Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
Curso de Graduação em Engenharia de Teleinformática

Ítalo Cavalcante Sampaio

Sistema de Monitoramento Remoto de Pacientes Implementado em Hardware de Arquitetura ARM

Fortaleza – Ceará
Dezembro 2011

Autor:
Ítalo Cavalcante Sampaio

Orientador:
Prof. Dr. Helano de Sousa Castro

Sistema de Monitoramento Remoto de Pacientes Implementado em Hardware de Arquitetura ARM

Monografia de Conclusão de Curso apresentada à Coordenação do Curso de Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de **Engenheiro de Teleinformática**.

Fortaleza – Ceará
Dezembro 2011

Ítalo Cavalcante Sampaio

Sistema de Monitoramento Remoto de Pacientes Implementado em Hardware de Arquitetura ARM

Esta Monografia foi julgada adequada para a obtenção do diploma de Engenheiro do Curso de Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

Ítalo Cavalcante Sampaio

Banca Examinadora:

Prof. Dr. Helano de Sousa Castro
Orientador

Prof. Alexandre Augusto da Penha Coelho

Prof. Ricardo Jardel Nunes da Silveira

Fortaleza, 12 de dezembro de 2011

Resumo

Com os avanços da computação, sobretudo na área de sistemas embarcados, é cada vez mais fácil realizar tarefas que necessitam de um poder computacional considerável utilizando hardware de baixo custo. Essa nova realidade faz com que outras áreas, não necessariamente relacionadas com a computação, sejam beneficiadas. Um bom exemplo deste fenômeno é a área de saúde. Hoje, é possível desenvolver equipamentos que realizam tarefas essenciais para a saúde de um paciente empregando um baixo custo, utilizando um baixo consumo de energia e com uma grande variedade de funcionalidades.

Neste contexto, este trabalho descreve o desenvolvimento de um sistema de **Home Care** de baixo custo que utiliza tecnologias sem fio para promover uma maior qualidade de vida ao paciente, permitindo que o mesmo tenha mobilidade enquanto é monitorado, ao invés de ficar preso a uma cama de hospital. O sistema foi desenvolvido utilizando um processador Freescale iMX53, que possui um núcleo ARM, rodando o Sistema Operacional Linux. A conexão com os sensores é feita através da tecnologia bluetooth e os sinais vitais do paciente são exibidos em uma página da Web, podendo ser acessados de qualquer lugar por um familiar ou por um médico, o que garante uma maior tranquilidade ao paciente e seus familiares.

Palavras-chaves: Sistemas Embarcados, Linux, ARM, Bluetooth, Telemedicina, Home Care.

Abstract

The current state of the computer field, specially on embedded systems, has allowed tasks that need high amount of computing power to be performed by low cost hardware without major difficulty. This new reality benefits a great amount of areas, not necessarily related to computing, such as the medicine. Nowadays, it is possible to develop an equipment that performs tasks that are essential to the health of a patient with low cost, low power and yet with a wide variety of features.

This work describes the development of a low cost Home Care system, which uses wireless technology in order to provide a better life quality to the patient, allowing some mobility while the monitoring is performed, in contrast to the classic situation, where the patient has to be in a hospital bed during long periods of time. The system was developed by using a Fresscale iMX53 processor, which uses an ARM core, running Linux Operation System. The connection with the sensor uses bluetooth technology and the patient's vital signals are presented through a Web page, which allows them to be accessed from anywhere by a family member or by a doctor, which brings tranquility to both the patient and the family.

Keywords: Embeded Systems, Linux, ARM, Bluetooth, Telemedicine, Home Care.

Dedico este trabalho aos meus pais, Cláudia e James, à minha irmã, Thais e à minha namorada, Eveline, por todo o amor e pelo apoio ao longo da graduação e por toda a vida.

Agradecimentos

Agradeço primeiramente aos meus pais, responsáveis diretos por todas as minhas conquistas, por terem sempre deixado claro a importância do estudo no crescimento enquanto ser humano e profissional, pelo apoio incondicional e pela compreensão nos momentos de ausência.

À minha namorada, Eveline, por estar sempre ao meu lado ao longo desta caminhada, nos momentos de glória e nos mais difíceis, pela compreensão, pelo companheirismo, amor e dedicação.

A todos os colegas e amigos do LESC, pelas incontáveis contribuições a esse e outros trabalhos, sempre que tive necessidade.

Aos meus amigos, por sempre acreditar que eu conseguiria atingir meus objetivos.

Ao professor Helano Castro, por aceitar ser meu orientador e fornecer sua valiosa contribuição para a conclusão deste trabalho.

A todos os alunos do curso de Graduação em Engenharia de Teleinformática, por compartilhar momentos tão especiais ao longo desses 5 anos.

Seu trabalho vai ocupar grande parte da sua vida, e o único jeito de se sentir verdadeiramente satisfeito é fazer o que você acredita ser um ótimo trabalho. E o único jeito de fazer um ótimo trabalho é amar aquilo que você faz.

Steve Jobs

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

SO Sistema Operacional

GPL GNU General Public License

GUI Interface Gráfica de Usuário (*Graphic User Interface*)

API Interface de Programação de Aplicativos (*Application Programming Interface*)

IDE Ambiente Integrado de Desenvolvimento (*Integrated Development Environment*)

RISC Computação com Conjunto de Instruções Reduzido (*Reduced Instruction Set Computing*)

CISC Computação com Conjunto de Instruções Complexo (*Complex Instruction Set Computing*)

PDS Processamento Digital de Sinais

CPSR Registro de Status do Programa Atual(*Current Program Status Register*)

SIMD *Single Instruction, Multiple Data*

SMS Serviço de Mensagens Curtas (*Short Message Service*)

LTIB Linux Target Image Builder

BSP *Board Support Package*

HDP Perfil para Dispositivos de Saúde (*Health Device Profile*)

TFTP Protocolo de Transferência de Arquivos Trivial (*Trivial File Transfer Protocol*)

NFS Sistema de Arquivos em Rede (*Network File System*)

ALU Unidade Lógica Aritimética (*Aritimetic Logic Unit*)

MAC Unidade Multiplicador-Acumulador (*Multiply-Accumulate Unit*)

ECG Eletrocardiograma

GPS Sistema de Posicionamento Global (*Global Positioning System*)

GSM Sistema Global para Comunicações Móveis (*Global System for Mobile Communications*)

MMU Unidade de Gerenciamento de Memória (*Memory Management Unity*)

Capítulo 1

Introdução

1.1 Motivação

Um dos grandes problemas enfrentados pelo Brasil atualmente na área de Saúde é a alta taxa de ocupação nos hospitais da rede pública. Muitos desses hospitais, possuem taxa de ocupação superior a 100%, já que, com todos os leitos ocupados, são obrigados a fazer uso de macas que são colocadas nos corredores para conseguir atender mais pacientes que a capacidade do hospital (BITTENCOURT, 2011). Uma consequência grave desta situação é que, além dos pacientes que conseguem internação não receberem um tratamento adequado, já que os profissionais e os equipamentos não estão presentes em número suficiente para atender à demanda, muitos acabam não conseguindo vagas nos leitos, morrendo em corredores de hospitais ou em consequência das longas esperas por uma vaga (GAZETA, 2011; G1, 2011b; G1, 2011a).

Desta forma, é necessário que sejam criadas soluções para que a demanda por leitos hospitalares seja suprida. A solução óbvia para este problema é a construção de mais hospitais e ampliação dos já existentes. Contudo, esta solução implica num alto custo para os cofres públicos para financiar as obras necessárias, além de impacto na população, já que, durante as reformas, os pacientes que já estão internados precisam ser realocados, gerando muito transtorno para a população. Além disso, o tempo que leva para implementar esse tipo de ação geralmente é muito alto. Outra solução para o problema é diminuir a necessidade dos pacientes utilizarem os leitos já existentes. Uma forma de fazer isto é através de sistemas de *Home Care*, que permitem que pacientes em situações de menor gravidade sejam monitorados em

casa, não necessitando ocupar um leito hospitalar. A adoção desta solução é menos custosa e mais rápida que a construção de novos leitos, além de ser normalmente mais agradável para o paciente, que pode ficar em casa com seus familiares.

Neste trabalho, foi desenvolvido um sistema de *Home Care* que se utiliza de tecnologias sem fio para realizar as medições dos sinais vitais do paciente. Desta forma, o dispositivo pode ficar fixo em algum local da casa, enquanto o paciente, com o sensor acoplado ao seu corpo, tem liberdade para movimentar-se pela casa, sem estar conectado por fios ao dispositivo. Isto promove uma maior qualidade de vida, já que o paciente pode realizar suas atividades normalmente sem se preocupar com o monitoramento.

O desenvolvimento do sensor não faz parte do escopo deste trabalho. Todo o desenvolvimento foi realizado partindo da premissa que está disponível um eletrocardiógrafo portátil compatível com o perfil Bluetooth HDP (LATUSKE, 2009).

1.2 Objetivos

O objetivo principal deste projeto de conclusão de curso é desenvolver o protótipo de um sistema de *Home Care*, capaz de monitorar sinais vitais de um paciente usando tecnologias sem fio e exibir as informações atavés de uma página WEB, permitindo o acesso a essas informações a partir de qualquer lugar.

O projeto tem como objetivos específicos portar e personalizar o kernel do Sistema Operacional Linux para a placa de desenvolvimento utilizada, desenvolver o Device Driver para o receptor bluetooth utilizado, desenvolver a aplicação que colhe e armazena os sinais vitais do paciente via bluetooth, criar uma aplicação que realiza medidas relevantes a partir dos dados colhidos, desenvolver uma solução de servidor WEB e a página WEB para exibir os dados e desenvolver o aplicativo que avisa, via SMS, a família do paciente caso algum valor esteja fora do normal.

1.3 Organização deste trabalho

O restante deste trabalho está dividido em capítulos. No Capítulo 2 é feita uma rápida revisão sobre os principais conceitos utilizados no desenvolvimento deste trabalho. O capítulo fala inicialmente sobre o sistema operacional Linux e sua aplicação em sistemas embarcados. Em seguida, é dada uma visão geral sobre a arquitetura de processadores ARM, suas principais características e as famílias de

processadores ARM. Finalmente, é feita uma revisão sobre os conceitos básicos de sistemas de *Home Care* e eletrocardiografia.

No Capítulo 3 é detalhada a metodologia de desenvolvimento do projeto. Inicialmente, o capítulo mostra a arquitetura proposta para o projeto. Em seguida, é apresentado o ambiente de desenvolvimento, detalhando as ferramentas utilizadas. O capítulo segue apresentando a plataforma de hardware utilizada para o desenvolvimento, a metodologia utilizada para customização do Sistema Operacional utilizado, para o desenvolvimento do software e para a validação do projeto.

Finalmente, o Capítulo 4 apresenta os resultados obtidos com o trabalho, uma análise desses resultados, as conclusões e previsões de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Sistema Operacional Linux

2.1.1 Introdução

O Sistema Operacional Linux foi criado como um projeto pessoal do programador finlandês Linus Torvalds. Em agosto de 1991, Linus postou pela primeira vez no grupo de notícias *comp.os.minix* que estava trabalhando em um Sistema Operacional (SO) livre baseado em Unix para computadores IBM-compatíveis baseados no processador intel 80386 (MINIX, 1991). Aos poucos, o sistema foi evoluindo, ganhando novas funcionalidades como memória virtual e um sistema de arquivos mais sofisticado. Rapidamente, foram criados portes para outras plataformas, como Alpha, SPARC, Motorola MC680x0, PowerPC, e IBM System/390 (BOVET; CESATI, 2000).

O principal diferencial do Linux com relação aos outros SOs da época era seu modelo de negócio baseado em software livre. Todo o código era, e continua sendo até hoje, publicado sob a **GNU General Public License (GPL)** (FREE SOFTWARE FUNDATION, 2007). Esta licença diz que o código fonte pode ser modificado e reutilizado por qualquer pessoa, desde que qualquer software feito usando pedaços de código GPL também seja distribuído junto com o código fonte. O resultado prático da utilização desta licença no Linux é que o desenvolvimento do sistema não fica centralizado. Qualquer usuário pode contribuir com código, customizando e melhorando o sistema à sua maneira, realizando portes para diferentes plataformas e corrigindo defeitos que possam vir em cada novo lançamento. Assim, à medida que

a base de usuários foi crescendo, a quantidade de funcionalidades foi aumentando, assim como a confiabilidade do sistema como um todo.

Em 1994, foi lançada a versão 1.0 do Linux. Esta versão tinha aproximadamente 165.000 linhas de código (TANENBAUM, 2007) e incluia novas funcionalidades como um novo sistema de arquivos, arquivos mapeados em memória e suporte a rede com sockets e implementação da pilha TCP/IP. Esta versão incluia também vários novos *Device Drivers*. Nos dois anos seguintes, várias revisões foram lançadas.

Em 1996, foi lançada a versão 2.0. Nesta versão, havia cerca de 470.000 linhas de código em C e 8.000 linhas de código em *assembly*. As novidades incluiam suporte a arquiteturas de 64 bits, multiprogramação simétrica, novos protocolos de rede, dentre outras funcionalidades. Novamente, boa parte do código adicionado consistia em *Device Drivers*, fazendo com que o sistema suportasse cada vez mais dispositivos.

Por ser baseado em Unix, o Linux acabou herdando boa parte dos softwares desenvolvidos para Unix, já que os mesmos puderam ser portados sem grandes problemas. Estes programas incluem utilitários, o gerenciador de interface gráfica X, além de diversos aplicativos de rede. Além disso, foram desenvolvidas duas Interface Gráfica de Usuário (*Graphic User Interface*)s (GUIs) para Linux: GNOME e KDE. Assim, o Linux conseguiu uma relativa popularização herdando parte do ecossistema de usuários UNIX, somado ao ecossistema que se formou em torno do próprio Linux.

Somente em 2011, foi lançada a versão 3.0 do Kernel, e segundo o próprio Linus Torvalds, a atualização da numeração, de 2.X para 3.X não tem relação com nenhuma grande mudança estrutural no kernel, sendo mais ligada às comemorações de 20 anos da criação do Linux (TORVALDS, 2011).

Hoje, o Linux ocupa apenas uma pequena parcela do mercado de computadores pessoais, que era seu foco inicial. Por outro lado, o sistema é líder de mercado quando se trata de supercomputadores e grandes servidores Web. Segundo (TOP500, 2011), 82,6% dos 500 supercomputadores mais potentes do mundo rodam o SO Linux. Outro nicho onde o Linux tem se destacado é no mercado de computação móvel, já que o Android, Sistema Operacional mais usado em *smartphones* (PCWORLD, 2011), é construído sobre uma camada de kernel Linux, conforme pode ser visto na Figura 2.1.

Devido à sua característica de software aberto, o Linux já foi portado para

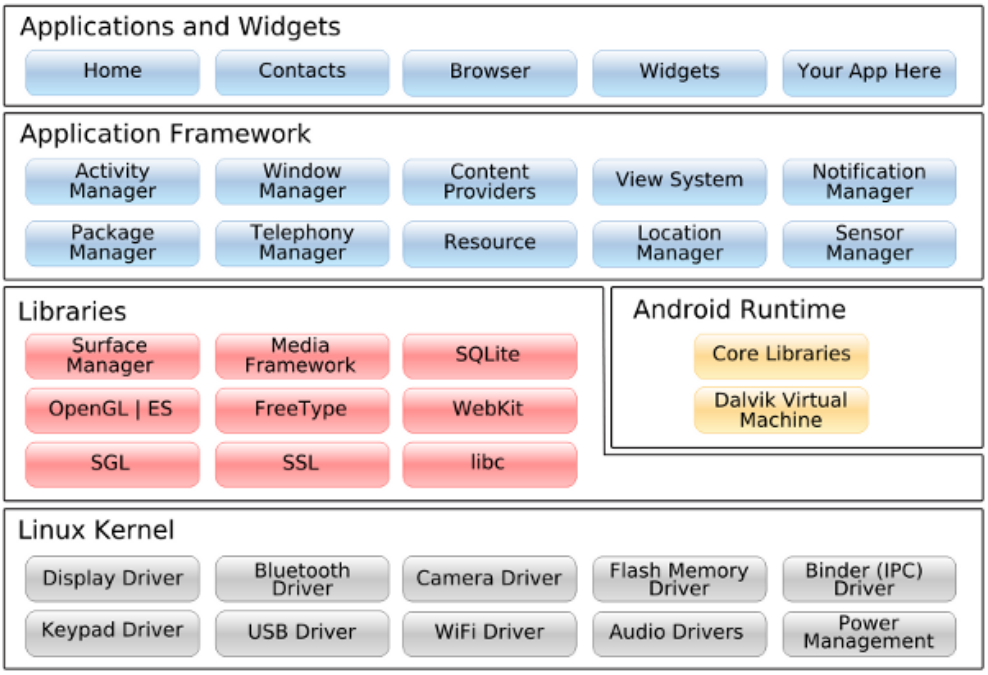

Figura 2.1: Visão em camadas da arquitetura do Android

uma série de arquiteturas, tornando o mesmo uma escolha interessante para o uso em sistemas embarcados. Algumas arquiteturas que suportam Linux são ARM, Atmel AVR32, Freescale 68k, IBM System/390 e Z/Architecture, Intel IA-64 e x86, Microblaze, MIPS, OpenRISC, PowerPC, dentre muitas outras. Podemos ver na lista de arquiteturas suportadas que é possível rodar Linux em sistemas para atender aos mais variados requisitos, desde grandes mainframes, passando pelos computadores pessoais high-end até sistemas embarcados de baixo consumo.

2.1.2 Descrição

Confirme mencionado na Seção 2.1.1, o Linux é um SO baseado no UNIX. Desta forma, apesar de não contar com código UNIX em seu kernel, muitas das funcionalidades do sistema são baseadas nas do UNIX. Como muitos outros sistemas baseados em UNIX, o Linux é compatível com alguns padrões de desenvolvimento, sendo o principal deles o padrão IEEE POSIX, que consiste em uma Interface de Programação de Aplicativos (*Application Programming Interface*) (API) que pode ser utilizada por todos os programas de usuário. O POSIX, no entanto, define apenas a interface de programação, que deve ser a mesma em todos os SOs que possuem suporte ao padrão, não definindo nenhum detalhe de implementação dos mesmos.

Apesar de não ser o único SO baseado no UNIX e de possuir uma série de semelhanças com outros SOs, podemos ver em (BOVET; CESATI, 2000) uma série de características que fazem com que o Linux tenha destaque em meio aos outros SOs baseados em UNIX. As principais características são citadas a seguir.

Linux é gratuito: Escolhendo o Linux para um projeto, corta-se o custo que poderia haver com o SO.

Completamente costumizável em todos os seus componentes: A GPL permite que o código fonte de qualquer componente do sistema seja modificado para se adequar às necessidades do projeto.

Capacidade de rodar em hardware de baixo custo: É possível construir um sistema simples utilizando hardware de baixíssimo custo rodando Linux. É possível também gerar imagens do Linux com um pequeno *footprint* de memória.

Eficiente: Sistemas Linux costumam explorar ao máximo a capacidade do hardware, rodando muito rápido. O projeto do SO em si tem foco no desempenho, direcionando as decisões de projeto para rejeitar qualquer funcionalidade que penalize o desempenho.

Alta qualidade do código fonte: Os desenvolvedores do kernel tendem a gerar código bastante estável, com baixa taxa de falhas e baixo tempo de manutenção.

Suporte: Graças à sua base de usuários sempre ativa, é muito fácil conseguir suporte para sistemas Linux. Normalmente, os usuários respondem às dúvidas em listas de discussão em pouco tempo e *device drivers* são facilmente encontrados na internet.

2.2 Linux em Sistemas Embarcados

Apesar de ter sido inicialmente desenvolvido com o intuito de ser utilizado em *desktops*, o Linux é bastante utilizado em sistemas embarcados. Neste trabalho, considera-se um sistema embarcado qualquer sistema de computação desenvolvido para um propósito bem definido, em oposição aos sistemas de propósito geral, que podem atender a uma grande quantidade de aplicações. Devido à sua natureza

específica, é desejável que um sistema embarcado tenha baixo custo para tornar-se vantajoso em relação a um sistema de propósito geral. Para garantir o baixo custo, utiliza-se a menor quantidade possível de componentes de hardware e software, tornando o sistema o mais otimizado possível.

Em sistemas embarcados simples, como controles remotos, calculadoras, dispositivos periféricos como teclados e monitores, dentre outros, costuma-se usar um *firmware*, que é um programa que implementa desde a camada de aplicação até a camada mais baixa, configurando os registros internos dos dispositivos de hardware. No entanto, à medida que o sistema evolui em complexidade, torna-se necessário o uso de ferramentas que não estão disponíveis em um *firmware*, mas podem ser encontradas em um SO, como sincronização de processos, computação concorrente, gerenciamento de memória e sistema de arquivos. Cabe ao desenvolvedor decidir até que ponto é vantajoso utilizar apenas um *firmware* e a partir de onde passa a ser melhor utilizar um SO.

2.2.1 Características de Sistemas Embarcados Linux

Sistemas Embarcados Linux têm diversas características que os diferem dos sistemas Linux de propósito geral, como os *desktops*. A seguir, são listadas algumas características que encontramos nesse tipo de sistema, conforme listadas em (YAGHMOUR, 2003).

Tamanho

O tamanho de um sistema embarcado Linux influencia e é influenciado por diversos fatores. Primeiramente, é importante notar que um sistema embarcado Linux pode ter diferentes tamanhos físicos, indo desde *clusters* que ocupam prédios inteiros até relógios de pulso. O tamanho físico influencia diretamente na quantidade e no tipo de componentes que podem ser acomodados no sistema, sendo esse um fator determinante na capacidade computacional do sistema como um todo. Assim, fatores como velocidade da CPU e quantidade de memória RAM e memória não volátil são fortemente influenciados pelo tamanho físico do sistema.

Requisitos de Tempo

Para alguns sistemas embarcados, existe uma necessidade que o sistema reaja dentro de um período de tempo pré-definido. Quando há essa necessidade, dizemos

que se trata de um sistema de Tempo Real. Um sistema de Tempo Real pode ser classificado em dois tipos básicos: sistemas de tempo real críticos, ou *hard real time systems*, e sistemas não-críticos, ou *soft real time systems*.

Um sistema de tempo real é considerado do tipo *hard* caso ele precise necessariamente reagir dentro do tempo estimulado, caso contrário algo catastrófico acontecerá. Alguns exemplos clássicos de um sistemas de tempo real críticos são computadores de bordo de aeronaves, sistemas de monitoramento de caldeiras, além de sistemas em que uma falha em seu funcionamento possa por em risco vidas humanas. Já os sistemas do tipo soft também possuem restrições temporais, mas nada de catastrófico ocorre caso os requisitos não sejam atendidos. Normalmente quando um sistema *soft* não cumpre seu requisito temporal, ocorre apenas uma degradação na experiência do usuário. Um exemplo deste tipo de sistema é um sistema multimídia como um *mp3 player*.

O Linux não é considerado um SO de tempo real. Existem, no entanto, variações do Linux desenvolvidas para dar suporte a aplicações de tempo real do tipo *hard*. Um exemplo notável é o RTLinux (YODAIKEN, 1999), que consiste basicamente em um SO que roda o Linux inteiro como um de seus processos, sendo totalmente preemptável. Este SO, no entanto é um projeto paralelo, não relacionado com o projeto do kernel do Linux.

Conectividade

A conectividade, ou seja, a capacidade do sistema estar conectado a uma rede, é cada vez mais uma necessidade para a maioria dos sistemas embarcados. Mesmo sistemas que tradicionalmente não precisam conectar-se a uma rede, como televisões, geladeiras e torradeiras, agora possuem acesso à internet. Desta forma, este é um dos itens essenciais no design de um sistema embarcado Linux. Os dispositivos podem ter uma ou mais formas de conectar-se à internet, podendo ser via cabos ou sem fio. O Linux incorpora nativamente diversos protocolos de rede, o que facilita bastante o *design* de dispositivos com estas características.

Interação com o usuário

O grau de interação com o usuário pode variar bastante de um sistema para outro. Alguns sistemas embarcados estão completamente focados na interação com o usuário, como é o caso dos leitores de livros eletrônicos. Estes dispositivos

apresentam uma tela para mostrar as informações ao usuário e várias interfaces de entrada para receber os dados. Por outro lado, existem dispositivos, como os de controle industrial, por exemplo, que possuem poucas formas de interação com o usuário, como LEDs e botões, ou mesmo nenhuma interação, realizando todo o trabalho sem a interferência de um operador.

2.2.2 Vantagens do Linux para Sistemas Embarcados

Existem uma série de motivos por trás da supremacia do Linux em sistemas embarcados em comparação com outros SOs. Esta seção lista alguns desses motivos. Algumas destas vantagens estão presentes tanto em ambiente *desktop* quanto em servidores e em ambientes corporativos, enquanto outras são específicas para ambientes embarcados.

Qualidade e confiabilidade do código

Em geral, a qualidade e a confiabilidade são duas grandezas desejáveis, porém difíceis de medir em um código, por serem muito subjetivas. Entretanto, é possível fazê-lo analisando uma série de características que a maioria dos programadores considera que contribuem para aumentar a qualidade e a confiabilidade. Algumas características que contribuem para a qualidade do código são a modularidade, a clareza do código, a extensibilidade e a configurabilidade. Por outro lado, para ser considerado confiável, o código deve ter características como previsibilidade, capacidade de recuperação de erros e longevidade.

A maior parte dos programadores concorda que o kernel do Linux se encaixa nas descrições de qualidade e confiabilidade. A razão principal disto é o modelo de software aberto, que conta com a contribuição da comunidade que está sempre debatendo os possíveis problemas de *design* e sugerindo melhorias. Geralmente, quando alguma decisão ruim de *design* é tomada, as várias pessoas espalhadas pelo mundo responsáveis pelos testes apontam o problema e o mesmo é rapidamente solucionado.

O Linux tem mostrado bastante longevidade ao longo do tempo, não apresentando grandes problemas de *design* com o tempo de uso. Além disso, é possível selecionar quais componentes do sistema serão instalados, e quais serão evitados. O mesmo vale para as funcionalidades do kernel, onde é possível escolher quais serão carregadas na imagem final do kernel. Uma maneira simples de atestar

a qualidade do código é observar as várias listas de discussão sobre o Linux. Nestas listas é possível ver que os problemas apontados costumam ser resolvidos muito rapidamente e novas funcionalidades são adicionadas com velocidade semelhante, o que mostra que o código atende às características listadas acima.

Disponibilidade do código

Um fator primordial para a adoção do Linux em um sistema embarcado é o fato de que todo o código fonte do Linux, bem como as ferramentas necessárias para sua compilação, estão disponíveis para todos sem nenhuma restrição de acesso. Isso permite que, quando ocorre algum problema do sistema, toda a comunidade possa procurar a causa do mesmo no código fonte, fazendo com que o problema seja resolvido rapidamente. Em outros SOs embarcados, cujo código fonte não está disponível ou precisa ser comprado, não existe essa possibilidade. Outra vantagem da disponibilidade do código é que o próprio desenvolvedor pode solucionar o problema sem ajuda externa, apenas investigando o código fonte. É possível também ter um entendimento melhor do funcionamento do SO quando se tem o código disponível, podendo criar rotinas mais otimizadas e de acordo com as rotinas internas do sistema.

A disponibilidade do código também contribui com a padronização das plataformas, uma vez que é possível desenvolver sistemas Linux completamente baseados em software de código aberto. Os desenvolvedores, então, tendem a adotar padrões de projetos para poder tirar proveito do que já foi desenvolvido dentro do mesmo padrão. Um exemplo desse tipo de comportamento, por parte dos desenvolvedores, é o projeto OpenMoko (OPENMOKO Inc, 2011), que consiste em um telefone celular baseado em Linux que já possui as funcionalidades básicas necessárias para este tipo de dispositivo. Assim, um desenvolvedor interessado em projetar um telefone celular baseado em Linux pode partir do projeto OpenMoko, utilizando as funcionalidades básicas oferecidas por este projeto e se concentrando apenas no diferencial do seu produto. Com essa abordagem, o desenvolvedor ganha bastante tempo no desenvolvimento.

Suporte a Hardware

O Linux suporta uma grande quantidade de plataformas e dispositivos de hardware. Embora muitos fabricantes não disponibilizem *drivers* Linux para seus

produtos, a comunidade costuma desenvolver e dar suporte a esses *drivers*. Assim, não há o risco de o fabricante descontinuar o suporte a determinado hardware, já que a comunidade é responsável pelo suporte. Além de suportar uma grande quantidade de dispositivos, o Linux já foi portado para uma grande quantidade de plataformas, rodando em diferentes arquiteturas. Mesmo que a arquitetura escolhida ainda não seja suportada, é sempre possível contactar alguém que teve uma experiência semelhante de porte do Linux para diferentes plataformas e obter ajuda durante o processo. Graças a essa portabilidade, o Linux pode ser utilizado em sistemas de diferentes tamanhos, com diferentes requisitos e com uma ampla gama de custos.

Protocolos de comunicação e padrões de software

Outro fator que contribui para o destaque do Linux como SO em sistemas embarcados é a variedade de protocolos e padrões de software suportados. Assim, é fácil integrar sistemas Linux com outros sistemas já existentes. Um sistema Linux pode, por exemplo, conectar-se a uma rede local Windows sem problemas, atuando inclusive como servidor sem que os clientes percebam mudanças. O mesmo vale para protocolos como o bluetooth, Sistema de Posicionamento Global (*Global Positioning System*) (GPS), Sistema Global para Comunicações Móveis (*Global System for Mobile Communications*) (GSM), dentre outros, onde um sistema Linux é facilmente integrado a uma infraestrutura previamente estabelecida.

Ferramentas disponíveis

Devido à própria natureza do sistema, usuários do Linux costumam seguir a filosofia do software de código aberto. Isto significa que, normalmente, quando alguém desenvolve uma ferramenta para Linux, o código é disponibilizado publicamente. Isto ajuda bastante o desenvolvimento, já que muitas ferramentas que podem ser necessárias, como compiladores, *device drivers* e aplicativos em geral já estão disponíveis, pelo menos em um estágio inicial. Em outros casos, existem ferramentas semelhantes que podem ser usadas como base para o desenvolvimento. Em todo caso, o esforço de desenvolvimento é reduzido devido à disponibilidade das ferramentas, bem como dos autores das mesmas, que costumam fornecer suporte no desenvolvimento.

Custo

Normalmente, um projeto tem três custos associados ao software. O primeiro é o custo da aquisição das licenças de desenvolvimento. Normalmente, estas licenças limitam o número de desenvolvedores que podem trabalhar com o projeto, além de possuirem uma data de expiração. O segundo custo é a aquisição de ferramentas adicionais, como Ambiente Integrado de Desenvolvimento (*Integrated Development Environment*)s (IDEs) e designs de referência. Finalmente, depois que o produto está pronto, é possível que a empresa cobre um custo por unidade produzida, chamado de *royalty*. Dependendo do projeto, estes custos de software podem impactar fortemente no custo total do projeto, fazendo com que o produto final fique menos competitivo.

Em um projeto que utiliza o Linux, este modelo não se aplica. Primeiramente, tanto o SO em si quanto as ferramentas de desenvolvimento são gratuitos, eliminando os dois primeiros custos. Em segundo lugar, a licença permite que o sistema seja distribuído livremente, não havendo o terceiro custo, vindo de *royalties*. Os custos de software que podem existir em um projeto Linux aparecem quando o desenvolvedor opta por adiquirir uma distribuição pronta, com todos os pacotes inclusos e devidamente compilados. Nesse caso, muitas vezes, as empresas cobram pela distribuição ou pelo suporte técnico. Em outros casos, as empresas cobram para realizar o porte do Linux para a plataforma desejada. Em todos os casos, o gasto adicional ocorre apenas caso o desenvolvedor queira economizar no tempo de projeto, não sendo um gasto absolutamente necessário.

É importante ter em mente que há uma tendência cada vez maior de convergência entre os sistemas de propósito geral, como computadores de mesa, e os sistemas embarcados. Com a grande propagação de *smartphones* e *tablets* no mercado, estes conceitos se confundem. No projeto destes dispositivos, estão presentes tanto preocupações do mundo embarcado, como grande autonomia de bateria, conectividade e portabilidade quanto preocupações comuns em *desktops*, como resposta rápida ao usuário e amplo suporte a aplicações. Assim, é necessário que os SOs sejam adaptados aos dois mundos, pegando as principais características de cada um. Neste sentido, o Linux é um bom candidato a SO utilizado nesses sistemas, já que ele tem se mostrado adaptado a ambas aplicações. De fato, o SO Android, que roda sobre um kernel Linux, é hoje o mais utilizado em *smartphones* (PCWORLD, 2011), além de ser o segundo mais utilizado em *tablets* (TGDAILY, 2011), o que

mostra que o Linux tem sido muito bem sucedido neste tipo de ambiente.

2.3 Processadores ARM

2.3.1 Visão Geral da Arquitetura de Processadores ARM

Nesta seção, são descritas resumidamente as características da arquitetura do *core* ARM. Do ponto de vista do programador, o *core* pode ser visto como um conjunto de unidades funcionais conectadas por barramentos de dados. Na Figura 2.2, pode-se ter esta visão, onde as linhas representam barramentos e as caixas representam uma unidade funcional ou uma área de armazenamento. Pode-se ver na figura que os dados entram no *core* sempre pelo barramento de dados. Estes dados podem ser tanto instruções que devem ser executadas quanto dados propriamente ditos, como parâmetros de funções. Na mesma figura, é mostrada uma implementação do ARM usando a arquitetura de Von Neumann, onde os dados e instruções dividem o mesmo barramento. Existe também a possibilidade de termos implementações do ARM utilizando arquitetura Harvard, ou seja, barramentos separados para instruções e dados.

Como todos os processadores RISC, o ARM usa uma arquitetura do tipo *load-store*. Isto quer dizer que existem dois tipos de instruções para mover dados para dentro ou para fora do computador: instruções de carregamento (ou *load*), que copiam dados da memória para os registros internos do processador e funções de armazenamento (ou *store*), que fazem a cópia de dados dos registros internos para a memória. Não há nenhuma instrução que faça a manipulação dos dados diretamente na memória, logo, todo o processamento é feito dentro do processador.

Cada instrução que chega no processador é tratada inicialmente pelo *Instruction Decoder*. Dentre outras coisas, este bloco é responsável por identificar a qual conjunto de instruções cada instrução pertence. O *Register File* é responsável por armazenar os dados que chegam ao processador. Este bloco é composto por registros de 32 bits. O bloco *Sign extend* trata dados de 8 e 16 bits com sinal, transformando-os em 32 bits para que sejam armazenados no *Register File*. Esta etapa é importante porque o ARM é um processador de 32 bits, ou seja, ele interpreta os dados presentes nos registros internos como dados de 32 bits.

A maior parte das instruções do ARM utilizam dois registros de entrada, R_m e R_n, e um de saída, R_d. Os operandos de entrada de uma instrução são lidos dos

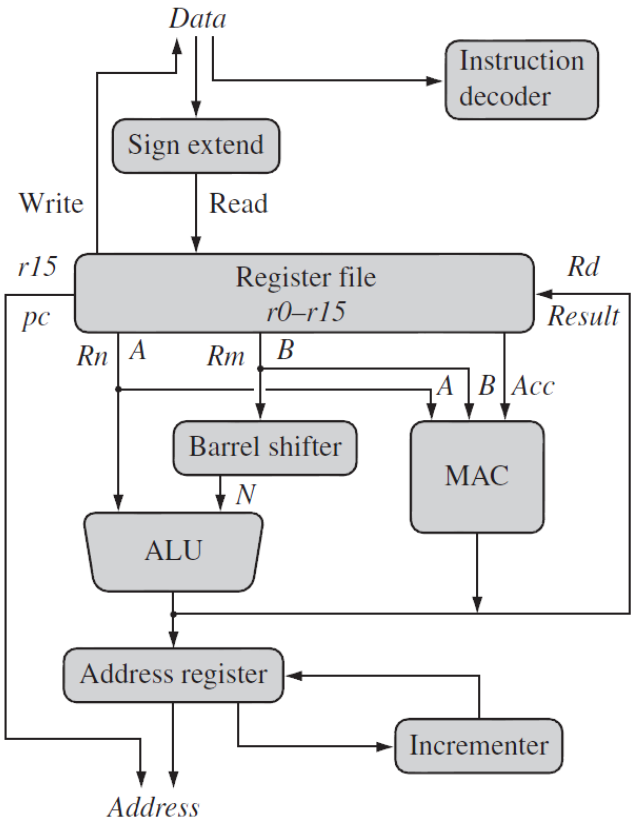

Figura 2.2: Fluxo dos dados no core do processador ARM

registros através dos barramentos A e B, mostrados na Figura 2.2.

Após o armazenamento dos dados de entrada, a Unidade Lógica Aritimética (*Aritimetic Logic Unit*) (ALU) ou a Unidade Multiplicador-Acumulador (*Multiply-Accumulate Unit*) (MAC) carrega os valores R_m e R_n e computa o resultado. Instruções de processamento de dados escrevem o resultado R_d diretamente no *Register File*, enquanto instruções do tipo *load-store* usam a ALU para gerar um endereço que será armazenado no *Address Register* para, em seguida, ser colocado no barramento de endereços.

Depois de passar por todos os blocos funcionais, o resultado R_d é escrito de volta no *Register File* usando o barramento marcado como *result*. Para instruções do tipo *load-store*, o bloco *Incrementer* atualiza o registro de endereço antes que o *core* leia ou escreva o próximo valor para ou da memória. Este processo continua até que ocorra uma exceção ou uma interrupção que modifique o fluxo normal de execução.

Registros Internos

Conforme mencionado anteriormente, a maior parte dos registros internos do ARM são registros de propósito geral. Estes registros podem ser utilizados para guardar tanto dados quanto endereços. Geralmente, os registros são identificados pela letra `r` seguida do número do registro. A Figura 2.3 mostra o mapa dos registros internos do ARM que estão disponíveis em **modo de usuário**, que é o modo que normalmente é usado pelas aplicações. É possível ver na Figura a existência de um total de 16 registros de dados, 13 dos quais são usados para dados (r_0 a r_{12}). Os registros r_{13}, r_{14} e r_{15} têm propósitos específicos, funcionando, respectivamente, como *stack pointer* (sp), que aponta para o último elemento da pilha no momento atual; *link register* (lr), que guarda o endereço de retorno das funções e *program counter* (pc), que guarda o endereço da próxima instrução que deve ser buscada pelo processador.

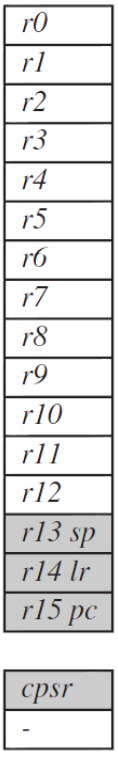

Figura 2.3: Mapa dos registros do ARM disponíveis em modo usuário

Apesar de terem funções bem definidas, os registros r_{13} e r_{14} podem ser usados também como registros de propósito geral. Isto pode ser especialmente útil, já que estes registros são salvos automaticamente quando ocorre uma mudança de modo no

processador. No entanto, não é uma boa prática usar o registro r_{13} como propósito geral quando existe um SO rodando no sistema, já que os SOs costumam assumir que há um endereço válido de pilha no registro r_{13}. Uma característica importante dos registros r_0 a r_{12} é que eles são ortogonais, ou seja, qualquer instrução pode ser aplicada da mesma forma em qualquer um desses registros. No entanto, algumas instruções operam de modo diferenciado nos registros r_{14} e r_{15}.

Além dos 16 registros de dados, o ARM possui ainda dois registros de status: o *cpsr* e *spsr*, que guardam os status do programa atual e salvo, respectivamene. O *Register File* possui todos os registros que estão disponíveis no momento. Quais registros estão disponíveis depende do modo em que o processador está operando.

Registro de Status do Programa Atual

O Registro de Status do Programa Atual(*Current Program Status Register*) (CPSR) é um registro de 32 bits utilizado pelo ARM para monitorar e controlar operações internas. Este registro é dividido em 4 blocos, com 8 bits cada, segundo mostra a figura 2.4. As partes sombreadas na figura são reservadas para uso futuro. As quatro regiões nas quais o registro está dividido são: flags, status, extensão e controle. O campo de controle contém o modo do processador, estado do mesmo e os bits de máscara de interrupção. O campo de flags contém os chamados flags de condições.

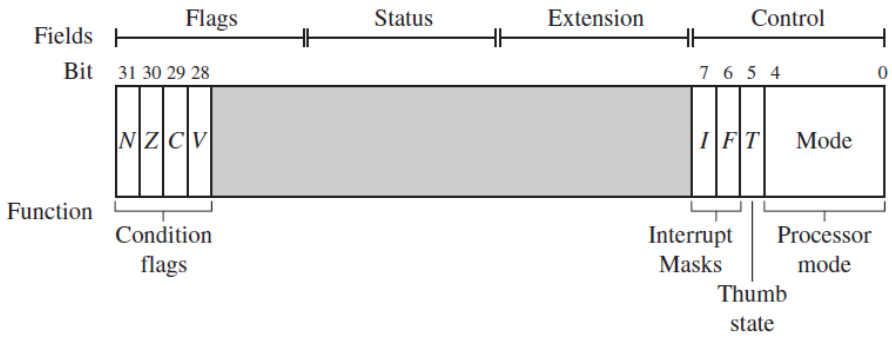

Figura 2.4: Descrição dos bits do registro CPSR

Modos do Processador O modo do processador determina quais registros estão ativos e os direitos de acesso ao CPSR. Modos privilegiado possuem acesso total, em modos de escrita e leitura, ao CPSR. Por outro lado, modos não privilegiados

podem apenas ler o campo de controle do CPSR, mas possuem acesso de leitura e escrita aos flags de condições.

No total, o ARM possui sete modos do processador, sendo seis privilegiados: *abort, fast interrupt request, interrupt request, supervisor, system* e *undefined*, e um modo não privilegiado: o modo de usuário (*user mode*). O modo *abort* é usado quando ocorre uma falha de acesso à memória. O *fast interrupt mode* e o *interrupt mode* são os dois níveis de interrupção disponíveis no ARM. O modo *supervisor* é o modo no qual o ARM começa sua execução, e é também o modo no qual o kernel trabalha. O modo *system* é uma versão especial do modo de usuário que possui permissões de leitura e escrita no registro CPSR. O modo *undefined* é usado quando o processador encontra uma instrução desconhecida ou não suportada. Finalmente, o modo de usuário é usado por programas e aplicações de usuário.

Banked Registers A Figura 2.5 mostra os 37 registros, dos quais 20 estão escondidos dos programas na maior parte do tempo, ficando disponíveis apenas em alguns modos. Estes registros, identificados pelo sombreado na Figura 2.5, são chamados de *Banked Registers*. O modo no qual os registros estão disponíveis é mostrado na Figura 2.5 através de um prefixo que representa o modo. Assim, no modo *abort*, por exemplo, estão disponíveis os registros **r13_abt**, **r14_abt** e **spsr_abt**.

Programas rodando em modo privilegiado podem mudar o modo atual de execução escrevendo diretamente os bits de modo do CPSR. Cada modo do processador, exceto o modo *system*, possui seu próprio conjunto de *banked registers*. Cada um deles está relacionado a um registro em modo de usuário, contudo uma alteração feita em um *banked register* em modo privilegiado não altera o valor do registro em modo de usuário, já que é feita uma troca de contexto quando o processador transita entre modo de usuário e modo privilegiado.

Estado e Conjuntos de Instruções O estado do processador indica qual conjunto de instruções está sendo usado no momento. Existem três conjuntos de instruções possíveis: ARM, Thumb e Jazelle. O conjunto de instruções ARM só está ativo quando o processador está no estado ARM. Da mesma forma ocorre para o conjunto de instruções Thumb, que só está ativo no estado Thumb. Uma vez no estado Thumb, o processador só pode executar instruções Thumb de 16 bits, não sendo possível misturar instruções Thumb, ARM e Jazelle sequencialmente.

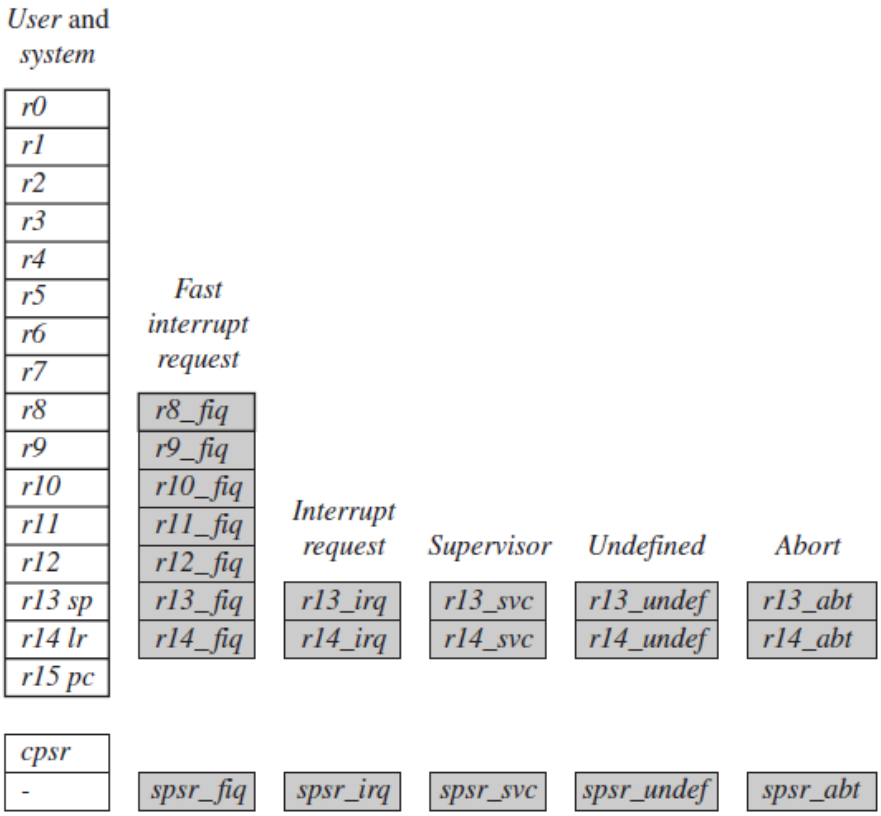

Figura 2.5: Descrição dos bits do registro CPSR

Os bits T e J definem o estado atual do processador. Quando ambos estão em zero, que é o estado inicial do processador quando o mesmo é energizado, o conjunto de instruções ARM é selecionado. Quando um dos bits está setado, o conjunto de instruções correspondente é selecionado. Para fazer a transição entre os estados do processador, é usada uma instrução específica.

Máscaras de Interrupções Os bits de máscara de interrupções são utilizados para que o processador deixe de atender determinadas interrupções. No ARM, duas interrupções estão disponíveis: *interrupt request* (IRQ) e *fast interrupt request* (FIQ). Para mascarar estas interrupções, são usados os bits 6 e 7 (I e F), desabilitando respectivamente a IRQ e a FIQ.

Flags Condicionais Os Flags Condicionais são atualizados quando é executada alguma instrução que possui o sufixo S. Por exemplo, se a instrução `SUBS` é executada e tem um zero como resultado, o flag Z é setado. Estes flags são avaliados para indicar rapidamente que determinadas condições foram atingidas, permitindo saltos

condicionais no programa. A Tabela 2.1 mostra os Flags Condicionais mais usados.

Tabela 2.1: Principais Flags Condicionais no processador ARM

Flag	Nome da Flag	É setada quando...
Q	Saturation	o resultado gera uma saturação e/ou um overflow
V	Overflow	o resultado da operação causa um overflow com sinal
C	Carry	o resultado possui um carry sem sinal
Z	Zero	o resultado é zero, também usado para indicar igualdade
N	Negative	o bit 31 do resultado é 1

Execução Condicional A execução condicional controla se o core vai ou não executar determinada instrução dependendo do estado atual dos Flags Condicionais. Antes da execução, o processador compara o atributo condicional associado à instrução com a condição presente nos flags, caso a condição esteja sendo atendida, a instrução é executada. Caso contrário, a mesma é descartada.

Os atributos condicionais são pós-fixados ao mnemônico das instruções. A Tabela 2.2 mostra a lista dos atributos condicionais suportados, seu nome e o(s) flag(s) avaliado(s) na ocorrência do atributo. Caso nenhum atributo seja passado junto com a instrução, é assumido o atributo padrão AL (executar sempre).

2.3.2 Processadores ARM em Sistemas Embarcados

A filosofia de *design* RISC

O core do ARM utiliza uma arquitetura do tipo Computação com Conjunto de Instruções Reduzido (*Reduced Instruction Set Computing*) (RISC). Esta filosofia de *design* consiste em fornecer instruções do processador que executam tarefas bem simples, de modo que cada instrução é executada em apenas um ciclo de clock. Assim, a complexidade do hardware é reduzida, sendo transferida para o software. A justificativa desta abordagem é que é mais fácil prover flexibilidade em software que em hardware, ficando sobre o compilador a responsabilidade de quebrar operações complexas (como uma divisão, por exemplo) em diversas operações mais simples. Como resultado disto, a grande demanda em processadores RISC está sobre o compilador. As arquiteturas Computação com Conjunto de Instruções Complexo (*Complex Instruction Set Computing*) (CISC), por outro lado, contam com instruções complexas implementadas em hardware, o que garante

Tabela 2.2: Lista dos atributos condicionais suportados pelo ARM

Mnemônico	Nome	Flags Condicionais testados
EQ	equal	Z
NE	not equal	z
CS HS	carry set/unsigned higher or same	C
CC LO	carry clear/unsigned lower	c
MI	minus/negative	N
PL	plus/positive or zero	n
VS	overflow	V
VC	no overflow	v
HI	unsigned higher	zC
LS	unsigned lower or same	Z or c
GE	signed greater than or equal	NV or nv
LT	signed less than	Nv or nV
GT	signed greater than	NzV or nzv
LE	signed less than or equal	Z or Nv or nV
AL	always (não condicional)	ignorado

uma performance maior, mas por outro lado diminui a flexibilidade e aumenta complexidade e a área em silício do processador, o que resulta em um custo maior. A figura 2.6 mostra a relação de complexidade nas duas abordagens.

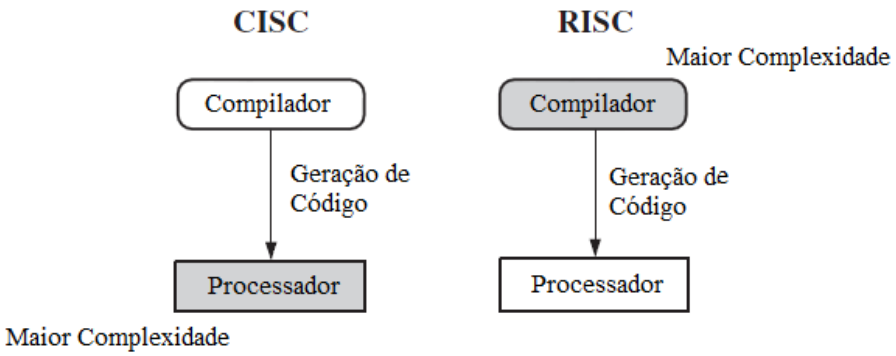

Figura 2.6: Comparação da localização da complexidade em sistemas RISC e CISC

No desenvolvimento de processadores RISC, são levadas em consideração as seguintes regras de *design*:

Instruções Conforme mencionado anteriormente, processadores RISC possuem poucas instruções que realizam tarefas simples. Cada instrução possue um tempo de execução fixo, permitindo que uma estrutura de *pipeline* inicie a busca das próximas instruções enquanto a atual está sendo executada. Em processadores CISC, por outro lado, algumas instruções têm tempo de execução variável e podem levar vários ciclos para terminar de executar;

Pipelines O processamento das instruções é quebrado em tarefas menores, que podem ser executadas em paralelo pelo *pipeline*. Assim, as fases de buscar, decodificar e executar são realizadas todas simultaneamente, idealmente em um só ciclo de clock, garantindo uma alta velocidade de execução. Não há a necessidade de utilizar um microcódigo para executar as instruções, como ocorre em processadores CISC;

Registros Processadores RISC possuem vários registros de propósito geral, que podem ser usados tanto para armazenar dados quanto endereços. Desta forma, os registros servem como uma pequena porção de memória local para ser utilizada durante o processamento das instruções. Isto não ocorre nas máquinas CISC, que possuem registros dedicados para propósitos específicos;

Arquitetura *load-store* Com esta arquitetura, o processador tem instruções do tipo *load and store*, ou seja, carregar e guardar, que fazem a transferência de dados entre a memória externa e os registros internos do processador. Como acessos à memória externa são mais custosos, separar acessos à memória do processamento dos dados permite uma maior eficiência na execução, fazendo com que várias instruções seguidas operem sobre os dados armazenados nos registros internos, sem a necessidade de fazer um acesso à memória externa. Em processadores CISC, por outro lado, instruções de processamento de dados podem operar diretamente na memória externa.

As regras de *design* mostradas acima permitem que processadores RISC sejam mais simples, o que permite que o core opere com uma frequência de clock mais alta que os CISC, que possuem uma complexidade maior. A tendência atual, no entanto, não é optar por nenhuma das abordagens especificamente, fazendo um balanceamento das características de cada abordagem dependendo do objetivo do projeto, se é uma simplicidade maior e redução dos custos ou uma complexidade maior e maior robustez.

A filosofia de *design* ARM

O *design* do processador ARM é norteado por uma série de parâmetros que devem ser levados em consideração. Inicialmente, a maior parte dos sistemas embarcados devem ser portáteis, por isso sua alimentação costuma vir de uma bateria. Assim, o ARM deve ser tão pequeno quanto possível, de modo a consumir o mínimo de energia e proporcionar uma maior vida útil para a bateria. Outro requisito importante é uma alta densidade de código, ou seja, o código gerado não deve ocupar muito espaço na memória. Isto é importante porque os dispositivos embarcados normalmente possuem restrições de tamanho e de custo, o que limita a quantidade de memória disponível. Adicionalmente, é desejável que a área do *die* seja a menor possível, já que o custo de manufatura de um chip é determinado pela sua área. Quanto menor a área do *core*, mais espaço fica disponível para os periféricos, aumentando a integração e reduzindo o custo total do projeto.

Normalmente, os processadores ARM contam com ferramentas de debug integradas em hardware, o que permite que o desenvolvedor tenha total controle sobre o código que está executando no processador, podendo ler os valores dos registros internos do processador e de regiões de memória, enquanto o processador está executando código. Desta forma, é possível resolver os problemas que surgem durante o desenvolvimento com muita rapidez, o que resulta numa redução nos custos de produção.

Devido às restrições impostas pelo uso em sistemas embarcados, o ARM não utiliza uma arquitetura RISC pura. Em vez disso, ele utiliza muitos conceitos RISC, mas com algumas características adicionais que permitem um bom desempenho em sistemas embarcados. A seguir estão listadas algumas características que diferem o ARM de um sistema RISC tradicional.

Tempo de execução variável para algumas instruções Nem todas as instruções do ARM executam em um único ciclo de *clock*. As instruções do tipo *load-store*, por exemplo, têm tempo de execução variável, dependendo do número de registros transferidos. Pode ocorrer variação também caso a transferência ocorra em endereços de memória consecutivos, já que neste caso a transferência é mais rápida em comparação com acessos a endereços aleatórios.

Suporte a instruções *Thumb 16-bit* Conforme detalhado anteriormente, o

ARM possui mais de um conjunto de instruções. Além do seu conjunto nativo com instruções de 32 bits, ele suporta o conjunto de instruções *Thumb*, de 16 bits. Estas instruções de 16 bits aumentam a densidade do código em cerca de 30% em comparação com as instruções de tamanho fixo de 32 bits (SLOSS; SYMES; WRIGHT, 2004).

Execução condicional Algumas instruções só são executadas quando alguma condição específica é atendida. Isto aumenta a performance e a densidade de código. Contudo, gera instruções mais complexas, o que é contrário ao que prega a filosofia RISC.

Instruções Aprimoradas O conjunto de instruções do ARM suporta algumas instruções complexas, usadas para Processamento Digital de Sinais (PDS), como multiplicações rápidas de 16 por 16 bits. Estas instruções eliminam a necessidade de utilizar um DSP externo junto com o ARM.

Devido a esta mistura das características dos sistemas RISC com alguns elementos de sistemas CISC, os processadores ARM possuem um bom equilíbrio entre flexibilidade, custo e desempenho, sendo bastante adequados para o uso em sistemas embarcados.

2.3.3 Famílias do Processador ARM

As diferentes versões dos processadores ARM costumam ser divididas em famílias, de acordo com o core utilizado. Algumas famílias populares do ARM são o ARM7, ARM9, ARM10 e ARM11. Os números indicam diferentes versões do core, com números ascendentes representando um aumento de sofisticação e potência. É importante notar que os números pós-fixados nos nomes das famílias não representam exatamente a versão a qual o processador pertence, apesar de números diferentes indicarem versões diferentes. Por exemplo, o ARM7 também é conhecido como ARMv4T, ou a versão 4 do core do processador ARM (o T presente no nome indica o suporte ao conjunto de instruções *Thumb*, já apresentado neste trabalho). Já a arquitetura ARMv5E apareceu inicialmente no processador ARM9. Esta versão apresentava instruções de DSP aprimoradas (em inglês, *enhanced*), daí o E presente no nome. O ARM11, por sua vez, utiliza o core ARMv6. Esta versão do core inclui algumas funcionalidades no sistema de gerenciamento de memória e instruções do tipo *Single Instruction, Multiple Data* (SIMD), ou seja, uma única instrução pode

fazer operações sobre um bloco de dados, acelerando operações de movimentação de dados da memória.

Depois da introdução da família ARM11, foi decidido que era necessário desenvolver processadores focados na aplicação. Assim, o portifólio de processadores ARM passou a possuir desde representantes de baixo consumo, utilizados em aplicações sensíveis a custo, até processadores de alta performance, utilizados em aplicações que cobram um grande poder de processamento.

Assim, a partir do ARMv7, foram desenvolvidos três perfis de processadores ARM:

Perfil A: Desenvolvido para aplicações de alta performance. Estes processadores são desenvolvidos para executar os principais SOs, utilizando alto poder de processamento, sistema de memória virtual com suporte a Unidade de Gerenciamento de Memória (*Memory Management Unity*) (MMU). Produtos que utilizam esse perfil incluem telefones celulares *high-end* e outros dispositivos embarcados com alto poder de processamento.

Perfil R Desenvolvido especialmente para aplicações em tempo real, onde são necessários alto poder de processamento, baixa latência e alta confiabilidade. A capacidade de cumprir tarefas de tempo real, no entanto depende do SO que roda sobre o processador, não sendo suficiente o uso de um ARMv7-R.

Perfil M Processadores voltados para aplicações de baixo custo, onde os fatores principais são a eficiência, o baixo consumo de potência, baixa latência de interrupção e facilidade de uso. Devido a essas características, processadores com perfil M são muito utilizados em aparelhos celulares.

Os processadores da família Cortex foram os primeiros produzidos com a arquitetura ARMv7. Esta família inclui os Cortex-A8, utilizados nos smartphones *high-end*, o Cortex-R4, utilizado para aplicações de tempo real e o Cortex-M3, amplamente utilizado em sistemas embarcados devido ao seu baixo custo e baixo consumo de energia.

2.4 Sistemas de Home Care

Segundo (LEME, 2011), Home Care é definido como uma modalidade contínua de serviços na área de saúde, cujas atividades são dedicadas aos pacientes em um

ambiente extra-hospitalar. Ou seja, os cuidados são realizados fora do hospital, normalmente na casa do paciente.

A principal vantagem dos serviços de Home Care em relação aos tratamentos realizados dentro do hospital é a possibilidade de manter o paciente o tempo todo próximo da família. Diferente de um ambiente hospitalar, onde o paciente ficará a maior parte do tempo em contato com desconhecidos e possíveis infecções hospitalares, num sistema de Home Care ele tem a possibilidade de ficar instalado em sua própria casa e cercado dos familiares, o que promove uma melhor qualidade de vida para o paciente durante o tratamento. Outra grande vantagem desse sistema, conforme explicado na seção 1.1, é o alívio na superlotação de leitos hospitalares, já que cada paciente tratado em regime de Home Care é um leito hospitalar a menos a ser ocupado num hospital. Esta segunda vantagem é especialmente importante em países emergentes, como o Brasil, onde a superlotação dos hospitais é um problema constante.

Obviamente, o sistema tem também algumas desvantagens, como por exemplo o fato de que, em regime hospitalar, o paciente tem acesso imediato a médicos e a todo tipo de equipamento que possa ser necessário no caso de uma emergência. Num regime de Home Care, por outro lado, isso não ocorre, sendo necessário chamar uma ambulância para fazer o atendimento emergencial, o que pode deixar o paciente sem cuidados por alguns minutos, o que, por sua vez, pode fazer a diferença entre a sobrevivência ou não do paciente. Desta forma, é muito importante que seja feita uma análise da gravidade do estado do paciente, para que seja tomada a decisão de deixá-lo em regime hospitalar ou em regime de Home Care. A decisão depende também da vontade e disponibilidade da família, já que o paciente deve receber um acompanhamento enquanto está nesse regime.

Classicamente, para manter um paciente em regime de Home Care, era necessário contratar um profissional da saúde que ficasse o tempo todo acompanhando o paciente. Isso gera um grande custo adicional para a família. Hoje em dia, é possível utilizar equipamentos de telemedicina para fazer o acompanhamento do paciente remotamente. Com isso, um médico pode acompanhar o paciente 24h por dia sem a necessidade de se deslocar para a casa do paciente.

Segundo (D'ANGELO *et al.*, 2010), a primeira tentativa de desenvolver um sistema que coleta dados de ECG de um paciente e transferir os dados remotamente para um cardiologista ocorreu na década de 70. Hoje, há várias soluções no mercado,

utilizando diferentes metodologias, que podem ser classificadas de forma simplificada em duas abordagens.

A abordagem clássica consiste de um gravador de ECG, que fica na casa do paciente, um canal de comunicação (telefone ou GSM) e uma unidade receptora que fica com o cardiologista. Essa abordagem tem uma série de desvantagens. A primeira delas é que o gravador utilizado é desenvolvido para uso dentro de uma clínica ou hospital, por isso acaba tendo uma operação muito complicada para um usuário normal. Outra desvantagem é a falta de compatibilidade entre os aparelhos, já que a unidade receptora só funciona com determinado gravador. Além disso, somente essa unidade receptora pode ler os dados enviados, não sendo possível o acesso aos dados fora do hospital nem por outra pessoa que não seja o médico. Finalmente, a abordagem é muito sujeita a erros, já que o paciente fica responsável por enviar os dados, correndo o risco de esquecer de enviar; o canal utilizado é muito instável e sujeito a perda de pacotes e a análise dos dados é feita manualmente pelo médico.

A outra abordagem, mais moderna, consiste em um sistema de ECG com gerenciamento de dados centralizado. Esta abordagem é uma evolução da clássica e está mais alinhada com a tecnologia disponível atualmente. O gerenciamento de dados centralizado significa que os dados são enviados para um servidor através da internet. Assim, o sistema passa a ser dividido em duas partes: o cliente, que fica instalado localmente em um PC, e uma plataforma Web, que é acessada pelo cliente. Essa abordagem permite uma série de avanços, como disponibilizar as medidas automaticamente na tela, gravação do histórico do paciente no banco de dados e acesso aos dados a partir de qualquer computador. Além disso, a interface é construída de modo a ser mais familiar para o usuário, evitando as dificuldades presentes na abordagem clássica.

2.4.1 Fundamentos de Eletrocardiografia

Para desenvolver a funcionalidade de exibir leituras dos sinais vitais dos pacientes na tela, mencionada na seção anterior, é necessário um conhecimento básico acerca dos sinais que se deseja medir. Devido a este fato, esta seção descreverá de forma resumida as principais características de um sinal obtido através de eletrocardiograma, que será o tipo de sinal medido neste trabalho. Como não é objetivo do sistema desenvolvido a detecção de problemas cardíacos, ficando esta parte a cargo do médico, a descrição do sinal feita aqui será focada apenas no

básico, suficiente para compreender as principais características do sinal gerado por um eletrocardiograma.

Fisiologia do Coração

O coração é um órgão muscular que tem como função principal impulsionar o sangue através dos vasos sanguíneos, fazendo com que este circule por todo o organismo. A Figura 2.7, adaptada de (BONSOR, 1999), mostra o corte de um coração humano. É possível observar na figura que o coração é formado por quatro câmaras internas: dois átrios e dois ventrículos. Cada uma destas câmaras possui em sua saída uma válvula unidirecional, que faz com que o sangue possa fluir sempre na mesma direção.

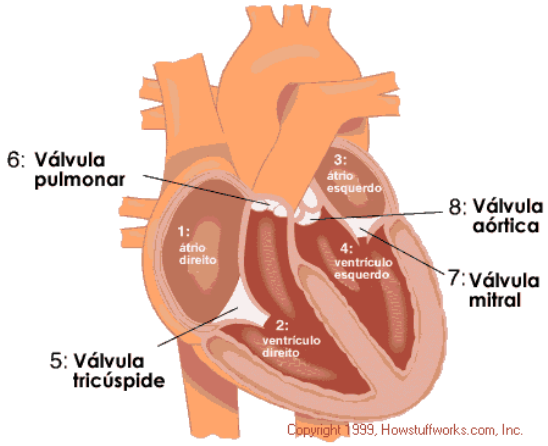

Figura 2.7: Visão detalhada de um coração humano, com suas cavidades e as válvulas que as interligam (BONSOR, 1999)

O funcionamento do coração se dá em dois movimentos: um movimento de contração, chamado **sístole** e um de relaxamento, chamado **diástole**. A sístole, por sua vez, se divide em dois momentos: um primeiro em que os dois átrios se contraem simultaneamente, expulsando o sangue para os ventrículos, e um segundo onde os ventrículos se contraem, expulsando o sangue para fora do coração. Estes movimentos de contração ocorrem com força suficiente para bombear o sangue por todo o corpo. Após a sístole, ocorre o movimento de relaxamento ou diástole, que permite que o sangue entre novamente no coração. Este conjunto de movimentos do coração se repete periodicamente e recebe o nome de **ciclo cardíaco**.

Todos esses movimentos do coração são controlados por impulsos elétricos que vêm do sistema nervoso. Durante um eletrocardiograma, sensores elétricos

são posicionados em vários pontos do corpo do paciente, no intuito de fazer a leitura desses sinais elétricos que comandam o coração. Assim, o exame de eletrocardiograma é, na verdade, uma medida dos impulsos elétricos que controlam o coração, e não dos batimentos do coração propriamente ditos.

O Eletrocardiograma

O Eletrocardiograma (ECG) é, conforme dito na seção anterior, o registro de todas as atividades elétricas que ocorrem no coração, obtido por meio de eletrodos conectados à superfície corporal do paciente. O ECG é uma ferramenta de diagnóstico não invasivo largamente utilizada para detectar problemas como arritmias e distúrbios em geral no coração.

A Figura 2.8, adaptada de (NETO, 2010), mostra uma onda típica de ECG. Como é possível ver na figura, a onda é dividida em várias componentes, que representam os diversos impulsos elétricos necessários para controlar os movimentos descritos na seção anterior.

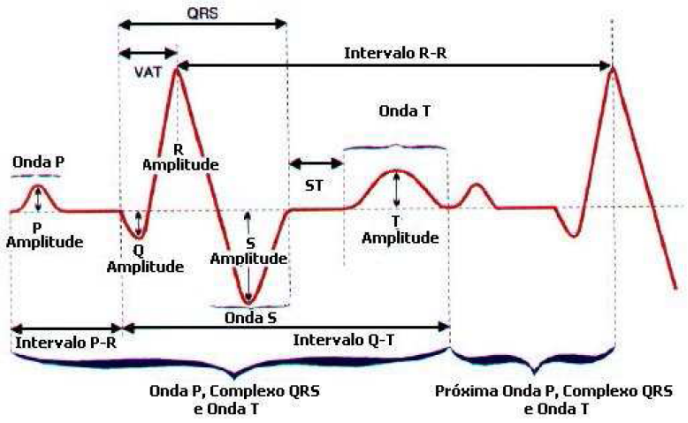

Figura 2.8: Típico sinal de ECG, com indicação das componentes da onda

Na onda mostrada na Figura 2.8, é possível ver, inicialmente, a onda P, que coincide com a propagação da atividade elétrica nos átrios e com o início de sua contração. O complexo QRS, formado pelas ondas Q, R e S, coincide com a propagação da atividade elétrica nos ventrículos e com o início de sua contração. Finalmente, a onda T coincide com o relaxamento das cavidades. Através da análise dessas componentes, o médico é capaz de detectar possíveis anomalias no comportamento do coração.

Uma medida de particular interesse que pode ser obtida facilmente através do ECG é a frequência cardíaca, que é simplesmente a quantidade de vezes que o coração realiza um ciclo cardíaco em um minuto. Para aferir o ciclo cardíaco, basta localizar qualquer uma das ondas que formam o ciclo e contar quantas vezes esse tipo de onda aparece em um minuto. A metodologia utilizada para medir a frequência cardíaca é detalhada na Seção 3.5.2.

Capítulo 3

Metodologia de Desenvolvimento

Este capítulo apresenta a metodologia de desenvolvimento utilizada para este projeto, bem como os detalhes da arquitetura proposta. Na Seção 3.1 é apresentada a arquitetura proposta para o trabalho, em forma de diagrama de blocos funcionais. A seção 3.2 mostra o ambiente de desenvolvimento utilizado, bem como as ferramentas de desenvolvimento de hardware e software.

A seção 3.4 mostra como foi feita a customização do SO para o projeto, enquanto a seção 3.3 mostra em detalhes o hardware utilizado. Finalmente, a seção 3.5 detalha o software que foi desenvolvido.

3.1 Arquitetura Proposta

A arquitetura proposta consiste basicamente em um sistema que tem como elemento de processamento um microcontrolador Freescale iMX53, que possui um core ARM Cortex-M3. O sistema todo pode ser visto na Figura 3.1.

O iMX53 recebe, via bluetooth, os sinais vitais do paciente, medidos por um sensor cujo desenvolvimento não faz parte do escopo deste trabalho. Quando os sinais são recebidos, o primeiro bloco, chamado **aquisição** trata de organizar e armazenar os dados em um banco de dados. Estes dados ficam disponíveis para consultas futuras, montando assim um histórico dos sinais vitais do paciente. Além de armazenados, os dados referentes aos sinais vitais do paciente são enviados para o segundo bloco, chamado **análise**. Neste bloco funcional, a medida do sinal é comparada com valores pré-definidos pelo médico. Caso algum sinal esteja fora da faixa determinada como segura pelo médico, um alerta é acionado, utilizando a

conectividade presente no sistema. Assim, o alerta pode ser mandado via Serviço de Mensagens Curtas (*Short Message Service*) (SMS), email ou mesmo uma ligação utilizando serviços de VoIP, para a família do paciente, para o médico ou para o serviço de emergência.

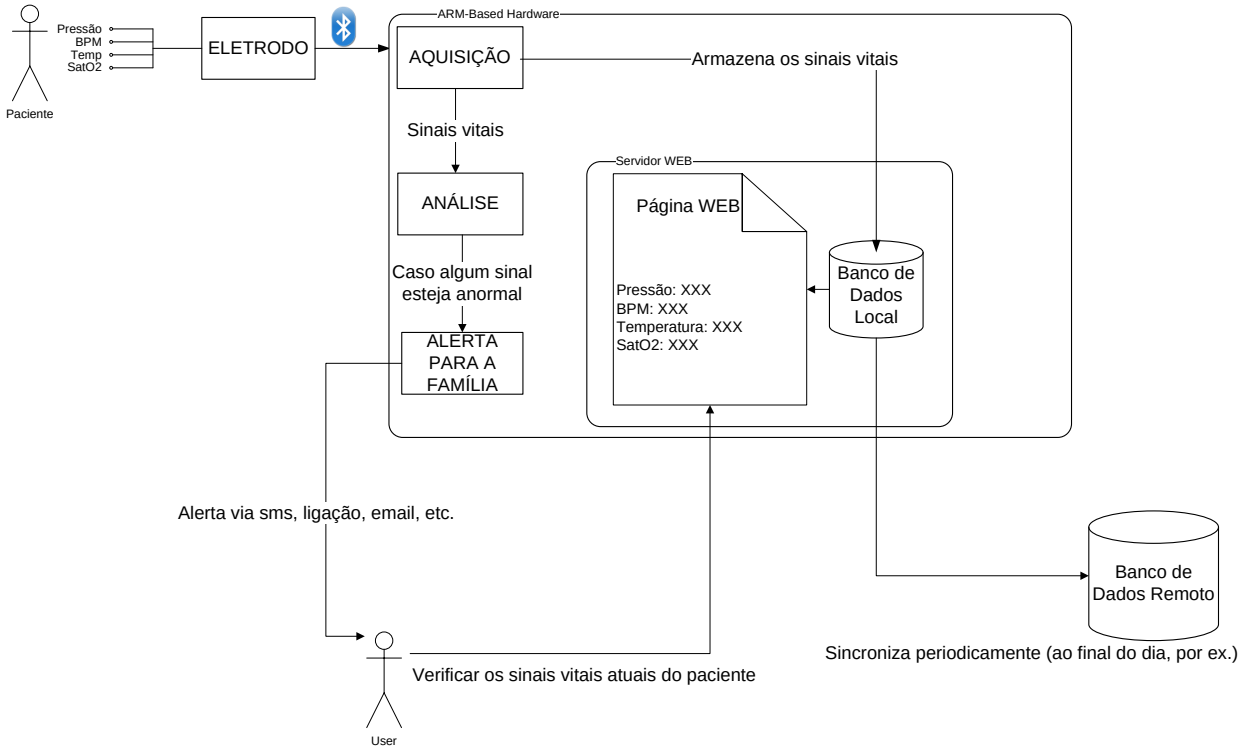

Figura 3.1: Arquitetura proposta para o sistema de Home Care

Enquanto faz a checagem descrita no parágrafo anterior, o sistema trata de formatar uma página Web, que é publicada através de um servidor Web que roda no próprio iMX53. A página Web é a principal forma de interação entre o sistema e o usuário, que pode ser um familiar do paciente, um médico ou alguém do plano de saúde. Nesta página, é exibido o estado atual dos sinais vitais, através de gráficos temporais. É também através desta página Web que o médico pode configurar os limiares aceitáveis para cada sinal vital do paciente. Assim, o sistema não é capaz de realizar um diagnóstico por si só. Em vez disso, o médico faz os exames pertinentes e determina através de metodologias tradicionais quais são os limiares aceitáveis para cada paciente. O que o sistema faz é verificar o tempo todo se os sinais vitais estão

dentro desse limiar determinado pelo médico.

Ainda na Figura 3.1, é possível ver que são usados dois bancos de dados: um local, que armazena um histórico das medidas dos sinais vitais feitas pelo dispositivo. O outro banco é externo ao sistema e seria, num cenário real, o banco de dados do plano de saúde, onde seria possível guardar um histórico dos sinais vitais do paciente, para consulta por parte do plano. A sincronia entre o banco local e o banco remoto é feita periódicamente, podendo ser realizada, por exemplo, ao final do dia. Além de servir para manter um histórico atualizado, o banco de dados remoto serve para fornecer uma maior confiabilidade ao sistema, servindo como cópia de segurança do conteúdo do banco de dados local. Assim, caso haja algum problema com o dispositivo ou o mesmo precise ser trocado, o custo para a troca é mínimo, já que tanto os dados históricos dos sinais vitais do paciente quanto as configurações realizadas pelo médico estão guardados no banco de dados remoto, sendo necessário apenas fazer a sincronização entre os bancos para voltar a ter todos os dados localmente.

É importante ressaltar que a arquitetura do projeto foi pensada para monitorar qualquer sinal vital de um paciente. Contudo, o desenvolvimento do software exige que sejam escolhidos os sinais vitais que serão monitorados. Assim, para este trabalho, foi escolhido trabalhar apenas com sinais de eletrocardiograma. Assim, todo o software e a página da Web foram desenvolvidos para dar suporte à monitoração deste tipo de sinal. A arquitetura, no entanto, permite a adição de outros tipos de sinal para monitoramento no futuro.

As próximas seções detalham o hardware e o software desenvolvidos para implementar a arquitetura discutida nesta seção. Antes, porém, é importante mencionar as ferramentas utilizadas para o projeto, tanto na parte de software quanto para o hardware.

3.2 Ferramentas Utilizadas

O sucesso de um projeto depende fortemente das ferramentas que são utilizadas. Sempre que possível, deve-se selecionar cuidadosamente quais ferramentas serão utilizadas em cada fase do projeto, evitando que escolhas erradas acabem impactando no tempo de execução do projeto. Esta seção detalha as ferramentas que foram utilizadas para este processo, justificando o uso das mesmas e mostrando o impacto de cada uma no resultado final do projeto. Sempre que possível, foi

dada a preferência ao uso de ferramentas livres e de código aberto, principalmente no desenvolvimento do software. Isto foi feito no intuito de diminuir o custo final do projeto e para permitir, na medida do possível, que qualquer pessoa que tenha interesse em extender este projeto no futuro tenha o mínimo possível de dificuldades para começar a trabalhar, partindo inicialmente de um ambiente de desenvolvimento de baixíssimo custo.

3.2.1 Software

Esta seção descreve as ferramentas utilizadas para o desenvolvimento da parte de software do projeto.

U-Boot

O programa **Das U-Boot**, ou simplesmente U-Boot (Universal Bootloader) é um firmware gerenciador de boot. Isto significa que ele é o primeiro programa executado quando o sistema é inicializado, sendo o responsável por selecionar o SO que será executado e configurá-lo de acordo com as configurações do usuário, passando os parâmetros de inicialização adequados. Na prática, o gerenciador de boot serve como uma interface através da qual o usuário pode fazer algumas configurações iniciais no sistema, deixando-o pronto para rodar o SO sem problemas.

O U-Boot é distribuído sob a licença GPL (FREE SOFTWARE FUNDATION, 2007), o que significa que seu código fonte está disponível e pode ser modificado à vontade, o que dá uma grande flexibilidade ao mesmo, permitindo que seja utilizado em uma grande variedade de plataformas diferentes. De fato, o U-Boot hoje dá suporte a uma grande variedade de plataformas, incluindo vários exemplos de plataforma de arquiteturas ARM, AVR32, Blackfin, Microblaze, MIPS, Powerpc, x86, dentre outras. A lista completa de plataformas suportadas pode ser vista explorando o código fonte, disponível em (U-BOOT, 2011). O foco principal do desenvolvimento do U-Boot é o suporte ao SO Linux e, como resultado, o código como um todo é bastante otimizado para esse SO.

A escolha do U-Boot como gerenciador de boot para este projeto foi motivada pelos dois fatores discutidos acima: o fato de ser regido pela licença GPL, permitindo que fossem feitas modificações para adaptar o firmware à plataforma utilizada, caso fosse necessário; e a grande integração com o Linux, que já evita o esforço de integração entre bootloader e Sistema Operacional. Adicionalmente, conforme

será mostrado na seção seguinte, o U-Boot é suportado nativamente pelo Linux Target Image Builder (LTIB), o que também facilita o processo de desenvolvimento. Finalmente, um último fator a favor do U-Boot é que, diferente de outros *bootloaders* como o RedBoot, o U-Boot dá suporte a uma grande variedade de sistemas de arquivos, incluindo o sistema FAT, presente em mídias de armazenamento como pendrives e sd cards e sistema ext2, presente em sistemas Linux, possibilitando o uso de técnicas o que permitem que a imagem do SO fique armazenada em um local remoto, sendo carregada via rede para o sistema alvo, eliminando a necessidade de realizar gravações na memória flash do microcontrolador.

Para este projeto, durante o desenvolvimento, o U-Boot ficava armazenado em um sdcard, enquanto o kernel e o sistema de arquivos linux ficavam em uma máquina remota de desenvolvimento, sendo carregados ambos via rede. Na versão final, tanto o kernel quanto o sistema de arquivos foram transferidos para o sdcard, permitindo que o sistema funcione de modo independente. Esta flexibilidade do U-Boot permitiu uma otimização do tempo de projeto, já que dependendo do bootloader utilizado, talvez fosse necessário fazer uma nova gravação no sdcard em cada alteração no sistema de arquivos, por exemplo.

LTIB

O Linux Target Image Builder (LTIB) (LTIB, 2011) é uma ferramenta que automatiza o processo de compilação, personalização e instalação de uma imagem do SO Linux. Utilizando o LTIB, é possível gerar uma imagem do Linux com todos os pacotes desejados já devidamente adicionados, bem como todas as alterações necessárias no kernel, como *device drivers* e seleção da plataforma de destino.

Além de configurar o Linux, conforme adiantado na seção anterior, o LTIB permite, através da seleção de algumas opções por meio de uma interface gráfica, que seja gerada uma imagem personalizada do U-Boot, também pronta para a plataforma de destino. Apesar de não ser estritamente necessário para o desenvolvimento de projetos com Linux embarcado, já que todas as operações que são feitas pelo LTIB também podem ser feitas manualmente editando os arquivos de configuração do Linux e do U-Boot, o LTIB facilita bastante essas operações, pois fornece uma interface gráfica na qual é possível ter uma visão geral de quais opções estão selecionadas no momento.

Outra vantagem do uso do LTIB é que ele já incorpora à imagem do Linux o

Board Support Package (BSP) correspondente à plataforma de destino, fazendo isso de maneira praticamente transparente para o usuário. Assim, uma vez configurado o ambiente, fica muito simples realizar mudanças e personalizações, tanto no kernel quanto no sistema de arquivos, no bootloader ou no próprio BSP.

Assim como o U-Boot, o LTIB também é distribuído sob licença GPL, incorporando todas as vantagens das ferramentas de código aberto.

Eclipse

O Eclipse (ECLIPSE, 2011) é uma IDE de código aberto criada inicialmente para desenvolvimento em linguagem de programação Java. No entanto, devido à sua natureza de código aberto, rapidamente foram criadas extensões para várias linguagens, sendo bastante usada para desenvolvimento em Ada, C, C++, COBOL, Perl, PHP, Python, Ruby, Scala, dentre outras linguagens.

Para este projeto, o Eclipse foi escolhido por ter uma vasta gama de opções para desenvolvimento em linguagem C, principal linguagem utilizada no kernel Linux, por ser código aberto e também pela familiaridade anterior com a IDE. O uso de uma IDE é fundamentalmente importante em um projeto que envolve alterações no kernel do Linux, já que o kernel possui seu código distribuído entre muitos arquivos, vários deles específicos para cada arquitetura ou plataforma. Desta forma, sem uma boa IDE, é possível que o desenvolvedor encontre problemas para manter seu código organizado e para depurar as funções quando algo dá errado.

Apache

Apache é um servidor HTTP de código aberto e é o servidor mais utilizado para páginas Web. Em maio de 2010, o Apache foi usado para disponibilizar 54,68% dos sites em geral e 66% dos sites mais acessados (NETCRAFT, 2010). O Apache foi escolhido pela sua popularidade, por ser livre e por sua compatibilidade com o linux. Além disso, o uso de um servidor HTTP pronto, em vez de desenvolver um proprietário, facilita bastante o projeto, diminuindo o tempo total de projeto.

MySQL

MySQL é um sistema de gerenciamento de banco de dados amplamente utilizado em diversas aplicações no mundo inteiro. O sistema funciona basicamente através da troca de mensagens (chamadas *queries*) utilizando strings ASCII. As mensagens

possuem uma sintaxe própria, mas são geralmente simples de implementar na maioria das linguagens de programação. Neste trabalho, o MySQL foi utilizado para guardar a base de pacientes cadastrados, os valores aceitáveis para os sinais vitais bem como qualquer informação que precisasse de um armazenamento a longo prazo.

PHP

PHP é uma linguagem interpretada utilizada normalmente em servidores para gerar conteúdo dinâmico na Web. Basicamente, o PHP é uma linguagem de programação que possui facilidades para formatar a saída dos scripts como páginas Web ou elementos html que possam ser facilmente formatados por um navegador Web.

Para este trabalho, o PHP foi utilizado para realizar diversas funções no servidor, como alteração de dados de pacientes, definição dos limiares aceitáveis para os sinais vitais e geração do gráfico em tempo real.

É interessante notar que o Apache, o MySQL e o PHP juntos rodando em um servidor Linux são uma configuração bastante comum, usada por muitos sites na Web. Normalmente, esta combinação recebe o nome de LAMP (Linux Apache MySQL PHP). Assim, ao utilizar este conjunto de ferramentas, pode-se usufruir da grande disponibilidade de suporte disponível na rede.

3.2.2 Hardware

A plataforma de Hardware utilizada foi a Freescale i.MX53 Quick Start Board (FREESCALE, 2011), que conta com um processador i.MX53, além de uma série de interfaces já montadas, o que facilita bastante o desenvolvimento. O hardware final do projeto, no caso de um produto comercial, seria uma versão bastante reduzida da Quick Start Board, já que a maior parte das conexões não é utilizada para este projeto. Contudo, esta plataforma de hardware foi escolhida para o protótipo para eliminar a necessidade de desenvolver uma placa proprietária. Além disso, a Quick Start Board já fornece algumas ferramentas como um BSP padrão, que facilita a personalização do kernel do Linux, além do suporte que pode ser obtido diretamente da Freescale.

3.3 Descrição do Hardware

Conforme mencionado na seção 3.2.2, o hardware utilizado foi um kit de desenvolvimento Freescale i.MX53 Quick Start Board, o que eliminou a necessidade de desenvolver o hardware. Adicionalmente, foram utilizados módulos comerciais para as interfaces de bluetooth e wifi, ambos conectando-se à Quick Start Board através de conexões USB. A intenção inicial do projeto era de desenvolver placas de expansão para estas duas interfaces, contudo, por restrições de tempo do projeto, optou-se por utilizar módulos prontos, o que reduziu o tempo total de desenvolvimento, mesmo que isso tenha levado a um aumento no custo final do protótipo. Em versões posteriores, a ideia é desenvolver todo o hardware, diminuindo o custo de produção por unidade. No entanto, o projeto desenvolvido permitiu validar o conceito da solução.

3.4 Customização do Sistema Operacional

O SO utilizado no projeto foi desenvolvido a partir do BSP fornecido pela Freescale, fabricante do processador e do kit de desenvolvimento utilizados. O BSP inclui uma versão do kernel do Linux já portada para o processador, e devidamente customizado para operar com os periféricos presentes na placa. Desta forma, o esforço necessário para realizar o porte e customização do kernel diminui consideravelmente. O BSP conta ainda com o U-Boot devidamente portado para o processador e uma série de pacotes que podem ser adicionados ao Linux, aumentando as funcionalidades do SO.

Partindo do BSP fornecido, foram feitas inicialmente customizações no U-Boot e no kernel, fazendo com que ele operasse corretamente no ambiente de desenvolvimento. O ambiente é, basicamente, o mostrado na Figura 3.2, consistindo de um laptop conectado na mesma rede que a Quick Start Board. O U-Boot fica gravado em um cartão de memória conectado à placa, enquanto a imagem do kernel e o sistema de arquivos ficam no laptop. Assim, ao ser energizado, o sistema inicialmente copia o U-Boot do cartão de memória para a memória RAM e começa a executá-lo. O U-Boot está configurado para buscar o kernel do Linux no laptop, via rede utilizando o Protocolo de Transferência de Arquivos Trivial (*Trivial File Transfer Protocol*) (TFTP). O kernel, depois de carregado, também está configurado para receber o sistema de arquivos via rede, utilizando o Sistema de Arquivos em

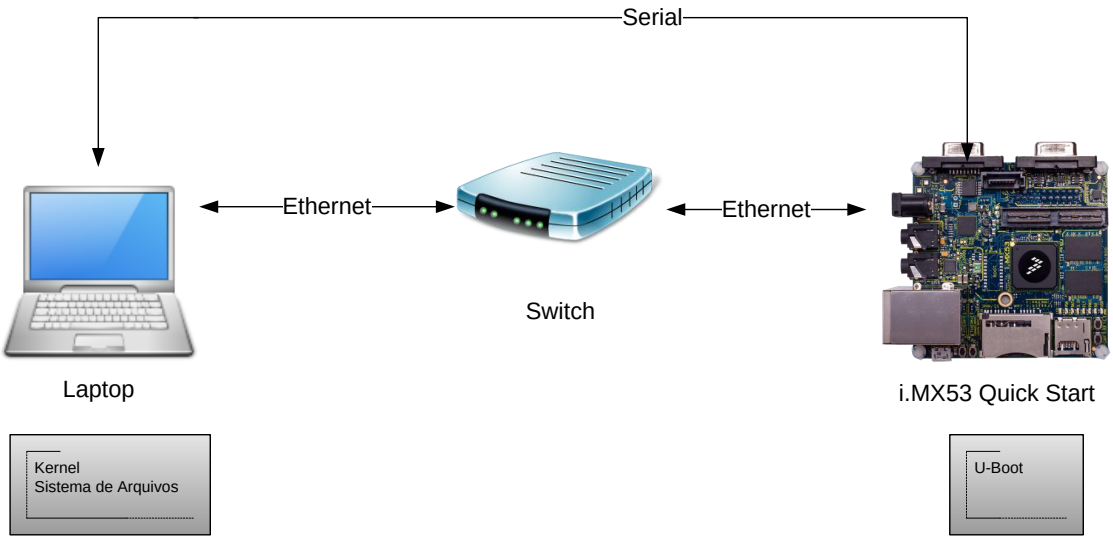

Figura 3.2: Ambiente de desenvolvimento do SO

Rede (*Network File System*) (NFS). O terminal do Linux fica configurado para funcionar via porta serial, também conectada ao laptop.

A principal vantagem desse ambiente é a facilidade para fazer alterações no sistema de arquivos, mesmo enquanto programas são utilizados. Como é preciso modificar diversos arquivos, instalar programas e ajustar configurações do sistema muitas vezes ao longo do desenvolvimento, não é interessante ter que gravar uma nova imagem diretamente na memória da placa alvo, como muitas vezes precisa ser feito quando se está utilizando um *firmware*. Esta flexibilidade fornecida pelo NFS é mais uma vantagem da utilização de um SO em vez de um *firmware* em um sistema embarcado.

Na versão final, tanto o kernel quanto o sistema de arquivos foram gravados diretamente no cartão de memória conectado à placa, deixando o funcionamento do sistema completamente independente do laptop.

Após a configuração inicial do ambiente, foi necessário adicionar uma série de pacotes ao Linux de modo a fornecer as funcionalidades desejadas, fazendo um papel semelhante ao das empresas que desenvolvem as distribuições Linux, como Ubuntu, Debian, Red Hat, dentre outras. A principal vantagem de partir do kernel puro e adicionar os pacotes necessários ao invés de utilizar uma distribuição pronta é o tamanho final da imagem que acaba ficando menor, o que, além de ocupar menos espaço na memória, oferece um melhor desempenho, já que não há serviços

desnecessários rodando no sistema.

3.5 Descrição do Software

O software desenvolvido para este projeto foi dividido em quatro blocos funcionais, conforme mostrado na Figura 3.1. Cada bloco corresponte a uma rotina que é executada em loop durante o funcionamento do sistema. Os blocos interagem entre si indiretamente, através da escrita e leitura de dados no banco de dados. Ao longo desta seção, cada bloco funcional será detalhado.

3.5.1 Aquisição de Dados

O primeiro bloco funcional realiza a aquisição dos dados do paciente. A aquisição é feita via bluetooth, de acordo com o Perfil para Dispositivos de Saúde (*Health Device Profile*) (HDP). O perfil HDP (LATUSKE, 2009) foi lançado em 2008, mas somente em 2011, com o lançamento da versão 2.6.38 do kernel, o mesmo foi incorporado ao Linux. O perfil foi criado para suprir a demanda por uma padronização do protocolo de comunicação entre dispositivos médicos bluetooth, já que, antes de haver esta padronização, cada fabricante desenvolvia seu próprio protocolo, acoplando o *design* de sistemas de monitoramento ao sensor que pode ser utilizado. A vantagem do uso do novo perfil é que é possível ter certeza que o sistema será compatível com qualquer sensor que seja desenvolvido pensando na compatibilidade com o HDP.

Na Figura 3.3, é apresentado o fluxograma do software de aquisição de dados. O programa inicia sua execução fazendo uma busca por todos os sensores compatíveis com HDP presentes no raio de alcance do bluetooth. Quando acha algum sensor, uma nova *thread* é criada para tratar da aquisição dos dados desse sensor, enquanto a rotina principal do programa continua procurando periodicamente por novos sensores.

A *thread* responsável pela aquisição dos dados inicia sua execução verificando se o sensor localizado já está cadastrado no sistema. Caso seja um novo sensor, é feito um cadastro do mesmo no banco de dados. A *thread* inicia então a aquisição de dados propriamente dita, adquirindo amostras numa taxa determinada pela capacidade do sensor. Cada nova amostra colhida é armazenada na tabela correspondente no banco de dados. Quando o sensor para de responder, a *thread* é finalizada.

Este bloco funcional garante que, a cada momento, haverá, no banco de

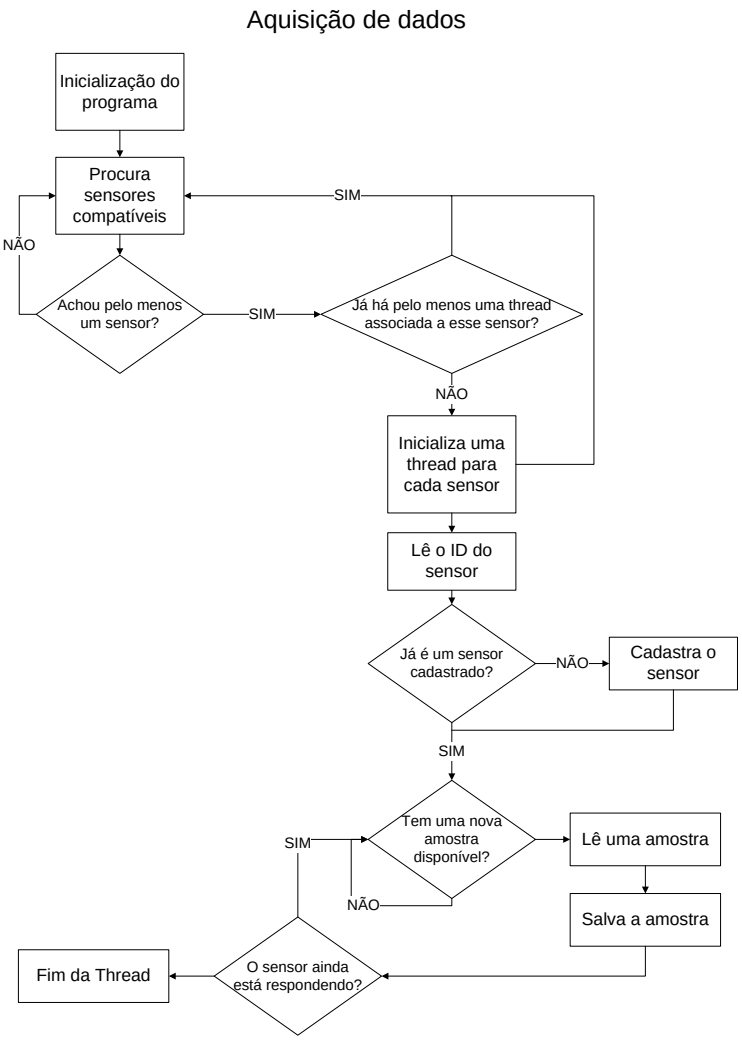

Figura 3.3: Fluxograma do software que realiza a aquisição dos dados

dados, informações atualizadas de todos os sensores presentes dentro do alcance do dispositivo, e que cada novo sensor que entra no raio de alcance será cadastrado no banco de dados. No entanto, este bloco não faz a associação entre cada sensor e um paciente. Cabe ao usuário (médico ou familiar) fazer, através da página Web, a associação entre sensores e pacientes. Por padrão, é assumido que todos os sensores presentes pertencem ao mesmo paciente, mas é possível, no momento do cadastro de um paciente, explicitar quais sensores estão ligados a este paciente.

3.5.2 Análise dos Dados

O segundo bloco funcional trata as amostras obtidas pelo primeiro, gerando dados úteis para o usuário. Conforme dito na seção 3.1, o software desenvolvido para

este trabalho é focado na aquisição de sinais de batimentos cardíacos. Desta forma, este bloco funcional lê amostras referentes à tensão medida pelo eletrocardiograma e gera como saída a frequência cardíaca atual do paciente.

Observando novamente a Figura 2.8, podemos ver que o complexo QRS tem uma peculiaridade que pode ser utilizada na medição da frequência cardíaca: é nessa parte do sinal que ocorre a variação mais rápida da tensão. Devido a esse fenômeno, a derivada será usada para localizar os complexos QRS no sinal, e a distância média entre dois complexos adjacentes será a frequência cardíaca.

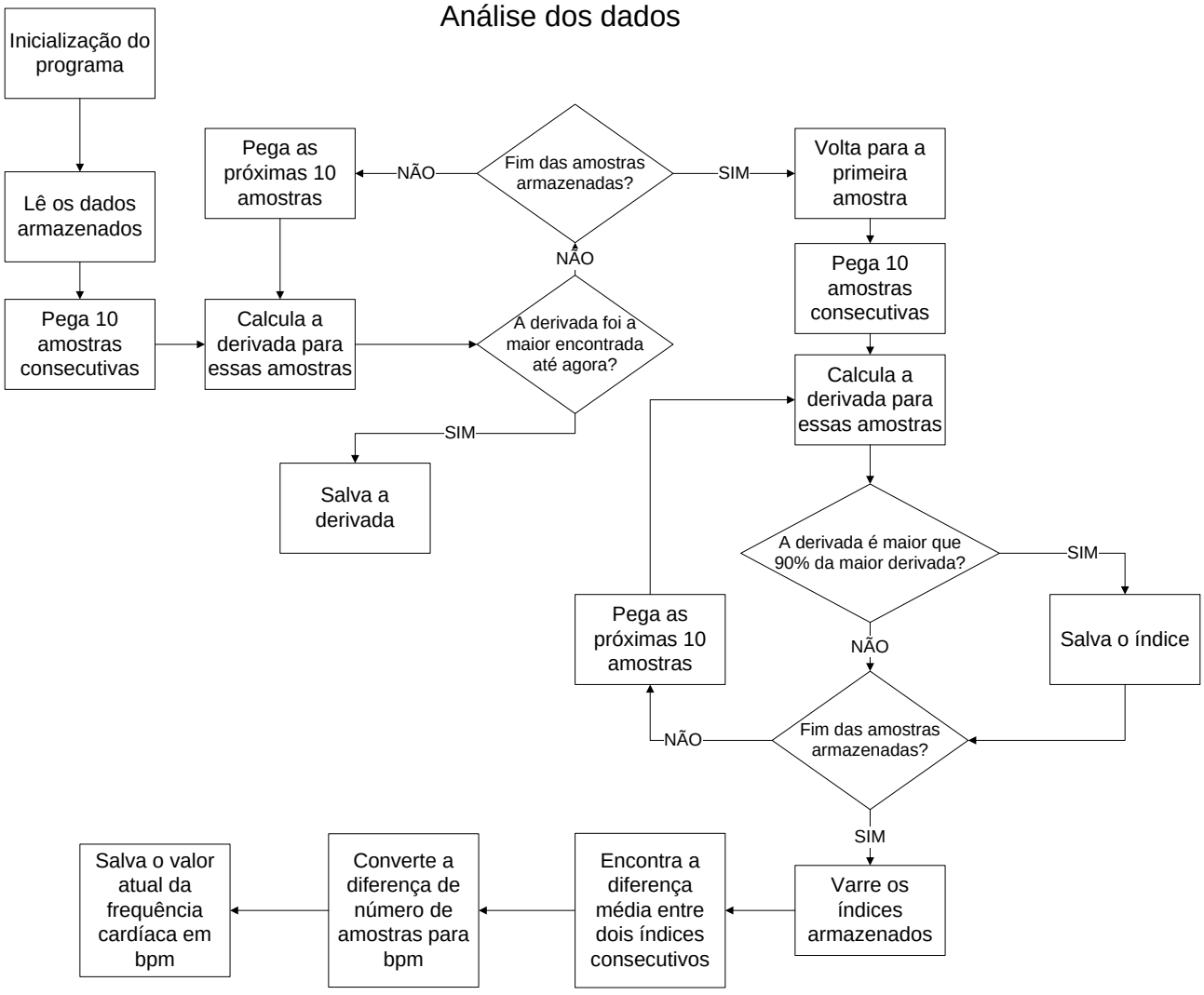

Figura 3.4: Fluxograma do software que realiza a análise dos sinais vitais

A Figura 3.4 mostra o fluxograma deste programa. A execução do programa começa com a leitura dos dados gerados pelo bloco Aquisição. As amostras são lidas

em blocos de 10, tamanho este que foi definido empiricamente. O primeiro laço que se observa no fluxograma tem como objetivo encontrar a região do sinal onde ocorre a maior derivada. Uma vez localizada a maior derivada, o sinal é varrido novamente em busca de todos os pontos onde aparece um valor de derivada igual a pelo menos 90% da maior derivada encontrada. Esses pontos são as localizações dos complexos QRS. Finalmente, é feita uma média entre as distâncias entre os complexos e o resultado, obtido em número de amostras, é convertido para unidades de tempo e finalmente para batimentos por minuto, que é a unidade usual para frequência cardíaca. O valor obtido é salvo em um arquivo, de onde pode ser lido por todos os outros programas.

3.5.3 Geração da Página Web

O terceiro bloco funcional tem a função de ler tanto as amostras geradas pelo primeiro bloco quanto o valor da frequência cardíaca gerado pelo segundo bloco e apresentar todas essas informações em uma página Web. A Figura 3.5 mostra o fluxograma para este programa.

No início de sua execução, o programa busca no banco de dados a lista de pacientes cadastrados no sistema. Em seguida, o programa verifica qual paciente está atualmente selecionado na página Web. Feito isso, o software busca no banco de dados as informações pessoais do paciente, como nome, idade e sexo e as configurações, como limites aceitáveis para frequência cardíaca e os sensores associados ao paciente. Em seguida, é gerado um gráfico com as amostras obtidas do primeiro bloco. O gráfico é então copiado para a árvore de diretórios do servidor Web, de modo que possa ser acessado pelo servidor na hora da montagem da página. Caso não haja mudança no paciente, o programa segue buscando as amostras e gerando o gráfico. Cada vez que o usuário seleciona um paciente diferente, os dados referentes a esse novo paciente são carregados e segue o ciclo.

Tanto o software quanto o servidor Web têm participação na montagem da página Web. A função do software é deixar transparente para o servidor o processo de geração do gráfico, fornecendo para este apenas uma figura com o gráfico pronto para ser exibido na página. O servidor, por outro lado, é responsável por tratar as requisições do usuário, atualizar o banco de dados caso o usuário faça alguma alteração cadastral nos pacientes e montar a página Web, cuidando da formatação e da animação do gráfico. A página Web é a interface entre o usuário e o sistema.

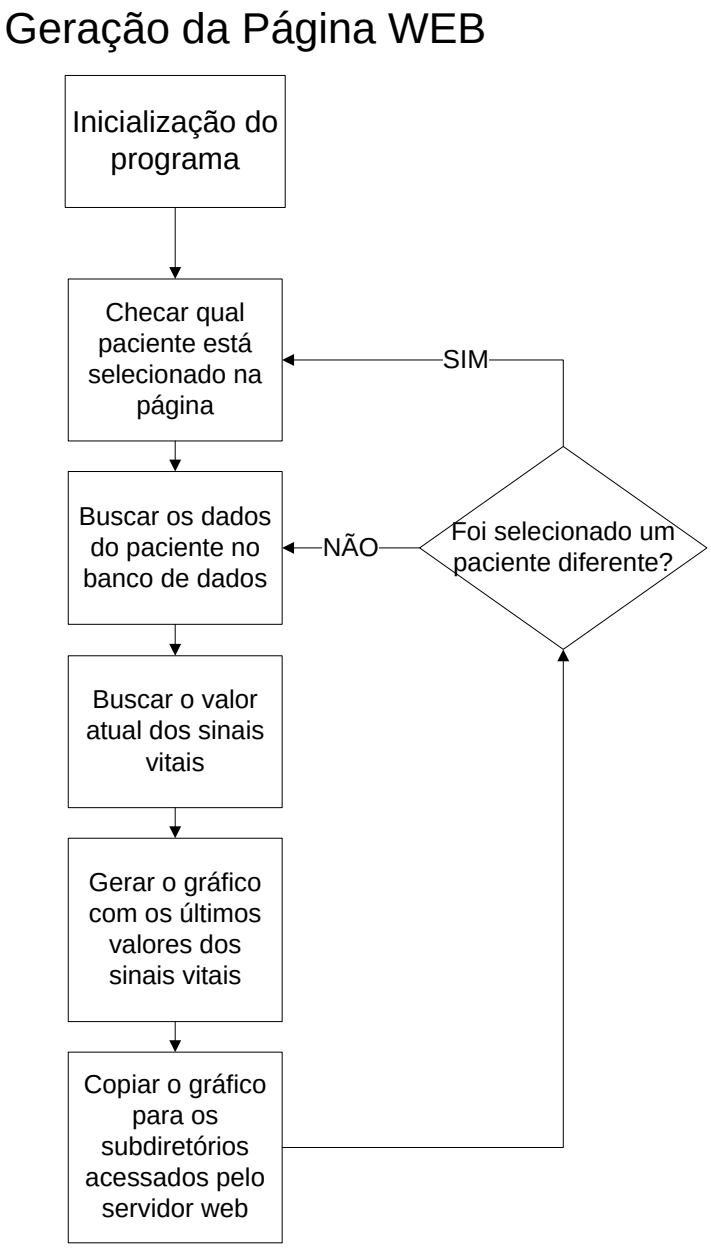

Figura 3.5: Fluxograma do software que gera a página Web

É através dela que o usuário faz a monitoração do paciente e também configura os valores máximo e mínimo que os sinais vitais podem atingir sem oferecer risco para o paciente.

3.5.4 Alerta para a Família

O último bloco funcional tem a importante função de monitorar o valor dos sinais vitais dos pacientes, disparando um alerta para a família caso o valor ultrapasse os valores pré-definidos pelo médico como valores seguros. A Figura 3.6 mostra o fluxograma para este programa.

O programa funciona lendo continuamente os sinais vitais de todos os pacientes. Como já foi explicado, para este trabalho, o único sinal vital monitorado é a frequência cardíaca. Assim, o programa fica lendo o valor atual da frequência cardíaca de cada paciente e comparando com os limites definidos pelo médico para cada paciente. Caso alguma frequência cardíaca ultrapasse o limite superior ou inferior, são lançados diversos alertas. O primeiro deles é um SMS, que é enviado para um telefone celular que pode ser cadastrado através da página Web. Em seguida, é mandado um email, também para o endereço cadastrado através da página Web. Finalmente, a ocorrência do evento é registrada no banco de dados. Além disso, o sistema exibirá, na página Web, um alerta que informa ao usuário qual evento ocorreu e com qual paciente.

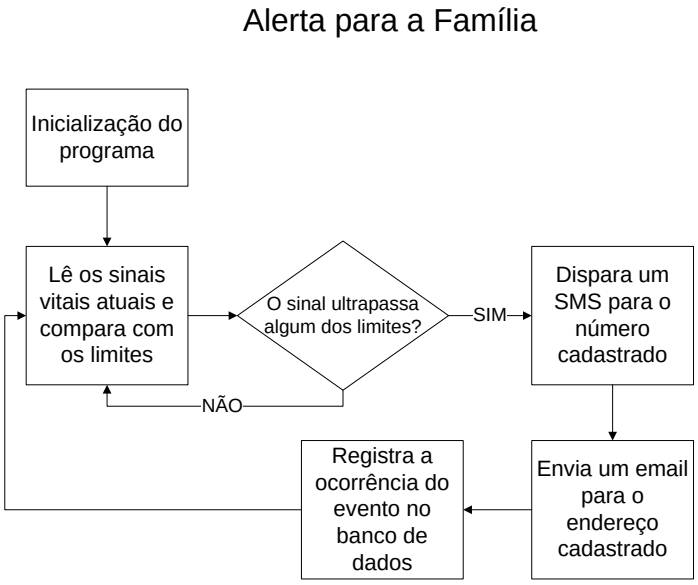

Figura 3.6: Fluxograma do software que envia um alerta para a família do paciente

3.6 Metodologia de Validação

O projeto foi desenvolvido com suporte ao perfil bluetooth HDP. Por ser um perfil relativamente novo, adicionado somente esse ano à árvore oficial do Linux, ainda não há uma grande disponibilidade de equipamentos compatíveis com esse padrão. Devido a isto, foi necessário emular um sensor utilizando um laptop rodando Linux. O laptop executa um programa que simula um sensor, enviando amostras geradas computacionalmente. As amostras pertencem a um sinal de frequência variável e adicionado de ruído [1], de modo a reproduzir, o mais fielmente possível, um sinal real.

Com esse ambiente de validação, podemos ter uma reprodução fiel, já que ambos os dispositivos comunicam-se através de um protocolo bem estabelecido, que seria o mesmo utilizado por um sensor real. O software que emula um sensor foi desenvolvido utilizando a biblioteca BlueZ (BLUEZ, 2011), disponível na árvore do Linux e que, a partir de sua versão 4.8, disponibiliza rotinas para trabalhar com dispositivos HDP.

[1] Para a validação, foi gerado um Ruído Branco Gaussiano Aditivo, tipo de ruído que é comumente encontrado em medições de sinais elétricos em geral.

Capítulo 4

Resultados e sua Análise

4.1 Resultados

Nesta seção serão apresentados os resultados do trabalho. Para uma melhor organização, os resultados serão divididos em três módulos: a página Web, que faz a interação com o usuário e apresenta os sinais vitais do paciente, o software, que faz a aquisição dos dados e toda a análise dos sinais e o hardware sobre o qual a aplicação é executada.

4.1.1 Página Web

A Figura 4.1 mostra a visão principal da página Web. É nesta tela que é mostrado o ECG do paciente em tempo real. A página fica disponível para acesso a partir da rede local e pode ser disponibilizada para acesso externo. Desta forma, o médico pode fazer o acompanhamento do paciente à distância, não havendo a necessidade de manter um profissional em contato direto com o paciente.

O gráfico mostrado na Figura 4.1 foi gerado computacionalmente, mas possui todas as características presentes em um sinal de ECG real, conforme discutido nas seções anteriores. Além do gráfico, pode-se observar que a página mostra os dados principais para identificação do paciente: o nome, sexo e idade. Além disso, a página mostra a frequência cardíaca atual do paciente. Isto é uma vantagem em relação a exames clássicos de ECG, nos quais o médico tinha que aferir a frequência cardíaca contando os quadrados ocupados por um ciclo em papel milimetrado. Em seguida, a página mostra um registro da maior e menor frequência cardíaca já registradas para este paciente e os limites inferior e superior para a frequência cardíaca do paciente.

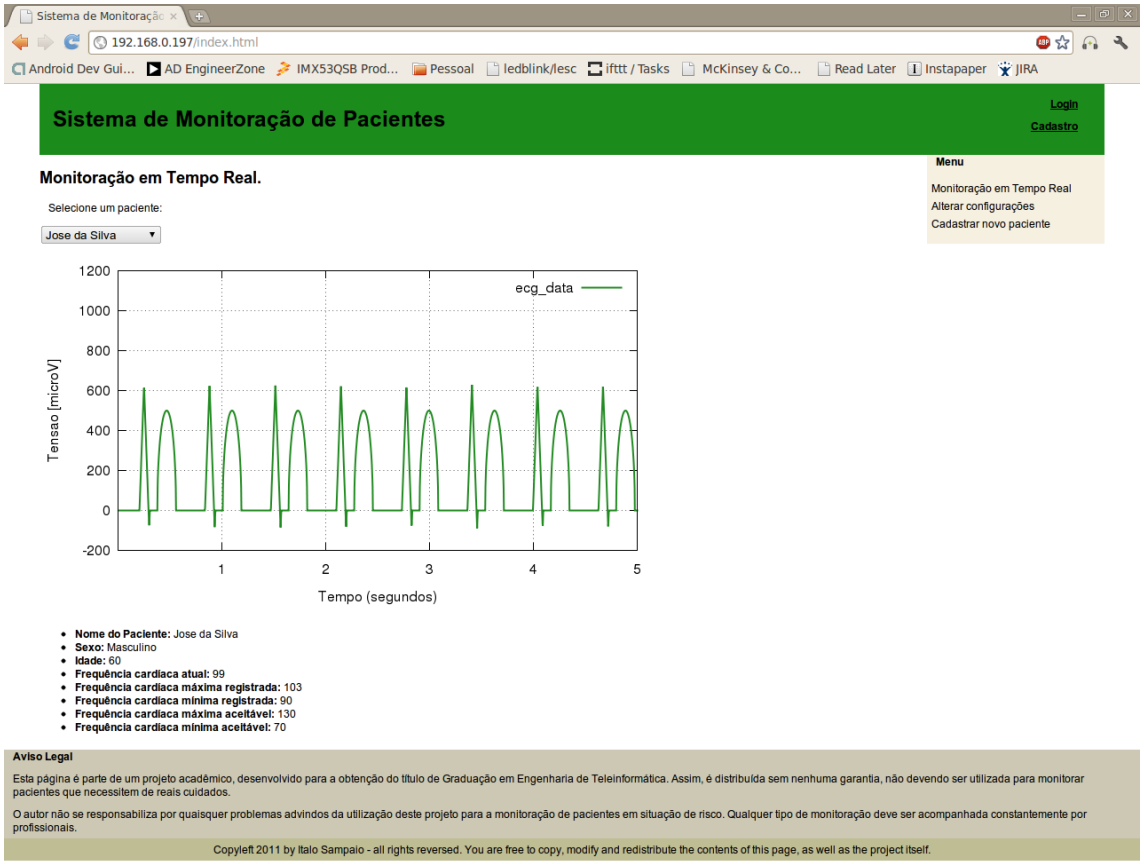

Figura 4.1: Página Web com o eletrocardiograma em tempo real

A Figura 4.2 mostra a tela de cadastro de pacientes. É nessa tela que o médico faz o cadastro de novos pacientes que serão monitorados pelo sistema. Conforme discutido antes, o médico deverá, através de exames tradicionais, determinar os limites seguros para a frequência cardíaca do paciente. uma vez determinados os limites, o médico entra com os valores nesta tela e o sistema se encarrega de enviar um alerta caso a frequência cardíaca ultrapasse um desses limites.

Já a Figura 4.3 mostra a tela de alteração de configurações do paciente, na qual o médico pode modificar ou atualizar os dados dos pacientes cadastrados.

A página Web oferece as ferramentas necessárias não só para o acompanhamento em tempo real do estado do paciente, mas também para manter o sistema atualizado, tudo isso sem que o médico tenha que se deslocar para o domicílio do paciente. Além disso, a página Web pode ser acessada por várias pessoas simultaneamente, podendo ser acessada pelo médico, por vários familiares e por funcionários do plano de saúde,

Figura 4.2: Página de cadastro de novos pacientes

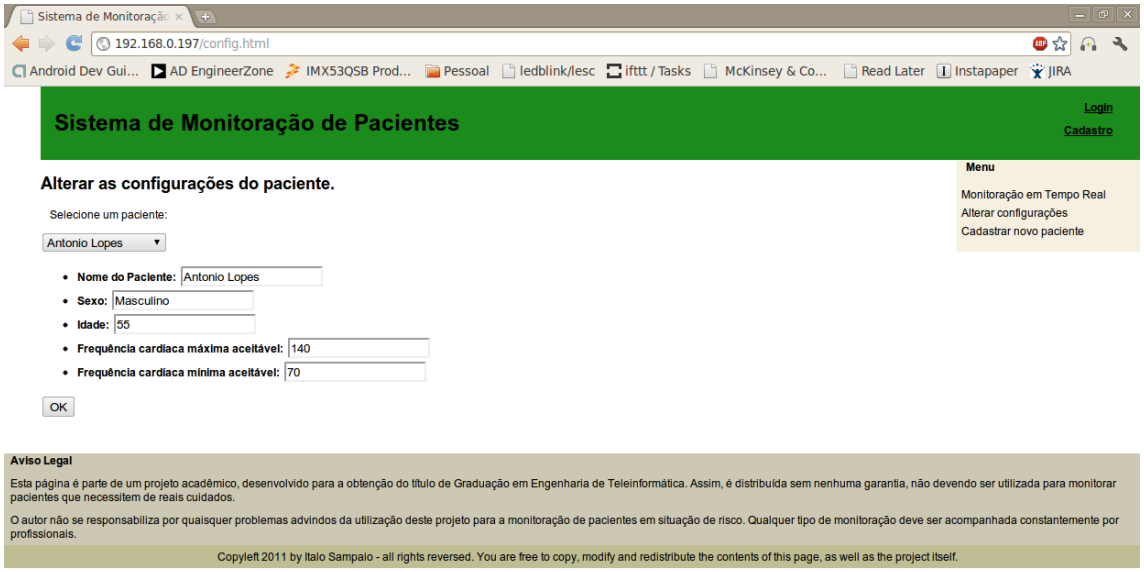

Figura 4.3: Página de alteração dos dados do paciente

por exemplo. A Tabela 4.1.1 mostra a carga no sistema quando temos vários clientes acessando a página Web ao mesmo tempo.

Observando os resultados apresentados na Tabela 4.1.1, podemos ver que, mesmo com 4 clientes conectados simultaneamente, o que é um cenário superestimado, o sistema não está operando em sua carga máxima, sendo capaz de responder às requisições sem degradação na experiência do usuário e sem correr o risco de perder

# Clientes	Carga no processador (%)	Memória ocupada (MB)
0	55	36,228
1	62	37,300
2	59	37,672
3	72	39,680
4	83	40,052

Tabela 4.1: Influência de múltiplos clientes acessando a página Web na performance do sistema

amostras.

4.1.2 Software

Nesta seção, serão apresentados os resultados ligados ao software. Conforme dito anteriormente, o software resultante é uma série de programas e scripts que desempenham as funções detalhadas na seção 3.5.2, além de alguns scripts que realizam a comunicação entre os diversos módulos de software.

Módulo de Software	Tempo de execução(ms)
Aquisição de Dados [1]	2,272
Análise de Dados	405
Geração da Página Web	1.026
Alerta para a Família	781

Tabela 4.2: Tempo de execução para cada bloco funcional de software

A Tabela 4.1.2 mostra o tempo de execução para cada módulo do software, medidos com o sistema em funcionamento. Podemos ver nessa tabela que o principal gargalo no tempo de execução do sistema é a geração da página Web, cujo tempo de execução é algumas ordens de grandeza maior que o tempo dos outros módulos. Mesmo assim, podemos ver que a execução de todos os módulos de Software leva cerca de 1 segundo para ser concluída. Como o tempo razoável para realização do atendimento a uma vítima de uma parada cardíaca, por exemplo, é da ordem de minutos, o tempo de execução da ordem de 1 segundo é perfeitamente aceitável.

[1] O tempo de aquisição dos dados é limitado pelo sensor. Para este trabalho, foi considerado um sensor de frequência de amostragem igual a 440 amostras por segundo

4.1.3 Hardware

Conforme mencionado anteriormente, o hardware utilizado foi a Quick Start Board da Freescale, não tendo havido necessidade de desenvolver hardware extra. O único hardware extra utilizado foi o receptor bluetooth. Adicionalmente, é possível utilizar um receptor wifi para dar uma maior mobilidade ao dispositivo. No entanto, como o sistema deve ficar fixo em um local da casa do paciente, não há necessidade de utilizar a conexão wifi, sendo suficiente uma rede cabeada.

Apesar de não ser um projeto de hardware propriamente dito, este trabalho tem um forte apelo comercial, o que leva à conclusão natural de que, em trabalhos futuros, deve ser desenvolvido um protótipo de hardware para comportar o trabalho, eliminando a necessidade do uso da Quick Start Board e diminuindo o custo por unidade produzida.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Este trabalho descreveu o desenvolvimento de um sistema de *Home Care*, contemplando o porte do Sistema Operacional Linux, personalização do kernel do sistema, desenvolvimento de software para aquisição, processamento e exibição de dados, além de um servidor Web completo rodando em hardware de baixo custo. O sistema desenvolvido é capaz de realizar todas as tarefas propostas em um tempo baixo o suficiente para garantir a segurança do paciente que está sendo monitorado.

O sistema desenvolvido representa um avanço em relação aos sistemas de *Home Care* disponíveis no mercado atualmente, já que ele pode funcionar de forma completamente auto-suficiente, recebendo todos os comandos via Web, o que permite que um operador configure o sistema à distância. Ainda devido à essa característica, o sistema elimina a necessidade de um profissional da saúde estar acompanhando o paciente em sua casa, o que acarreta em um custo menor do tratamento, um desafogamento de leitos hospitalares e um maior bem estar para o paciente, que fica cercado apenas por seus familiares. Outra vantagem do sistema proposto nesse trabalho é o fato de trabalhar com sensores sem fio, que permitem que o paciente se movimente livremente pela casa sem que a monitoração seja interrompida. Finalmente, o sistema oferece uma maior tranquilidade para a família do paciente, devido à funcionalidade de envio de alerta caso os valores dos sinais vitais do paciente ultrapassem os valores definidos pelo médico como seguros.

Desta forma, pode-se dizer com segurança que tanto o objetivo geral do trabalho

quanto os específicos foram cumpridos, resultando em um sistema completo de acordo com os objetivos estabelecidos. No entanto, conforme veremos na seção seguinte, algumas melhoras ainda precisam ser feitas antes que o dispositivo possa ser considerado pronto para ser posto no mercado.

5.2 Trabalhos Futuros

Apesar de ter cumprido com os objetivos do trabalho, o sistema proposto ainda não pode ser considerado completo o suficiente para ser competitivo no mercado. A seguir são listadas algumas melhoras que serão feitas futuramente no sistema, aumentando sua eficiência, confiabilidade e seu valor agregado:

- testar o sistema com um sensores compatíveis com perfil HDP ligados a pacientes reais;
- incluir medidas de outros sinais vitais, como pressão arterial, nível de saturação de oxigênio, frequência respiratória, etc;
- adicionar um bloco funcional de software capaz de fazer o processamento dos sinais vitais e detectar possíveis anomalias no paciente;
- desenvolver um hardware dedicado para o sistema, utilizando apenas os componentes necessários e eliminando o custo adicional de utilizar um kit de desenvolvimento;
- disponibilizar todo o software desenvolvido neste trabalho para a comunidade, permitindo que outras pessoas aprimorem o sistema.

Referências Bibliográficas

BITTENCOURT, R. *A situação medieval dos hospitais*. Junho 2011. Disponível em http://www.horadopovo.com.br/2011/junho/2967-15-06-2011/P4/pag4f.htm#, acessado em 01/11/2011.

BLUEZ. *BlueZ - Official Linux Bluetooh protocol stack*. Novembro 2011. Disponível em http://www.bluez.org/, acessado em 03/11/2011.

BONSOR, K. *Como funcionam os corações artificiais*. Novembro 1999. Disponível em http://saude.hsw.uol.com.br/coracao-artificial1.htm, acessado em 07/11/2011.

BOVET, D. P.; CESATI, M. *Understanding the Linux Kernel*. [S.l.]: O'Reilly, 2000.

D'ANGELO, L. T. *et al.* A system for intelligent home care ecg upload and priorisation. In: *In Proc. of The 32nd Annual International Conference of the IEEE EMBS*. [S.l.: s.n.], 2010.

ECLIPSE. *About Eclipse*. Junho 2011. Disponível em http://www.eclipse.org/org/, acessado em 14/10/2011.

FREE SOFTWARE FUNDATION. *GNU General Public License*. Junho 2007. Disponível em http://www.gnu.org/licenses/gpl.html, acessado em 28/08/2011.

FREESCALE. *IMX53QSB: i.MX53 Quick Start Board*. Outubro 2011. Disponível em http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX53QSB&parentCode=i.M acessado em 01/11/2011.

G1. *Idosa que foi à Justiça para ter vaga em UTI será enterrada nesta tarde*. Outubro 2011. Disponível em

http://g1.globo.com/rio-de-janeiro/noticia/2011/10/idosa-que-foi-justica-para-ter-vaga-em-uti-sera-
acessado em 01/11/2011.

G1. *Morre paciente que teria esperado atendimento em corredor de hospital*. Outubro 2011. Disponível em http://g1.globo.com/rio-de-janeiro/noticia/2011/10/morre-paciente-que-teria-esperado-atendimento
acessado em 01/11/2011.

GAZETA. *Família de aposentado que morreu em corredor de hospital pretende acionar a Justiça*. Março 2011. Disponível em http://gazetaonline.globo.com/_conteudo/2011/03/a_gazeta/minuto_a_minuto/812515-familia-de
acessado em 01/11/2011.

LATUSKE, R. *Bluetooth Health Device Profile (HDP)*. Agosto 2009. Disponível em http://www.ars2000.com/Bluetooth_HDP.pdf, acessado em 28/08/2011.

LEME, E. de O. *O que é Home Care?* Novembro 2011. Disponível em http://www.portalhomecare.com.br/home-care/o-que-e-o-home-care, acessado em 03/11/2011.

LTIB. *Linux Target Image Builder*. Junho 2011. Disponível em http://ltib.org/, acessado em 14/10/2011.

MINIX. *MINIX OS Discussion List*. Agosto 1991. Disponível em http://groups.google.com/group/comp.os.minix/msg/b813d52cbc5a044b, acessado em 28/08/2011.

NETCRAFT. *May 2010 Web Server Survey*. Maio 2010. Disponível em http://news.netcraft.com/archives/2010/05/14/may_2010_web_server_survey.html, acessado em 01/11/2011.

NETO, L. A. de L. *Eletrocardiógrafo Portátil com uma Derivação e Comunicação Bluetooth*. Dissertação (Mestrado) — Universidade Federal do Ceará, 2010.

OPENMOKO Inc. *OpenMoko*. Junho 2011. Disponível em http://http://www.openmoko.com/, acessado em 14/09/2011.

PCWORLD. *Android Market Share Growth Accelerating*. Junho 2011. Disponível em http://www.pcworld.com/article/226339/android_market_share_growth_accelerating_nielsen_fi
acessado em 28/08/2011.

SLOSS, A. *et al. ARM System Developer's Guide: Designing and Optimizing System Software*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN 1558608745.

TANENBAUM, A. S. *Modern Operating Systems*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN 9780136006633.

TGDAILY. *Android tablet market share tumbles to 26.8%*. Setembro 2011. Disponível em http://www.tgdaily.com/mobility-features/58463-android-tablet-market-share-tumbles-to-268, acessado em 19/09/2011.

TOP500. *Top 500 Supercomputer sites - Operating System share for 06/2011*. Junho 2011. Disponível em http://top500.org/charts/list/37/os, acessado em 28/08/2011.

TORVALDS, L. *Linux 3.0 release*. Julho 2011. Disponível em http://lwn.net/Articles/452531/, acessado em 28/08/2011.

U-BOOT. *Das U-Boot Source Code Tree*. Junho 2011. Disponível em http://git.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=tree, acessado em 14/10/2011.

YAGHMOUR, K. *Building Embedded Linux Systems*. [S.l.]: O'Reilly Media, Inc., 2003. ISBN 059600222X.

YODAIKEN, V. The rtlinux manifesto. In: *In Proc. of The 5th Linux Expo*. [S.l.: s.n.], 1999.