])).

Estas diferenças de potenciais são captadas fixando-se eletrodos na superfície do corpo (geralmente aplica-se um gel para melhorar o contato com o eletrodo metálico) e medindo a tensão entre eles.

Para descrever completamente a atividade elétrica do coração, várias derivações devem ser registradas. Na prática, as derivações são tomadas no plano frontal (permite a visualização do coração pela parte frontal do indivíduo) e no plano transversal (permite a visualização do coração como se o indivíduo fosse cortado ao meio). O uso de diferentes derivações permite avaliar o estado de regiões específicas do coração.

Existem três derivações básicas medidas no plano frontal, denominadas bipolares [1]. Estas são derivadas de pares de eletrodos, quando estes encontram-se localizados no braço direito (RA), no braço esquerdo (LA) e na perna esquerda (LL), resultando em:

- Derivação I = LA - RA (LA: entrada positiva do medidor, RA: entrada negativa do medidor);
- Derivação II = LL - RA (LL: entrada positiva do medidor, RA: entrada negativa do medidor); e
- Derivação III = LL - LA (LL: entrada positiva do medidor, LA: entrada negativa do medidor).

Os vetores dessas três derivações podem ser representados como um triângulo equilátero no plano frontal do corpo, conhecido como Triângulo de Eindhoven [1].

As derivações aumentadas e precordiais são denominadas unipolares, pois registram a média dos sinais de dois ou mais eletrodos. As derivações aumentadas (aVL, aVR e aVF) são medidas da seguinte forma:

- aVL: LA na entrada positiva do medidor, RA e LL na entrada negativa do medidor;
- aVR: RA na entrada positiva do medidor, LA e LL na entrada negativa do medidor; e
- aVF: LL na entrada positiva do medidor, RA e LA na entrada negativa do medidor.

As derivações precordiais (V1, V2, V3, V4, V5 e V6) são medidas no plano transversal, fixando-se eletrodos em posições pré-definidas do tórax.

A Figura 2.4 ilustra todas as doze derivações descritas.

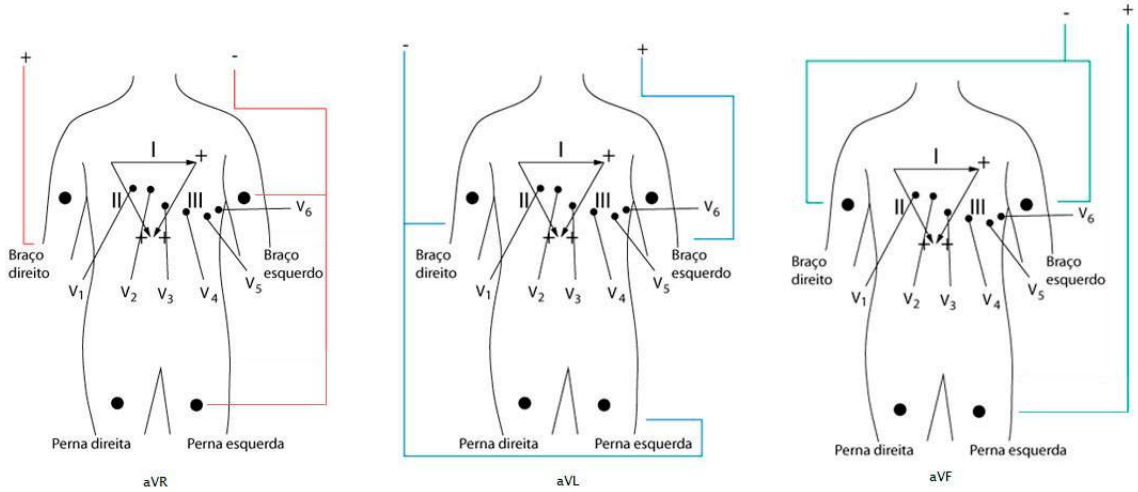

Figura 2.4: A imagem ilustra o Triângulo de Eindhoven com os vetores para as derivações DI, DII e DIII. Apresenta também as derivações aumentadas e as precordiais (modificado de [2]).

Para o registro dos sinais das doze derivações padronizadas (DI, DII, DIII, aVR, aVL, aVF, V1, V2, V3, V4, V5 e V6) são utilizados dez eletrodos. Entretanto, com três eletrodos já é possível obter as seis principais derivações (DI, DII, DIII, aVR, aVL e aVF).

Os sinais captados pelos eletrodos apresentam uma amplitude na faixa de 0,05 a 3 mV (Tabela 2.1). A largura de banda do sinal a ser captado deve ser definida conforme a aplicação (Figura 2.5), podendo variar entre 0,05 e 100 Hz (registro das doze derivações), 0,5 e 50 Hz (monitoração do ECG) e a faixa de frequência centrada em 17 Hz (para a determinação da frequência cardíaca) [36].

Tabela 2.1: Características dos biopotenciais [1, 10].

Biopotencial	**Faixa de frequências (Hz)**	**Faixa de amplitudes (mV)**
Eletrocardiograma	0,01 - 100	0,05 - 3
Eletroencefalograma	0,1 - 80	0,001 - 1
Eletro-oculograma	0,01 - 10	0,001 - 0,3
Eletromiograma	50 - 3000	0,01 - 100

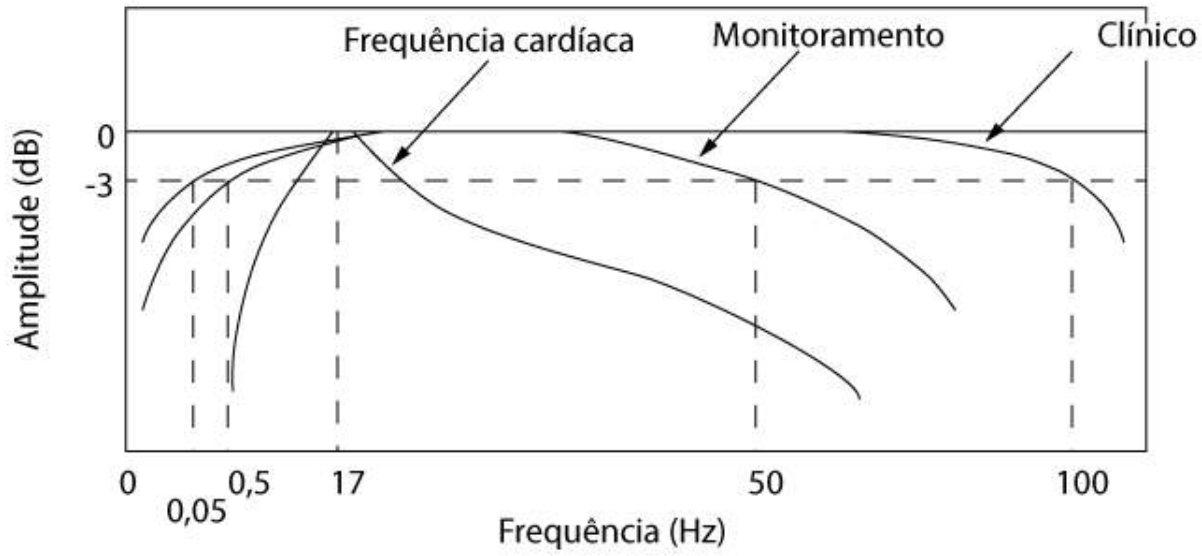

Figura 2.5: Diferentes larguras de banda utilizadas em eletrocardiógrafos (modificado de [3]).

Vale ressaltar que quando o sinal é adquirido, ele sempre está misturado com ruídos que podem inutilizar o sinal de ECG, tais como interferência da rede elétrica, ondas eletromagnéticas captadas pelo corpo, sinais bioelétricos gerados em outros lugares do corpo e movimentos do indivíduo. Devido a estes ruídos, filtros são necessários para melhorar o sinal registrado, permitindo extrair parâmetros que quantificam os sinais de ECG de forma mais fácil. A Figura 2.6 mostra as etapas envolvidas na aquisição de sinais bioelétricos que normalmente precisam ser realizadas por circuitos analógicos.

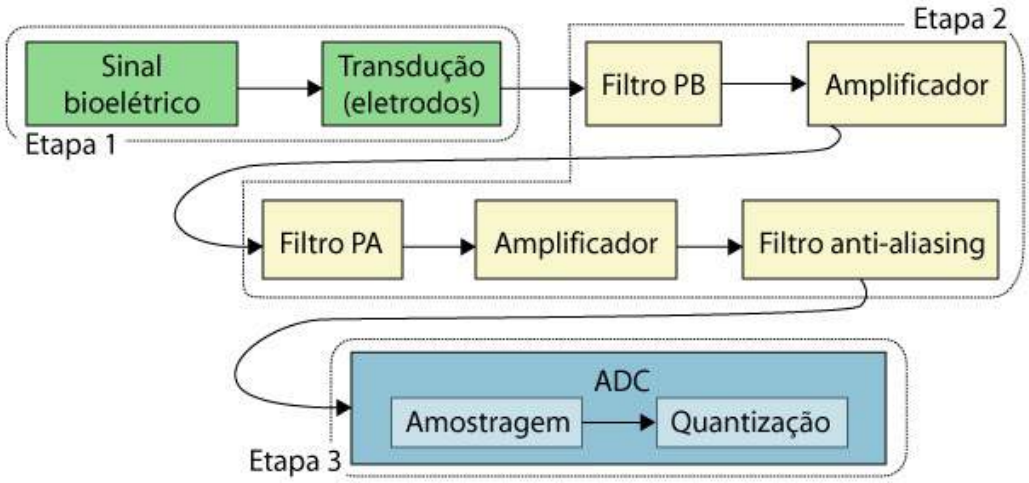

Figura 2.6: Representação das etapas envolvidas na aquisição de sinais de ECG (modificado de [4]).

- Na etapa 1 a variável (informação de interesse) passa pelo transdutor, um dispositivo que converte a variável física em elétrica, gerando como saída uma tensão analógica proporcional à variável física monitorada [38–40];
- Na etapa 2 ocorre o condicionamento do sinal através da filtragem e da amplificação do mesmo. A filtragem é realizada através de circuitos que selecionam ou rejeitam uma faixa de frequências (normalmente construídos com Amplificadores Operacionais (op-amps)). Geralmente ela é utilizada para eliminar componentes espectrais indesejadas (ruídos), sendo que, na aquisição de sinais de ECG, comumente, empregam-se filtros passa-baixas e passa-altas. O filtro passa-baixas analógico (Figura 2.7.a) permite apenas a passagem de frequências inferiores à frequência de corte do filtro (atenua a amplitude das frequências acima da frequência de corte), enquanto que no filtro passa-altas analógico (Figura 2.7.b) ocorre a passagem de frequências superiores à frequência de corte do filtro, atenuando a amplitude das frequências abaixo da frequência de corte. Existe ainda a classe de filtro passa-baixas *Butterworth*, normalmente utilizado como filtros *anti-aliasing* para limitar o sinal à faixa de frequência desejada. A amplificação é realizada devido à baixa amplitude dos sinais captados e após esse processo de condicionamento, o sinal pode ser digitalizado; e
- Na etapa 3 ocorre a digitalização do sinal, realizada através da amostragem e quantização por um conversor analógico-digital (ADC).

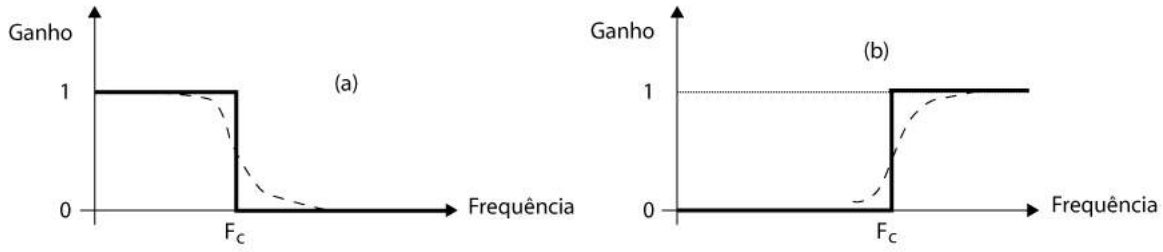

Figura 2.7: a) Filtro passa-baixas: para frequências inferiores à frequência de corte do filtro, o ganho é unitário, portanto, a amplitude do sinal de saída é igual ao da entrada. Já quando as frequências são superiores o ganho é zero. b) Filtro passa-altas: o ganho é unitário apenas para sinais acima da frequência de corte do filtro (modificado de [5]). Cabe observar que os filtros descritos são ideais e na prática, obtém-se comportamentos como os descritos pelas linhas tracejadas.

Portanto, a amplificação e a filtragem servem para eliminar as interferências e adequar a amplitude dos sinais ao ADC, enquanto que a digitalização transforma o sinal analógico em digital e possibilita o seu processamento por computadores [4].

A amostragem é a medição do valor de um único ponto do sinal, sendo este ponto denominado amostra, em intervalo de tempo pré-determinado, conhecido por período de amostragem. A quantidade de amostras obtidas em um segundo é denominada taxa de amostragem ou frequência de amostragem (F_a), sendo calculada pelo inverso do período de amostragem. Vale ressaltar que uma série de amostras possibilita que o sinal seja reconstruído, assim enunciado pelo teorema de *Shannon* (teorema da amostragem): a frequência de amostragem deve ser no mínimo o dobro da frequência máxima do sinal amostrado ($F_a \geq 2.Fsinal$), para que as informações não sejam perdidas e o sinal possa ser reconstruído [4, 38, 39].

Se o teorema da amostragem não for satisfeito, ocorre um efeito chamado de *aliasing* (ou falseamento), comprometendo a reconstrução com fidelidade do sinal analógico original. Para exemplificar, caso seja definido que a maior frequência contida em um sinal é 5 kHz, esse sinal deve ser amostrado a uma taxa de 10000 amostras por segundo. Caso ocorra a presença de um sinal de entrada de 6 kHz, um sinal falso de 4 kHz seria adquirido, ou seja, uma componente de 6 kHz pareceria uma componente de frequência de 4 kHz (10 kHz - 6 kHz). Como não é possível garantir que o sinal medido não contenha componentes acima deste limite, realiza-se uma filtragem passa-baixas, também conhecida por filtragem *anti-aliasing*, antes da amostragem [4, 38].

A quantização consiste em associar cada amostra do sinal analógico a um dos 2^B valores possíveis, onde B corresponde à resolução do ADC (número de bits). Quanto maior a resolução do ADC, maior será a exatidão no processo de quantização.

2.3 *Field Programmable Analog Array*

A *Field Programmable Analog Array* (FPAA) é um dispositivo utilizado na implementação de sistemas analógicos que oferece a possibilidade de ter todos os circuitos analógicos necessários em um único componente programável. A FPAA é comparada a FPGA, pois ambas consistem de uma matriz de células as quais realizam funções reconfiguráveis, e que precisam de células de comutação para a configuração e o roteamento, sendo a FPGA aplicada em sistemas digitais.

Os principais benefícios oferecidos pela FPAA são [10, 41–43]:

- Simplificação do projeto;
- Redução de dimensões da placa de circuito impresso;
- Fabricação simplificada;
- Aumento da confiabilidade do sistema;
- Alta precisão; e
- Permite a reconfiguração, proporcionando a atualização em campo (flexibilidade).

Comercialmente, existem poucas fabricantes de FPAA, sendo as mais presentes no mercado: *Cypress* (família CY8C2XXXX), *Lattice Semiconductor* (família ispPAC) e *Anadigm* (famílias: AN120E04, AN121E04 e AN131E04). O custo de uma FPAA no mercado varia entre USD3,00 e USD13,00, um valor relativamente baixo quando considera-se as vantagens e recursos disponíveis [44–46].

2.3.1 Arquitetura

Uma FPAA é constituída internamente por elementos programáveis e uma rede de interconexões. Geralmente, os elementos programáveis - Op-amps, capacitores, multiplexadores analógicos, entre outros - estão agrupados em blocos, denominados Blocos Analógicos Configuráveis (CAB). A rede de interconexões são chaves eletrônicas que interconectam os CABs e as células de entrada e saída, que fazem a interface com o meio externo e podem executar funções de condicionamento de sinais [47].

Os CABs possuem acesso a Tabela de Busca (*Look-up-table* (LUT)), amplamente utilizada na linearização de sensores e em processos de auto-calibração, e um gerador de tensão de referência que fornece as tensões necessárias [45].

A seguir será apresentada a FPAA selecionada para ser usada no sistema proposto. A escolha desta FPAA deve-se ao fato dela ser amplamente utilizada nos trabalhos encontrados, além de todo suporte técnico realizado pela fabricante.

2.3.2 AN221E04

A FPAA utilizada no trabalho é a AN221E04, que usa uma fonte de 5 volts e trata-se de um processador analógico de sinal programável dinamicamente (*dynamically programmed Analog Signal Processors* (dpASP)), permitindo que o dispositivo seja reconfigurado (totalmente ou parcialmente) de maneira dinâmica, durante sua operação. Esta reconfiguração é denominada *on-the-fly*, onde uma nova configuração pode ser enviada para a FPAA enquanto a atual configuração encontra-se ativa e operante. Com isso, a reconfiguração dinâmica permite que as características da FPAA possam ser alteradas e carregadas em tempo real, sem a necessidade de reiniciar o sistema. Esta é uma solução eficiente para aplicações como: circuitos analógicos de filtragem, circuitos condicionadores de sinais de sensores e sistemas de controle de malha fechada, onde há necessidade de alteração de parâmetros do sistema [41]. A Figura 2.8 mostra a arquitetura interna deste circuito integrado.

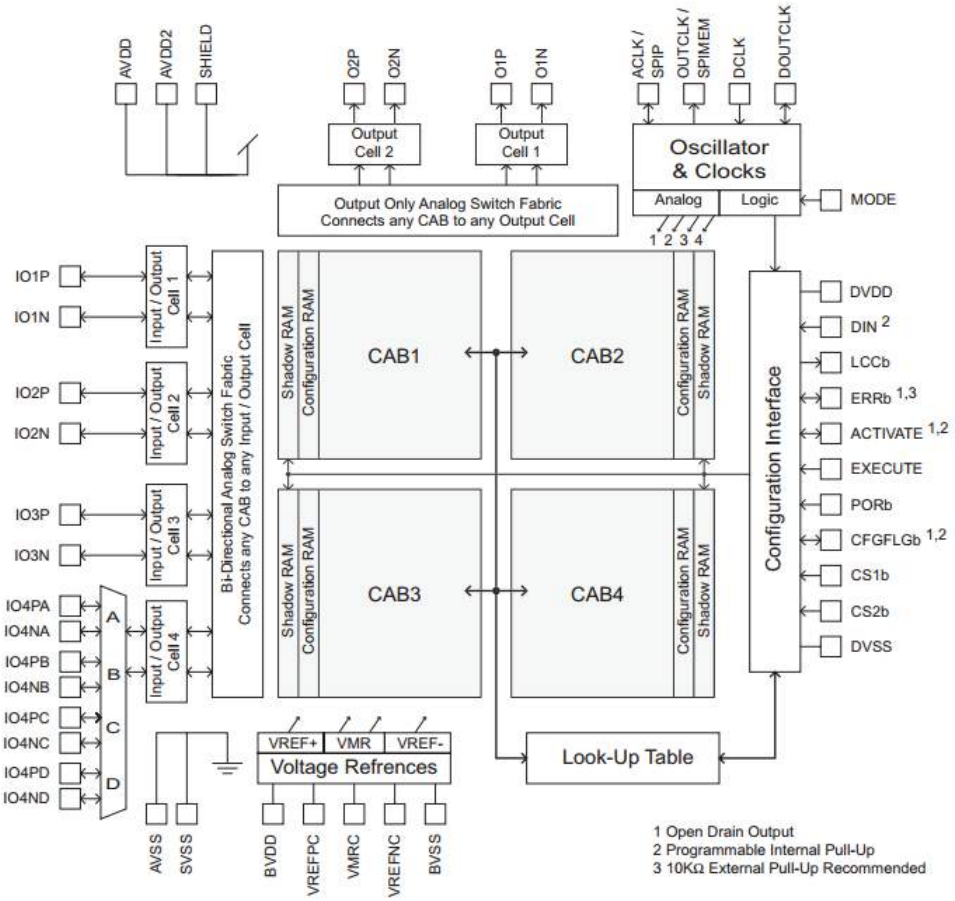

Figura 2.8: Visão geral da arquitetura da FPAA Anadigm AN221E04 [6].

Analisando a Figura 2.8, pode-se observar que a FPAA é formada por:

- Quatro CABs em uma matriz 2x2;

- Quatro células configuráveis de entrada e saída;
- Duas células dedicadas para interface de saída;
- Uma *Look-up-table* (LUT);
- Um bloco de tensão de referência;
- Um bloco de osciladores; e
- Um bloco de interface de configuração.

A FPAA possui quatro células configuráveis de Entrada/Saída (E/S), sendo que a quarta célula possui um multiplexador bidirecional que permite a conexão de até quatro sinais diferenciais distintos. Cada célula de E/S configurável possui um conjunto de recursos que possibilita conexões com o mundo exterior, sem a necessidade de componentes adicionais. Todos os sinais roteados e processados dentro destas células são diferenciais, podendo existir uma conexão interna para ligar a entrada negativa ao *Voltage Main Reference* (VMR), referenciando a entrada positiva ao VMR.

Cada célula de E/S, quando configurada como entrada, disponibiliza alguns recursos que podem ser aplicados ao sinal de entrada:

- Filtro *anti-aliasing*: trata-se de um filtro passa-baixas para evitar o *aliasing*;
- Amplificador *Chopper*: amplificador que reduz consideravelmente a tensão de *offset* de entrada, sendo útil para sinais com baixa amplitude e que necessitam de alto ganho (variando entre 2^4 a 2^7); e
- *Buffer* de ganho unitário: quando o filtro *anti-aliasing* e o amplificador *Chopper* não são utilizados, o sinal é conectado diretamente ao barramento (*bypass*). Como os Op-amps contidos nos CABs apresentam baixa impedância de entrada, necessita-se de um *buffer* de ganho unitário.

As duas células de saída são dedicadas a sinais analógicos diferenciais, possuindo características semelhantes às células de E/S. Seus recursos são:

- Filtro *anti-aliasing*: conforme descrito anteriormente; e
- Filtro programável: caso não seja utilizado o sinal diferencial, o sinal de saída será referenciado ao VMR.

A memória de configuração *Static Random Access Memory* (SRAM) permite a modificação das conexões analógicas dentro de cada CAB. Cada CAB possui um banco de oito capacitores programáveis, sendo cada capacitor um grande banco de pequenos capacitores, dois Op-amps e um comparador.

Quando o *Successive Approximation Register* (SAR) está ativado, utiliza-se o comparador do CAB para implementar um ADC de 8 bits. A saída deste conversor pode ser roteada para

o mesmo CAB ou para a geração de endereços para a NUV (no final de cada conversão, os 8 bits resultantes são reconhecidos pela NUV como um novo endereço), permitindo a criação de funções não-lineares, linearizações, filtragem e controle automático de ganho [22].

O processamento analógico é realizado tendo como tensão de referência a VMR, a qual tem seu valor nominal de 2 V. A tensão VMR é derivada de uma fonte de referência de alta precisão. Além da tensão VMR são geradas outras duas referências, VREF+ (1,5 V acima de VMR) e VREF- (1,5 V abaixo de VMR).

A configuração é realizada utilizando-se dados gerados pela ferramenta de *software* *AnadigmDesigner2*. A interface permite a configuração através de uma memória *Electrically Erasable Programmable Read-Only Memory* (EEPROM) ou de um microcontrolador mestre. Ao utilizar uma memória EEPROM, a configuração é realizada logo depois que a FPAA é energizada, carregando os dados automaticamente da EEPROM. Quando um microcontrolador realiza a configuração, este apenas envia os dados para a FPAA, que neste caso, funciona no modo escravo. Vale ressaltar que nas duas formas de configuração utiliza-se o barramento *Serial Peripheral Interface* (SPI) ou a comunicação serial EIA/TIA-232 (níveis TTL), no caso do MCU.

Ao ser reiniciada, a memória SRAM é totalmente limpa e realiza-se a transferência dos dados da configuração primária para ela. Inicialmente, os dados são lidos e encaminhados para a memória *Shadow* SRAM. A memória *Shadow* permite que uma configuração diferente do circuito seja carregada, sem perturbar a funcionalidade da configuração atual. Para que a FPAA inicie a transferência do conteúdo da *Shadow* SRAM para a memória de configuração é necessário apenas um ciclo de *clock*.

A Figura 2.9 mostra a interface do *software* *AnadigmDesigner2*, que permite projetar e implementar circuitos analógicos, gerando os arquivos de dados necessários para a configuração da FPAA. Para a criação de circuitos são utilizados módulos denominados CAM, onde cada CAM implementa uma função analógica configurável via parâmetros, como: filtros, multiplicadores, somadores, detectores de pico, comparadores, retificadores, entre outros. Os CAMs necessários para o projeto precisam apenas ser arrastados para a área de trabalho do projeto. A ferramenta ainda inclui um simulador, composto por um gerador de sinais e um osciloscópio, que permitem avaliar o comportamento do circuito criado, antes de utilizar a FPAA.

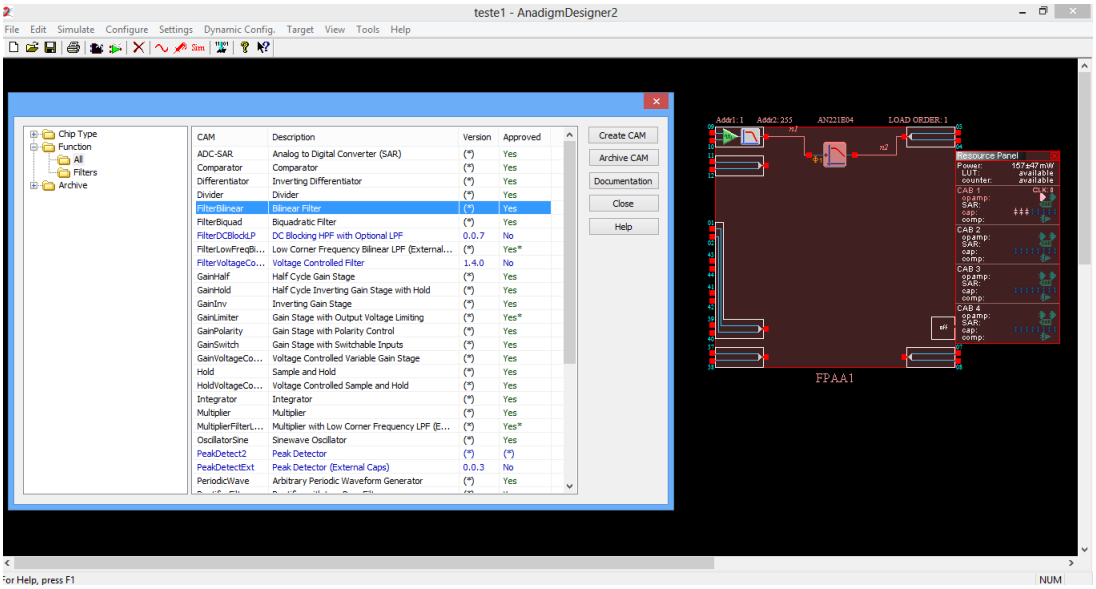

Figura 2.9: Tela do *software AnadigmDesigner2*.

O *software AnadigmDesigner2* possui também duas ferramentas - uma para a criação de filtros e outra para sistemas de controle automático *Proportional-Integral-Derivative* (PID). Elas permitem que sejam desenvolvidos sistemas apenas com a inserção de parâmetros, gerando automaticamente o diagrama do circuito. A partir deste diagrama, os respectivos CAMs são carregados no *AnadigmDesigner2*, possibilitando a geração dos dados de configuração da FPAA.

Reconfiguração Dinâmica

A AN221E04 suporta dois métodos de configuração dinâmica, isto é, através desses dois métodos ela permite controlar dinamicamente os parâmetros dos circuitos analógicos (alterar durante seu funcionamento): o método algorítmico e o método dirigido a estados [6].

O método algorítmico utiliza um algoritmo e um código implementado em linguagem C, que gera os dados de configuração da FPAA a partir de chamadas de funções. Este método permite utilizar melhor os recursos da FPAA, gerando os seguintes arquivos: *API*.c, *API*.h, *CAM*.c e *CAM*.h. Os arquivos denominados "*API*" controlam os dados e as funções necessárias para a configuração inicial, enquanto que os arquivos denominados "*CAM*" implementam os algoritmos utilizados para definir as conexões e os valores dos capacitores associados. A partir destes códigos é possível chamar qualquer função "*CAM*" e alterar qualquer parâmetro em qualquer momento, pois o código calculará e preparará um fluxo de dados para a configuração desejada.

As chamadas de funções dentro do código "*API*" são utilizadas para realizar a transferência da configuração para a FPAA. O fluxo de dados de configuração pode realizar uma configuração primária (configuração inicial da FPAA) ou uma configuração parcial (utilizada para atualizar parâmetros específicos de um CAM específico). Vale ressaltar que a configuração parcial permite atualizar um parâmetro dentro de poucos microsegundos.

A única limitação deste método é a utilização de funções matemáticas complexas (o CAM filtro biquadrático utiliza funções trigonométricas e raiz quadrada). Estas funções são calculadas rapidamente em uma plataforma IBM-PC, porém, em um MCU de baixa capacidade de processamento será necessário um tempo para o cálculo, que poderá chegar a alguns segundos.

É importante constar que a ferramenta *AnadigmDesigner2* fornece um recurso para prototipagem rápida utilizando um IBM-PC, conhecida como "*Visual C++ Prototype*". Este gerador de código é uma extensão do método algorítmico, onde através da utilização da *Integrated Development Environment* (IDE) *Microsoft Visual Studio* - nota-se a necessidade do uso de um IBM-PC - criam-se projetos completos, incluindo todo o código C da FPAA e as funções de suporte necessárias para executar uma aplicação [6].

O método dirigido a estados é direcionado a MCUs com baixo poder de processamento, possibilitando a geração de arquivos de configuração primária e secundárias. Os circuitos neste método precisam primeiramente ser criados no IBM-PC, em seguida, o gerador de código os analisará e produzirá um código de tamanho mínimo para permitir que cada "estado do circuito" seja configurado.

Neste método, existem dois formatos do arquivo de saída: o arquivo de texto contendo código C ou os arquivos de texto formatados em *Anadigm Hex File* (AHF).

Os arquivos de texto em código C contêm blocos de AHF (equivale ao *Intel Hex*), onde o primeiro bloco de código descreve o circuito primário e as partes restantes, as configurações parciais necessárias para ajustar o circuito para o próximo estado escolhido pelo MCU. Já os arquivos de texto formatados em AHF contêm simplesmente os dados de configuração.

Vale ressaltar que o AHF contém os dados de configuração em hexadecimal, os quais são enviados bit a bit para o pino DIN (entrada serial de dados) da FPAA, sincronizados pela borda de subida do pino DCLK. Um arquivo .AHF possui entre 150 e 800 *bytes*, dependendo da complexidade do circuito [45].

2.4 Microcontroladores *Mixed-Signal*

Os microcontroladores chamados de *Mixed-Signal* são microcontroladores que possuem internamente componentes analógicos e digitais em um mesmo *chip* [48, 49]. Os principais benefícios proporcionados pelos MCUs *Mixed-Signal* encontram-se resumidos na Tabela 2.2 [7, 48, 49].

Tabela 2.2: Benefícios proporcionados pelo uso de MCUs *Mixed-Signal*.

Benefícios	**Descrição**
Confiabilidade	Proporciona um aumento de confiabilidade, pois a principal causa de falhas são as interconexões (soldas e conexões mecânicas) que são reduzidas, juntamente com a diminuição de componentes discretos.
Menor consumo de energia	O reduzido consumo de energia é proporcionado pelos baixos requisitos de energia de um único *chip*.
Redução de ruído	O nível de emissão de ruído é menor do que um sistema que possui componentes discretos.
Menor custo do sistema	Um sistema altamente integrado propicia um custo reduzido, devido à redução dos componentes, custos dos *dies* e testes. Existem ainda outras melhorias, tais como a diminuição da complexidade do *software* e do projeto em geral, que resultam em menores tempo e custos de desenvolvimento.

A Figura 2.10 ilustra um exemplo de um sistema embarcado, onde os sensores fornecem sinais analógicos de entrada que precisam passar por um ADC e por comparadores e amplificadores de ganho programável (*Programmable-gain amplifier* (PGA)), antes de serem processados por um MCU. As saídas do MCU são tipicamente digitais e muitas vezes precisam ser convertidas de volta para um sinal analógico - através de um conversor digital/analógico (*Digital-to-Analog Converter* (DAC)) - antes que possam ser utilizados para controlar componentes analógicos. Também existem outras funções de natureza analógica em um sistema embarcado - sensor de temperatura, osciladores, entre outras - sendo que a maioria dessas funções normalmente são implementadas em *chips* analógicos discretos, os quais são integrados em um único MCU *Mixed-Signal* [7].

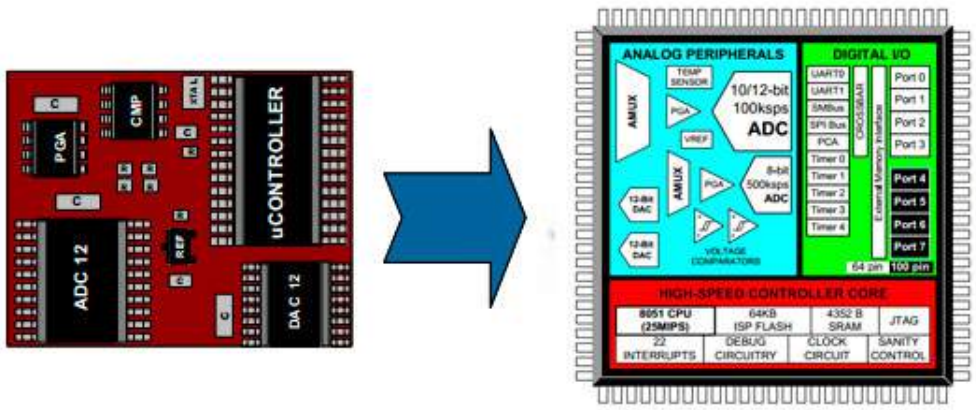

Figura 2.10: À esquerda encontra-se uma placa de um sistema embarcado com seus componentes discretos. À direita observa-se um MCU *Mixed-Signal* que contém internamente todos os circuitos da placa (modificado de [7]).

Atualmente, existem no mercado diversos fabricantes de MCUs *Mixed-Signal*, incluindo: *Texas Instruments*, *Cypress*, *Freescale*, *Silabs*, entre outros. Cada MCU apresenta diferentes características, tais como: presença de núcleo ARM ou 8051, *clocks* variando de 8 a 67 MHz, ADC de até 24 *bits* de resolução, inclusão de Op-amps, comparadores analógicos, amplificadores de ganho programável, entre outros. Vale ressaltar que quanto ao preço, estes microcontroladores variam entre USD 3,20 e USD 20,00 [44].

A seguir será apresentado o MCU utilizado no sistema, cuja escolha foi motivada pelos requisitos do sistema e pelas experiências anteriores com esse *chip*.

2.4.1 Microcontrolador C8051F320

O microcontrolador selecionado para este trabalho foi o C8051F320, que possui um núcleo compatível com o conjunto de instruções da família MCS-51. Esse núcleo possui um *pipeline* que proporciona uma aceleração de aproximadamente 12 vezes em relação ao núcleo original MCS-51. Além dos recursos originais da família MCS-51, o C8051F320 apresenta em seu interior: oscilador de 24,5 MHz, memória de dados (RAM) de 2304 *Bytes* (256 B originais, 1 kB para FIFO da USB e 1 kB de RAM extra), *FLASH* para programa de 16 kB programável no sistema, portas SPI e SMBus, interface USB 2.0, dois temporizadores/contadores adicionais, ADC com referência de tensão interna (10 bits, 200 kSPS, 17 entradas multiplexadas), regulador de tensão de 3,3 V, detector de *brown-out*, sensor de temperatura e *crossbar* que permite configurar a função de cada pino de E/S. Quanto ao encapsulamento do microcontrolador, o F320 possui invólucro do tipo SMD (LQFP) de 32 pinos.

A Figura 2.11 apresenta a arquitetura interna do MCU C8051F320.

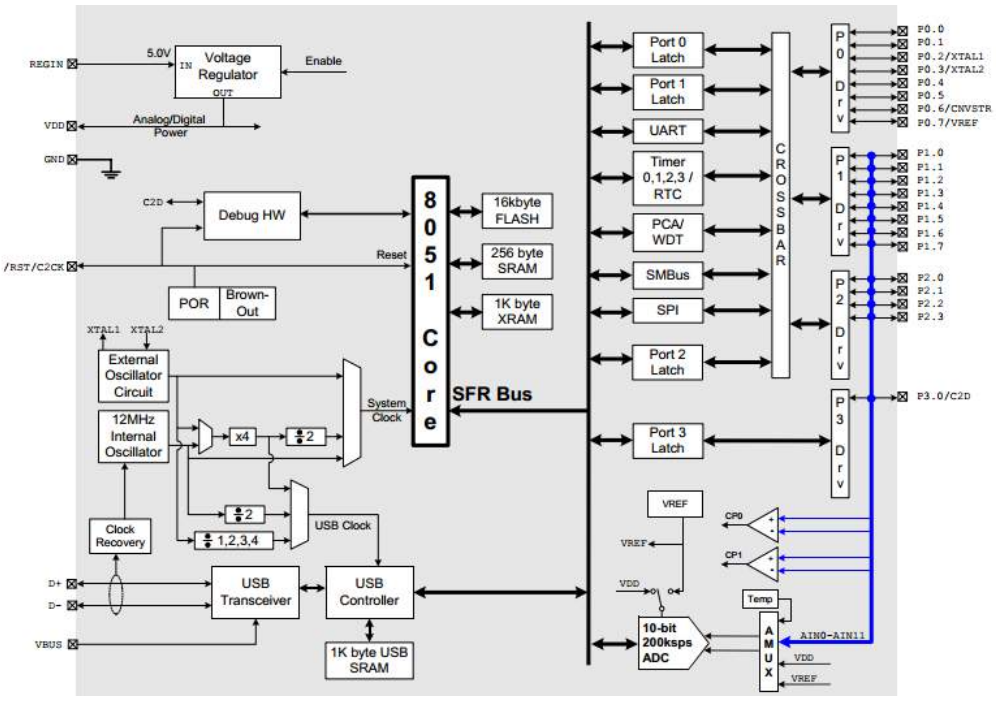

Figura 2.11: Arquitetura interna do MCU *Mixed-Signal* Silabs C8051F320 [8].

Comunicação Serial

O F320 possui uma UART (*Universal Asynchronous Receiver/Transmitter*), que consiste em um barramento serial completamente implementado em *hardware* e que faz uso intensivo de interrupções, exigindo pouca intervenção por parte da Unidade Central de Processamento (CPU).

A UART é *full-duplex* e oferece os modos 1 e 3 da família MCS-51 original. Por se tratar de uma UART aprimorada, o F320 permite uma ampla faixa de fontes de *clock* para gerar a taxa de transmissão/recepção, sendo elas: *System Clock* (SYSCLK) (*clock* do sistema), SYSCLK/4, SYSCLK/12, SYSCLK/48, *clock* do oscilador externo ou uma entrada externa. Entre as taxas de transmissão de dados (*baudrate*) possíveis estão: 1200, 2400, 9600, 14400, 28800, 57600, 115200 ou 230400 bps [8].

O F320 permite que a UART utilize 8 bits ou 9 bits:

- 8 bits: utiliza um total de 10 bits para enviar cada *byte* de dados (1 de início, 8 de dados, sendo primeiro o *Least Significant Bit* (LSB) (*little-endian*), e 1 de parada). A transmissão começa após uma operação de escrita no registrador SBUF; ou
- 9 bits: utiliza um total de 11 bits por *byte* de dados (1 de início, 8 de dados, sendo primeiro o LSB, 1 programável e 1 de parada).

Existe um *buffer* de dados que permite a recepção de um segundo *byte* de entrada, antes que o *software* tenha terminado de ler o primeiro *byte* recebido. Através da habilitação das interrupções da UART, uma interrupção é gerada cada vez que uma transmissão é concluída ou um *byte* de dados é recebido. A Figura 2.12 mostra a interconexão entre dispositivos para estabelecer a comunicação serial.

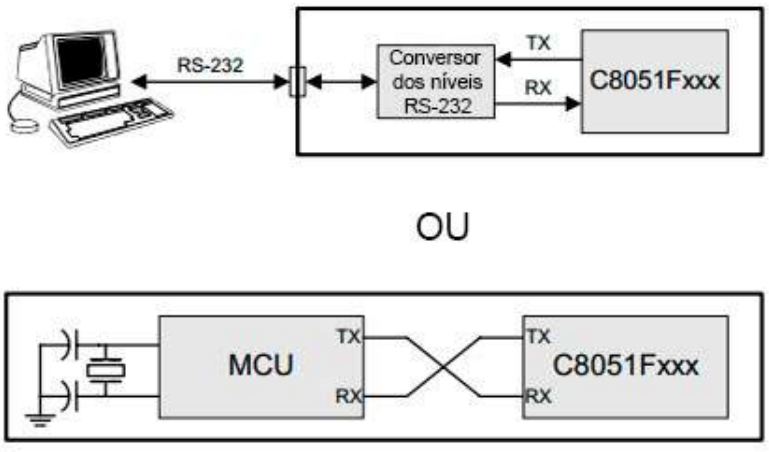

Figura 2.12: Diagrama de interconexão através da UART. A imagem superior ilustra o microcontrolador C8051F320 conectado a um conversor de níveis RS-232, a fim de interligar a interface UART com um IBM-PC. A imagem inferior apresenta uma interconexão entre o C8051F320 e um outro microcontrolador [8]. Observa-se que a comunicação serial necessita apenas de dois pinos, o pino Tx (transmissão) e o pino Rx (recepção).

Conversor Analógico-Digital

Por se tratar de um MCU *Mixed-Signal*, o C8051F320 possui entre os seus periféricos analógicos, um ADC SAR de 10 bits, com uma taxa de amostragem de até 200 kSPS.

O ADC opera nos modos *Single-ended* e *Differential*, e inclui um multiplexador analógico que seleciona as entradas positiva e negativa, permitindo medir sinais aplicados nos pinos das portas 1, 2 e 3, na saída do sensor de temperatura interno, ou na tensão de alimentação (VDD), em relação a pinos das portas 1, 2 e 3, VREF ou GND. Para aplicações que necessitam de baixo consumo de energia, o ADC pode ser desativado pelo *firmware*.

As conversões podem ser iniciadas de seis formas: por *software* (escrita no bit AD0BUSY do registrador ADC0CN), por *overflow* nos temporizadores 0, 1, 2 ou 3, ou através de um sinal externo aplicado em um pino do microcontrolador. Esta flexibilidade permite que a conversão seja iniciada por eventos de *software*, por evento periódico (*overflow* dos temporizadores) ou por sinais externos. Quando a conversão é concluída, ocorre a alteração de um bit de **status** e uma interrupção é gerada (caso esteja habilitada). A palavra binária resultante de 10 bits aparece nos registradores de função especial (*Special Function Register* (SFR)) do ADC, ADC0H e ADC0L.

Um recurso que pode ser explorado quando o ADC está habilitado é o detector de janela programável, o qual possibilita que o ADC seja configurado para interromper o MCU quando o valor digitalizado encontra-se dentro ou fora de uma faixa de valores especificada pela janela de comparação. O ADC monitora uma tensão qualquer continuamente sem interromper o microcontrolador, a menos que o valor convertido esteja dentro ou fora do intervalo especificado em SFRs específicos. A Figura 2.13 apresenta os blocos funcionais do ADC.

2.5 *Bluetooth*

O *Bluetooth* é uma tecnologia de comunicação sem fio que surgiu através da cooperação de diversas empresas, com a finalidade de eliminar cabos de conexão entre os dispositivos eletrônicos. Os grupos 802.15 da IEEE e o *Bluetooth Special Interest Group* (SIG) foram criados para definir um padrão e aprimorar a tecnologia *Bluetooth*. Atualmente, o SIG conta com cerca de 18 mil empresas como membros associados ou afiliados, sendo que as afiliadas usufruem da permissão para utilizar a tecnologia *Bluetooth*, e somente as empresas associadas participam na elaboração de melhorias da especificação e na padronização das aplicações do *Bluetooth*.

Algumas características do padrão *Bluetooth* são [50]:

- Curto alcance;
- Trabalha com baixas tensões, permitindo baixo consumo;
- *Hardware* com dimensões reduzidas;
- Baixo custo;

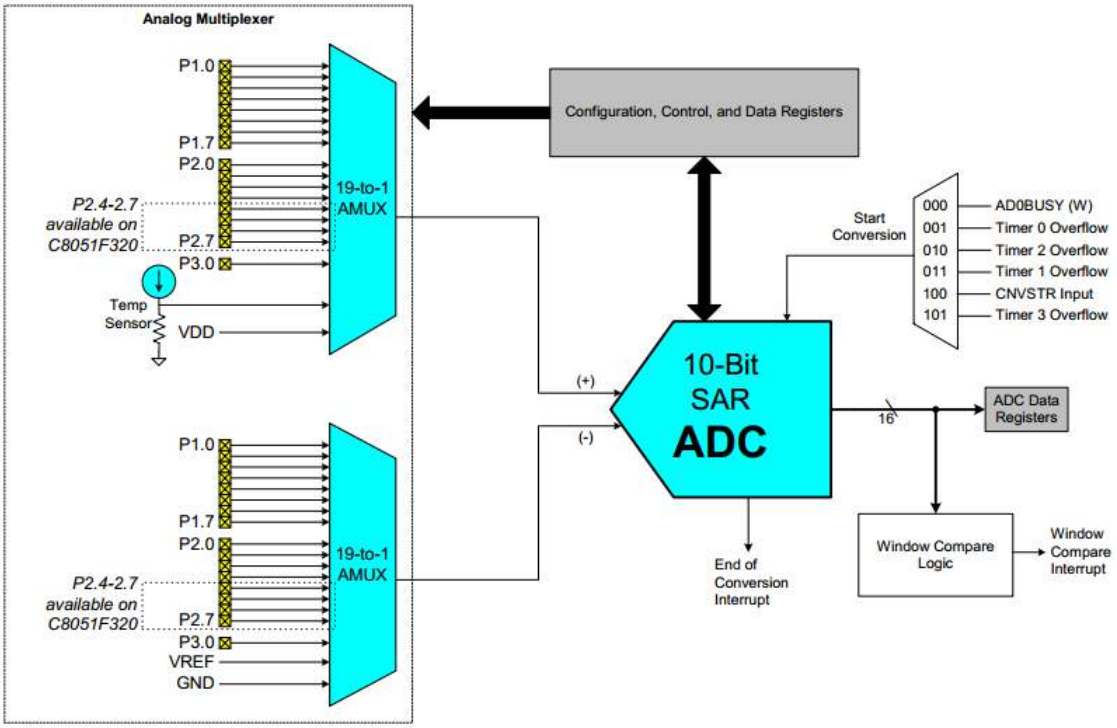

Figura 2.13: Diagrama de blocos do ADC presente no F320 [8].

- Suporte a transmissão de dados e voz;
- Especificação disponível publicamente e livre de *royalties*; e
- A faixa utilizada do espectro de RF pode ser usada sem a necessidade de uma licença especial em todo o mundo (faixa de licença livre *Industrial, Scientific, Medical* de 2,4 GHz).

A fim de evitar interferências com outras fontes de RF que podem estar usando a faixa de 2,4 GHz, a tecnologia *Bluetooth* utiliza a técnica *Adaptive Frequency Hopping* (AFH) e pequenos pacotes de dados. A AFH é um mecanismo de espalhamento espectral que proporciona maior segurança, robustez e eficiência na comunicação. Para isso, a faixa de frequências que varia entre 2400 MHz e 2483,5 MHz é dividida em 79 canais de 1 MHz cada, além de uma banda de proteção superior e outra inferior (Tabela 2.3). Os dados a serem transmitidos são divididos em pequenos pacotes para serem enviados em sequência, onde após o envio de um pacote o dispositivo seleciona um outro canal para transmitir o próximo pacote da sequência. Esta mudança de canal é realizada em até 1600 vezes por segundo [3, 51].

Tabela 2.3: Divisão da faixa ISM para implementar o *Bluetooth* (retirado de [3]).

Proteção inferior	2400 MHz
Canal 0	2402 MHz
Canal 1	2403 MHz
Canal 2	2404 MHz
Canal 3	2405 MHz
.	.
.	.
.	.
Canal 77	2479 MHz
Canal 78	2480
Proteção superior	2481 + 2,5 MHz

A confiabilidade no protocolo é garantida através da retransmissão de pacotes perdidos ou corrompidos. O *Bluetooth* suporta pacotes de voz (*Synchronous Connection-Oriented* (SCO)) e dados (*Asynchronous Connection-Less* (ACL)), onde apenas nos ACL há a retransmissão de pacotes (mediante sinal de confirmação do receptor, através do *Cyclic Redundancy Check* (CRC) do pacote recebido).

A Tabela 2.4 mostra a especificação *Bluetooth* de acordo com a potência de transmissão, enquanto que a Tabela 2.5 apresenta as versões existentes para o *Bluetooth*, com a sua respectiva taxa de transmissão.

Tabela 2.4: Níveis de potência de transmissão do *Bluetooth*.

Classe	**Potência Máxima permitida (mW)**	**Alcance (aproximado)**
1	100 mW	até 100 metros
2	2,5 mW	até 10 metros
3	1 mW	até 1 metro

Tabela 2.5: Versões existentes do padrão *Bluetooth*. A redução na taxa de transmissão da versão 3.0 para a versão 4.0 deve-se à economia de energia obtida.

Versão	**Taxa de transmissão (Mbit/s)**
1.2	1
2.0 + *Enhanced Data Rate* (EDR)	3
3.0	24
4.0	1

Os dispositivos *Bluetooth* possuem 4 estados básicos de operação, sendo eles descritos abaixo:

- Mestre: controla uma *piconet*;

- Escravo ativo: conectado e participando ativamente de uma rede *piconet*;
- Escravo passivo: conectado a uma *piconet*, mas em modo de baixa prioridade (apenas monitora); e
- Em espera: não está conectado a uma *piconet*.

O conjunto de dispositivos conectados em uma rede *ad-hoc* que utiliza os protocolos *Bluetooth* é denominado *piconet*. Eles são estabelecidos automaticamente e dinamicamente de acordo com a entrada ou saída dos dispositivos no raio de cobertura. Um dispositivo mestre pode se comunicar com até sete dispositivos, os quais obedecem a sequência de canais definida pelo dispositivo mestre da *piconet*. Pode ocorrer a união de *piconets*, denominada *scatternet* (Figura 2.14).

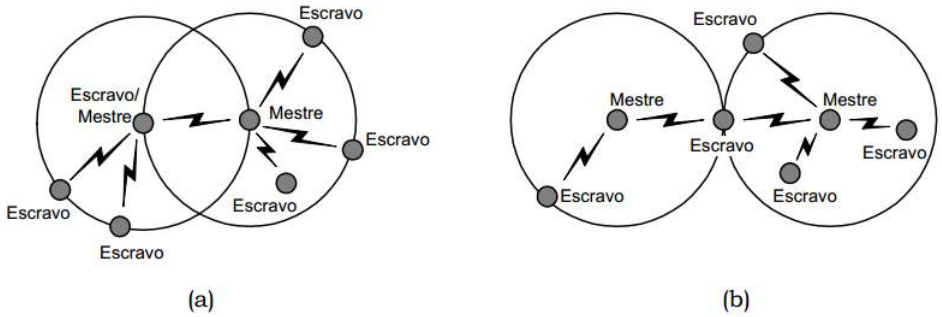

Figura 2.14: (a) O dispositivo em comum na *scatternet* atua como escravo em uma rede e mestre em outra. (b) O dispositivo em comum opera como escravo nos dois *piconets* (retirado de [3]).

Existem diversas identificações para os dispositivos *Bluetooth*, por exemplo: endereço, classe, lista de serviços disponíveis, fabricante, entre outras. Para estabelecer uma conexão entre dispositivos é necessário o pareamento ou aceitação, mediante uma senha em comum inserida nos dispositivos. Cada dispositivo possui um número único de 48 bits que serve de endereço.

A seguir, o dispositivo *Bluetooth* HC-05 será apresentado. Esse módulo *Bluetooth* foi selecionado, principalmente por se tratar de uma solução barata e amplamente adotada em projetos que necessitam de comunicação *Bluetooth*.

2.5.1 Módulo HC-05

O HC-05 é um módulo *Bluetooth Serial Port Protocol* (SPP) que permite a comunicação serial sem fio de forma transparente. Ele é compatível com o padrão *Bluetooth* versão 2.0 e mede 12,7 mm x 27 mm, o que ajuda a simplificar o desenvolvimento do projeto. As principais especificações deste dispositivo são: interface UART (níveis TTL), baixo consumo de energia, antena integrada e tensão de alimentação de 3,3 V à 6 V [52]. A Figura 2.15 apresenta o módulo utilizado no trabalho.

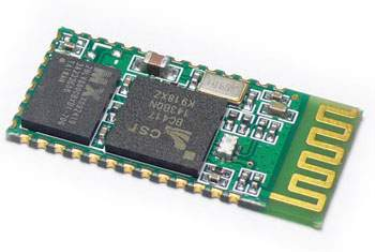

Figura 2.15: Módulo *Bluetooth* HC-05 utilizado para a comunicação serial sem fio.

O HC-05 possui dois modos de funcionamento, *order-response* e conexão automática. Quando o módulo está no modo *order-response*, o usuário pode enviar comandos AT (cadeias de caracteres com formato padronizado) para configurar e controlar o módulo. Quando o módulo está no modo de conexão automática, ele seguirá a última configuração para transmitir os dados automaticamente, podendo ser mestre ou escravo.

2.5.2 Android

O Android é uma plataforma (composta por um SO e ferramentas para o desenvolvimento de aplicações) para dispositivos móveis, criada pelo Google em parceria com uma aliança de diversas empresas do ramo tecnológico - por exemplo: *Sprint Nextel, T-Mobile, Vodafone, Acer, Asus, Dell, Garmin, HTC Corporation, LG, Motorola, Samsung, Sharp, Sony Ericsson, Toshiba, Intel, Nvidia*, entre outras - denominada *Open Handset Alliance* (OHA). Atualmente o Android possui o código-fonte aberto, sob a licença *Apache Software License* (permite a qualquer desenvolvedor a alteração do código-fonte, mas não exige a exposição das alterações feitas) [9].

A Figura 2.16 mostra a arquitetura do sistema operacional Android, a qual é composta por cinco camadas: *Kernel, Libraries, Android Runtime, Application Framework* e *Applications*.

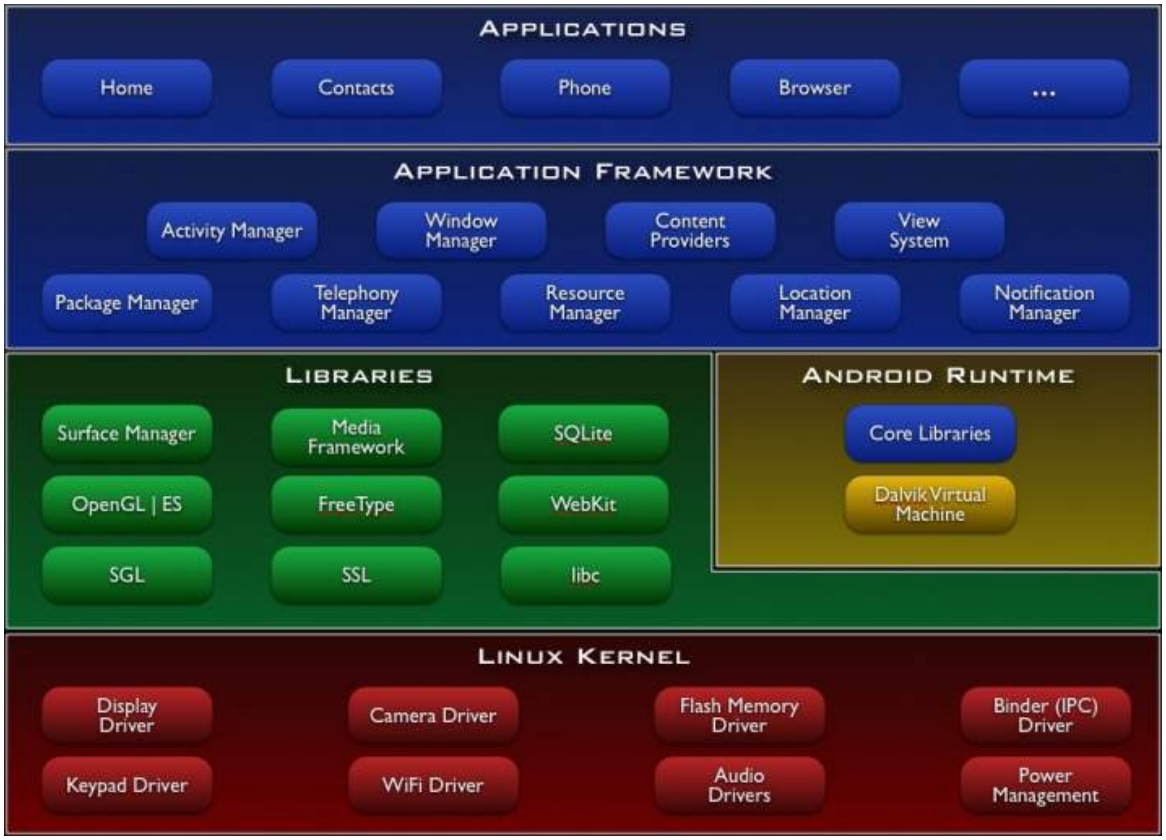

Figura 2.16: Arquitetura do sistema operacional Android (retirado de [9]).

A camada denominada Linux *Kernel* é composta pelo *Kernel* versão 2.6 e se responsabiliza pelos serviços, segurança, gerenciamento de memória e processos, rede e *drivers* (*display* LCD, câmera, teclado, memória *Flash*, entre outros). Essa camada também se responsabiliza pela abstração entre o *hardware* e o resto da pilha do sistema.

A camada *Libraries* é composta por um conjunto de bibliotecas escritas em C/C++, usadas por diversos componentes do sistema. Estas bibliotecas são nativas e fornecem vários recursos aos desenvolvedores. Algumas das principais bibliotecas são:

- **Sistema de biblioteca C:** implementação da biblioteca C padrão (*libc*) sintetizada para dispositivos embarcados com Linux;
- ***Media Libraries:*** bibliotecas que suportam a reprodução e gravação de diversos formatos populares de áudio, vídeo e imagens, por exemplo: MPEG-4, H.264, MP3, AAC, JPG, PNG, entre outros;
- ***Surface Manager:*** gerencia o acesso à visualização 2D e 3D;
- ***LibWebCore:*** funções para navegadores *Web*;
- **SGL:** funções para gráficos;
- **OpenGL | ES:** permite a renderização de gráficos 2D e 3D, geralmente acelerados via *hardware*;

- ***FreeType:*** fontes *BitMap* e vetorizadas; e
- **SQLite:** banco de dados relacional disponível para todas as aplicações.

A camada Android *Runtime* inclui um conjunto de bibliotecas que fornecem a maioria das funções disponíveis nas principais bibliotecas da linguagem de programação Java. Cada aplicação Android é executada em um processo, permitindo que um dispositivo execute várias máquinas virtuais de forma eficiente, executando os arquivos *.DEX* de forma otimizada para se ter um consumo mínimo de memória, bateria e processador.

A camada *Application Framework* possibilita a simplificação na reutilização dos componentes, possibilitando que qualquer aplicação publique suas funcionalidades e qualquer outra aplicação possa fazer o uso destas (sujeito a restrições de segurança impostas pelo *Framework*). Além disso, esta camada agrega todas as *Application Programming Interface* (API)s e os recursos utilizados pelos aplicativos, como provedores de conteúdo (troca de recursos entre aplicativos) e gerenciadores de recursos, de notificação e pacotes [9].

A camada localizada no topo é a *Application*, na qual se encontram todos os aplicativos escritos em Java, por exemplo: clientes de *e-mail*, calendário, mapas, navegador *Web*, contatos e outros. Quando se desenvolve aplicativos para esta plataforma, estes são escritos em linguagem Java para serem executados pela *Dalvik Virtual Machine* (*Dalvik VM*).

Atualmente, o Android encontra-se na versão *4.4* e a API *Level 19* é disponibilizada para acessar as funcionalidades necessárias no desenvolvimento de aplicativos, como as usadas neste trabalho: transmissão e recepção de dados via *Bluetooth* e funções gráficas para plotar na tela do *smartphone*, os sinais de ECG enviados pelo sistema embarcado. Vale ressaltar que no Android *4.4* existe a possibilidade de escolha da máquina virtual - *Dalvik VM* ou ART (*Android Runtime*, o qual possui um desempenho superior à *Dalvik VM*, devido a uma pré-preparação do código a ser executado - o aplicativo torna-se praticamente uma ferramenta nativa do sistema) [53].

2.6 Considerações Finais

Neste capítulo foram apresentados diversos conceitos necessários para o entendimento do trabalho. Uma breve análise dos sistemas *M-Health* aplicados à aquisição de sinais de eletrocardiograma foi realizada, sendo posteriormente abordados as características dos sinais de ECG e os principais componentes necessários para o desenvolvimento do sistema proposto, tais como a FPAA, os MCUs *Mixed-Signal* e o módulo *Bluetooth*. O próximo capítulo apresenta as características do sistema embarcado desenvolvido, bem como os detalhes do *hardware*, *firmware* e *software* implementados neste trabalho.

Capítulo 3

Metodologia

Este capítulo apresenta a metodologia utilizada no desenvolvimento do Sistema Embarcado Reconfigurável para Aquisição de Sinais de ECG (SERAS-ECG). Todos os detalhes relacionados aos materiais e métodos serão apresentados a seguir, detalhando o *hardware*, o *software* e o *firmware* que constituem o SERAS-ECG. Finalmente, na Seção 4.3 são apresentados algumas considerações finais.

3.1 SERAS-ECG

A Figura 3.1 mostra o diagrama de blocos do sistema proposto para aquisição de sinais de ECG, o qual foi montado em uma matriz de contatos. Pode-se observar que o sistema capta os sinais do indivíduo através de 3 eletrodos, sendo que todo o circuito eletrônico necessário para o condicionamento do sinal de ECG foi construído em uma FPAA. Através do barramento SPI, o microcontrolador realiza a configuração da FPAA, a partir dos dados de configuração recebidos pelo *Bluetooth*. O sinal amplificado e filtrado pela FPAA é digitalizado por um microcontrolador que transmite o sinal para o *smartphone* através de um módulo *Bluetooth*. O sistema funciona utilizando uma fonte de 5 V.

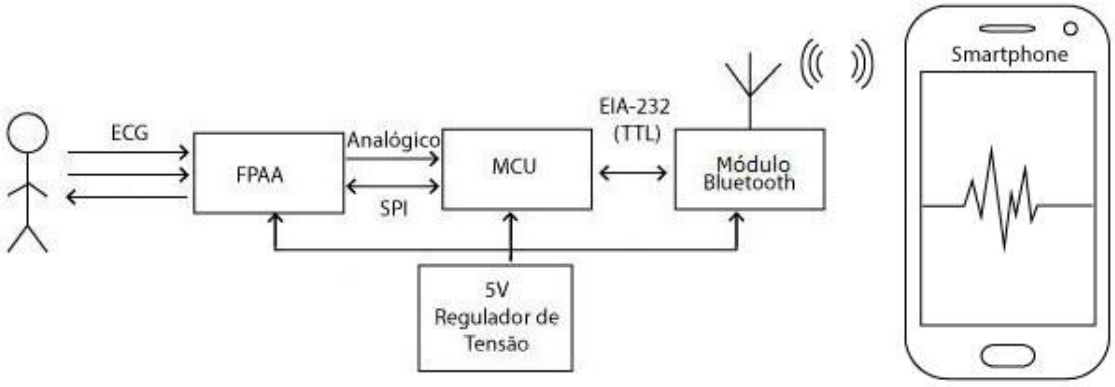

Figura 3.1: Diagrama de blocos do sistema para aquisição de sinais de ECG.

A Figura 3.2 apresenta o esquema eletrônico montado em uma matriz de contatos. Pode-se verificar que o microcontrolador atua como um dispositivo mestre e a FPAA como um dispositivo escravo. As portas DCLK (P0_0) e DIN (P0_2), localizadas na FPAA, recebem o vetor de *Bytes* gerado pela ferramenta *AnadigmDesigner2* e enviado pelo microcontrolador. Caso a FPAA seja configurada corretamente, o led no pino LCCb da FPAA é aceso. Além disso, um *clock* de 100 kHz (P0_7) é gerado pelo MCU e transmitido para o pino ACLK da FPAA.

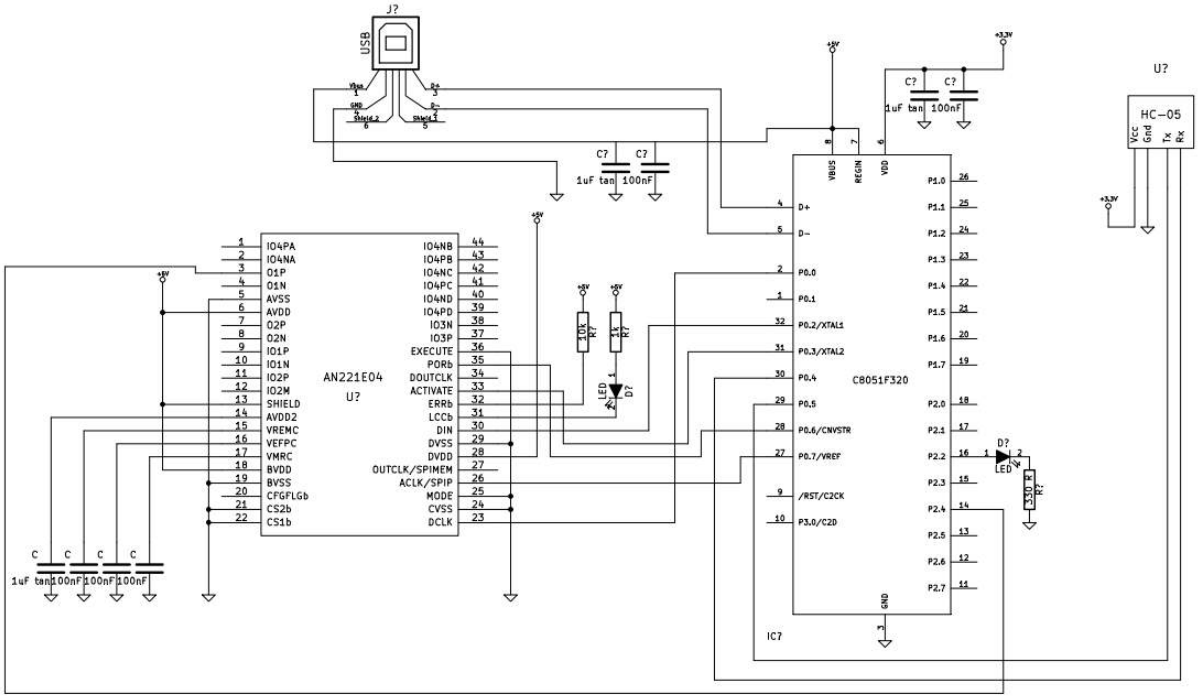

Figura 3.2: Esquema eletrônico do sistema.

Observa-se também que o módulo *Bluetooth* encontra-se conectado ao microcontrolador através dos pinos P0_4 (TX) e P0_5 (RX), além das tensões GND e 3,3 V fornecida pelo regulador de tensão interno do microcontrolador. O módulo HC-05 foi utilizado com as configurações *default* (9600, 8, N, 1). Para testar a interface UART foram transmitidos alguns caracteres pelo microcontrolador, os quais foram recebidos corretamente através dos aplicativos desenvolvidos e do terminal *BlueTerm* [54].

O ADC foi configurado tendo como entrada o pino P2_4, para realizar a digitalização dos sinais amplificados e filtrados pela FPAA. Cada conversão do ADC é iniciada por uma interrupção do temporizador 2, o qual foi configurado para gerar uma amostragem de 480 amostras por segundo.

A Figura 3.3 mostra o osciloscópio virtual (*National Instruments myDAQ*) conectado à porta USB do *notebook*, a matriz de contatos com os dispositivos utilizados e os eletrodos usados nos testes. Existem dois leds, sendo um explicado anteriormente e o outro para realizar testes de código (P2_2), além de dois *push-buttons*, um para reiniciar o microcontrolador e o outro para entrar no modo *bootloader*.

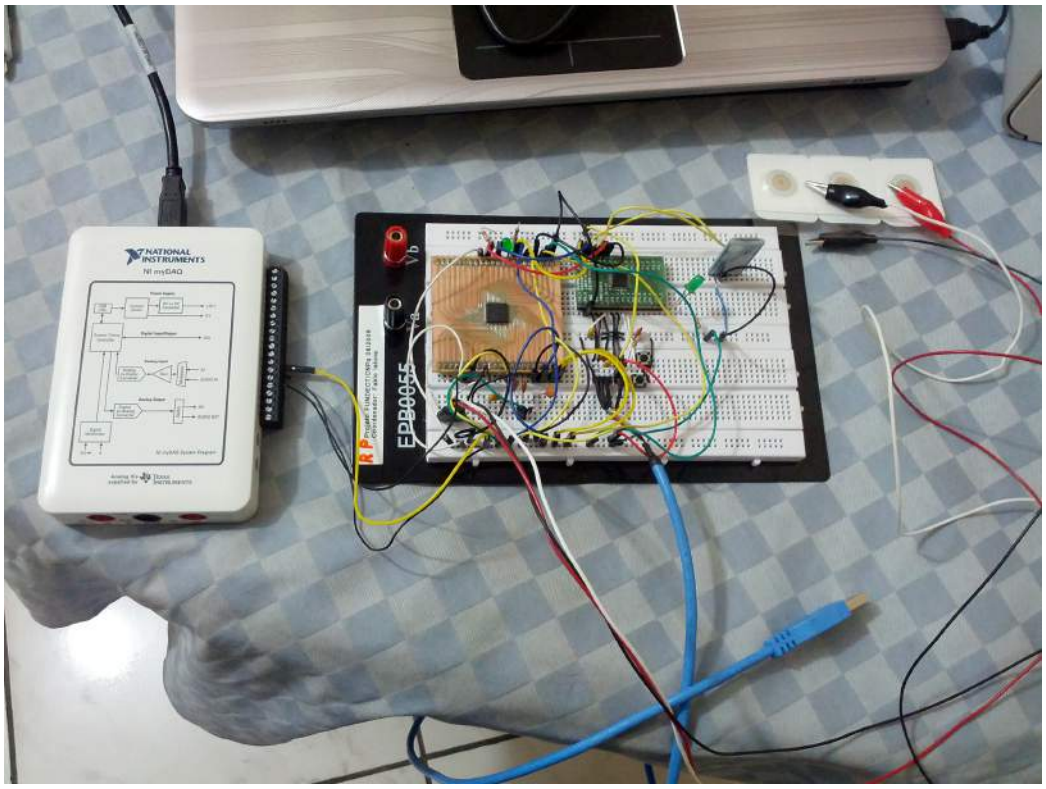

Figura 3.3: À esquerda encontra-se o osciloscópio virtual, no centro a matriz de contatos com os dispositivos utilizados no circuito e à direita, os eletrodos usados nos testes.

Conforme visto na Figura 3.3, para utilizar a FPAA em uma matriz de contatos, foi necessário construir uma placa de circuito impresso adaptadora, pois o invólucro da FPAA é do tipo QFP (SMD). O *layout* dessa placa adaptadora (QFP para DIP) foi confeccionado no aplicativo KiCad [55] e a fabricação da placa foi executada em uma prototipadora ProtoMat E33.

O *Smartphone* utilizado foi o *Samsung Galaxy Nexus I9250* que possui como principais especificações: CPU *dual-core* ARM Cortex-A9 com *clock* de 1,2 GHz, 1 GB de memória RAM, 16 GB de memória de dados, *Bluetooth* 3.0 compatível com as versões anteriores, Android 4.0 nativo (atualizável para 4.3), entre outras. Já o *Notebook* utilizado foi o *HP Pavilion dv6-3181nr Entertainment*, cuja configuração principal é: processador i7 1,6GHz, 8GB de memória RAM e sistema operacional *Windows 8 Pro 64 bits*.

A seguir, o circuito responsável pelo condicionamento do sinal de ECG na FPAA será descrito.

3.1.1 Circuito de ECG na FPAA

A partir da Figura 2.6, na Etapa 2, observam-se os blocos necessários para o condicionamento do sinal. A ferramenta *AnadigmDesigner2* foi utilizada para criar a configuração (Figura 3.4) que implementa as funções analógicas necessárias para o sistema proposto.

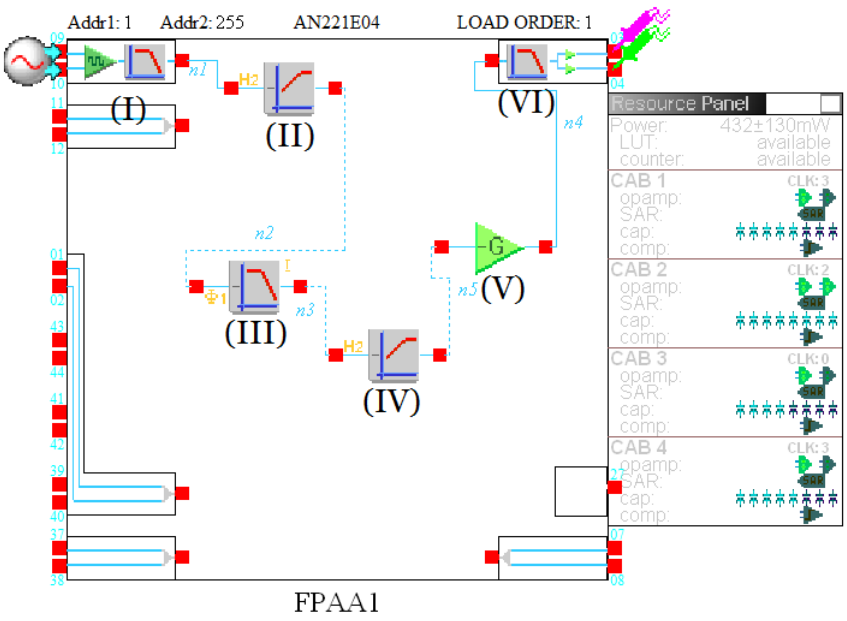

Figura 3.4: Configuração desenvolvida no *software* *AnadigmDesigner2* para realizar o condicionamento do sinal no SERAS-ECG. Este circuito foi desenvolvido selecionando e roteando os vários CAM utilizados.

A Figura 3.5 mostra que o *clock* aplicado na FPAA é de 100 kHz (gerado pela porta P0_7 do microcontrolador, conforme explicado anteriormente), mas o mesmo pode ser reduzido através de divisores de *clock* internos. Os divisores utilizados no projeto foram: $clock0 = 4$, $clock1 = 1$, $clock2 = 8$ e $clock3 = 64$. Conforme o *clock* é alterado, as frequências de corte e ganho de cada CAM também se alteram, definindo-se o *clock* principal para 100 kHz com base nos requisitos do projeto e testes realizados.

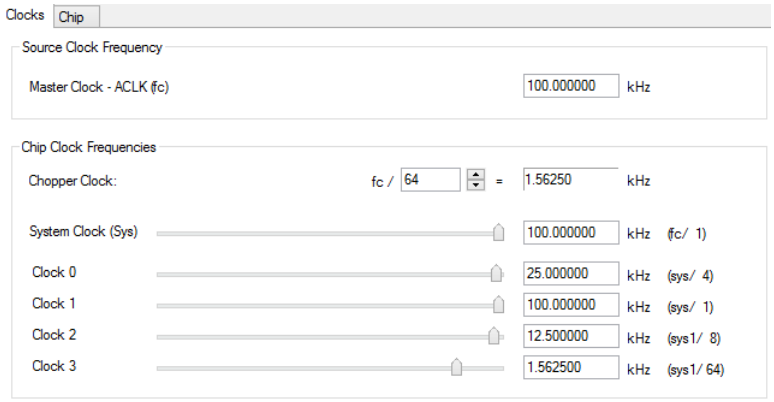

Figura 3.5: Valores de todos os *clocks* utilizados na configuração que realiza o condicionamento do sinal de ECG. Observa-se que o *Chopper* e o *System Clock* foram definidos, de acordo com o *clock* aplicado na VPAA (*Master Clock*), enquanto que os divisores de *clocks* internos utilizaram como base o valor do *System Clock*.

Além da faixa permitida para a frequência de corte e o ganho dos CAMs, os *clocks* escolhidos podem injetar ruídos no circuito da VPAA. Para evitar tais ruídos, a fabricante recomenda que sejam utilizados divisores múltiplos de 2^n [56].

A Tabela 3.1 descreve as principais características dos CAMs utilizados no projeto.

Tabela 3.1: CAMs e parâmetros utilizados.

CAM	Nome	Descrição	Parâmetros	Clocks
I	Célula de Entrada	Amplificador diferencial com baixo *offset* e filtro passa-baixas	Ganho: 16; Frequência de corte: 76 kHz	*Chopper clock*: 1,5625 kHz
II	Filtro Bilinear	Filtro passa-altas	Frequência de corte: 0,781 Hz; Ganho: 1	1,563 kHz (*Clock* 3)
III	Filtro Biquadrático	Filtro passa-baixas com saída invertida	Frequência de corte: 40 Hz; Ganho: 32,5	12,5 kHz (*Clock* 2)
IV	Filtro Bilinear	Filtro passa-altas	Frequência de corte: 0,781 Hz; Ganho: 1	1,563 kHz (*Clock* 3)
V	Amplificador Inversor	Aplica um ganho	Ganho: -1	25 kHz (*Clock* 0)
VI	Célula de Saída	Saída diferencial e filtro passa-baixas	Frequência de corte: 76 kHz	

Em relação aos filtros utilizados, o filtro bilinear é de primeira ordem, enquanto que o filtro biquadrático é de segunda ordem. O filtro biquadrático utiliza a aproximação *Butterworth*. Vale relembrar a existência da ferramenta integrada à *AnadigmDesigner2* para a construção de filtros, denominada *AnadigmFilter*.

A respeito dos parâmetros escolhidos em cada CAM do circuito da FPAA, no CAM I, o ganho e a frequência de corte tiveram os menores valores possíveis selecionados. O baixo ganho deve-se à possibilidade do amplificador saturar devido ao potencial de meia-célula dos eletrodos (realizaram-se testes para ajustar o ganho), já o filtro passa-baixas faz parte da célula de entrada e remove os ruídos de alta frequência. No CAM II, o filtro passa-altas elimina a componente CC do sinal (potencial de meia-célula dos eletrodos) permitindo uma amplificação posterior do sinal sem saturar os amplificadores. A frequência de corte selecionada foi a menor possível. No CAM III, a frequência de corte foi escolhida por intermédio da norma [57]. Esse filtro passa-baixas possui a função de atenuar a amplitude das frequências acima da frequência de corte, servindo como filtro *anti-aliasing*. O CAM IV elimina as possíveis tensões de *offset* (componentes CC) inerentes aos amplificadores, antes da aplicação do ganho final. O CAM V serve para proporcionar o ganho final, permitindo a alteração do ganho total do circuito de condicionamento do sinal de ECG. Utilizou-se um amplificador inversor para que a polaridade na saída da FPAA seja igual a da entrada, já que o CAM III tem sua saída invertida. O CAM VI é uma célula de saída que possui um filtro passa-baixas cuja frequência de corte foi configurada no menor valor possível.

3.1.2 Programação do *Firmware*

Para a transferência dos *firmwares* para o MCU utilizou-se um *firmware bootloader* - que ocupa 5 kB da memória *FLASH* do microcontrolador e permite a gravação de *firmwares* utilizando-se a porta USB do computador -, e o aplicativo *USBbootloader* instalado em um IBM-PC (*Windows 8*), sendo ambos fornecidos pelo fabricante do microcontrolador. O *firmware bootloader* foi gravado previamente no microcontrolador utilizando-se o adaptador/gravador serial EC2 (*Silicon Laboratories*).

Para a programação em C do MCU foi utilizado o ambiente integrado de desenvolvimento (IDE) *Silicon Laboratories 4.50*. Esta IDE permite criar programas para MCUs da família MCS-51 utilizando-se o compilador *open-source Small Device C Compiler* (SDCC). O compilador SDCC é distribuído sob licença *General Public License* (GPL) e foi utilizada a versão 3.2.0.

Anteriormente à descrição do *firmware*, dois temas são necessários para o correto entendimento do fluxograma: a memória *Flash* interna do microcontrolador e o cálculo de CRC.

Como foi apresentado anteriormente, o microcontrolador C8051F320 possui 16 kB de memória *Flash* interna reprogramável, a qual é organizada em páginas contendo 512 *Bytes* cada. Para realizar qualquer operação na memória *Flash*, necessita-se escrever no registrador FLKEY dois códigos-chave em sequência (0xA5 e 0xF1), antes da realização da operação na *Flash*. Caso os códigos-chave sejam incorretos, as operações na *Flash* serão desabilitadas até o próximo *reset* do microcontrolador.

Além disso, para efetuar uma operação de escrita, precisa-se executar uma operação para apagar as páginas que serão utilizadas. Após apagá-las, os códigos-chave precisam ser escritos novamente, para posteriormente realizar a escrita. Reforça-se que estes passos não são necessários na operação de leitura.

A Figura 3.6 ilustra o mapa da memória *Flash*, apresentando a área reservada, o *Lock Byte* e as páginas destravadas, as quais podem ser manipuladas pelo *firmware*.

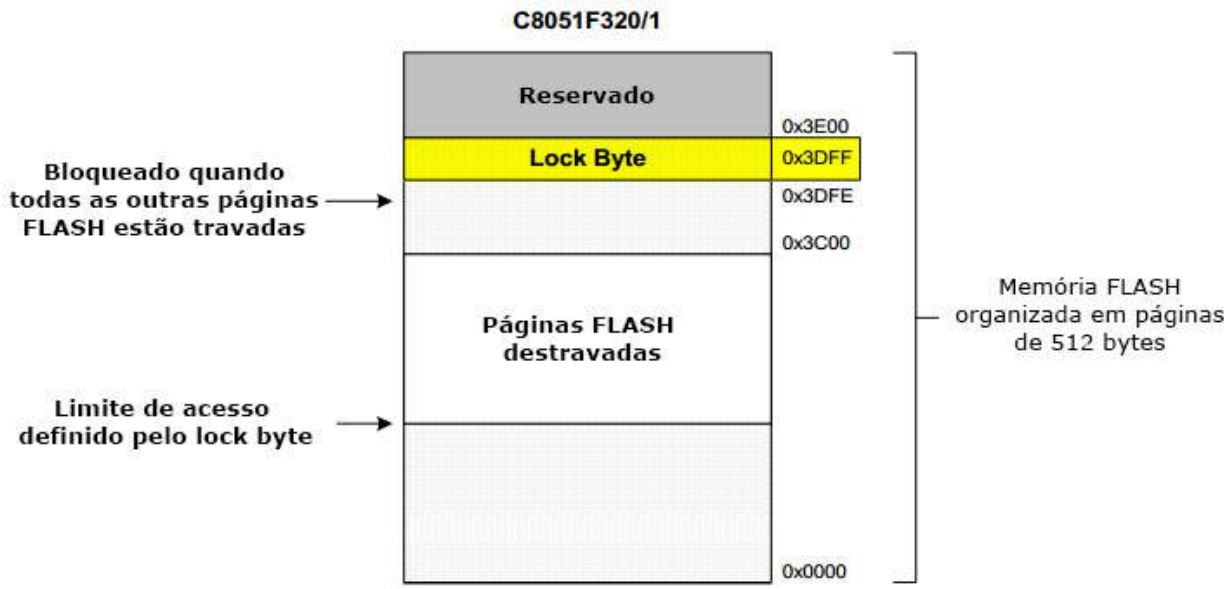

Figura 3.6: Mapa da memória *Flash* contendo a área reservada, o *Lock Byte* (oferece proteção através do travamento de páginas) e as páginas destravadas (utilizadas sem a proteção adicional).

Já o CRC - utilizado no SERAS-ECG - possui como finalidade a detecção de erros nas transmissões dos *Bytes* de configuração da FPAA. A ferramenta *AnadigmDesigner2* possibilita gerar *Bytes* CRC, para posterior verificação da integridade dos *Bytes* de configuração, mas devido à pouca documentação existente a respeito do algoritmo CRC16 implementado na ferramenta, além da existência de diversos formatos para a implementação do CRC, preferiu-se implementar o algoritmo CRC16-ANSI (cujo o polinômio é $x^{16} + x^{15} + x^2 + 1$). A partir deste algoritmo, aumenta-se a confiabilidade na transmissão dos dados enviados pelo *smartphone* através do *Bluetooth*, para o microcontrolador. Vale ressaltar que além desse CRC implementado, a tecnologia *Bluetooth* garante a confiabilidade através de 2 níveis de CRC, retransmitindo os pacotes perdidos ou corrompidos [58].

Para o melhor entendimento, a Figura 3.7 resume as principais características a respeito dos *firmwares* e *softwares* desenvolvidos.

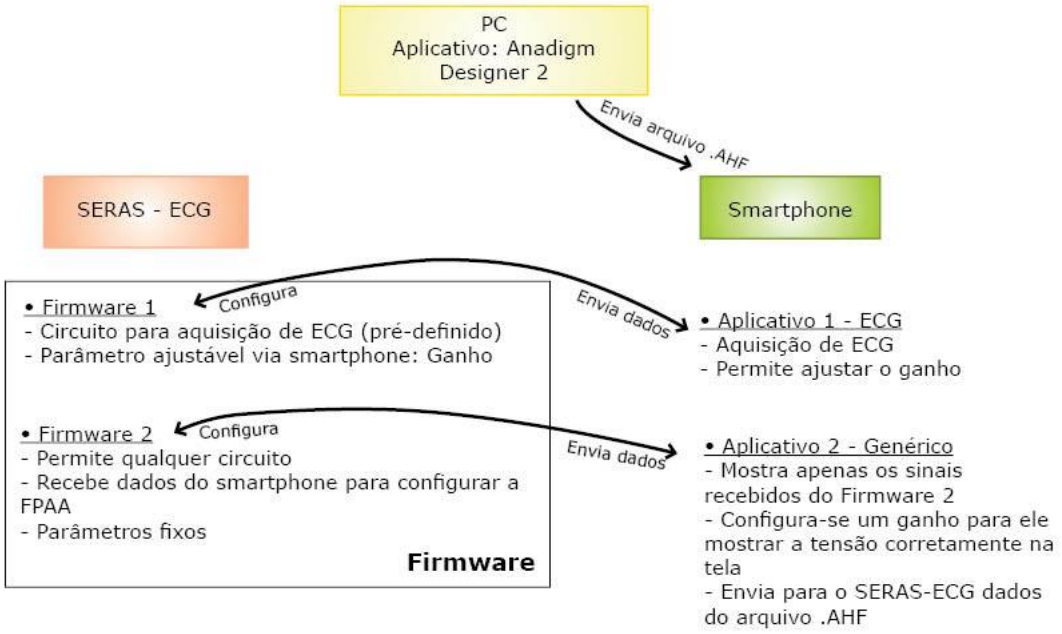

Figura 3.7: Ilustração resumindo os *firmwares* e *softwares* desenvolvidos.

A Figura 3.8 apresenta o fluxograma do *firmware* desenvolvido em linguagem C. Os códigos-fonte do *firmware* encontram-se no Apêndice.

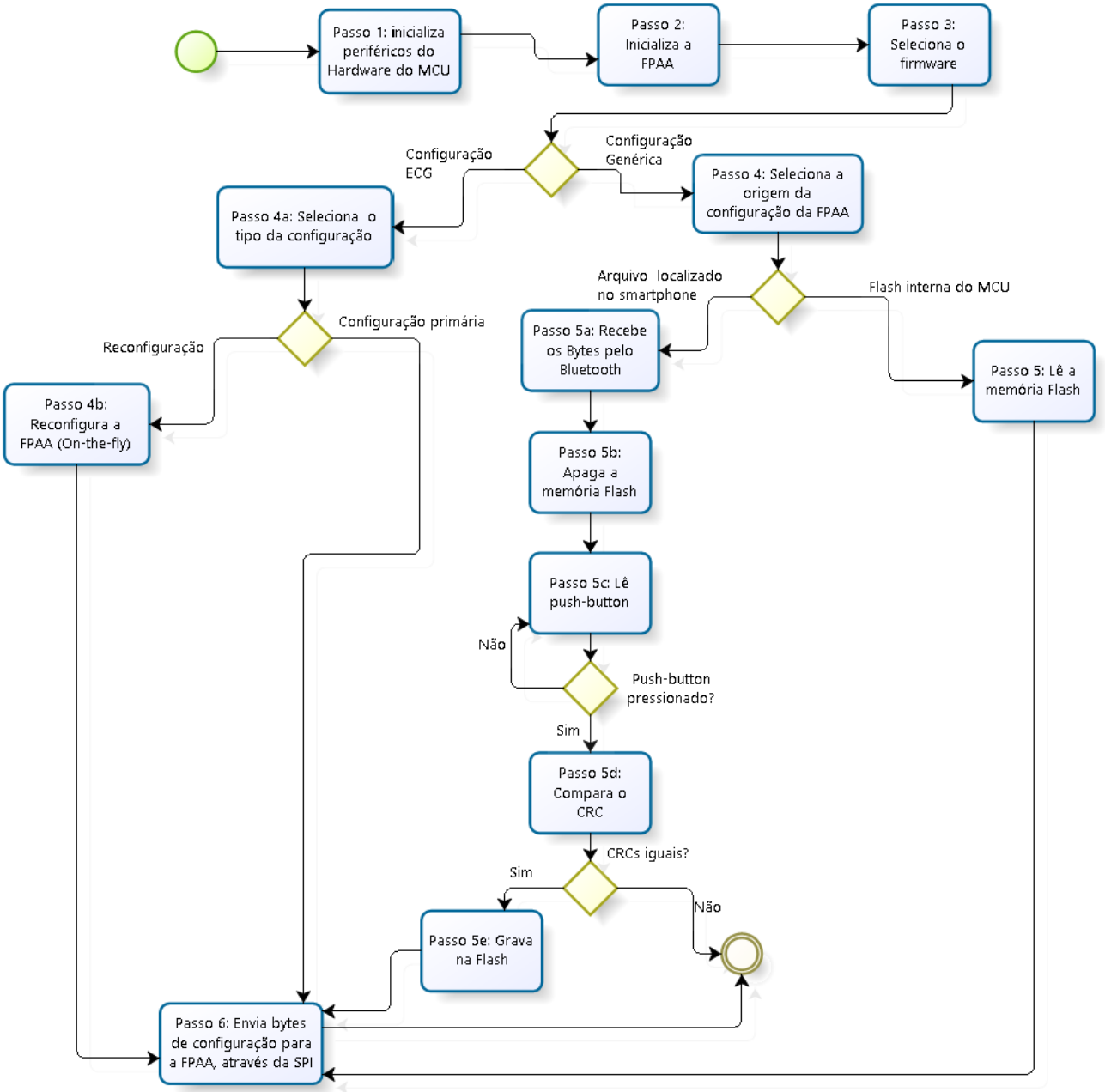

Figura 3.8: Fluxograma do *firmware* desenvolvido.

Os passos realizados pelo *firmware* são descritos com mais detalhes a seguir:

- No primeiro passo, todos os periféricos do *hardware* são inicializados (*Programmable Counter Array*, Temporizador, UART, ADC, Portas de E/S, Oscilador e Interrupções). Para facilitar o entendimento do código, o arquivo "*Registradores.c*" (presente no Apêndice) contém todos os registradores e seus devidos valores necessários para inicializar os periféricos. Os valores aplicados em cada registrador foram baseados de acordo com a configuração de cada periférico. O *Programmable Counter Array* foi configurado utilizando o *clock* do sistema como fonte de base de tempo, o módulo 0 de captura/comparação com frequência de saída igual a 100 kHz e o *watchdog* desabilitado.

Foram utilizados 2 temporizadores, sendo um para gerar a taxa de transmissão de 9615 bps da UART (modo de operação com 8 bits *auto-reload*) e o outro, para definir a taxa de amostragem em 480 amostras/segundo (modo de operação com 16 bits *auto-reload*). A UART foi configurada no modo 8 bits e com permissão para realizar a recepção de dados. O ADC foi habilitado com o modo de conversão, inicializado através de um *overflow* do temporizador 2, utilizando a porta P2_4 como entrada positiva e a tensão GND como entrada negativa. As portas de E/S foram configuradas como digitais, com exceção da porta P2_4 que foi configurada como analógica (realiza a conversão do ADC). O oscilador configurou o *clock* do sistema em 12 MHz. Já as interrupções foram desabilitadas visando configurar a FPAA, antes de transmitir os valores do ADC pela UART para o módulo *Bluetooth*;

- No passo 2, o procedimento "*InitFPAA()*" é invocado para inicializar os registradores internos (DIN e DCLK) da FPAA e, realiza-se um *reset* forçado na FPAA para apagar totalmente a memória SRAM, eliminando possíveis configurações armazenadas;

- No passo 3, o *firmware* fica no modo *polling* aguardando a comunicação com o aplicativo Android. Após a comunicação ser realizada, o *firmware* selecionado pode ser a versão genérica (permite configurar a FPAA através de qualquer configuração criada no *software AnadigmDesigner2*) ou a versão ECG (configura a FPAA com o circuito específico para condicionar o sinal de ECG);

- No passo 4, o *firmware* aguarda o usuário informar se a configuração desejada encontra-se presente em um arquivo com extensão *.AN* (localizado no *smartphone*) ou na memória *Flash* do microcontrolador. Essa informação é transmitida através da seleção realizada no aplicativo Android.

  - Caso a configuração presente na memória *Flash* seja selecionada, o passo 5 é executado. Com isso, o vetor de *Bytes* da configuração armazenado na *Flash* é transferido para a FPAA (passo 6);
  - Se o usuário desejar que um arquivo *.AN* seja selecionado, os *Bytes* presentes no arquivo *.AN* são enviados para o microcontrolador através do *Bluetooth*, e no passo 5a, ocorre o completo recebimento do vetor de configuração. Posteriormente, no passo 5b, duas páginas da memória *Flash* são apagadas (totalizam 1024 *Bytes*, um espaço suficiente para armazenar qualquer configuração da FPAA). No passo 5c, o *push-button* deve ser pressionado para prosseguir com a configuração. A seguir, no passo 5d, o *firmware* realiza o cálculo do CRC dos *Bytes* recebidos e compara-os com o CRC recebido. Caso os CRCs estejam corretos (idênticos) realiza-se a gravação do vetor recebido na memória *Flash* (passo 5e). No último passo, se a configuração da FPAA foi realizada com sucesso, o caractere 's' é transmitido para o aplicativo Android, caso contrário, uma indicação de falha (através do envio do caractere 'n') é efetuada, exigindo que o microcontrolador seja reiniciado.

- De maneira alternativa ao passo 4, no passo 4a, o usuário deve informar (através do aplicativo Android) se deseja realizar a configuração primária na FPAA. Caso a configuração primária seja selecionada, o passo 6 é executado e os *Bytes* são enviados para a FPAA entrar em funcionamento. Caso a reconfiguração *On-the-fly* seja preferida

(passo 4b), o *firmware* aguardará os *Bytes* enviados pelo *smartphone*, os quais são responsáveis pela alteração no ganho do circuito.

3.1.3 Programação do Aplicativo Android

A programação do aplicativo Android foi realizada na IDE *Eclipse Juno* com o Android *Software Development Kit* (SDK) 4.2.2 (API 17). Vale ressaltar que foi instalado no Eclipse o pacote *Google USB Driver*, sendo este compatível com o *Smartphone Samsung Galaxy Nexus* (GT-I9250) utilizado no trabalho. Este pacote permite que os aplicativos sejam testados diretamente no *Smartphone* através do cabo USB. Isto garante os testes utilizando *Bluetooth*, pois o emulador não suporta o *Bluetooth*.

Foram desenvolvidas duas aplicações Android, denominadas "*ECG*" e "*Genérico*", sendo ambas responsáveis pela configuração da FPAA, visualização e armazenamento dos sinais. A Figura 3.9 apresenta a tela inicial do aplicativo Android. Deve-se destacar que os dois aplicativos possuem a mesma interface gráfica.

Figura 3.9: Tela inicial do aplicativo Android. Pode-se observar que o usuário deve selecionar entre visualizar os sinais de ECG em tempo real ou a partir de um arquivo, ou configurar a FPAA.

A diferença entre as duas aplicações deve-se ao fato do aplicativo "*ECG*" utilizar apenas a configuração do circuito de ECG, além de possibilitar a reconfiguração parcial do circuito, através da implementação de um algoritmo que altera a configuração da FPAA, enquanto que o aplicativo "*Genérico*" permite a reconfiguração completa do circuito da FPAA e a possibilidade do usuário informar o ganho total da FPAA. Através da informação do ganho total, o aplicativo imprime as tensões limites superior e inferior. A partir do ganho informado, a leitura do ADC é convertida para Volts. As Figuras 3.11 e 3.10 ilustram um exemplo do desenho das tensões na tela.

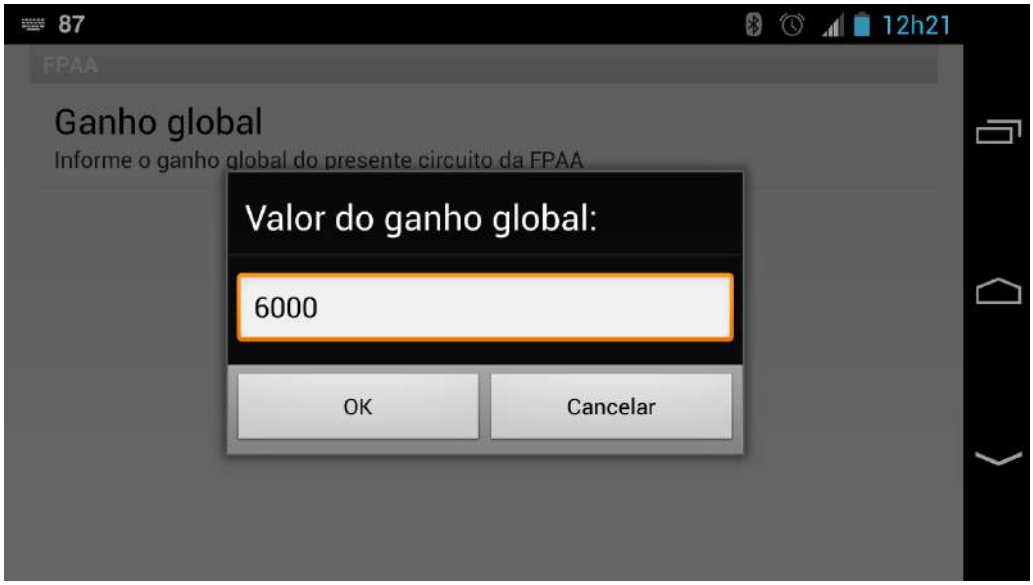

Figura 3.10: Tela da caixa de texto com o ganho total informado pelo usuário.

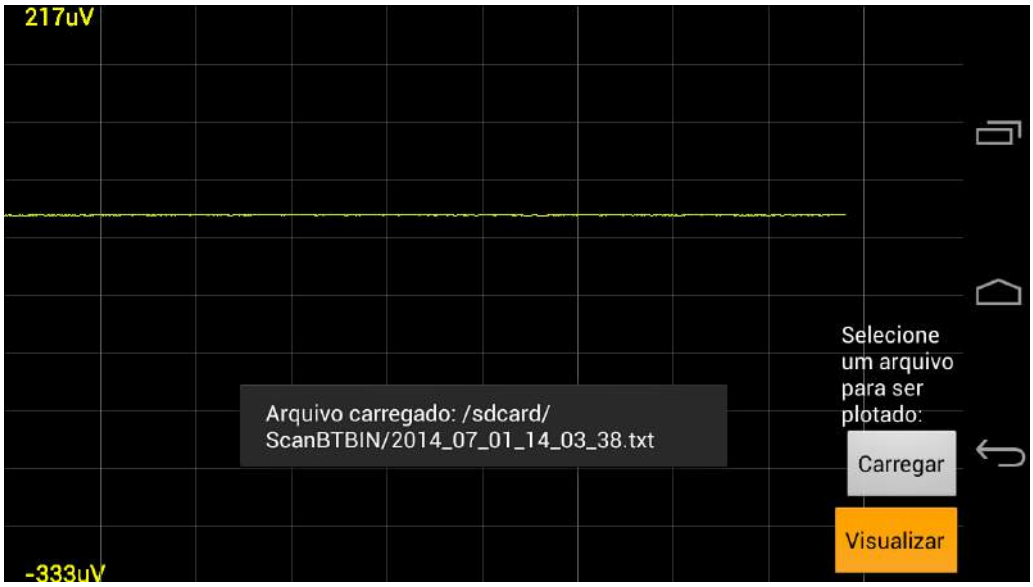

Figura 3.11: Limites superior e inferior com as suas respectivas tensões. Para carregar as tensões salvas no momento da aquisição, existe um arquivo de texto que contém o ganho do respectivo registro armazenado.

Através da ferramenta *AnadigmDesigner2* foram gerados os seguintes arquivos: *API*.c, *API*.h, *CAM*.c e *CAM*.h. Estes arquivos em linguagem C possuem todo o código necessário para reconfigurar a FPAA através do método algorítmico, entretanto, o código gerado pode ser compactado e simplificado para reduzir o consumo de memória [56].

Para criar o código compactado utilizado no SERAS-ECG, realizaram-se as seguintes etapas:

1. Geração do código em linguagem C do circuito ECG (método algorítmico), a partir da ferramenta *AnadigmDesigner2*;

2. Criação de um novo *chip* idêntico ao circuito ECG, alterando-se apenas o valor do ganho do CAM Amplificador Inversor, tornando alguns valores do vetor de configuração diferentes no segundo *chip*;

3. Geração do arquivo de código em linguagem C, do método dirigido a estados (*AnadigmDesigner2*);

4. Identificação das posições dos vetores (gerados na etapa anterior), que tiveram seus valores alterados; e

5. Desenvolvimento de uma função baseada nas posições identificadas anteriormente, nos demais valores não alterados e em trechos do código algorítmico, específicos para alterar o CAM desejado.

A Figura 3.12 representa o fluxograma executado pelos aplicativos Android.

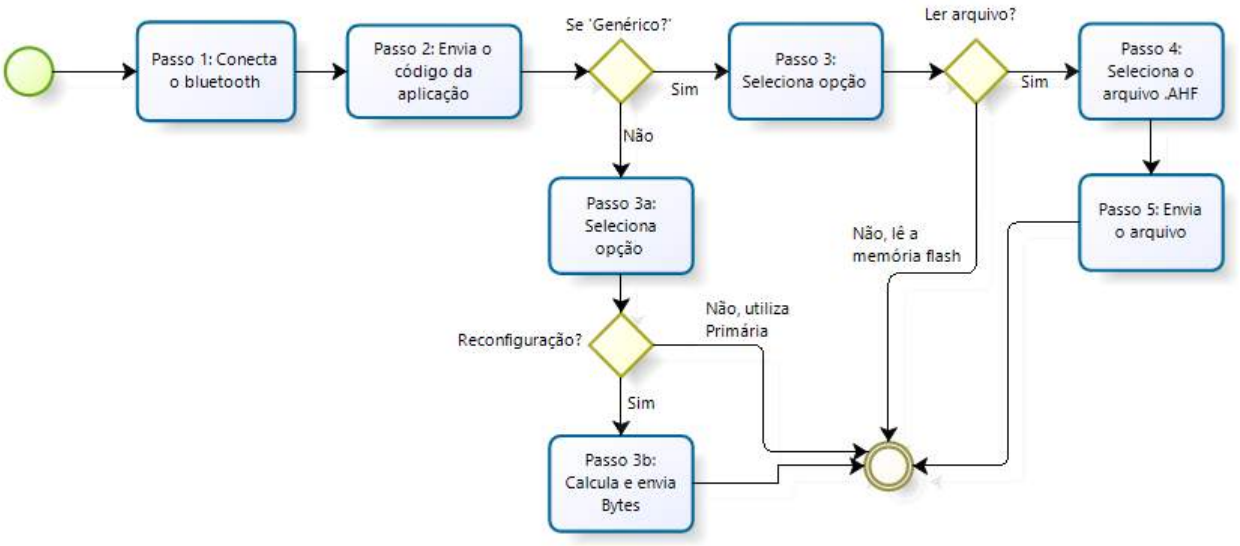

Figura 3.12: Fluxograma dos aplicativos Android desenvolvidos.

O fluxograma da Figura 3.12 é descrito a seguir:

- No primeiro passo ocorre o pareamento do *Bluetooth*;
- Conforme visto no fluxograma do *firmware*, o aplicativo Android envia o código da aplicação, possibilitando a escolha do *firmware* que será executado (passo 2);
  - Caso o código seja do aplicativo "*Genérico*", no passo 3, o usuário pode selecionar entre a configuração da FPAA através de dados já gravados na memória *Flash* do microcontrolador ou a partir de um arquivo localizado no *smartphone*;

  - Caso contrário (aplicativo "*ECG*"), o aplicativo permite a escolha entre reconfigurar o ganho da VPAA (passo 3b) ou utilizar a configuração primária do circuito ECG.

- Se o usuário optou pela configuração via arquivo, deve-se selecionar o arquivo .HEX desejado (passo 4) e enviar os dados da configuração para o microcontrolador (passo 5).

Visualização dos sinais

A visualização dos sinais de ECG na tela do *smartphone* é realizada através dos métodos da classe *"Canvas"*, que são utilizados para controlar a interface gráfica do aplicativo. Pelo fato dos aplicativos desenvolvidos necessitarem de atualizações na tela constantemente, exigiu-se a utilização da subclasse especial *"SurfaceView"*.

A *"SurfaceView"* é uma classe que oferece uma área (superfície) de desenho dedicada dentro da classe que representa os componentes gráficos de uma interface gráfica, denominada *View*. O objetivo é fornecer esta área de desenho para que uma *thread* secundária da aplicação se responsabilize pelo gerenciamento da atualização gráfica do objeto *"SurfaceView"*, evitando que a aplicação espere que a hierarquia da *"View"* esteja disponível para desenhar (geralmente, esta espera causa a finalização da execução do aplicativo, devido ao tempo em que a aplicação fica sem resposta). Como os aplicativos desse trabalho realizam atualizações da interface gráfica em tempo real, uma *thread* separada foi necessária para gerenciar um objeto *"SurfaceView"* e executar o método *Draw* (da classe *android.graphics.Canvas*) na máxima velocidade que a *thread* permite.

A Figura 3.13 apresenta o funcionamento do sistema de coordenadas x e y empregado para desenhar na tela. A partir dela, observa-se que o ponto $x = 0$ e $y = 0$ localiza-se no topo da tela (o eixo y é invertido em relação à necessidade da aplicação).

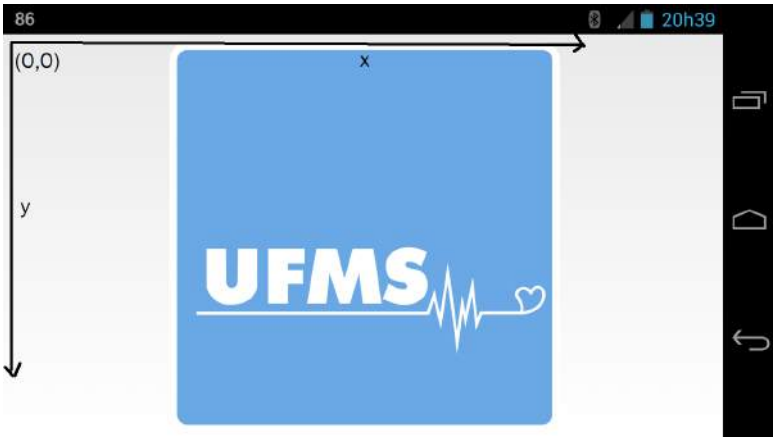

Figura 3.13: Sistema de coordenadas utilizado na tela de aplicativos Android.

Através do cálculo da equação da reta e, baseado na altura máxima que o aplicativo permite, obteve-se a seguinte equação: $y = (-2.505882352941176 * sample) + 639$, a qual é

utilizada para desenhar na tela qualquer amostra recebida. Para o melhor entendimento, o seguinte exemplo é válido, pois o ADC foi configurado com 8 bits de resolução. Com isso, os valores das amostras recebidas pelo *smartphone* variam entre 0 e 255. Através desses valores, a equação realiza a conversão para desenhar corretamente na tela, onde a amostra 0 é desenhada na posição 639 do eixo y, e a amostra 255 é desenhada na posição 0 do mesmo eixo.

3.2 Considerações Finais

Neste capítulo foram apresentados todos os detalhes referentes ao sistema desenvolvido, denominado SERAS-ECG. Além do diagrama de blocos do sistema e do esquema eletrônico montado em uma matriz de contatos, todos os blocos necessários para o condicionamento do sinal, presentes no circuito de ECG da FPAA, foram explicados minuciosamente. Para finalizar o capítulo, os passos envolvidos no desenvolvimento do *firmware* e do *software* foram descritos de acordo com as suas particularidades, como as ferramentas de *software* utilizadas e os passos executados em seu fluxograma. O próximo capítulo apresenta os testes realizados no SERAS-ECG e os resultados obtidos.

Capítulo 4

Resultados e Discussão

O presente capítulo apresenta os procedimentos de teste e os resultados obtidos no desenvolvimento do SERAS-ECG, visando compará-los com os resultados esperados. Na Seção 4.1 são descritos os testes realizados no circuito para a aquisição dos sinais de ECG, sendo estes, baseados nas recomendações da norma ANSI/AAMI EC13:2002 [57], específica para monitores cardíacos. Na Seção 4.2 foram analisados cinco circuitos distintos de filtros criados através da ferramenta *AnadigmFilter*, os quais possibilitaram a avaliação de diferentes configurações implementadas na FPAA. Já na Seção 4.3 são apresentadas algumas considerações finais deste capítulo.

4.1 Teste da Configuração para Condicionamento do Sinal de Eletrocardiograma

Os testes no circuito de ECG foram realizados tendo como base as recomendações da norma ANSI/AAMI EC13:2002 [57]. Essa norma foi publicada em 2002 e definiu os requisitos mínimos de segurança e desempenho para alguns equipamentos que monitoram o ECG. Os testes realizados neste trabalho são baseados nos requisitos especiais para monitores que exibem as formas de onda do ECG.

A seguir, serão descritos os testes e seus respectivos resultados. Vale ressaltar que os testes abordados nas Subseções 4.1.1 a 4.1.5 utilizaram o circuito de testes ilustrado na Figura 4.1.

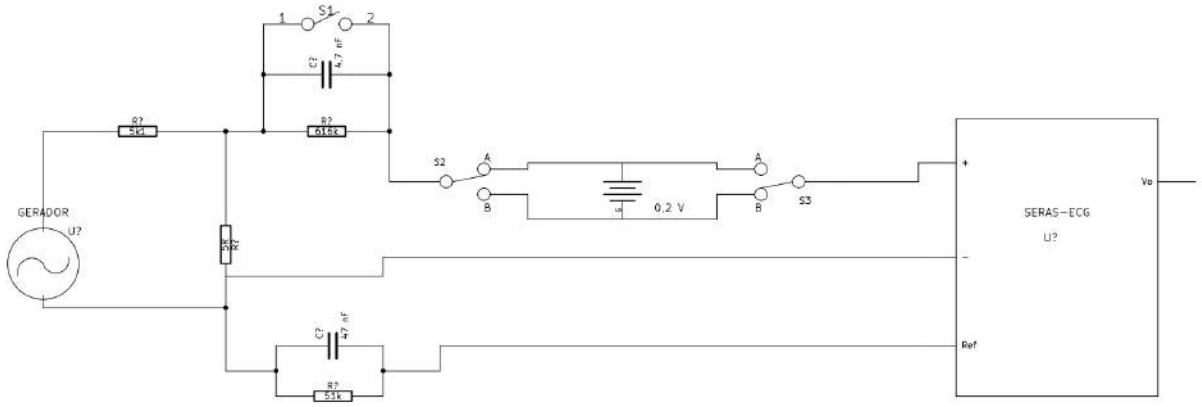

Figura 4.1: Modelo diagramático do circuito de testes utilizado.

4.1.1 Teste 1 - Faixa de Entrada

Para esse teste, a norma define que o dispositivo deve ser capaz de responder e exibir as tensões diferenciais que variam na faixa de -5 mV a +5 mV. Quando uma tensão de +200 mV e -200 mV (simulando o potencial de meia-célula) for inserida em série com o sinal de ±5 mV, a variação na amplitude do sinal registrado deve ser no máximo ±10%.

A partir do circuito mostrado na Figura 4.1 foram realizados três testes para verificar a faixa de entrada. O gerador de sinais foi ajustado para gerar uma onda senoidal de 16 Hz da seguinte forma:

I - Registro sem potencial de meia célula (chave S1 fechada, S2 e S3 na posição A);

II - Registro com potencial de meia célula (S1 fechada, S2 em A e S3 em B); e

III - Registro com potencial de meia célula e polaridade invertida (S1 fechada, S2 em B e S3 em A).

A Tabela 4.1 apresenta os dados obtidos.

Tabela 4.1: Análise realizada pelos picos e vales registrados (em NCAD$_{pp}$), os quais foram verificados se existiam distorções.

Registro I (NCAD$_{pp}$)	Registro II (NCAD$_{pp}$)	Registro III (NCAD$_{pp}$)	Registro I - Registro II (NCAD$_{pp}$)	Registro I - Registro III (NCAD$_{pp}$)	Diferença percentual entre I e II (%)	Diferença percentual entre I e III (%)
191	191	190	0	1	0	0,52
192	191	191	1	1	0,52	0,52
191	191	191	0	0	0	0
193	191	191	2	2	1,03	1,03
192	191	191	1	1	0,52	0,52
192	191	190	1	2	0,52	1,04
193	191	191	2	2	1,03	1,03
193	192	191	1	2	0,51	1,03
191	191	191	0	0	0	0
191	191	191	0	0	0	0
192	192	192	0	0	0	0
192	192	190	0	2	0	1,04
192	191	191	1	1	0,52	0,52
192	191	192	1	0	0,52	0
193	191	191	2	2	1,03	1,03
191	190	191	1	0	0,52	0
192	191	192	1	0	0,52	0
192	192	192	0	0	0	0
191	190	191	1	0	0,52	0
192	191	191	1	1	0,52	0,52
192	191	192	1	0	0,52	0
192	191	192	1	0	0,52	0
193	191	192	2	1	1,03	0,52
192	192	191	0	1	0	0,52
193	191	192	2	1	1,03	0,52
192	192	191	0	1	0	0,52
191	191	190	0	1	0	0,52
192	191	191	1	1	0,52	0,52
192	191	191	1	1	0,52	0,52
191	191	191	0	0	0	0
$191,9 \pm 0,358(\alpha = 99\%)$	$191,13 \pm 0,255(\alpha = 99\%)$	$191,13 \pm 0,316(\alpha = 99\%)$	$0,76 \pm 0,389(\alpha = 99\%)$	$0,76 \pm 0,366(\alpha = 99\%)$	$0,41 \pm 0,18(\alpha = 99\%)$	$0,41 \pm 0,19(\alpha = 99\%)$

Através dos dados obtidos, verificou-se que não ocorreram distorções/saturações no sinal registrado quando a amplitude de entrada foi de ± 5 mVp. Além disso, na presença de um potencial de meia-célula simulado de 0,2 V, a amplitude de saída não extrapolou o limite aceitável de $\pm 10\%$ do valor registrado sem esse potencial.

4.1.2 Teste 2 - Resposta em Frequência

O teste de resposta em frequência deve satisfazer dois tipos de sinais de entrada, sendo eles resumidos na Tabela 4.2.

Tabela 4.2: Métodos a serem seguidos no teste de resposta em frequência.

Método	Amplitude de Entrada (mVpv)	Frequência e Forma de onda
A	1,0	0,67 Hz a 40 Hz, Senóide
B	1,5	1 Hz, Triangular

No método A, o limite máximo aceitável na saída é de 10% superior em relação à um sinal registrado de 5 Hz. Para o método B, a resposta de saída é obtida para uma onda repetitiva, triangular, com uma largura da base de 200 ms e uma frequência de 1 Hz. No caso do método B, a cada 10 ciclos consecutivos com largura base de 200 ms, geram-se 10 ciclos consecutivos com largura de 20 ms, cujos picos devem apresentar uma amplitude mínima de 75% da amplitude dos picos com base de 200 ms (Figura 4.2).

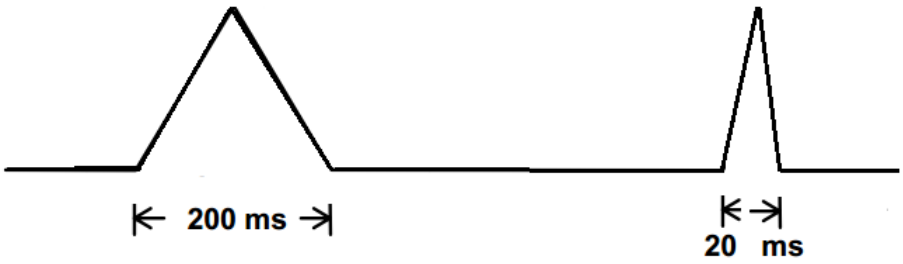

Figura 4.2: Ciclos consecutivos gerados no método B.

Através do método A, observou-se que a amplitude dos sinais registrados de todas as frequências (0,67 Hz, 5 Hz, 10 Hz, 20 Hz e 40 Hz), em relação à tensão da frequência considerada como referência (5 Hz), não excederam mais do que 10% da frequência base. Pelo método B, os dois sinais injetados (com a mesma amplitude) registraram sinais diferentes e as ondas de 20 ms, quando comparadas com as ondas de 200 ms, não ultrapassaram o limite inferior aceitável (75%). Vale ressaltar que a análise envolveu a subtração de cada pico e de cada ponto médio das ondas geradas.

As Tabelas referentes ao teste da resposta em frequência encontram-se no Apêndice A.

4.1.3 Teste 3 - Ruído

De acordo com a norma, o ruído presente no sinal registrado não deve exceder 30 uV, quando referenciado à entrada (RTI). Para a realização desse teste, as entradas da FPAA foram conectadas em série com um resistor de 51 KOhms em paralelo com um capacitor

de 47 nF (Figura 4.3). Vale ressaltar que a diferença aceitável entre o máximo e o mínimo registrados, deve ser inferior a 30 uVpv (RTi), em pelo menos 9 de 10 segmentos de 10 segundos.

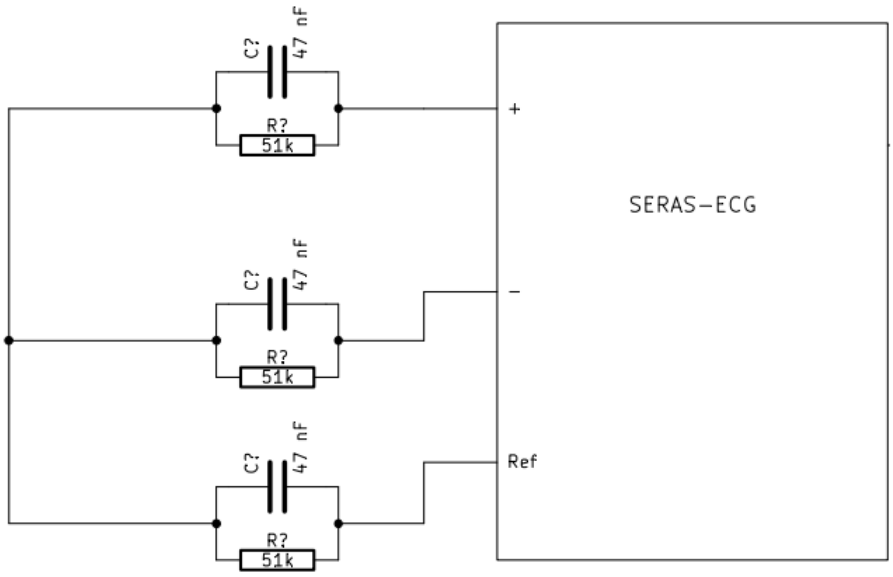

Figura 4.3: Modelo diagramático do circuito utilizado no teste para verificar o ruído presente na entrada da PPAA.

A Tabela 4.3 mostra as diferenças obtidas em 30 segmentos de 10 segundos.

Tabela 4.3: Resultados obtidos através do teste realizado. Observa-se que em todos os segmentos analisados, o ruído presente é superior à diferença aceitável presente na norma.

Segundos	Vin A/D (V)	Vin PPAA (V)	Inferior a 30uVpv?
0 - 10	0,067	6,47912E-05	NÃO
10 - 20	0,081	7,77494E-05	NÃO
20 - 30	0,081	7,77494E-05	NÃO
30 - 40	0,067	6,47912E-05	NÃO
40 - 50	0,081	7,77494E-05	NÃO
50 - 60	0,067	6,47912E-05	NÃO
60 - 70	0,067	6,47912E-05	NÃO
70 - 80	0,067	6,47912E-05	NÃO
80 - 90	0,067	6,47912E-05	NÃO
90 - 100	0,067	6,47912E-05	NÃO
100 - 110	0,067	6,47912E-05	NÃO
110 - 120	0,081	7,77494E-05	NÃO
120 - 130	0,081	7,77494E-05	NÃO
130 - 140	0,067	6,47912E-05	NÃO
140 - 150	0,067	6,47912E-05	NÃO
150 - 160	0,081	7,77494E-05	NÃO
160 - 170	0,054	5,18329E-05	NÃO
170 - 180	0,067	6,47912E-05	NÃO
180 - 190	0,067	6,47912E-05	NÃO
190 - 200	0,067	6,47912E-05	NÃO
200 - 210	0,067	6,47912E-05	NÃO
210 - 220	0,067	6,47912E-05	NÃO
220 - 230	0,067	6,47912E-05	NÃO
230 - 240	0,067	6,47912E-05	NÃO
240 - 250	0,067	6,47912E-05	NÃO
250 - 260	0,067	6,47912E-05	NÃO
260 - 270	0,067	6,47912E-05	NÃO
270 - 280	0,067	6,47912E-05	NÃO
280 - 290	0,067	6,47912E-05	NÃO
290 - 300	0,067	6,47912E-05	NÃO
	$0,069 \pm 0,003 (\alpha = 99\%)$	$67 \pm 3uV (\alpha = 99\%)$	

Conforme observado na Tabela 4.3, o ruído presente no sistema foi superior à 30 uVpv (não satisfazendo à norma), porém, deve-se observar que o sistema foi montado em uma matriz de contatos e que não foi realizada nenhuma análise profunda na configuração do circuito para melhorar esse parâmetro.

4.1.4 Teste 4 - Rejeição em Modo Comum

Esse teste verificou a capacidade do sistema em rejeitar os sinais em modo comum, como os encontrados na superfície do corpo. Para a medição da Taxa de Rejeição em Modo Comum (CMRR) em 60 Hz foram aplicados 2 sinais: 1,0 mVpv / 60 Hz / modo diferencial (Figura 4.1) e 3,8 Vpv / 60 Hz / modo comum (Figura 4.4). Cada registro realizado teve a duração de 60 segundos, sendo que o segundo possui as seguintes particularidades: registro com S1 e S2 fechadas, registro com S1 aberta e S2 fechada, e registro com S1 fechada e S2 aberta.

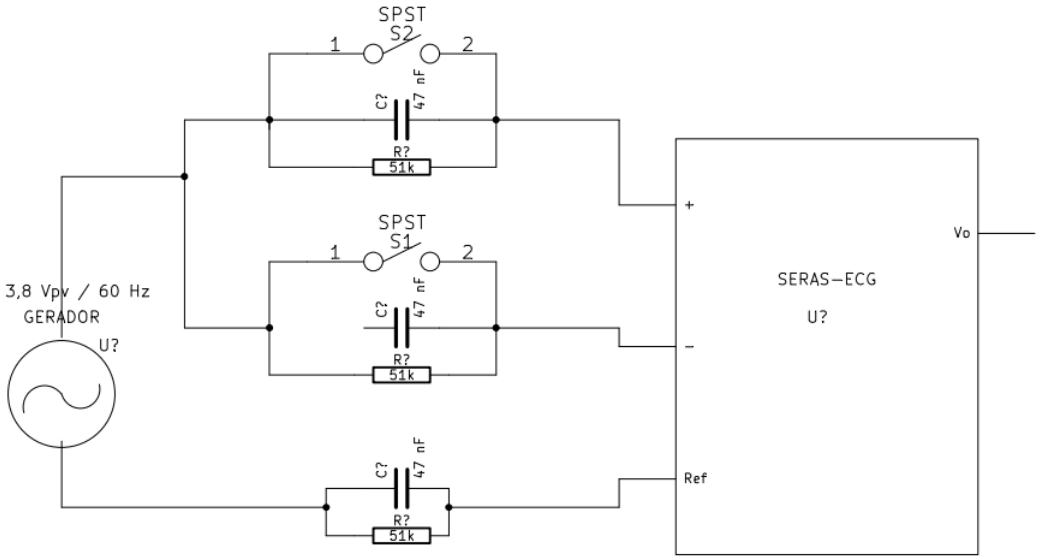

Figura 4.4: Diagrama esquemático do circuito de testes utilizado para medição da CMRR.

Através da análise dos sinais registrados obteve-se a tensão pico a pico para cada registro e calculou-se a CMRR (CMRR = 20 ∗ log(Ganho no modo diferencial / Ganho no modo comum)), que resultou na Tabela 4.4.

Tabela 4.4: Taxa de Rejeição em Modo Comum do SERAS-ECG medida para sinais de 60 Hz.

Método utilizado	Valor (dB)
Chaves fechadas	81,29460378
S1 aberta, S2 fechada	54,76788656
S1 fechada, S2 aberta	63,74089563

A partir desses valores, observou-se que o teste de rejeição em modo comum resultou em um valor adequado de CMRR.

4.1.5 Teste 5 - Impedância de Entrada

Nesse teste simulou-se um aumento na impedância pele-eletrodo, através de um resistor de 0,62 KOhms, em paralelo com um capacitor de 4,7 nF, ligados em série com a entrada positiva (Figura 4.1). Nessa simulação, os sinais registrados (S1 aberta - Figura 4.1) não devem apresentar uma redução na amplitude superior a 20%, em relação a obtida sem a impedância simulada (S1 fechada - Figura 4.1), dentro da faixa de frequências de 0,67 até 40 Hz.

A Tabela 4.5 apresenta os valores dos sinais registrados.

Tabela 4.5: Resultados obtidos pela comparação dos dois sinais. O sinal 1 foi registrado sem a impedância simulada, enquanto que o sinal 2 foi registrado com a simulação da impedância.

Sinal 1 (NCADpv)	Sinal 2 (NCADpv)	20% do Sinal 1 (NCADpv)	Redução (%)	Reduziu 20%?
150	149	30	0,67	NÃO
150	149	30	0,67	NÃO
150	150	30	0,00	NÃO
137	137	27,4	0,00	NÃO
198	190	39,6	4,04	NÃO
199	192	39,8	3,52	NÃO
198	193	39,6	2,53	NÃO
199	192	39,8	3,52	NÃO
198	194	39,6	2,02	NÃO
198	193	39,6	2,53	NÃO
199	194	39,8	2,51	NÃO
202	185	40,4	8,42	NÃO
202	185	40,4	8,42	NÃO
201	187	40,2	6,97	NÃO
202	186	40,4	7,92	NÃO
202	188	40,4	6,93	NÃO
200	186	40	7,00	NÃO
203	186	40,6	8,37	NÃO
202	187	40,4	7,43	NÃO
201	188	40,2	6,47	NÃO
202	186	40,4	7,92	NÃO
164	132	32,8	19,51	NÃO
164	134	32,8	18,29	NÃO
166	133	33,2	19,88	NÃO
164	134	32,8	18,29	NÃO
166	133	33,2	19,88	NÃO
163	134	32,6	17,79	NÃO
166	133	33,2	19,88	NÃO
162	134	32,4	17,28	NÃO
167	134	33,4	19,76	NÃO
163	135	32,6	17,18	NÃO

57	47	11,4	17,54	NÃO
60	49	12	18,33	NÃO
57	46	11,4	19,30	NÃO
55	45	11	18,18	NÃO
58	48	11,6	17,24	NÃO
58	48	11,6	17,24	NÃO
57	46	11,4	19,30	NÃO
59	48	11,8	18,64	NÃO
58	47	11,6	18,97	NÃO
58	48	11,6	17,24	NÃO
56	45	11,2	19,64	NÃO
56	45	11,2	19,64	NÃO
60	49	12	18,33	NÃO
59	49	11,8	16,95	NÃO
57	46	11,4	19,30	NÃO
$141,36 \pm 24,369 (\alpha = 99\%)$	$126,93 \pm 23,89 (\alpha = 99\%)$		$12,2 \pm 2,9 (\alpha = 99\%)$	

Portanto, verificou-se que em nenhum registro ocorreu a redução de 20% no sinal devido à presença de uma elevada impedância pele-eletrodo (simulada).

4.1.6 Teste 6 - Consumo de Energia

O consumo de energia foi medido, desconsiderando os leds presentes no circuito, através de um amperímetro.

A média obtida considerando-se 5 medidas foi de $88{,}6 \pm 0{,}6$ mA (nível de confiança = 99%), resultando em uma potência de 443 ± 3 mW (nível de confiança = 99%).

Cabe observar que o consumo de energia do sistema obtido não envolveu otimizações na configuração do circuito (devido ao tempo restrito para a execução do trabalho), com o objetivo de diminuí-lo. Entretanto, em comparação com alguns trabalhos utilizando FPAA, o consumo obtido foi menor. No trabalho de [31] o sistema consumia aproximadamente 12 W (composto por uma placa de desenvolvimento da FPGA, uma placa de desenvolvimento da FPAA e uma tela LCD), enquanto que no trabalho de [41] o consumo simulado para o condicionamento de sinais de eletromiograma, apenas da FPAA, foi de 700 mW. Cabe observar que os trabalhos [10, 21–23, 32] não mencionaram o consumo de energia.

4.1.7 Teste 7 - Sinais Reais de ECG

O teste em questão envolveu a aquisição de sinais de ECG em um indivíduo real. Para isso, utilizou-se a derivação DI (braço esquerdo com a entrada positiva do medidor e braço

direito com a entrada negativa do medidor) e o aplicativo "*ECG*". A Figura 4.5 mostra o sinal visualizado e armazenado no *Smartphone*.

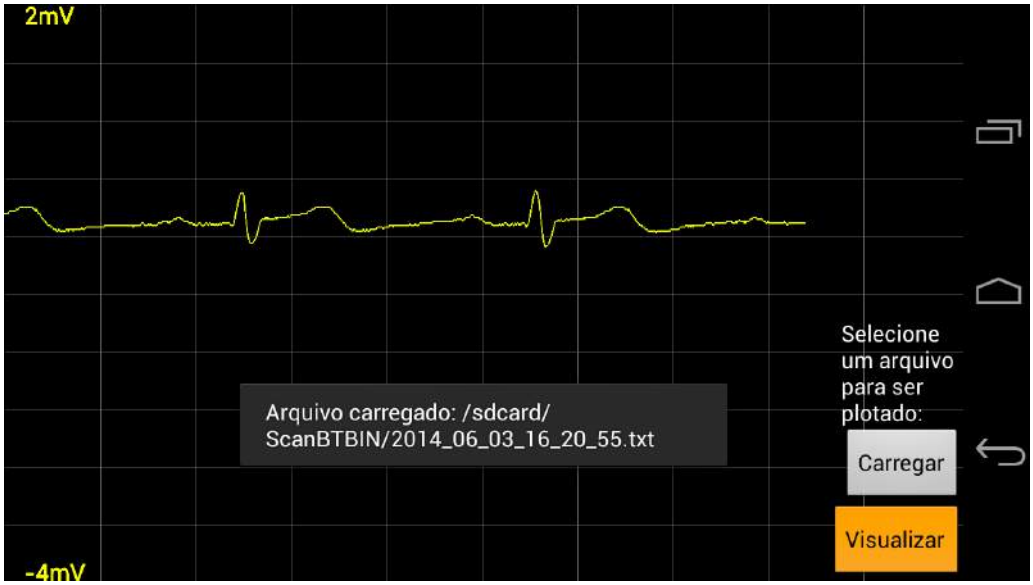

Figura 4.5: Sinal visualizado no *Smartphone* exibindo o sinal de ECG real captado por eletrodos em um indivíduo (Ganho total = 260).

Após a aquisição desse sinal, realizou-se uma reconfiguração do ganho da FPAA para 1040. A Figura 4.6 apresenta a tela do aplicativo que permite a entrada do novo ganho a ser usado na reconfiguração.

Figura 4.6: Tela do aplicativo "*ECG*" referente à reconfiguração da FPAA. Observa-se que o ganho do CAM Amplificador Inversor foi alterado para 4, tornando o ganho total da FPAA igual a 1040.

A Figura 4.7 mostra o sinal de ECG adquirido com o ganho alterado.

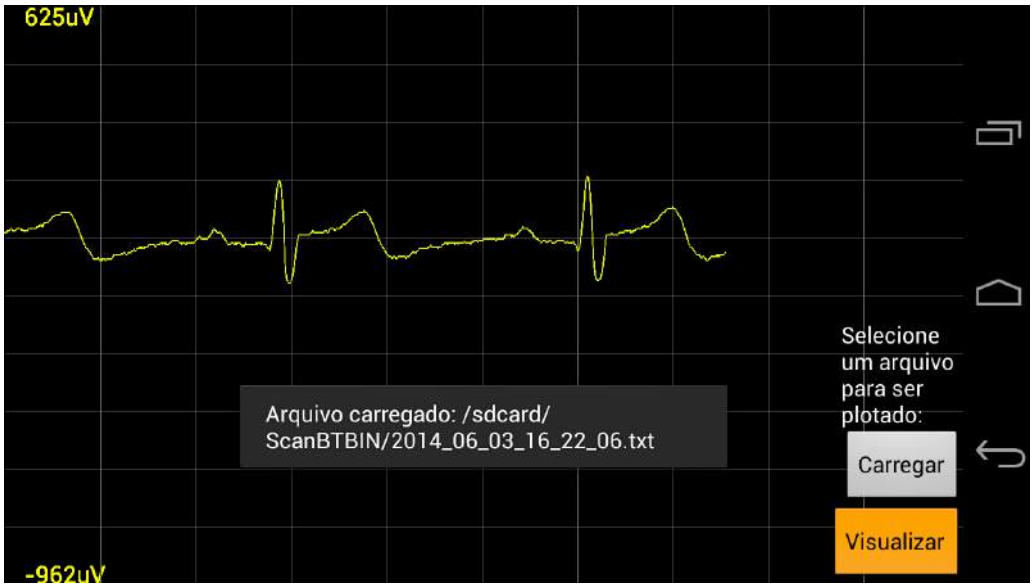

Figura 4.7: Sinal de ECG real registrado com um ganho de 1040.

Os sinais visualizados no aplicativo "*ECG*" apresentaram uma reprodução satisfatória do sinal real captado em um indivíduo, principalmente levando-se em consideração que o circuito foi montado em uma matriz de contatos, situação que o torna mais suscetível aos ruídos eletromagnéticos externos e gerados pelo próprio circuito (parte digital).

4.1.8 Consumo dos recursos internos da FPAA

A Figura 4.8, gerada pela ferramenta *AnadigmDesigner2*, apresenta a capacidade e a utilização da FPAA AN221E04 para o circuito de condicionamento do sinal de ECG. Já a Tabela 4.6 resume as principais informações referentes ao consumo dos elementos internos da FPAA.

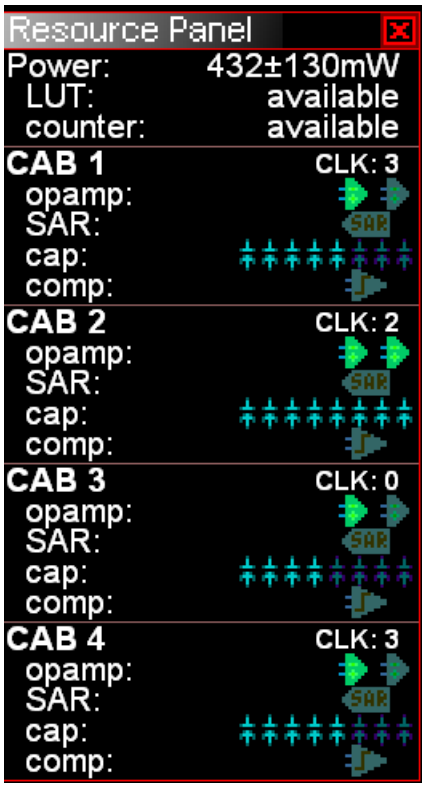

Figura 4.8: Painel mostrando os recursos utilizados pelo circuito de condicionamento de ECG, onde pode-se observar o consumo de cada tipo de elemento em cada CAB.

Tabela 4.6: Resumo das capacidades e utilização da FPAA.

CAB	Op-amps	SAR	Capacitores	Comparador
1	1 de 2	Não	5 de 8	Não
2	2 de 2	Não	8 de 8	Não
3	1 de 2	Não	4 de 8	Não
4	1 de 2	Não	5 de 8	Não

Portanto, através da análise realizada no circuito de condicionamento de ECG, pôde-se observar que do total de 48 elementos, foram utilizados 27, consumindo 56,25% da AN221E04.

4.2 Teste da Configuração Genérica

Para explorar a flexibilidade da FPAA e testar a configuração genérica foram criados cinco filtros distintos na ferramenta *AnadigmFilter*, pois esse tipo de circuito pode ter seu funcionamento verificado pela simples medição da resposta em frequência. Após finalizados, cada filtro foi importado para seu respectivo projeto no *AnadigmDesigner2*, possuindo em comum os seguintes componentes: uma entrada diferencial com um filtro passa-baixas (Fc

= 76 kHz) e ganho = 16, além de uma saída de tensão também com filtro passa-baixas (Fc = 76 kHz) e ganho = 1.

Após a conclusão das cinco configurações da FPAA, realizou-se a geração e transferência dos arquivos *.NFF* para a memória do *smartphone*. Observa-se que cada circuito possui diferentes CAMs, os quais são atribuídos automaticamente através da seleção dos parâmetros de cada filtro na ferramenta *AnadigmFilter*. Vale ressaltar que a quantidade de CAMs e FPAAs variam de acordo com os requisitos do filtro, que todos os filtros projetados visaram a utilização de uma única FPAA, e que a aproximação utilizada foi a *Butterworth*.

A Figura 4.9 mostra os parâmetros utilizados na criação do filtro 1 e a respectiva resposta em frequência visualizada pela ferramenta *AnadigmFilter*.

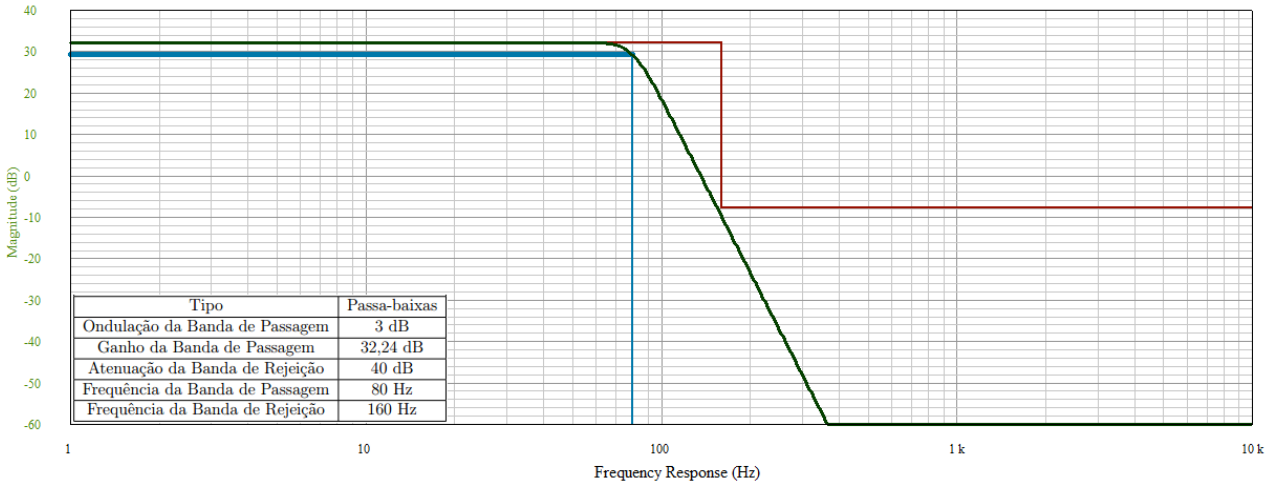

Figura 4.9: Resposta em frequência simulada (teórica) do filtro passa-baixas (filtro 1).

A Figura 4.10 mostra a configuração criada automaticamente para o filtro 1.

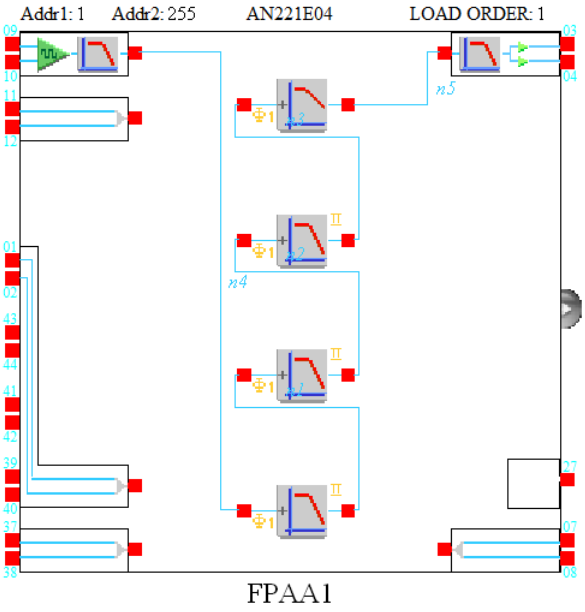

Figura 4.10: Configuração criada para a realização dos experimentos com o filtro 1.

A Figura 4.11 mostra os parâmetros utilizados na criação do filtro 2 e a respectiva resposta em frequência visualizada pela ferramenta *AnadigmFilter*.

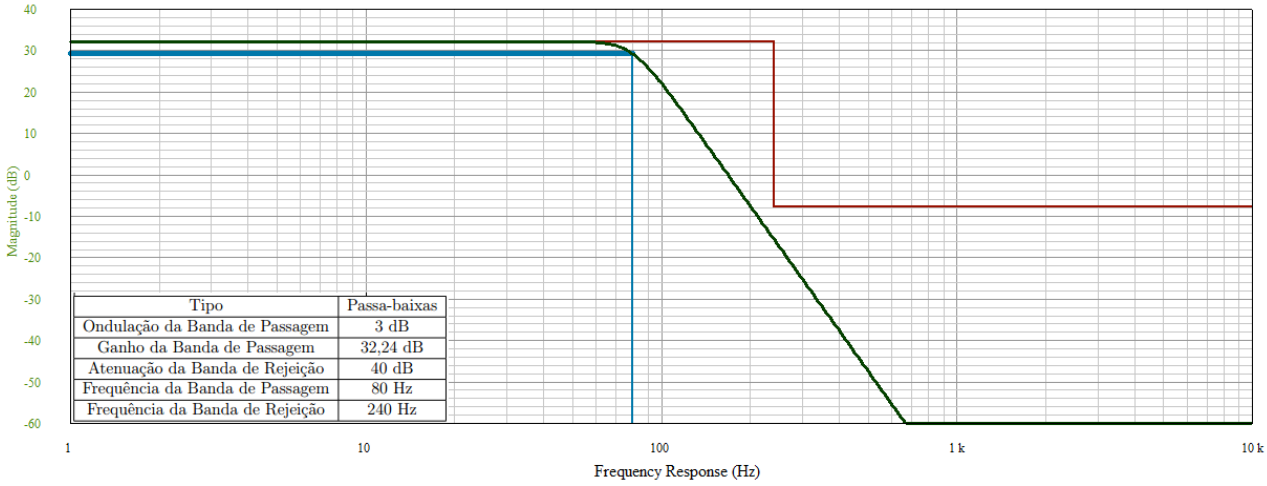

Figura 4.11: Resposta em frequência simulada para o filtro passa-baixas (filtro 2). Observa-se que a única diferença para o filtro 1 (Figura 4.9), refere-se à frequência da banda de rejeição, a qual o filtro 2 possui uma frequência maior (240 Hz).

A Figura 4.12 mostra a configuração criada automaticamente para o filtro 2.

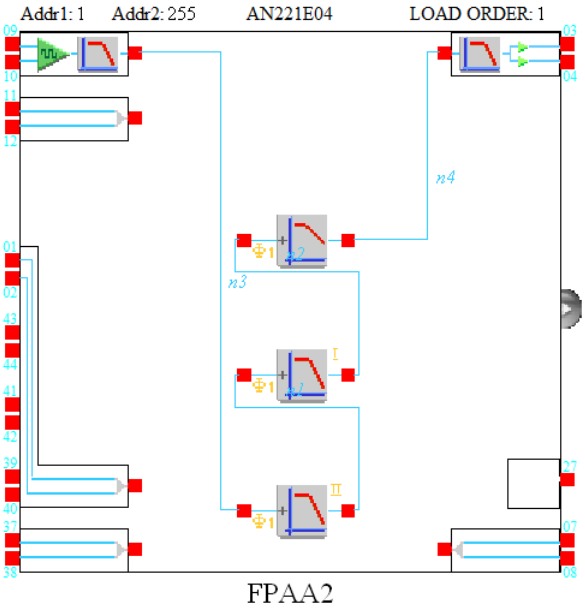

Figura 4.12: Configuração criada para a realização dos experimentos com o filtro 2.

A Figura 4.13 mostra os parâmetros utilizados na criação do filtro 3 e a respectiva resposta em frequência visualizada pela ferramenta *AnadigmFilter*.

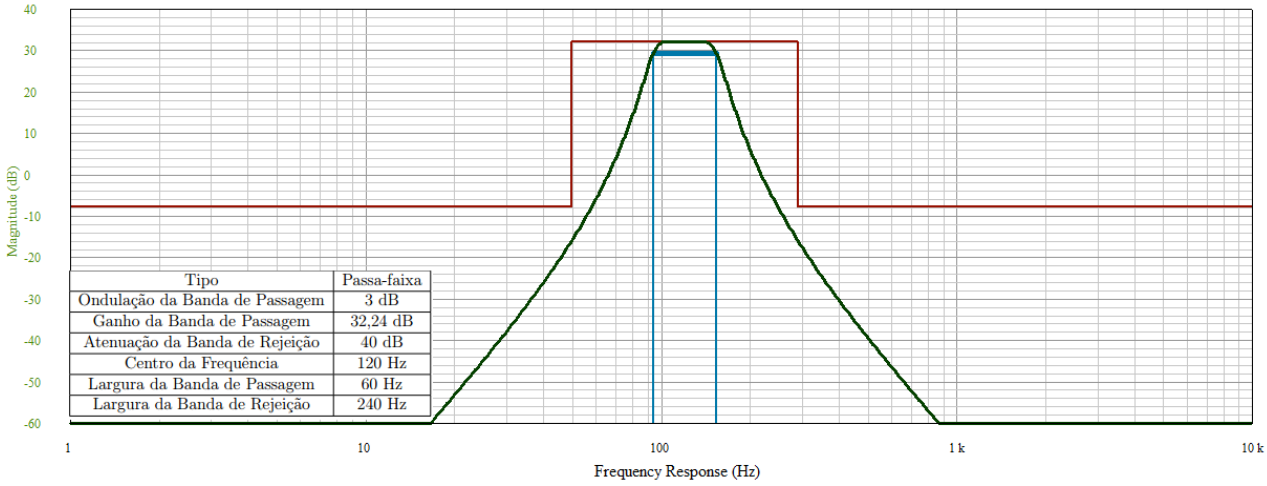

Figura 4.13: Resposta em frequência do filtro passa-faixa (filtro 3).

A Figura 4.14 mostra a configuração criada automaticamente para o filtro 3.

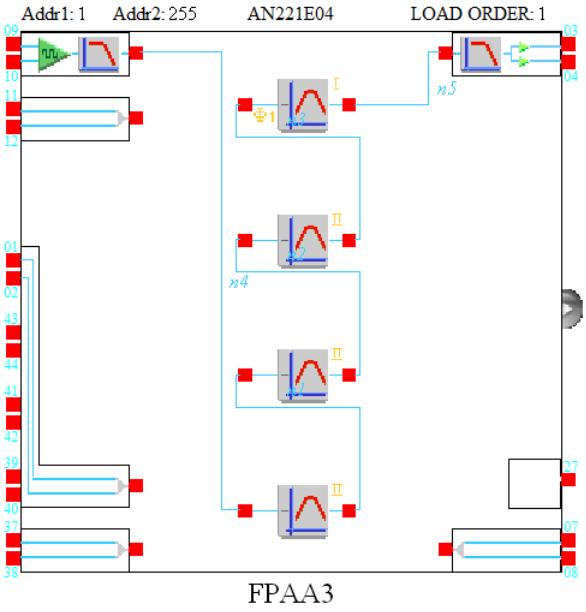

Figura 4.14: Configuração criada para a realização dos experimentos com o filtro 3.

A Figura 4.9 mostra os parâmetros utilizados na criação do filtro 4 e a respectiva resposta em frequência visualizada pela ferramenta *AnadigmFilter*.

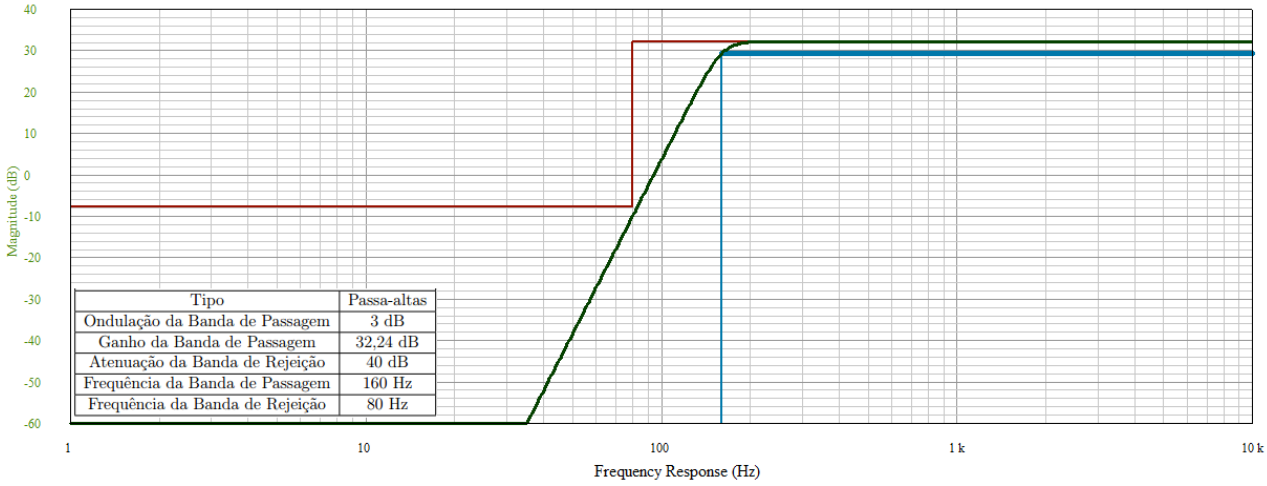

Figura 4.15: Resposta em frequência do filtro passa-altas (filtro 4).

A Figura 4.16 mostra a configuração criada automaticamente para o filtro 4.

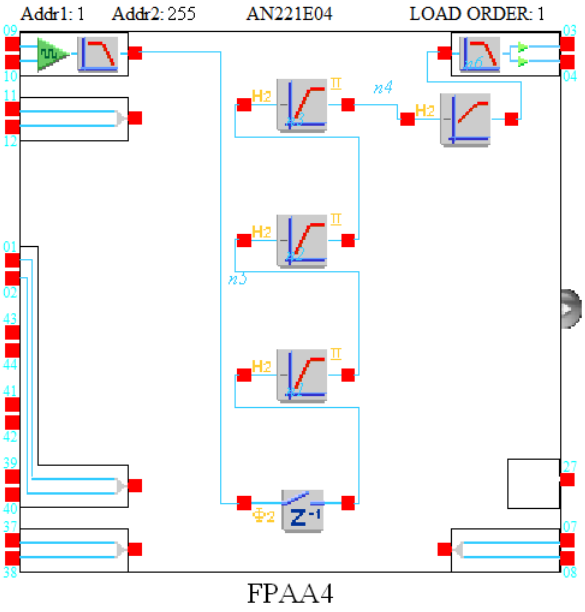

Figura 4.16: Configuração criada para a realização dos experimentos com o filtro passa-altas (filtro 4).

A Figura 4.17 mostra os parâmetros utilizados na criação do filtro 5 e a respectiva resposta em frequência visualizada pela ferramenta *AnadigmFilter*.

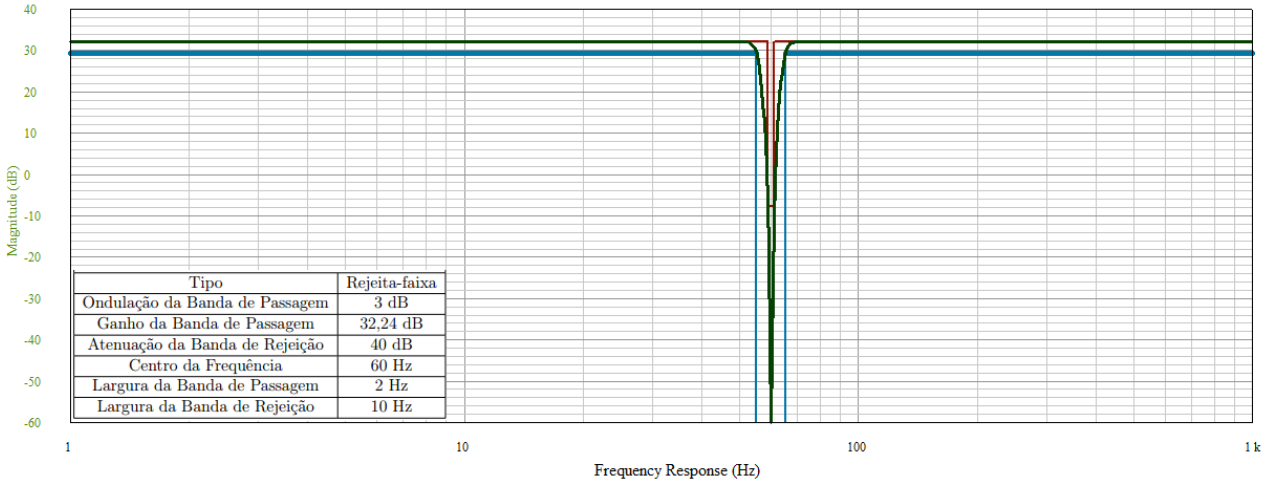

Figura 4.17: Resposta em frequência do filtro rejeita-faixas (filtro 5).

A Figura 4.10 mostra a configuração criada automaticamente para o filtro 5.

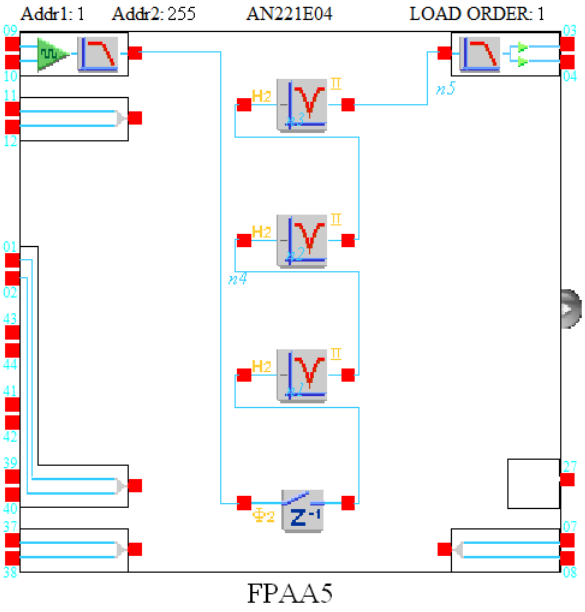

Figura 4.18: Configuração criada para a realização dos experimentos com o filtro rejeita-faixa (filtro 5).

A partir das 5 configurações (circuitos) obtidas, realizou-se uma série de 30 testes em cada circuito, onde em cada teste eram seguidos os seguintes passos:

1. A configuração do circuito em questão era transferida para a FPAA pelo aplicativo "*Genérico*";

2. A ferramenta "*Bode Analyzer*" (fornecida juntamente com o módulo *NI myDAQ* e configurada para frequência inicial de 1 Hz, frequência final de 1 kHz, 5 passos por década e tensão de pico de 4 V) era executada e armazenava a resposta em frequência da FPAA. Nessa medição, o módulo *NI myDAQ* teve seu sinal aplicado à FPAA através de um atenuador resistivo, que reduziu a amplitude do sinal em 60 dB (1000 vezes), e mediu o sinal na saída da FPAA, conectada na entrada do conversor A/D do microcontrolador; e

3. Um sinal senoidal (confeccionado no *software "Arbitrary Waveform Generator"*, com uma tensão fixa de 5 Vpv e variando de 1 Hz a 240 Hz) era aplicado através do gerador de sinais presente no módulo *NI myDAQ*, sendo os sinais visualizados em tempo real e armazenados no *smartphone* em um arquivo binário. Ressalta-se que esses sinais também passaram pelo atenuador resistivo de 1000 vezes. O arquivo binário era então analisado através do *script* presente nos Apêndices, o qual foi executado no *software Matlab* [59]. O *script* realizava a conversão do arquivo binário e, em seguida, a medição do valor eficaz do sinal. A partir desses valores (trechos de 500 em 500 amostras) se obteve a resposta em frequência.

A Figura 4.19 mostra as respostas em frequência da FPAA obtidas nos passos 2 e 3 para o filtro 1.

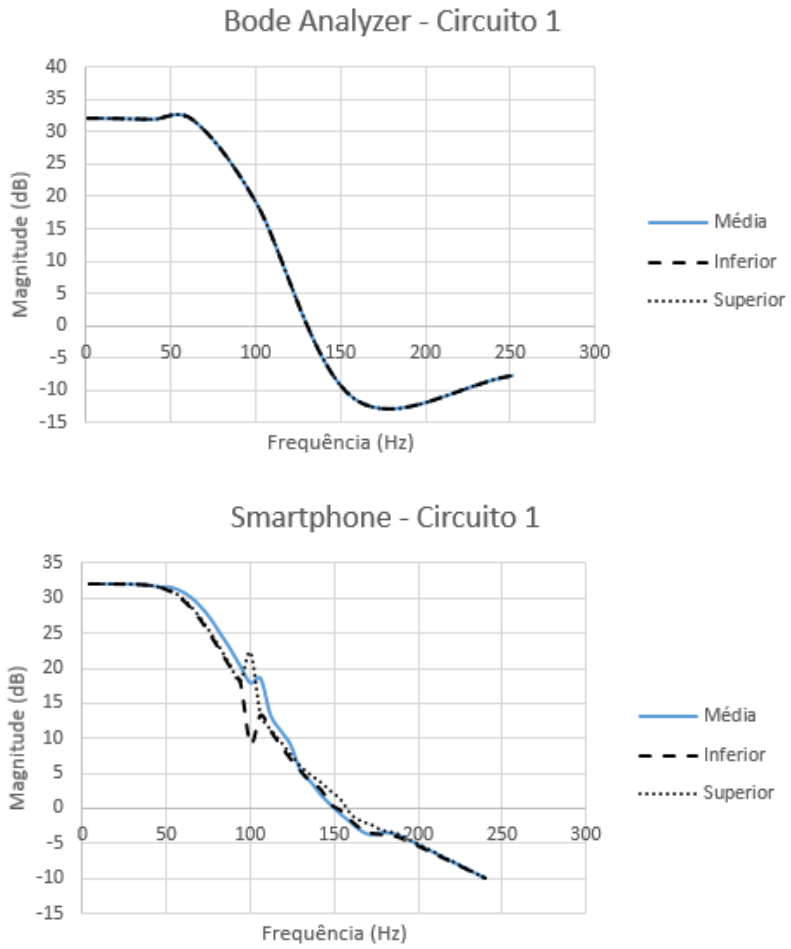

Figura 4.19: Respostas em frequência obtidas pelos passos 2 e 3. Vale ressaltar que a linha contínua indica a média da amostra com 30 medidas e as linhas tracejadas indicam o intervalo de confiança para um nível de confiança de 99%. Observa-se que as linhas se encontram muito próximas.

A Figura 4.20 mostra as respostas em frequência da FPAA obtidas nos passos 2 e 3 para o filtro 2.

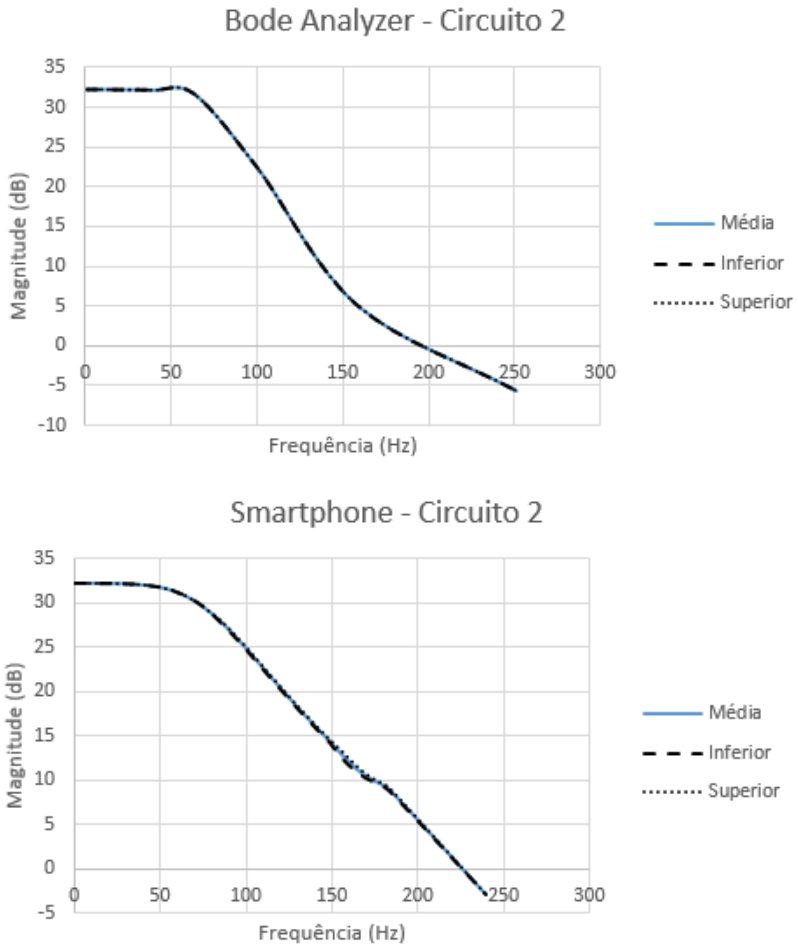

Figura 4.20: Resposta em frequência para o circuito 2. Observa-se que a linha contínua (representa a média da amostra com 30 medidas) e as linhas tracejadas (indicam o intervalo de confiança para um nível de confiança de 99%) encontram-se praticamente sobrepostas.

A Figura 4.21 mostra as respostas em frequência da FPAA obtidas nos passos 2 e 3 para o filtro 3.

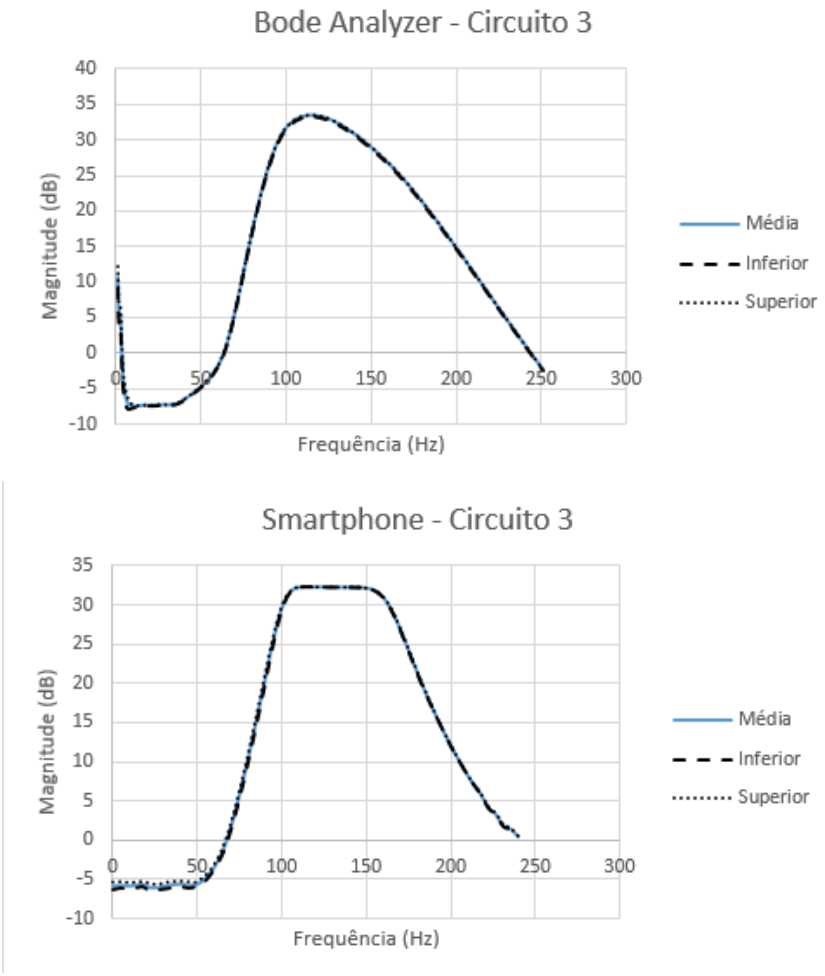

Figura 4.21: Respostas em frequência medidas para o circuito 3. Vale ressaltar que as linhas contínuas indicam a média da amostra com 30 medidas e as linhas tracejadas indicam o intervalo de confiança para um nível de confiança de 99%.

A Figura 4.22 mostra as respostas em frequência da FPAA obtidas nos passos 2 e 3 para o filtro 4.

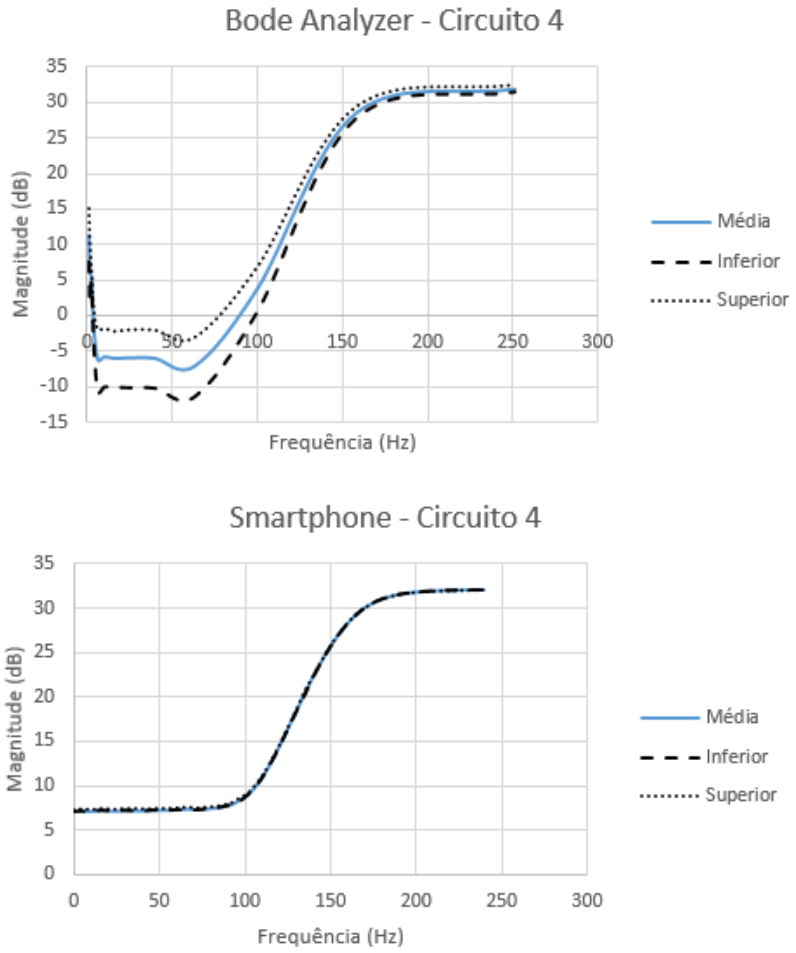

Figura 4.22: Respostas em frequência do circuito 4. Observa-se que as linhas contínuas indicam a média da amostra com 30 medidas e as linhas tracejadas indicam o intervalo de confiança para um nível de confiança de 99%.

A Figura 4.23 mostra as respostas em frequência da FPAA obtidas nos passos 2 e 3 para o filtro 5.

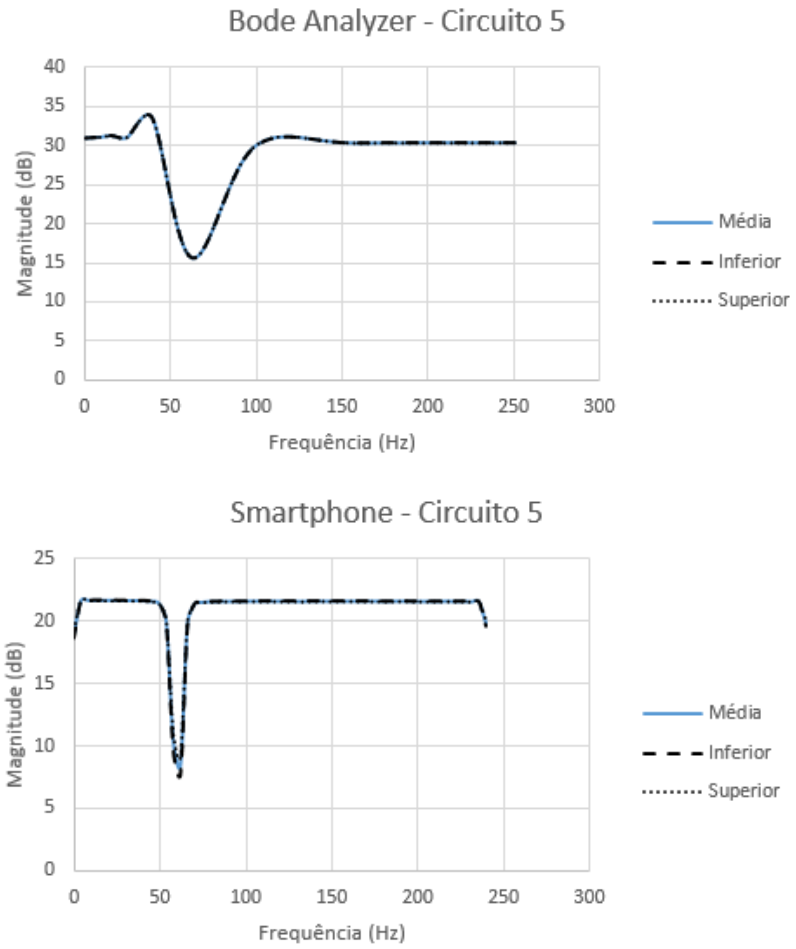

Figura 4.23: Respostas em frequência obtidas através do circuito 5. As linhas contínuas indicam a média da amostra com 30 medidas e as linhas tracejadas indicam o intervalo de confiança para um nível de confiança de 99%.

A Figura 4.24 mostra as respostas em frequência desejadas (teóricas) e as respostas em frequência medidas para os 5 filtros construídos.

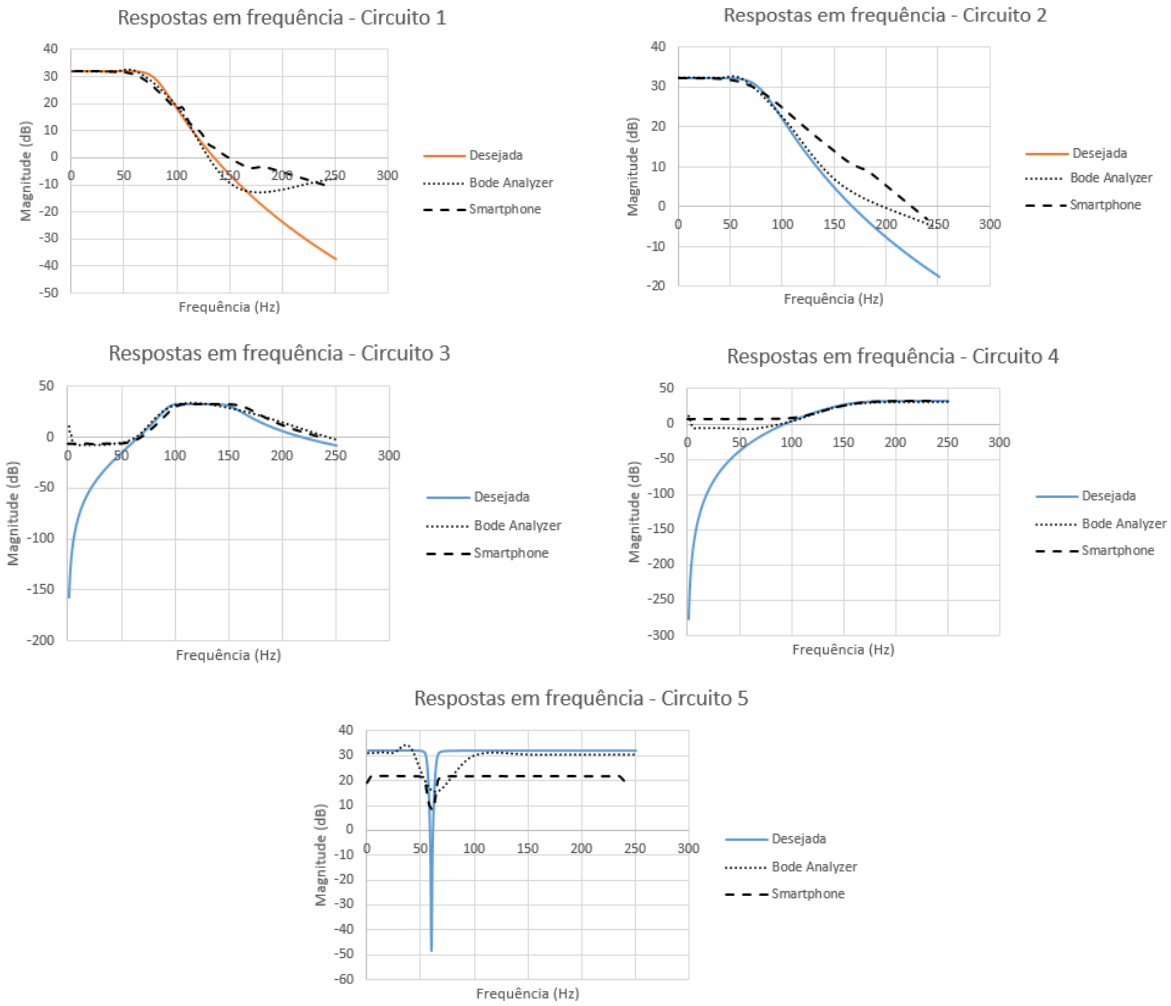

Figura 4.24: Comparativo entre as respostas em frequência desejadas e as obtidas através da ferramenta *"Bode Analyzer"* e dos sinais armazenados no *smartphone*.

Em relação às respostas em frequência obtidas pelos cinco circuitos genéricos, obteve-se um resultado considerado satisfatório, pois as respostas em frequência representam, com fidelidade, as respostas dos filtros projetados. Atribui-se as diferenças entre as respostas teóricas e medidas aos erros de medição inerentes a qualquer instrumento eletrônico, no caso desse trabalho, o próprio sistema desenvolvido e o módulo *NI myDAQ*.

4.3 Considerações Finais

Neste capítulo foram apresentados os testes realizados no circuito para a aquisição dos sinais de ECG e os testes envolvendo cinco circuitos distintos de filtros. Os testes do circuito de ECG utilizaram como base a norma ANSI/AAMI EC13:2002 [57], resultando em valores satisfatórios - com exceção do teste de ruído. Já os testes envolvendo os cinco filtros distintos criados na ferramenta *AnadigmFilter*, resultaram em respostas em frequência muito próximas às respostas teóricas. É importante destacar, ainda, que o custo total aproximado dos

principais componentes do SERAS-ECG é de USD 22,61 (MCU: USD 5,37, FPAA: USD 12,00 e módulo *Bluetooth*: USD 5,24). O próximo capítulo apresenta as conclusões relativas ao trabalho.

Capítulo 5

Conclusão

Dado o exposto, percebe-se que o trabalho apresentado utilizou, com sucesso, um dispositivo analógico reconfigurável (FPAA) para captar os sinais do coração, os quais possuem baixa amplitude (cerca de 3 mV). Pode-se verificar a grande flexibilidade proporcionada pela FPAA, pois através da reconfiguração é possível mudar o ganho de amplificadores, frequências de corte de filtros, funções de aproximação de filtros e até mesmo o circuito de condicionamento do sinal. Apesar das vantagens geradas pela FPAA, poucos trabalhos utilizando-a são encontrados na literatura.

Os parâmetros medidos nos testes (resposta em frequência, imunidade ao potencial de meia célula, CMRR, impedância de entrada e consumo) e a qualidade dos sinais de ECG registrados em um indivíduo foram satisfatórios. Cabe ressaltar que todos os parâmetros medidos (incluindo o ruído do sistema que não atendeu à norma [57]), certamente seriam melhorados caso houvesse um processo de otimização na configuração do circuito e se o sistema não tivesse sido montado em uma matriz de contatos.

O aplicativo Android *"ECG"*, com o circuito específico de ECG, permite visualizar e armazenar os sinais de ECG em tempo real no *smartphone*, e alterar o ganho total do circuito. Enquanto isso, o outro aplicativo desenvolvido permite carregar qualquer arquivo de configuração da FPAA, possibilitando a troca do circuito de condicionamento em campo, sem a necessidade de alterar o *firmware* do microcontrolador. Ambos aplicativos realizam o armazenamento dos sinais recebidos, possibilitando o posterior envio dos dados através da Internet para um médico analisá-los (telemedicina). Vale destacar que o SERAS-ECG poderia ainda se tornar um dos módulos de um sistema para *Home care*.

Portanto, conclui-se que a FPAA é um dispositivo recomendado para o uso em sistemas de aquisição de sinais bioelétricos, assim como outros, principalmente pela facilidade da reconfiguração (possui diversos blocos analógicos implementados e prontos para o uso).

5.1 Contribuições

A seguir são citadas as principais contribuições do presente trabalho:

- Desenvolvimento de um sistema para aquisição de sinais de ECG utilizando *smartphone*. Esse sistema pode servir como base para o desenvolvimento de trabalhos futuros na área de *m-Health* envolvendo ECG;
- Desenvolvimento de um sistema reconfigurável para aquisição de sinais analógicos, que poderá ser utilizado no registro de outros sinais bioelétricos (eletroencefalograma, eletromiograma, eletrooculograma e outros). Assim, o sistema construído poderá ser usado em trabalhos futuros na área médica (*m-Health*) e também no desenvolvimento de interfaces homem-máquina, área com bastante destaque atualmente [64] [65] [66] [67];
- Criação de métodos em Java para o aplicativo Android reconfigurar um CAM de um circuito da FPAA através do *Bluetooth*; e
- Elaboração de uma dissertação usando FPAA que pode ser usada como base para trabalhos futuros, inclusive contendo informações pouco encontradas sobre a reconfiguração.

Além das contribuições citadas, comparando-se com os trabalhos relacionados, vale ressaltar que o trabalho desenvolvido não utilizou uma placa de desenvolvimento fornecida pelo fabricante (diferente da maioria dos trabalhos encontrados), e que nenhum trabalho encontrado permite visualizar os sinais e reconfigurar a FPAA através do *smartphone*.

5.2 Dificuldades Encontradas

A maior dificuldade encontrada no trabalho foi a programação do aplicativo Android. Inicialmente, a programação foi realizada tendo como base a biblioteca *open-source "AChartEngine"* [60]. O aplicativo foi testado com tensões fixas na entrada do ADC (0 V e 3,3 V), onde constatou-se que o desempenho com a biblioteca em questão não foi satisfatório, pois as amostras recebidas pelo *Bluetooth* não eram "plotadas" em tempo real, isto é, precisava-se de um *buffer* para armazenar várias amostras e na sequência, atualizar a tela.

A fim de encontrar uma solução rápida para o desenvolvimento do aplicativo, houveram outras tentativas utilizando bibliotecas *open-source*. As outras bibliotecas testadas foram a *"Androidplot"* [61] e a *"GraphView"* [62], mas os resultados também não foram adequados.

Devido ao resultado inadequado destas tentativas, buscou-se implementar o aplicativo através da classe *"View"* e através da *OpenGL for Embedded Systems* (OpenGL ES). A classe *"View"* é o bloco básico para a construção de interfaces, sendo responsável pelo desenho e manipulação de eventos da aplicação, mas não recomendada para aplicações em tempo real. Já a OpenGL ES é um subconjunto da API *Open Graphics Library* (OpenGL) aplicada para renderizações 2D e 3D, geralmente utilizando a *Graphics Processing Unit* (GPU). Cabe observar que a API OpenGL ES não é equivalente à biblioteca OpenGL.

Novamente, as alternativas aplicadas não corresponderam às expectativas. Com isso, realizou-se uma nova implementação, dessa vez, utilizando a classe *"SurfaceView"*. Devido ao cenário de tempo real, essa implementação foi a que obteve o melhor desempenho na visualização dos sinais, porém, verificou-se que a taxa máxima de atualização da tela na

plataforma Android é de 60 Frames por Segundo (FPS) [63]. Sendo assim, a "plotagem" correta das amostras em tempo real não era possibilitada por essa taxa máxima. Devido a essa limitação, verificou-se qual o número de amostras que deveriam ser "plotadas" para cada atualização de tela (*frame*), contornando esta limitação.

O cálculo realizado considerou a largura da tela do aparelho utilizado (1175 *pixels*), a taxa de amostragem (480 amostras/segundo) e o tempo necessário para "plotar" em toda a tela (aproximadamente 2,45 segundos). Portanto, através da taxa máxima de 60 FPS, o aplicativo precisaria "plotar" cerca de 8 amostras por atualização de tela, entretanto, através das medições de tempo realizadas, o tempo gasto para "plotar" toda a tela era superior, necessitando que fossem "plotadas" 25 amostras para cada atualização.

Outra dificuldade encontrada no desenvolvimento do projeto refere-se à perda de diversas amostras recebidas pelo *Bluetooth*, sendo detectadas através da depuração dos *logs* do aplicativo Android. Essa perda de dados era causada por um erro de programação utilizando as APIs *Bluetooth*, onde o *buffer* do *smartphone* era sobrescrito com novas amostras recebidas, antes de realizar a "plotagem" das antigas. Com isso, o método *clone()* foi aplicado com êxito, realizando uma cópia dos *bytes* recebidos para outro *buffer*. A perda de dados foi resolvida através da "plotagem" do *buffer* auxiliar, enquanto que o outro *buffer* recebia as novas amostras sem sobrescrever as antigas.

Outro ponto importante ocorreu na implementação da reconfiguração dinâmica do modo algorítmico, onde tentou-se gerar o vetor de reconfiguração (para permitir a alteração do ganho da configuração de ECG) no *firmware* do microcontrolador. Inicialmente, foram encontrados problemas relacionados ao tamanho do espaço das variáveis, ocupado pelo código gerado pela ferramenta *AnadigmDesigner2*. Assim, o compilador avisava frequentemente que a memória RAM interna do microcontrolador estava esgotada. Após diversas modificações para simplificar o código, a solução recomendada pelo suporte do fabricante da FPAA foi gerar um código compactado, possuindo apenas o necessário para gerar o vetor de *bytes* usado para reconfigurar a FPAA.

Através de comparações entre os vetores de *bytes* gerados pela ferramenta *AnadigmDesigner2* e os vetores gerados pelo *firmware* desenvolvido, notou-se que os vetores apresentavam alguns valores diferentes. Suspeitou-se que a causa desse problema poderia ser a ausência do tipo de dados primitivo *double* no microcontrolador ou a utilização de funções trigonométricas no código.

A fim de resolver este problema, realizaram-se implementações e comparações entre o vetor de configuração gerado pela ferramenta *AnadigmDesigner2*, o vetor de configuração gerado pelo algoritmo executado em um computador IBM-PC utilizando precisão dupla, o vetor gerado pelo algoritmo executado em um computador IBM-PC utilizando precisão simples e o vetor gerado pelo *firmware* executado no microcontrolador (utilizando precisão simples).

A partir dessas implementações, constatou-se que os vetores de configuração gerados pela ferramenta *AnadigmDesigner2* eram iguais aos vetores gerados no computador IBM-PC (tanto com precisão simples, quanto com precisão dupla). Ao compará-los com o vetor gerado pelo *firmware*, observou-se que a implementação estava gerando um *byte* diferente, o qual comprometia totalmente a reconfiguração correta da FPAA.

A outra suspeita, motivada pelo fato das funções trigonométricas trabalharem com muita precisão, não pôde ser verificada por falta de tempo. Sendo assim, a alternativa encontrada e adotada no trabalho foi portar o código em linguagem C para a linguagem Java, tornando-o parte do aplicativo Android.

A maior limitação existente no sistema de visualização do sinal ocorre na "plotagem" de amostras que alternam entre 0 e 255, rapidamente e continuamente ao longo do tempo, preenchendo toda a tela do *smartphone*. Nessa sistuação, a aplicação não consegue "plotar" em tempo suficiente, tornando a visualização consideravelmente lenta. Vale ressaltar que essa situação não ocorre na visualização do sinal do ECG.

5.3 Trabalhos Futuros

Algumas sugestões de possíveis trabalhos futuros são apresentadas a seguir:

I - Fabricação de uma placa de circuito impresso: envolve a confecção do *layout* da placa, a fabricação da placa e a montagem do protótipo, que melhoraria várias características do sistema (ruído, CMRR, entre outras);

II - Desenvolvimento de um sistema para *Home care*: aproveitar a flexibilidade da FPAA tornando o sistema uma plataforma para aquisição de outros sinais bioelétricos, como: eletromiograma, eletroencefalograma, eletrooculograma, entre outros;

III - Acrescentar algoritmos de processamento de sinais e Inteligência Artificial (IA) no aplicativo Android: alguns exemplos existentes na literatura são voltados à: detecção do QRS e classificação de sinais [68], detecção de arritmias [69], estimação dos estágios de sono [70], detecção de epilepsias [71], entre outras;

IV - Verificar a possibilidade de comunicação entre o dispositivo *Google Class* e o SERAS-ECG: essa possibilidade existe pelo fato do dispositivo utilizar o SO Android e possuir a tecnologia *Bluetooth* inclusa; e

V - Explorar a ferramenta *Anadigm PID*: através desta ferramenta, diversos sistemas de controle podem ser implementados na FPAA, por exemplo, um projeto de sistema embarcado voltado à automação industrial.

Bibliografia

[1] J. Webster, *Medical Instrumentation: Application and Design*. Wiley, 3 ed., 1997.

[2] A. Atkielski, "Schematic representation of normal ecg," tech. rep., Atrial Fibrillation, 2007.

[3] D. Martincoski, "Sistema para telemetria de eletrocardiograma utilizando tecnologia bluetooth," Master's thesis, Universidade Federal de Santa Catarina, 2003.

[4] F. Faione, *Proposta e Implementação de Metodologia para Detecção de Hipoglicemia Baseada na Análise e Classificação do Eletroencefalograma*. Tese de Doutorado, Universidade Federal de Santa Catarina, 2003.

[5] F. N. R. Mussoi, "Resposta em frequência: Filtros passivos," tech. rep., Centro Federal de Educação Tecnológica de Santa Catarina (CEFET/SC), Julho 2004.

[6] Analog Inc., "An120e04 datasheet," tech. rep., Analog Inc., 2012.

[7] R. Bannatyne, "Maximizing design potential with a mixed-signal mcu," tech. rep., Silicon Laboratories, 2010.

[8] Silabs, "Mixed-signal, 8-bit and 32-bit microcontrollers (mcus)," tech. rep., Silabs, 2012.

[9] Google, "Android," tech. rep., Google, 2012.

[10] P. Rodriguez, H. Patiño, C. Jaramillo, and J. Medina, "Diseño de un microsistema para adquisición de señales cardiacas usando fpaas," *IBERCHIP*, 2006.

[11] Gartner, "Gartner says asia/pacific led worldwide mobile phone sales to growth in first quarter of 2013," tech. rep., Gartner, 2013.

[12] M. Cerny and M. Penhaker, "Biotelemetry," *14th Nordic-Baltic Conference on Biomedical Engineering and Medical Physics*, pp. 405–408, 2008.

[13] A. Raghavan, H. Ananthapadmanaban, M. Sivamurugan, and B. Ravindran, "Accurate mobile robot localization in indoor environments using bluetooth," *IEEE International Conference on Robotics and Automation*, 2010.

[14] S. Yong, "Design and implementation of electrical energy meter reading system based on bluetooth communication technology," *International Conference on Electrical and Control Engineering (ICECE)*, pp. 2111 – 2113, 2011.

[15] V. Pamplona, A. Mohan, M. Oliveira, and R. Raskar, "Netra: Interactive display for self-evaluation of an eye for visual accommodation and focal range," *SIGGRAPH*, 2010.

[16] V. Pamplona, E. Passos, J. Zizka, M. Oliveira, E. Lawson, E. Clua, and R. Raskar, "Catra: Cataract probe with a lightfield display and a snap-on eyepiece for mobile phones," *SIGGRAPH*, 2011.

[17] A. Khandoker, J. Black, and M. Palaniswami, "Smartphone-based low cost oximeter photoplethysmography," in *Electrical and Computer Engineering (ICECE), 2010 International Conference on*, pp. 634 –637, dec. 2010.

[18] Z. Aihua and H. Ninghao, "The system of pulse monitoring based on windows mobile," in *Business Management and Electronic Information (BMEI), 2011 International Conference on*, vol. 4, pp. 519 –522, may 2011.

[19] K. Watanabe, Y. Kurihara, T. Nakamura, and H. Tanaka, "Design of a low-frequency microphone for mobile phones and its application to ubiquitous medical and healthcare monitoring," *Sensors Journal, IEEE*, vol. 10, pp. 934 –941, may 2010.

[20] N. Mertz, "Ultrasound? fetal monitoring? spectrometer? there's an app for that!: Biomedical smart phone apps are taking healthcare by storm," *Pulse, IEEE*, vol. 3, pp. 16 –21, march 2012.

[21] D. Grzechca and J. Rutkowski, "The use of fpaa in signal processing laboratory for biomedical engineering students," in *Signals and Electronic Systems (ICSES), 2010 International Conference on*, pp. 453 –456, sept. 2010.

[22] D. Morales, A. García, E. Castillo, M. Carvajal, J. Banqueri, and A. Palma, "Flexible ecg acquisition system based on analog and digital reconfigurable devices," *Sensors and Actuators A: Physical*, vol. 165, no. 2, pp. 261 – 270, 2011.

[23] D. Morales, A. García, E. Castillo, M. Carvajal, N. Parrilla, and A. Palma, "An application of reconfigurable technologies for non-invasive fetal heart rate extraction," *Medical Engineering & Physics*, vol. 35, no. 7, pp. 1005 – 1014, 2013.

[24] W. H. Organization, "Cardiovascular disease," tech. rep., World Health Organization, 2013.

[25] Google, "Google i/o 2013," tech. rep., Google, 2013.

[26] A. Jesdanun, "Smartphone sales: Android extends lead over iphone," tech. rep., The Christian Science Monitor, 2012.

[27] Vital Wave Consulting, "m-health for development: The opportunity of mobile technology for healthcare in the developing world," *United Nations Foundation*, 2009.

[28] D. Patil, "Mobile for health (mhealth) in developing countries: Application of 4 ps of social marketing," *Journal of Health Informatics in Developing Countries*, pp. 317–326, 2011.

[29] P. Mechael, *Exploring Health-related Uses of Mobile Phones: An Egyptian Case Study*. PhD thesis, London School of Hygiene and Tropical Medicine, 2006.

[30] P. Mou, C. Chen, S. Pun, P. Mak, and M. Vai, "General purpose adaptive biosignal acquisition system combining fpga and fpaa," in *13th International Conference on Biomedical Engineering* (C. Nim and J. Goh, eds.), vol. 23 of *IFMBE Proceedings*, pp. 31–34, Springer Berlin Heidelberg, 2009.

[31] P. Mou, C. Chen, S. Pun, P. Mak, and M. Vai, "Portable intelligent bioelectric signals acquisition system with an adaptive frontend implemented using fpga and fpaa," *The 11th World Congress on Medical Physics and Biomedical Engineering (WC2009)*, pp. 348–351, September 2009.

[32] U. Chan, W. Chan, S.-H. Pun, M.-I. Vai, and P.-U. Mak, "Flexible implementation of front-end bioelectric signal amplifier using fpaa for telemedicine system," in *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pp. 3721–3724, 2007.

[33] B. Fuchs, S. Vogel, and D. Schroeder, "Universal application-specific integrated circuit for bioelectric data acquisition," in *Medical Engineering e Physics*, vol. 24, pp. 695–701, December 2002.

[34] C. Moraes and F. Faione, "Aplicativo para visualizar sinais bioelétricos em dispositivos móveis," in *Proceedings/Anais of the/do Congresso da Sociedade Brasileira de Computação - X Workshop de Informática Médica*, (Belo Horizonte), Biblioteca Digital Brasileira de Computação, 2010.

[35] C. M. Agulhari, "Compressão de eletrocardiogramas usando wavelets," Master's thesis, Universidade Estadual de Campinas - Faculdade de Engenharia Elétrica e de Computação, 2009.

[36] W. Tompkins, *Biomedical Digital Signal Processing*. Prentice Hall, 1995.

[37] A. N. Nopes, F. Motta, K. Abe, N. Brandão, V. Bassareze, V. Nouailhetas, and J. Aboulafia, "Eletrocardiograma," tech. rep., UNIFESP, 2003.

[38] R. Tocci, N. Widmer, and G. Moss, *Sistemas Digitais : Princípios e Aplicações*. Prentice Hall, 2007.

[39] E. Ferreira, "Análise da inferferência de ruídos e artefatos no processo de aquisição e processamento digital de um sinal biológico," Master's thesis, Universidade do Vale do Paraíba, 2007.

[40] T. Caparelli, "Projeto e desenvolvimento de um sistema multicanal de biotelemetria para detecção de sinais ecg, eeg e emg," Master's thesis, Universidade Federal de Uberlândia, 2012.

[41] P. R. Sanches, A. Muller, N. Carro, A. Susin, and P. Nohama, "Analog reconfigurable technologies for emg signal processing," *Revista Brasileira de Engenharia Biomédica*, vol. 23, pp. 153–157, abril 2007.

[42] G. Domenech-Asensi, J. Martinez-Alajarin, R. Ruiz-Merino, and J. Nopez-Alcantud, "Synthesis on fpaa of a smart stbetoscope analog subsystem," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pp. 1 –5, aug. 2006.

[43] Ni logic Pvt. Ntd, *Analogic User Manual Version 2.0*, 2013.

[44] Mouser Electronics Inc., "Produtos - cypress semiconductor," tech. rep., Mouser Electronics Inc., 2012.

[45] Anadigm Inc., "Anadigm fpaa," tech. rep., Anadigm Inc., 2012.

[46] N. S. Corporation, "Nattice programmable analog ics: isppac devices," tech. rep., Nattice, 2012.

[47] C. Petre, "Sim2spice, a tool for compiling simulink designs on fpaa and applications to neuromorphic circuits," Master's thesis, School of Electrical and Computer Engineering Georgia Institute of Technology, 2009.

[48] C. M. O. Sensors, "Mixed signal circuits," tech. rep., CMOS, 2012.

[49] Freescale, "S12 magniv mixed-signal microcontroller introduction," tech. rep., Freescale, 2012.

[50] P. McDermott-Wells, "What is bluetooth?," *IEEE Potentials*, vol. 23, pp. 33–35, 2005.

[51] N. Chevrollier, O. Rebala, and N. Golmie, "Bluetooth adaptive frequency hopping and scheduling," *IEEE Military Communications Conference (MILCOM)*, vol. 2, pp. 1138 – 1142, 2003.

[52] HC Serial Bluetooth Products, *HC-05 - User Instructional Manual*, 2010.

[53] Android, "Introducing art," tech. rep., Android, 2014.

[54] Desarrollos Tecnológicos Pymasde S.N., "Blueterm," tech. rep., Pymasde, 2013.

[55] KiCad, "Kicad eda software suite," tech. rep., KiCad, 2014.

[56] D. Novell, "Support anadigm." Private email, 2014.

[57] American National Standards Institute / Association for the Advancement of Medical Instrumentation, "ANSI/AAMI EC13:2002 - Cardiac monitors, heart rate meters, and alarms," 2002.

[58] J. Lynch, *Co-Channel Interference In Bluetooth Piconets*. PhD thesis, Virginia Polytechnic Institute and State University, 2002.

[59] MathWorks, "Matlab r2014," 2014.

[60] AChartEngine, "Achartengine," tech. rep., AChartEngine, 2014.

[61] Androidplot, "Androidplot," tech. rep., Androidplot, 2013.

[62] J. Gehring, "Android graphview," tech. rep., Jonas Gehring, 2014.

[63] R. Guy, "Android view refresh rate (fps)." Private email, 2010.

[64] Y. Zhang, N. Ni, and H. Yan, "The bci method for upper limb disabilities based on emg and gyros," in *Advanced Motion Control (AMC),2014 IEEE 13th International Workshop on*, pp. 434–439, March 2014.

[65] S.-N. Wu, N.-D. Miao, S.-W. Su, W.-N. Jiang, S.-A. Chen, and C.-T. Lin, "Controlling a human-computer interface system with a novel classification method that uses electrooculography signals," *Biomedical Engineering, IEEE Transactions on*, vol. 60, pp. 2133–2141, Aug 2013.

[66] H.-C. Seol, Y.-C. Kwon, S.-K. Hong, and O.-K. Kwon, "An emg readout front-end with automatic gain controller for human-computer interface," in *Biomedical Circuits and Systems Conference (BioCAS), 2013 IEEE*, pp. 170–173, Oct 2013.

[67] S. M. Patil and C. G. Patil, "An approach for human machine interaction using electromyography," *Journal of Medical Imaging and Health Informatics*, vol. 4, no. 1, pp. 71–75, 2014.

[68] S. Dilmac and M. Korurek, "A new ecg arrhythmia clustering method based on modified artificial bee colony algorithm, comparison with ga and pso classifiers," in *Innovations in Intelligent Systems and Applications (INISTA), 2013 IEEE International Symposium on*, pp. 1–5, June 2013.

[69] H.-N. Chan, M.-H. Hsu, W.-Y. Hsu, W.-K. Hsu, and S.-W. Chen, "Integrating physical activity detection in heart rate variability and cardiac arrhythmia analysis," in *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on*, pp. 1–3, Dec 2013.

[70] B. Rekha, A. Kandaswamy, and R. Keerthana, "Artificial intelligence based automated estimation of sleep stages using electrocardiograph signals: A perspective," *Applied Mechanics and Materials*, vol. 573, 2014.

[71] I. Mporas, V. Tsirka, E. Zacharaki, M. Koutroumanidis, and V. Megalooikonomou, "Online seizure detection from eeg and ecg signals for monitoring of epileptic patients," in *Artificial Intelligence: Methods and Applications* (A. Likas, K. Blekas, and D. Kalles, eds.), vol. 8445 of *Lecture Notes in Computer Science*, pp. 442–447, Springer International Publishing, 2014.

Capítulo 6

Apêndice A - Tabelas

6.1 Teste 2 - Resposta em frequência

Tabela 6.1: Teste da resposta em frequência realizada na configuração para condicionamento do circuito do sinal de ECG, obtida através do método A (descrito em 4.1.2).

0,67 Hz Arquivo 1 Pico-Vale (NCADpv)	Arquivo 2 Pico-Vale (NCADpv)	Arquivo 3 Pico-Vale (NCADpv)
42	42	43
42	43	42
42	43	42
42	43	42
42	43	42
42	43	42
42	43	43
42	43	42
42	42	42
42	42	42
42	42	43
42	42	42
42	42	43
42	43	42
42	42	42
42	42	42
42	42	43
42	42	42
42	43	42
42	43	43
42	43	42
42	43	42
42	43	42

42	43	42
42	42	42
42	42	43
42	42	42
42	42	42
42	42	42
42	42	43
$42 \pm 0 (\alpha = 99\%)$	$42,46 \pm 0,256 (\alpha = 99\%)$	$42,26 \pm 0,224 (\alpha = 99\%)$

5 Hz Arquivo 1 Pico-Vale (NCADpv)	Arquivo 2 Pico-Vale (NCADpv)	Arquivo 3 Pico-Vale (NCADpv)
95	95	96
96	95	96
95	96	95
96	95	95
95	95	95
95	96	95
95	96	95
96	96	95
96	96	95
95	95	95
96	95	95
96	95	95
95	96	95
96	96	95
95	96	95
96	95	95
96	95	95
96	95	96
95	96	96
96	96	95
95	95	96
95	95	96
95	95	95
95	96	95
95	96	95
95	95	95
95	95	95
95	96	95
95	96	95
95	95	96

$95,367 \pm 0,246 (\alpha = 99\%)$	$95,467 \pm 0,255 (\alpha = 99\%)$	$95,233 \pm 0,216 (\alpha = 99\%)$

16 Hz Arquivo 1 Pico-Vale (NCADpv)	Arquivo 2 Pico-Vale (NCADpv)	Arquivo 3 Pico-Vale (NCADpv)
97	97	98
97	97	95
97	97	97
97	97	97
97	97	97
97	97	97
96	97	96
96	97	96
96	97	96
97	97	97
97	97	97
97	97	97
97	97	97
97	97	97
97	97	97
97	97	97
97	97	96
97	97	96
97	97	96
96	97	97
96	97	97
96	97	97
97	97	97
97	97	97
97	97	97
97	97	97
97	97	97
97	97	97
96	97	97
96	97	96
96	97	96
$96,7 \pm 0,234 (\alpha = 99\%)$	$97 \pm 0 (\alpha = 99\%)$	$96,7 \pm 0,299 (\alpha = 99\%)$

20 Hz Arquivo 1	Arquivo 2	Arquivo 3

Pico-Vale (NCADpv)	Pico-Vale (NCADpv)	Pico-Vale (NCADpv)
95	95	95
95	95	95
95	95	95
94	94	95
94	94	95
94	93	95
94	95	95
94	95	94
94	95	94
95	95	95
95	95	95
95	95	95
94	94	94
94	94	94
94	94	94
94	94	93
94	94	93
94	94	93
95	95	93
95	95	94
95	95	94
95	95	94
95	94	94
95	94	94
95	94	95
95	95	95
95	95	95
93	94	94
93	95	94
93	95	94
$94,4 \pm 0,339 (\alpha = 99\%)$	$94,53 \pm 0,287 (\alpha = 99\%)$	$94,3 \pm 0,353 (\alpha = 99\%)$

40 Hz Arquivo 1 Pico-Vale (NCADpv)	Arquivo 2 Pico-Vale (NCADpv)	Arquivo 3 Pico-Vale (NCADpv)
68	69	69
68	69	69
68	69	69
69	69	69
69	69	69
69	69	69

68	69	69
68	69	69
68	69	69
69	69	69
69	69	69
69	68	69
69	68	69
69	69	69
69	70	69
69	69	69
69	69	69
69	69	69
68	69	69
68	69	69
68	69	69
69	69	69
69	69	69
69	69	69
69	69	69
69	69	69
69	69	69
69	68	69
$68,7 \pm 0,234(\alpha = 99\%)$	$68,93 \pm 0,183(\alpha = 99\%)$	$69 \pm 0(\alpha = 99\%)$

Tabela 6.6: Resposta em frequência obtida através do método B, descrito em 4.1.2.

Primeiro sinal Pico-Médio	Segundo sinal Pico-Médio	Compara sinal 1 com 2	%
18	21	Aceitável	16,67
18	20	Aceitável	11,11
17	21	Aceitável	23,53
18	19	Aceitável	5,56
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65

17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	19	Aceitável	5,56
17	20	Aceitável	17,65
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	19	Aceitável	5,56
18	21	Aceitável	16,67
18	19	Aceitável	5,56
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	19	Aceitável	5,56
17	19	Aceitável	11,76
17	20	Aceitável	17,65
16	19	Aceitável	18,75
17	20	Aceitável	17,65
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	19	Aceitável	11,76
17	20	Aceitável	17,65
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	20	Aceitável	17,65
17	21	Aceitável	23,53
17	19	Aceitável	11,76
17	21	Aceitável	23,53
18	20	Aceitável	11,11
17	21	Aceitável	23,53
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	19	Aceitável	5,56
18	20	Aceitável	11,11
18	20	Aceitável	11,11

17	20	Aceitável	17,65
18	21	Aceitável	16,67
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	19	Aceitável	5,56
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	19	Aceitável	5,56
18	20	Aceitável	11,11
18	20	Aceitável	11,11
16	19	Aceitável	18,75
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	18	Aceitável	0,00
18	21	Aceitável	16,67
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	20	Aceitável	17,65
16	19	Aceitável	18,75
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	21	Aceitável	16,67
18	19	Aceitável	5,56
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	19	Aceitável	11,76
17	18	Aceitável	5,88

18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	19	Aceitável	5,56
17	21	Aceitável	23,53
18	19	Aceitável	5,56
18	20	Aceitável	11,11
16	19	Aceitável	18,75
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
17	20	Aceitável	17,65
17	20	Aceitável	17,65
17	20	Aceitável	17,65
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	19	Aceitável	11,76
17	20	Aceitável	17,65
17	19	Aceitável	11,76
18	19	Aceitável	5,56
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	20	Aceitável	17,65
17	19	Aceitável	11,76
17	20	Aceitável	17,65
17	20	Aceitável	17,65
18	19	Aceitável	5,56

18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	19	Aceitável	11,76
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65
17	20	Aceitável	17,65
17	19	Aceitável	11,76
17	19	Aceitável	11,76
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	21	Aceitável	16,67
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	21	Aceitável	16,67
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	20	Aceitável	17,65
18	19	Aceitável	5,56
17	20	Aceitável	17,65
18	21	Aceitável	16,67

17	20	Aceitável	17,65
17	21	Aceitável	23,53
18	20	Aceitável	11,11
18	20	Aceitável	11,11
18	20	Aceitável	11,11
17	18	Aceitável	5,88
18	20	Aceitável	11,11
18	21	Aceitável	16,67
$17,59 \pm 0,268(\alpha = 99\%)$	$19,72 \pm 0,281(\alpha = 99\%)$		

Capítulo 7

Apêndice B - Códigos-fonte

Código-fonte 6.1: Código do *firmware* desenvolvido.

```
1  //-----------------------------------------------------------------------
2  // Includes
3  //-----------------------------------------------------------------------
4  #include "c8051F320.h"                  // SFR declarations
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include "Registradores.c" // clock da fpaa em 100kHz
8
9  //-----------------------------------------------------------------------
10 // Global
11 //-----------------------------------------------------------------------
12 #define SYSCLK        12000000          // SYSCLK frequency in Hz
13 #define BAUDRATE      9200             // Baud rate of UART in bps
14
15 // FPAA
16 #define DCLK P0_0
17 #define DIN P0_2
18 #define ACTIVATE P0_3
19 #define SYSCLK        12000000          // Internal oscillator frequency in Hz
20 #define CEX0_FREQUENCY  100000          // Frequency to output on CEX0 (Hz)
21
22 // CRC
23 #define VALORINICIAL 0
24 #define POLY   0xA001
25
26 unsigned int crc;  // utilizado para comparar o crc em decimal recebido pelo
       bt, com o crc gerado pelo fw
27 __xdata unsigned char vetor[900];
28
29 //-----------------------------------------------------------------------
30 // Function PROTOTYPES
31 //-----------------------------------------------------------------------
32 void InitFPAA (void);
33 void send_byte_to_chip (unsigned char data);
34 void get_crc (unsigned char in);
35
36 //-----------------------------------------------------------------------
37 // MCU
```

```
38 //---------------------------------------------------------------------
39 __code __at (0x3bfb) unsigned char DC = 0x81;    // Código de identificação do
       dispositivo
40 __code __at (0x3bfc) unsigned char FW1 = 1;      // Versão do firmware da
       aplicação: 1.0
41 __code __at (0x3bfd) unsigned char FW2 = 0;
42
43 //---------------------------------------------------------------------
44 // Firmware ECG
45 //---------------------------------------------------------------------
46 #define Primary_Config_Size    263
47 #define UpdateFilterSize     18 // Valores da transição (arquivo gerado) com os
       bytes comuns para todas as reconfigurações
48
49 __code unsigned char Primary_Data[263] =
50 {
51 0x00, 0x00, 0x00, 0x00, 0x00, 0xD5, 0xB7, 0x22,
52 0x00, 0x80, 0x01, 0x05, 0xCC, 0x00, 0x0C, 0x20,
53 0x00, 0x20, 0x04, 0x00, 0x02, 0x00, 0x7B, 0x00,
54 0x08, 0xFF, 0x03, 0x2A, 0xDE, 0x00, 0x01, 0x0F,
55 0x2A, 0xCE, 0x01, 0x1E, 0x01, 0x48, 0x00, 0x48,
56 0x00, 0x01, 0x0E, 0x10, 0xC3, 0x20, 0x00, 0x00,
57 0x00, 0x00, 0xF0, 0x43, 0x48, 0x00, 0xFF, 0x00,
58 0x10, 0xFF, 0x01, 0x81, 0xFF, 0x00, 0x10, 0xFF,
59 0x01, 0x81, 0x2A, 0xD1, 0x02, 0x01, 0x07, 0x2A,
60 0xDF, 0x02, 0x06, 0x04, 0x00, 0x00, 0xD0, 0x81,
61 0xAE, 0x2A, 0xCB, 0x03, 0x04, 0x98, 0x00, 0xC1,
62 0xAC, 0x2A, 0xDF, 0x03, 0x26, 0x10, 0xC3, 0x01,
63 0x81, 0x06, 0x01, 0x81, 0xD3, 0x00, 0x10, 0xD3,
64 0x00, 0x10, 0x00, 0x07, 0x0A, 0x01, 0x82, 0x07,
65 0x00, 0x00, 0x0A, 0x01, 0x82, 0xAD, 0x00, 0x20,
66 0x00, 0xAD, 0x00, 0x20, 0x00, 0x04, 0x00, 0x01,
67 0xAE, 0x81, 0xAC, 0x2A, 0xCB, 0x05, 0x0B, 0x98,
68 0x00, 0x00, 0x98, 0x00, 0x00, 0x00, 0x41, 0x9D,
69 0x81, 0xAC, 0x2A, 0xDB, 0x05, 0x17, 0x90, 0x00,
70 0x90, 0x00, 0x40, 0xA0, 0x00, 0x10, 0x9F, 0x00,
71 0x10, 0x01, 0x01, 0x81, 0x9F, 0x00, 0x10, 0x00,
72 0x00, 0x9F, 0x00, 0x10, 0x07, 0x2A, 0xDF, 0x06,
73 0x06, 0x04, 0x00, 0x00, 0xB0, 0x00, 0xB0, 0x2A,
74 0xCA, 0x07, 0x0A, 0xC1, 0xAC, 0x00, 0x00, 0x98,
75 0x00, 0x00, 0x00, 0x00, 0x98, 0x2A, 0xDF, 0x07,
76 0x13, 0x80, 0xA0, 0x00, 0x10, 0x9F, 0x00, 0x10,
77 0x01, 0x01, 0x81, 0x9F, 0x00, 0x10, 0x00, 0x00,
78 0x9F, 0x00, 0x10, 0x07, 0x2A, 0xDF, 0x08, 0x06,
79 0x04, 0x00, 0x00, 0x80, 0x00, 0x80, 0x2A, 0xCA,
80 0x09, 0x0A, 0xC1, 0xAC, 0x00, 0x00, 0x98, 0x00,
81 0x00, 0x00, 0x00, 0x98, 0x2A, 0x9F, 0x09, 0x01,
82 0x80, 0x2A, 0x00
83 };
84
85 void Reconfig(unsigned char *array, unsigned char size);
86 void Data_Write(unsigned char data);
87
88 //---------------------------------------------------------------------
89 // FLASH Memory
90 //---------------------------------------------------------------------
91 void FLASH_ByteWrite (unsigned int addr, char byte);
```

```
92 unsigned char FLASH_ByteRead (unsigned int addr);
93 void FLASH_PageErase (unsigned int addr);
94
95 //-----------------------------------------------------------------
96 // MAIN Routine
97 //-----------------------------------------------------------------
98 void main (void)
99 {
100   unsigned int i;
101   unsigned int contaMem = 0;
102   unsigned char byteTam[2]; // bytes utilizados para receber o valor do
        tamanho do vetor, para posteriormente converter para int
103   unsigned int tamanhoVetor;
104   unsigned int crcByCel;
105
106   unsigned char teste;
107   unsigned int valores = 0;
108
109   unsigned int endereco;
110
111   unsigned char Reconfig_Data[] = {0xD5, 0x01, 0x05, 0x80, 0x02, 0x0A, 0xFF, 0
        x00, 0x10, 0xFF,
112                     0x01, 0x81, 0xFF, 0x00, 0x10, 0xFF, 0x2A, 0x00}; // dados
                          default, firmware ECG;
113
114   unsigned int cont = 0;
115
116   Init_Device ();
117
118   InitPAA ();
119   P2_2 = 0;             // Apaga o led para indicar sucesso da inicialização
120
121   // PorB: forçar reset nessa porta
122   P0_6 = 0;
123   for(i = 0; i < 20; i++);
124   P0_6 = 1;
125
126   // seleciona firmware
127   while(RI0 == 0);
128   i = SBUF0;
129   RI0 = 0;
130
131   if(i == 1){
132     /*// enviar para o celular, a versão do firmware escolhida (depuração)
133     SBUF0 = i;
134     while(TI0 == 0);
135     TI0 = 0;*/
136
137     // Ler arquivo ou flash?
138     while(RI0 == 0);
139     i = SBUF0;
140     RI0 = 0;
141
142     /*// Depuração (arquivo ou flash)
143     SBUF0 = i;
144     while(TI0 == 0);
145     TI0 = 0;*/
```

```
146
147     if(i == 0) // ler FLASH
148     {
149       byteTam[0] = FLASH_ByteRead (0x37FF);
150       byteTam[1] = FLASH_ByteRead (0x37FE);
151
152       // Converte o valor para inteiro
153       tamanhoVetor = (byteTam[1] << 8) | byteTam[0];
154
155       for(i = 0x37FD; i > (0x37FD - tamanhoVetor + 2); i--){
156         teste = FLASH_ByteRead (i);
157
158         /* // depuração
159         SBUF0 = teste;
160         while(TI0 == 0);
161         TI0 = 0;*/
162         vetor[cont] = teste;
163         cont++;
164       }
165
166       for(i = 0; i < tamanhoVetor - 2; i++){
167         send_byte_to_chip (vetor[i]);
168       }
169
170       while (1) {
171         while(AD0INT == 0);
172         AD0INT = 0;
173         SBUF0 = ADC0H;
174         while(TI0 == 0);
175         TI0 = 0;
176         if(RI0 == 1){
177           if(SBUF0 == '1') P2_2 = !P2_2;
178           SBUF0 = 't';
179           while(TI0 == 0);
180           TI0 = 0;
181           RI0 = 0;
182         }
183       }
184     }
185     if(i == 1) // ler ARQUIVO
186     {
187       FLASH_PageErase(0x36B0); // 14000d
188       FLASH_PageErase(0x34BC); // 13500d
189
190       for(i = 0; i < tamanhoVetor - 2; i++){
191         get_crc(vetor[i]);
192       }
193
194       /*// Enviar o o valor crc calculado (depuração)
195       SBUF0 = crc;
196       while(TI0 == 0);
197       TI0 = 0;
198
199       SBUF0 = crc >> 8;
200       while(TI0 == 0);
201       TI0 = 0;
202
```

```
203         SBUF0 = 'M';  // caractere conhecido
204         while(TI0 == 0);
205         TI0 = 0;*/
206
207         while(P3_0 == 1);
208         P2_2 = 0; // apaga o led
209
210         SBUF0 = vetor[tamanhoVetor - 1];
211         while(TI0 == 0);
212         TI0 = 0;
213
214         SBUF0 = vetor[tamanhoVetor - 2];
215         while(TI0 == 0);
216         TI0 = 0;
217
218         crcByCel = (vetor[tamanhoVetor - 1] << 8) | vetor[tamanhoVetor - 2];
219
220         if(crcByCel == crc){
221           SBUF0 = 's';
222           while(TI0 == 0);
223           TI0 = 0;
224
225           endereco = 0x37FF;
226
227           // Capturar o tamanho do vetor
228           for(i = 0; i < 2; i++){
229             while(RI0 == 0);
230             byteTam[i] = SBUF0;
231             RI0 = 0;
232
233             FLASH_ByteWrite(endereco, SBUF0);
234             endereco--;
235           }
236
237           /*// Enviar o tamanho recebido (depuração)
238           SBUF0 = byteTam[0];
239           while(TI0 == 0);
240           TI0 = 0;
241
242           SBUF0 = byteTam[1];
243           while(TI0 == 0);
244           TI0 = 0;
245
246           SBUF0 = 'M';
247           while(TI0 == 0);
248           TI0 = 0;*/
249
250           // Converte o valor para inteiro
251           tamanhoVetor = (byteTam[1] << 8) | byteTam[0];
252
253           // Quando pressionar o push-button...
254           while(P3_0 == 1);
255           P2_2 = !P2_2; // deixa o led aceso
256
257           for(i = 0; i < tamanhoVetor; i++){
258             while(RI0 == 0);
259             vetor[i] = SBUF0;
```

```
260             RI0 = 0;
261
262             FLASH_ByteWrite(endereco, vetor[i]);
263             endereco--;
264           }
265
266           /*for(i = 0x3801; i > (0x3801 - tamanhoVetor); i--){
267             teste = FLASH_ByteRead (i);
268
269             SBUF0 = teste;
270             while(TI0 == 0);
271             TI0 = 0;
272           }*/
273
274           for(i = 0; i < tamanhoVetor - 2; i++){
275             send_byte_to_chip (vetor[i]);
276           }
277
278           while (1) {
279             while(ADCINT == 0);
280             ADCINT = 0;
281             SBUF0 = ADC0H;
282             while(TI0 == 0);
283             TI0 = 0;
284             if(RI0 == 1){
285               if(SBUF0 == '1') P2_2 = !P2_2;
286               SBUF0 = 't';
287               while(TI0 == 0);
288               TI0 = 0;
289               RI0 = 0;
290             }
291           }
292         }
293         else{
294           SBUF0 = 'n';
295           while(TI0 == 0);
296           TI0 = 0;
297         }
298       }
299     }
300     else{
301       /*// enviar para o celular, a versão do firmware escolhida (depuração)
302       SBUF0 = i;
303       while(TI0 == 0);
304       TI0 = 0;*/
305
306       // Utilizar a configuração primária (1) ou reconfigurar (0)?
307       while(RI0 == 0);
308       i = SBUF0;
309       RI0 = 0;
310
311       while(P3_0 == 1);
312       P2_2 = !P2_2; // deixa o led aceso até o fim da execução
313
314       if(i == 0) // reconfigurar
315       {
316         for(i = 0; i < Primary_Config_Size; i++){
```

```
317           Data_Write (Primary_Data[i]);
318         }
319
320         for(i = 0; i < UpdateFilterSize; i++){
321           while(RI0 == 0);
322           Reconfig_Data[i] = SBUF0;
323           RI0 = 0;
324         }
325
326         // depurar: testar se os bytes da reconfig estão corretos
327         for(i = 0; i < UpdateFilterSize; i++){
328           SBUF0 = Reconfig_Data[i];
329           while(TI0 == 0);
330           TI0 = 0;
331         }
332
333         Reconfig(Reconfig_Data, UpdateFilterSize);
334
335         while (1) {
336           while(ADCINT == 0);
337           ADCINT = 0;
338           SBUF0 = ADC0H;
339           while(TI0 == 0);
340           TI0 = 0;
341
342           // acender/apagar led através do botão no appAndroid
343           if(RI0 == 1){
344             if(SBUF0 == '1') {
345               P2_2 = !P2_2;
346
347               for(i = 0; i < UpdateFilterSize; i++){
348                 while(RI0 == 0);
349                 Reconfig_Data[i] = SBUF0;
350                 RI0 = 0;
351               }
352
353               Reconfig(Reconfig_Data, UpdateFilterSize);
354             }
355           }
356         }
357       }
358       if(i == 1) // primária
359       {
360         for(i = 0; i < Primary_Config_Size; i++){
361           Data_Write (Primary_Data[i]);
362         }
363
364         while (1) {
365           while(ADCINT == 0);
366           ADCINT = 0;
367           SBUF0 = ADC0H;
368           while(TI0 == 0);
369           TI0 = 0;
370
371           // acender/apagar led através do botão no appAndroid
372           if(RI0 == 1){
373             if(SBUF0 == '1') P2_2 = !P2_2;
```

```
374             TI0 = 0;
375             RI0 = 0;
376           }
377         }
378       }
379   }
380 }
381
382 void InitVPAA (void)
383 {
384   DIN = 0;
385   DCNK = 0;
386 }
387
388 void send_byte_to_chip(unsigned char data)
389 {
390   unsigned char bit = 8;
391   unsigned int teste = 0;
392
393       while (bit)
394       {
395             if (data & 0x80)
396           DIN = 1;
397             else
398           DIN = 0;
399
400       DCNK = 1;
401
402       for(teste=0;teste < 20;teste++); //Pequeno retardo no envio da
              configuração
403
404       DCNK = 0;
405             data <<= 1;
406             bit--;
407       }
408 }
409
410 void get_crc (unsigned char in)
411 {
412     unsigned char ctr, temp;
413
414     for (ctr = 8; ctr > 0; --ctr)
415     {
416         temp = in ^ (unsigned char) crc; // fazer no próximo bit
417         crc >>= 1; // atualiza a variável global crc
418         if (temp & 0x01) // se LSB XOR == 1
419         crc ^= POLY; // então faz o XOR polinomial com crc
420         in >>= 1; // próximo bit
421     }
422 }
423
424 void Data_Write(unsigned char data) // enviar os bytes pela SPI
425 {
426   unsigned char bit = 8;
427
428   while (bit--)
429   {
```

```
430     if (data & 0x80)
431       DIN = 1;
432     DCLK = 1;
433     DCLK = 0;
434     DIN = 0;
435     data <<= 1;
436   }
437 }
438
439 void Reconfig(unsigned char *array, unsigned char size) // envia os dados de
      reconfiguração para a FPAA
440 {
441   unsigned char i;
442
443   for (i = 0 ; i < size ; i++)
444     Data_Write(array[i]);
445 }
446
447 void FLASH_PageErase (unsigned int addr) // Apagar a memória FLASH
448 {
449    __xdata char *pwrite;           // ponteiro para acessar a FLASH
450
451    // altera a velocidade do clock, depois o restaura
452    CLKCON = 0x80;
453
454    RSTSRC = 0x02;
455
456    pwrite = (__xdata char*) addr;
457
458    FLKEY  = 0xA5;                  // Chave - parte 1
459    FLKEY  = 0xF1;                  // Chave - parte 2
460    PSCTL |= 0x03;                  // PSWE = 1; PSEE = 1
461
462
463    CLKCON = 0x80;
464
465    RSTSRC = 0x02;
466    *pwrite = 0;                    // apaga a página
467
468    PSCTL &= ~0x03;                 // PSWE = 0; PSEE = 0
469 }
470
471 void FLASH_ByteWrite (unsigned int addr, char byte) // gravar na memória FLASH
472 {
473    __xdata char *pwrite;           // ponteiro para escrever na FLASH
474
475    EA = 0;                            // desabilitar interrupções
476
477    // altera a velocidade do clock, depois o restaura
478    CLKCON = 0x80;
479    RSTSRC = 0x02;
480
481    pwrite = (__xdata char *) addr;
482
483    FLKEY  = 0xA5;                     // Chave - parte 1
484    FLKEY  = 0xF1;                     // Chave - parte 2
485    PSCTL |= 0x01;                     // PSWE = 1
```

```
486
487     VDM0CN = 0x80;
488     RSTSRC = 0x02;
489
490     *pwrite = byte;                     // escreve o byte
491
492     PSCTL &= ~0x01;                     // PSWE = 0
493 }
494
495 unsigned char FLASH_ByteRead (unsigned int addr) // lê a memória FLASH
496 {
497     __code char *pread;             // aponta a leitura FLASH
498     unsigned char byte;
499
500     EA = 0;                             // desabilita interrupções
501     pread = (__code char*) addr;
502
503     byte = *pread;                      // lê o byte
504
505     return byte;
506 }
507
508 //-----------------------------------------------------------------------
509 // End Of File
510 //-----------------------------------------------------------------------
```

Código-fonte 6.2: Código do arquivo "*Registradores.c*".

```
1  #include "compiler_defs.h"
2  #include "C8051F320_defs.h"
3
4  void PCA_Init()
5  {
6      PCA0CN    = 0x40;
7      PCA0MD    &= ~0x40;
8      PCA0MD    = 0x08;
9      PCA0CPM0  = 0x46;
10     PCA0CPH0  = 0x3C;
11 }
12
13 void Timer_Init()
14 {
15     TCON      = 0x40;
16     TMOD      = 0x20;
17     CKCON     = 0x11;
18     TH1       = 0x64;
19     TMR2CN    = 0x04;
20     TMR2RLL   = 0x58;
21     TMR2RLH   = 0x9E;
22     TMR2L     = 0x58;
23     TMR2H     = 0x9E;
24 }
25
26 void UART_Init()
27 {
28     SCON0     = 0x30;
29 }
```

```
30
31 void ADC_Init()
32 {
33     AMX0P      = 0x0C;
34     AMX0N      = 0x1F;
35     ADC0CF     = 0x5C;
36     ADC0CN     = 0x82;
37 }
38
39 void Voltage_Reference_Init()
40 {
41     REF0CN     = 0x08;
42 }
43
44 void Port_IO_Init()
45 {
46     // P0.0  -  Skipped,      Push-Pull,  Digital
47     // P0.1  -  Skipped,      Open-Drain, Digital
48     // P0.2  -  Skipped,      Push-Pull,  Digital
49     // P0.3  -  Skipped,      Open-Drain, Digital
50     // P0.4  -  TX0 (UART0), Push-Pull,  Digital
51     // P0.5  -  RX0 (UART0), Open-Drain, Digital
52     // P0.6  -  Skipped,      Open-Drain, Digital
53     // P0.7  -  CEX0 (PCA),   Push-Pull,  Digital
54
55     // P1.0  -  Unassigned,  Open-Drain, Digital
56     // P1.1  -  Unassigned,  Open-Drain, Digital
57     // P1.2  -  Unassigned,  Open-Drain, Digital
58     // P1.3  -  Unassigned,  Open-Drain, Digital
59     // P1.4  -  Unassigned,  Open-Drain, Digital
60     // P1.5  -  Unassigned,  Open-Drain, Digital
61     // P1.6  -  Unassigned,  Open-Drain, Digital
62     // P1.7  -  Unassigned,  Open-Drain, Digital
63     // P2.0  -  Unassigned,  Open-Drain, Digital
64     // P2.1  -  Unassigned,  Open-Drain, Digital
65     // P2.2  -  Unassigned,  Push-Pull,  Digital
66     // P2.3  -  Unassigned,  Open-Drain, Digital
67
68     P2MDIN     = 0xEF;
69     P0MDOUT    = 0x95;
70     P2MDOUT    = 0x04;
71     P0SKIP     = 0x4F;
72     XBR0       = 0x01;
73     XBR1       = 0x41;
74 }
75
76 void Oscillator_Init()
77 {
78     OSCICN     = 0x83;
79 }
80
81 void Init_Device(void)
82 {
83     PCA_Init();
84     Timer_Init();
85     UART_Init();
86     ADC_Init();
```

```
87      Voltage_Reference_Init();
88      Port_IO_Init();
89      Oscillator_Init();
90  }
```

Código-fonte 6.3: *Script* executado no *Matlab* para auxiliar na obtenção da resposta em frequência.

```
1  clear;clc;
2  fileID = fopen('C:\Users\Testes\Circuito 5\Celular\Circuito5Ok\2014
       _06_11_21_47_14.txt','rb');
3  A = fread(fileID);
4  tam = size(A);
5  B = A(2:2:tam); % desconsidera o caractere 10 (pular linha)
6  figure;plot(B); % plota na tela os sinais lidos
7
8  media = median(B); % extrai a média do vetor
9  tamanho = length(B);
10
11 for i=1 : tamanho
12   B(i) = B(i) - media; % subtrai a média de cada elemento do vetor, a fim de
         calcular corretamente o RMS (Root Mean Square)
13 end
14
15 tamanhoVetor = 500; % define uma "janela" de 500 posiçÕes
16
17 qtde = tamanho/tamanhoVetor;
18 inicio = 1;
19 fim = tamanhoVetor;
20
21 cont = 1;
22
23 for i = 1 : qtde % caminha entre todos os elementos do vetor para calcular o
       RMS de trechos (500 em 500)
24   a = B(inicio:fim);
25   inicio = inicio + 500;
26   fim = fim + 500;
27
28   y(cont) = rms(a);
29   cont = cont + 1;
30   null(a);
31 end
32
33 dlmwrite('C:\Users\Kyoto\Desktop\texto.txt',y,'\n') % salva o vetor y em um
       arçquivo
34
35 figure;plot(y); % plota na tela o vetor y (valores RMS)
```