UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

BRUNO SILVA GUEDES

Linux embarcado para Power Quality

Prof. Dr. João Cesar Netto
Orientador

Porto Alegre, dezembro de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Silva Guedes, Bruno

Linux embarcado para Power Quality / Bruno Silva Guedes. – Porto Alegre: Instituto de Informática da UFRGS, 2012.

57 f.: il.

Trabalho de Graduação – Universidade Federal do Rio Grande do Sul. Curso de Engenharia de Computação, Porto Alegre, BR–RS, 2012. Orientador: João Cesar Netto.

1. Sistemas embarcados. 2. Qualidade de energia. 3. Plataformas de prototipação. 4. Sistemas de tempo-real. I. Netto, João Cesar. II. Título.

“A reader lives a thousand lives before he dies.
The man who never reads lives only one”
— GEORGE R.R. MARTIN

AGRADECIMENTOS

Este Trabalho de Graduação encerra uma etapa de seis anos desde meu ingresso na UFRGS em 2007, e o sucesso dessa caminhada se deve também a participação de diversas pessoas que me acompanharam no processo e que recebem meus agradecimentos:

Meu orientador, Prof. João Cesar Netto, pelo apoio na concepção deste trabalho, desde sua definição até os últimos detalhes da monografia.

Meu chefe na empresa onde trabalho, a IMS Power Quality, Javier Aprea, por ter acreditado em mim desde o início mesmo compreendendo o quão desafiadora a proposta do trabalho era pra mim.

Meus colegas de trabalho, que pela compreensão nos momentos mais difíceis nunca deixaram de me apoiar nesta tarefa; em particular, meu veterano de duplo-diploma e colega de trabalho Bruno Dal Bó pela imensa cooperação e ajuda imprescindível em diversos detalhes da implementação e o colega Fábio Silva, pelo apoio com seu conhecimento de hardware na construção de uma etapa do trabalho.

Meus colegas de engenharia, principalmente Cassiana Fülber, Fernando Sousa, Henrique Klein, João Almeida, João Vicente, Jônatas Rech, Leonardo Faganello, Lucas Rosa, Matheus Proença, Thiago Santini e Tyron Scholem, pela camaradagem nos melhores e também nos mais complicados momentos do curso; foi uma grande honra para mim ter sido acompanhado por todos vocês nesta jornada.

Aos amigos de longa data e aos amigos que tive a oportunidade de conhecer na França, pelos momentos de descontração, pelas viagens explorando o velho continente, pelas dificuldades compartilhadas e pelo apoio anterior ao meu ingresso na universidade que culmina na diplomação.

Finalmente, meus pais, Carla e Milton, meu irmão Matheus e minha família pelo incentivo e pela compreensão nos momentos de ausência, seja durante minha estadia no exterior visando o aprimoramento da graduação, seja pelos trabalhos de final de semestre que tomavam dias e noites de dedicação.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ADC	*Analog-Digital Converter*
ANEEL	Agência Nacional de Energia Elétrica
ALSA	*Advanced Linux Sound Architecture*
ARM	*Advanced RISC Machine*
DSP	*Digital Signal Processor*
DVI	*Digital Visual Interface*
FIFO	*First in, first out*
GDB	*GNU Debugger*
GPIO	*General Purpose Input-Output*
HDMI	*High-Definition Multimedia Interface*
IDE	*Integrated Development Environment*
IEC	*International Electrotechnical Commission*
IHM	Interface homem-máquina
ISR	*Interrupt Service Routine*
JACK	JACK Audio Connection Kit
OMAP	*Open Multimedia Applications Platform*
OSAL	*Operating System Abstraction Layer*
PRODIST	Procedimentos de Distribuição de Energia Elétrica
RTOS	*Real-time operating system*
SSH	*Secure Shell*
SO	Sistema Operacional

LISTA DE FIGURAS

RESUMO

A definição da arquitetura de um equipamento de multimedição e análise de qualidade de energia consiste na escolha de diversos elementos que devem compor tal sistema, tais como os componentes que serão utilizados na construção do mesmo, o sistema operacional, a natureza das tarefas, os periféricos e as aplicações ligadas ao sistema. Este Trabalho de Graduação apresenta uma proposta de uma nova arquitetura para estes equipamentos, baseada em uma versão do sistema operacional Linux embarcada que incorpora aspectos de tempo real. Utilizando como plataforma de prototipação uma BeagleBoard-xM, procura-se demonstrar a viabilidade de uma arquitetura baseada em Linux para a construção destes equipamentos.

Palavras-chave: Sistemas embarcados, qualidade de energia, plataformas de prototipação, sistemas de tempo-real.

Embedded Linux for Power Quality

ABSTRACT

The definition of the architecture of a Power Quality meter consists in choosing several elements which will compose this system, such as the physical components that will be used in its construction, the operating system, the tasks which will be run, the peripherals and the applications embedded in the system. This study suggests a new architecture for this equipments, based on a embedded Linux distribution with real-time capabilities. Using the BeagleBoard-xM as prototyping platform, the feasibility of an architecture based on Linux for the construction of these equipments is pursued.

Keywords: embedded systems, power quality, prototyping platforms, real-time systems.

1 INTRODUÇÃO

Equipamentos de medição e análise de qualidade de energia fazem parte dos sistemas elétricos e industriais de diversas empresas. Estes equipamentos são aplicados em todos os níveis de aplicação de energia elétrica (geração, transmissão, distribuição e consumo). Sua construção envolve diversas áreas de conhecimento, como engenharia elétrica, transformadores, teoria eletromagnética, eletrônica embarcada, processamento digital de sinais e engenharia de software.

Diversas empresas no Brasil produzem tais equipamentos, e lidam diariamente com projetos que melhorem cada vez mais sua produção: estes projetos envolvem construção de hardware, visando reduzir custos de material e melhor performance energética; construção de software, com o objetivo de produzir soluções que monitoram e controlam os equipamentos, e principalmente eletrônica embarcada, onde estão os algoritmos de engenharia elétrica e de processamento digital de sinais que realizam a medição e os cálculos das grandezas envolvidas na aplicação.

Este Trabalho de Graduação foi realizado em parceria com a IMS Power Quality [1]. É um trabalho focado na construção de uma nova arquitetura embarcada para a implementação dos algoritmos de análise de qualidade de energia, que são cada vez mais complexos e exigem cada vez mais processamento de acordo com a evolução das normas e das classes de precisão desejadas para os equipamentos.

1.1 Objetivos

O objetivo deste Trabalho de Graduação, tal qual sua definição no artigo do TG1, é desenvolver um sistema embarcado para rodar sobre uma BeagleBoard, visando fornecer uma plataforma de serviços e ferramentas que possibilitem a construção de equipamentos de análise da qualidade de energia a partir dela. Esta solução consiste em integrar no sistema embarcado da placa (uma distribuição Linux para sistemas embarcados), os drivers dos periféricos necessários para a operação destes equipamentos, incluindo os mais básicos, como teclado e display, assim como as aplicações mais avançadas e específicas, como memória compartilhada e rotinas de tempo real para medição.

Esta nova arquitetura consiste em uma *proof of concept* para a IMS. A ideia é conseguir demonstrar que o uso de um sistema embarcado rodando uma distribuição Linux é viável para os projetos da empresa e que é possível agregar novas funcionalidades a esta arquitetura. Assim, a demonstração da realização do objetivo deste Trabalho de Graduação é a implementação das principais funções de um dos equipamentos já existentes da

[1] Empresa de Porto Alegre que atua na área de equipamentos de medição e análise de qualidade de energia há mais de 30 anos. http://www.ims.ind.br

IMS, como por exemplo, um multimedidor, em uma placa de prototipação, utilizando as ferramentas de sistemas embarcados na construção destas funções.

Uma vez que o sistema operacional escolhido para o trabalho foi o Linux, procurou-se respeitar a organização do trabalho de acordo com as camadas-padrão de um sistema Linux. A Figura 1.1 mostra os quatro blocos básicos explorados neste trabalho para a definição da arquitetura construída.

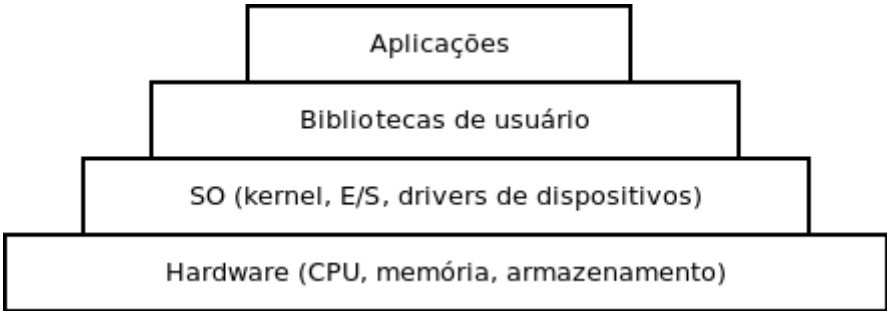


Figura 1.1: As camadas de um sistema Linux - adaptado de (Tanenbaum, 2010).

No Capítulo 2, é definida a primeira camada da arquitetura, que corresponde ao hardware utilizado neste trabalho. Uma apresentação do uso de placas de prototipação na indústria é realizada, e as características da placa escolhida, a BeagleBoard, são descritas.

O Capítulo 3 fornece a base teórica utilizada na construção de sistemas operacionais em tempo real. Estas definições são necessárias para a aplicação da teoria da amostragem que é usada para a medição de corrente e tensão, a operação básica de um equipamento de análise de qualidade de energia elétrica. A aplicação está descrita no Capítulo 4, e envolve mudanças na camada de SO, no uso de bibliotecas padrão e também na camada de aplicações em modo usuário.

O Capítulo 5 detalha a camada de hardware em relação a implementação de drivers para os dispositivos de entrada e saída presentes na arquitetura. No Capítulo 6, a arquitetura é posta a prova através do uso do *firmware* de um equipamento existente, o multimedidor de grandezas elétricas PowerNET M-500, exposto como uma aplicação de usuário que deverá rodar na placa reproduzindo o comportamento esperado do equipamento.

O Capítulo 7 analisa a prova de conceito desenvolvida neste trabalho de graduação, destacando possíveis melhorias, eventuais dificuldades e aplicações para a arquitetura proposta em futuros projetos de equipamentos de multimedição e análise de qualidade de energia.

Antes de entrar nos detalhes da construção das camadas do projeto, o contexto no qual este trabalho se enquadra é apresentado com mais detalhes no restante deste capítulo, com a apresentação dos equipamentos de medição e análise de qualidade de energia elétrica, quais as grandezas envolvidas e quais os desafios encontrados na construção dos mesmos.

1.2 Qualidade de energia

A construção de equipamentos multimedidores de energia e de analisadores de qualidade de energia é guiada por algumas normas nacionais e internacionais que definem como as grandezas envolvidas na análise devem ser medidas (regras em relação a amostragem do sinal, números de ciclo de tensão a ser considerados, etc.) e quais os graus de precisão na medição que definem as diversas classes de equipamentos. No Brasil, estas normas são regulamentadas pela ABNT, órgão de regulamentação de normas técnicas, e

pela ANEEL, agência reguladora da geração, transmissão e distribuição de energia elétrica no país. Internacionalmente, as normas que regulam este setor são administradas pela IEC, uma comissão que define os principais padrões a serem seguidos pelas agências reguladoras nos países. As normas brasileiras, como o PRODIST (PRODIST, 2010), normalmente referenciam normas da IEC, como a IEC 61000-4-7 (IEC 61000-4-7, 2010), IEC 61000-4-15 (IEC 61000-4-15, 2010) e IEC 61000-4-30 (IEC 61000-4-30, 2008).

A medição de energia elétrica consiste principalmente na medição de tensão e corrente, e a partir dos sinais de tensão e corrente medidos, a obtenção através de cálculos em um processador embarcado no equipamento, das demais grandezas elétricas (como potência e energia) e das grandezas de qualidade de energia (com harmônicos, flutuação e distúrbios na tensão). Neste trabalho, o desenvolvimento da arquitetura foi focado nos cálculos das grandezas mais simples, descritos no Capítulo 4. As grandezas de qualidade de energia possuem cálculos mais complexos, que podem ser aplicados à plataforma em um segundo momento.

De acordo com as especificações nas normas da IEC e da ANEEL, as principais grandezas que envolvem a qualidade da energia elétrica são:

- Flutuações de tensão (*flickers*) são variações de tensão, normalmente em baixas frequências e de baixa magnitude, cujo efeito é facilmente perceptível na forma de cintilação em lâmpadas e na taxa de atualização (*refresh*) de monitores de vídeo (Busatto, 2011). A norma PRODIST utiliza as definições da norma IEC 61000-4-15 para flutuação de tensão, e define os parâmetros que devem ser analisados.

- Harmônicos de tensão e de corrente são componentes senoidais cuja frequência difere da frequência nominal da rede. Elas distorcem o sinal fundamental através da inserção de sinais cuja frequência é o múltiplo da frequência fundamental, e são originados principalmente pela comutação do sinal na rede e da distorção da forma de onda em retificadores e conversores. Quando presentes em excesso na rede, podem causar superaquecimento de motores e geradores ou ainda perdas de energia em transformadores. A norma que regula a precisão e o modo de medição de harmônicos e da distorção harmônica total de um sinal é a IEC 61000-4-7.

- Distúrbios de tensão são eventos assíncronos que ocorrem na rede e que devem ser considerados de modo a validar ou invalidar medições que ocorreram depois de um evento. A norma IEC 61000-4-30 considera três tipos de distúrbios: *sag*, ou afundamento, que consiste em uma redução momentânea fora da tolerância aceitável do nível de tensão da rede; *swell*, ou elevação, que é o evento inverso ao *sag* e as interrupções, que são as quedas de energia em uma ou mais fases da rede.

1.3 Multimedidores e analisadores de energia

Os **multimedidores** de grandezas elétricas permitem a medição com grande precisão de todos os parâmetros de consumo da energia elétrica como tensão e corrente, a potência e sua demanda, e energia em todos os quadrantes; mostram as grandezas em displays multifuncionais, armazenam essas grandezas em memória de massa, permitem a parametrização por teclado e de forma remota e suportam protocolos de comunicação industrial. Esses medidores eletrônicos têm todas as características funcionais de medidores inteligentes e são comumente utilizados para porta ou fundo de painéis de automação, ou rateio de energia em prédios comerciais. O modelo mais recente de multimedidor desenvolvido

pela IMS é o PowerNET M-500 (Figura 1.2); a arquitetura desenvolvida neste trabalho busca reproduzir as funções deste equipamento.

Os **analisadores** da qualidade da energia elétrica são equipamentos que seguem normas nacionais (PRODIST) e internacionais (IEC) para a detecção e classificação de distúrbios no fornecimento de energia. São normalmente usados por concessionárias de energia elétrica para monitorar os parâmetros de qualidade de energia. Consultores usam este tipo de equipamento para realizar análises do consumo e sugerir melhorias em instalações comerciais e/ou industriais.

Esses equipamentos estão sendo desenvolvidos no Brasil para dar suporte as campanhas de medição que visam a criação da Rede Inteligente (*smart grid*) no país. Eles permitem medições remotas e automáticas usando protocolos inteligentes, são integrados a sistemas de aquisição e armazenamento de dados e normalmente instalados em condições atmosféricas adversas.

Exemplos de analisadores da IMS são o PowerNET P-600 e o PowerNET PQ-600 (Figura 1.2). O PQ-600 é o modelo mais recente e implementa o maior número de parâmetros de Power Quality (flutuação, distúrbios e desequilíbrios de tensão, por exemplo). A ideia é que no futuro a arquitetura deste trabalho seja aplicada também nestes equipamentos, que exigem mais processamento e mais robustez da plataforma. A seção 1.4 detalha esta necessidade.

Figura 1.2: Multimedidor PowerNET M-500 e Analisador PowerNET PQ-600 (IMS, 2012)

1.4 Arquiteturas de sistema

Uma grande parte dos equipamentos de multimedição e análise de qualidade de energia são desenvolvidos usando eletrônica embarcada, rodando sistemas operacionais de tempo real simples, com suporte a drivers de IHM (display e teclado) e com sistema de arquivos.

A arquitetura atual destes equipamentos contempla transdutores de sinais elétricos (tensão e corrente), circuitos de conformação de sinais (filtros e amplificadores), sua aquisição digital (ADC para discretização de valores no tempo e valor), DSP e/ou micro controladores pra processamento, visualização e controle, memórias para armazenamento e interfaces para supervisão por comunicação remota (Modbus, RS-232/485).

O grande desafio da arquitetura proposta neste trabalho é, além de contemplar estas características básicas de qualquer equipamento multimedidor ou analisador, integrar uma necessidade crescente de algoritmos mais precisos de processamento, no domínio tempo

e frequência, em tempo real, de quantidades massivas de amostras digitalizadas de sinais, em uma plataforma de baixo consumo e baixo custo (*low cost of ownership*).

Em equipamentos futuros para análise da qualidade da energia elétrica para instalação em painéis, a IMS estuda a possibilidade de adicionar características avançadas, como displays gráficos com oscilografia em tempo real, armazenamento massivo de dados, suporte a diversos Protocolos de Comunicação - como o DNP3 (DNP3, 2012) e o SIBMA [2] - e interfaces (USB, Ethernet), comunicação entre medidores, entre outras; funções que são facilmente suportadas por sistemas operacionais mais que possuem mais recursos (como o Linux) rodando em plataformas de hardware avançadas (por exemplo, o OMAP).

[2]Sistema Brasileiro de Medição Avançado, protocolo em desenvolvimento no Brasil para padronizar e automatizar a comunicação com equipamentos multimedidores

2 AMBIENTE DE PROTOTIPAÇÃO - BEAGLEBOARD

Este capítulo explora a base do sistema desenvolvido neste trabalho, de acordo com a Figura 1.1. O hardware escolhido para a implementação, a BeagleBoard, é apresentado, assim como algumas outras placas de prototipação também usadas na indústria e que fizeram parte da pesquisa em busca da escolha de uma placa. Por fim, o ambiente de desenvolvimento para a construção da arquitetura é descrito.

2.1 Placas de prototipação

O uso de placas de prototipação na construção de novas arquiteturas visando o desenvolvimento de hardware e software para um novo equipamento se justifica pela possibilidade de testar implementações e provar conceitos que podem ser exportados futuramente para o projeto do equipamento em si. No caso do desenvolvimento de software embarcado, o ponto interessante é poder escrever e testar código em um processador equipado por diversos componentes que pode ser utilizado posteriormente no projeto principal.

Deste modo, um dos objetivos deste trabalho foi utilizar uma placa de prototipação cujo processador fosse atraente no sentido de atender os requisitos de construção de equipamentos de multimedição, para que uma arquitetura desenvolvida sobre a placa pudesse ser utilizada na construção destes equipamentos.

Três placas que possuem características adequadas para a construção de equipamentos em geral para aplicações de medição de energia (como possuir projeto de hardware livre e gratuito, processadores com baixo consumo de potência e suporte a um sistema multitarefa capaz de executar múltiplas medições e executar diversas aplicações ao mesmo tempo) foram cogitadas para a implantação deste trabalho, e são descritas a seguir:

A **PandaBoard** (Figura 2.1) é uma placa equipada de um microprocessador ARM Cortex-A9 com clock de 1.2 GHz (PandaBoard, 2012). Possui ainda 1 GB de memória RAM e diversos periféricos. É uma placa cara, custando em torno de US$ 200, preço justificado pelo alto valor do processador (o Cortex A9 é utilizado, por exemplo, nos *system-on-a-chip* Apple A5 da Apple em alguns de seus modelos de *smartphones* e *tablets*. Além disso, este processador não possui um núcleo DSP programável que seria ideal para a aplicação deste trabalho.

A **Raspberry Pi** (Raspberry, 2012) vai na contramão da PandaBoard e é uma placa menor, com um processador mais simples (um ARM11 com clock de 700 MHz) e apenas 256 MB de memória RAM (Figura 2.1). Possui menos periféricos também, mas possui como vantagem um custo bem mais acessível (de 25 a 35 dólares) refletido pela maturidade do processador no mercado (equipado na maioria dos telefones da Nokia e em boa parte dos telefones da Samsung, por exemplo).

Finalmente, a placa de prototipação escolhida para o desenvolvimento da arquitetura

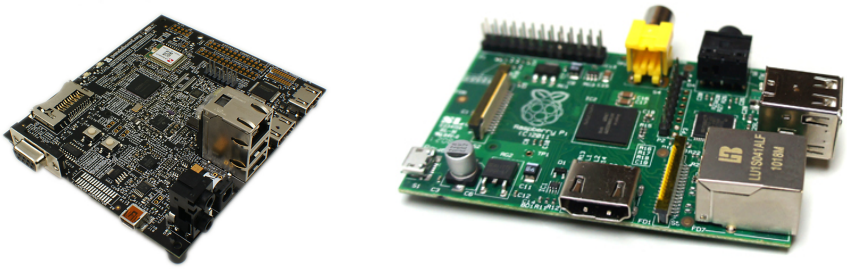


Figura 2.1: PandaBoard e Raspberry Pi

proposta neste trabalho foi a **BeagleBoard**. Sua principal vantagem sobre as concorrentes, para esta aplicação, é o ótimo custo/benefício (a placa custa aproximadamente 150 dólares). As características da placa são descritas na próxima seção.

2.2 Apresentação da BeagleBoard

A BeagleBoard é uma pequena placa de aproximadamente 8 cm x 8 cm, como mostrado na Figura 2.2. Construída por um time de engenheiros da Texas Instruments, ela é voltada para aplicações educacionais, já que suas características a classificam como um pequeno computador com baixo consumo de energia, e também como um modo de demonstrar as capacidades dos chips da família OMAP-DM da TI (TI, 2012).

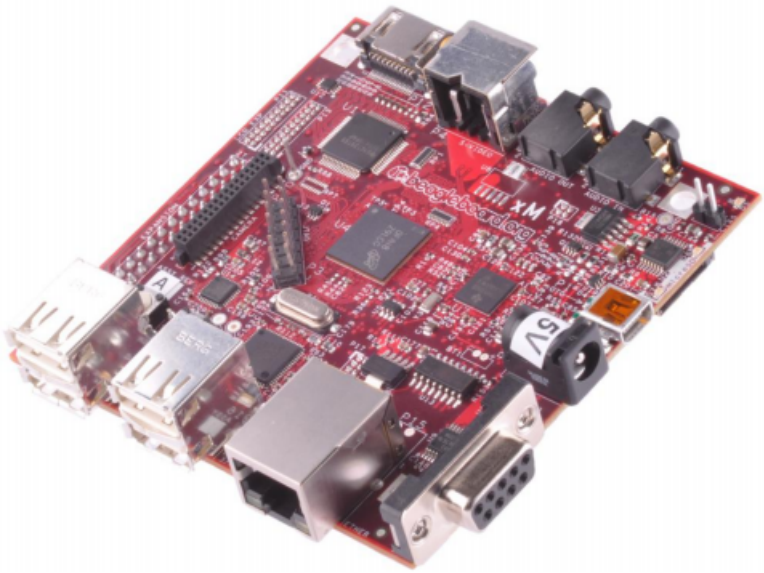


Figura 2.2: A placa BeagleBoard-xM revC.

A placa é equipada por um processador DM3730CBP, que contém dois núcleos, um núcleo ARM com um clock de 1 GHz e outro DSP, com um clock de 800 MHz. Possui ainda 512 MB de memória RAM, armazenamento através de um cartão microSD e suporte a diversas interfaces (USB, *stereo audio*, DVI (vídeo), Ethernet, JTAG e outras. A alimentação é fornecida através de uma fonte que forneça até 3A a uma tensão de 5V DC.

O processador DM3730 da BeagleBoard se destaca pelo consumo de energia extremamente baixo (735 mW a uma frequência de 800 MHz), tornando-o uma escolha ideal para nossa aplicação. Este balanço entre performance e baixo consumo faz com que ele já seja usado em diversas aplicações, como jogos, automação residencial, celulares e *tablets* (DM3730, 2012).

A BeagleBoard é capaz de rodar diversos sistemas operacionais, como Windows Embedded CE, RISC OS e também diversas distribuições de Linux, como Fedora e Ubuntu. A placa suporta boot diretamente do cartão microSD de 4GB fornecido com a placa, através do *bootloader* que já vem gravado na sua memória ROM.

O esquemático da Figura 2.3 consiste no diagrama de blocos de alto nível da BeagleBoard, somado a algumas das adições efetuadas neste trabalho visando atingir os objetivos propostos no Capítulo 1, e que serão descritas nos capítulos seguintes. Nesta figura, os blocos com fundo preto correspondem a circuitos existentes na placa; em branco, os principais componentes de entrada e saída, e em cinza claro, as adições efetuadas neste trabalho de graduação, que serão detalhadas nos capítulos seguintes.

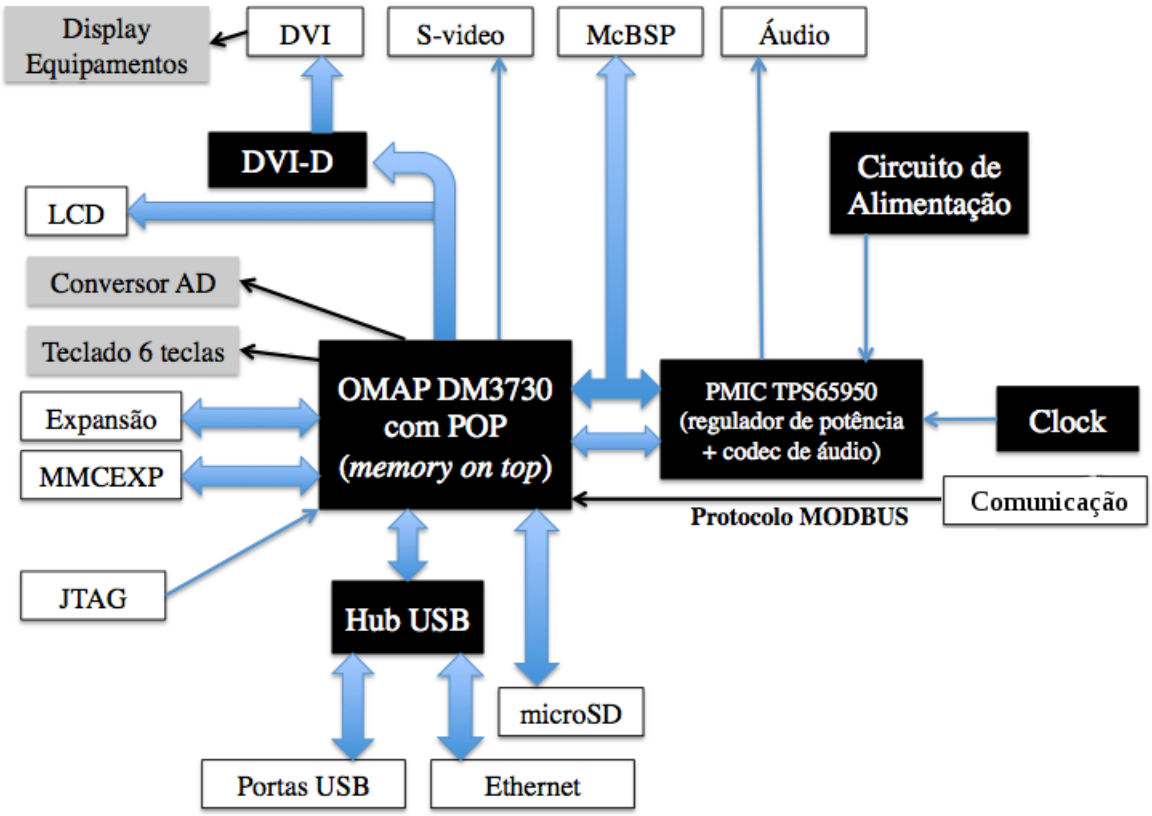


Figura 2.3: Diagrama de blocos de alto nível da BeagleBoard - adaptado de (Coley, 2010).

2.3 Ambiente de desenvolvimento

A preparação do ambiente de desenvolvimento consiste em uma etapa importante e essencial no desenvolvimento do trabalho. Visto que o desenvolvimento na BeagleBoard envolve o uso de diversas ferramentas, foi necessário dedicar um tempo razoável à preparação destas para a implementação do projeto.

Para tanto, foi utilizado um computador de trabalho rodando o sistema Ubuntu 12.04 RTS. Este computador foi escolhido pelo fato de fornecer suporte às ferramentas neces-

sárias de compilação cruzada para a BeagleBoard, já que o compilador utilizado para gerar código objeto para ela a partir de código-fonte em C precisava ser compatível com a plataforma ARM.

A distribuição Linux utilizada no trabalho é a Angstrom, que por sua vez é baseada no OpenEmbedded. O OpenEmbedded é um projeto open-source para a construção de uma distribuição Linux, que pode ser embarcada em diversas arquiteturas de hardware e que possibilita fácil customização de seu núcleo (OpenEmbedded, 2012). Para compilar o código-fonte do OpenEmbedded tendo como alvo a BeagleBoard, foi utilizado o BitBake (Bitbake, 2012), que constrói o núcleo do *kernel* e seleciona quais pacotes devem ser instalados no mesmo a partir de receitas pré-definidas em uma linguagem de alto nível (Python), de forma parecida com o que o script *configure* faz em conjunto com o programa *make*. Assim, a cada modificação do *kernel* o processo de gerar as novas imagens se resume a executar a receita adequada com o código modificado e copiar a imagem do *kernel* gerada mais os arquivos de *boot* para o cartão microSD da BeagleBoard.

A seleção dos pacotes incluídos pelo BitBake na compilação cruzada do sistema operacional depende dos requisitos do sistema a ser construído. Neste trabalho, foram incluídos pacotes como o GDB (para depuração de código) e o Jack (servidor para aplicações de áudio). O BitBake se encarrega de resolver as dependências dos pacotes durante o processo de construção, incluindo estas na imagem do Linux que foi instalada na placa. Além disso, o BitBake gera as ferramentas necessárias para a construção de aplicações para a plataforma alvo escolhida (neste caso, um compilador e um montador compatíveis com a arquitetura ARM).

A lista abaixo foi extraída dos arquivos de receitas do BitBake e de configuração da construção da imagem listando os pacotes utilizados na sua montagem:

```
1  // tarefas basicas selecionadas da task-base para uma imagem de console do
       OpenEmbedded:
2
3    ${@base_contains('MACHINE_FEATURES', 'kernel26','task-base-kernel26','task-base-
       kernel24',d)} \
4    ${@base_contains('MACHINE_FEATURES', 'apm', 'task-base-apm', '',d)} \
5    ${@base_contains('MACHINE_FEATURES', 'acpi', 'task-base-acpi', '',d)} \
6    ${@base_contains('MACHINE_FEATURES', 'keyboard', 'task-base-keyboard', '',d)} \
7    ${@base_contains('COMBINED_FEATURES', 'alsa', 'task-base-alsa', '',d)} \
8    ${@base_contains('COMBINED_FEATURES', 'ext2', 'task-base-ext2', '',d)} \
9    ${@base_contains('COMBINED_FEATURES', 'vfat', 'task-base-vfat', '',d)} \
10   ${@base_contains('COMBINED_FEATURES', 'pci', 'task-base-pci', '',d)} \
11   ${@base_contains('COMBINED_FEATURES', 'pcmcia', 'task-base-pcmcia', '',d)} \
12   ${@base_contains('COMBINED_FEATURES', 'usbgadget', 'task-base-usbgadget', '',d)} \
13   ${@base_contains('COMBINED_FEATURES', 'usbhost', 'task-base-usbhost', '',d)} \
14   ${@base_contains('COMBINED_FEATURES', 'uboot', 'task-base-uboot', '',d)} \
15   ${@base_contains('COMBINED_FEATURES', 'redboot', 'task-base-redboot', '',d)} \
16   ${@base_contains('DISTRO_FEATURES', 'nfs', 'task-base-nfs', '',d)} \
17   ${@base_contains('DISTRO_FEATURES', 'cramfs', 'task-base-cramfs', '',d)} \
18   ${@base_contains('DISTRO_FEATURES', 'smbfs', 'task-base-smbfs', '',d)} \
19   ${@base_contains('DISTRO_FEATURES', 'ipv6', 'task-base-ipv6', '',d)} \
20   ${@base_contains('DISTRO_FEATURES', 'ipsec', 'task-base-ipsec', '',d)} \
21   ${@base_contains('DISTRO_FEATURES', 'ppp', 'task-base-ppp', '',d)} \
22   ${@base_contains('DISTRO_FEATURES', 'raid', 'task-base-raid', '',d)} \
23
24   // pacotes adicionais usados neste trabalho
25
26 ANGSTROM_EXTRA_INSTALL += "jack-server
27               kernel-modules
28               libsndfile1
29               alsa-dev
30               libsamplerate0
31               libsamplerate0-dev
32               gdb
33               gdbserver"
```

O desenvolvimento de aplicações foi feito na mesma máquina de trabalho, com auxílio da IDE Eclipse (Axelson, 2012). O Eclipse permite o *cross-compiling* de aplicações escritas em C na máquina de trabalho utilizando o compilador *gcc* para ARM e o envio do código objeto para a placa via SSH. Além disso, ele facilita o desenvolvimento de aplicações por possibilitar a depuração do código rodando na placa (através de uma conexão com um servidor do GDB na BeagleBoard). É possível ainda gerenciar de forma simples nesta IDE as dependências de bibliotecas compiladas para ARM geradas no processo de compilação do *kernel* e que são necessárias nas aplicações desenvolvidas.

O diagrama da Figura 2.4 sintetiza o processo de desenvolvimento utilizado no trabalho.

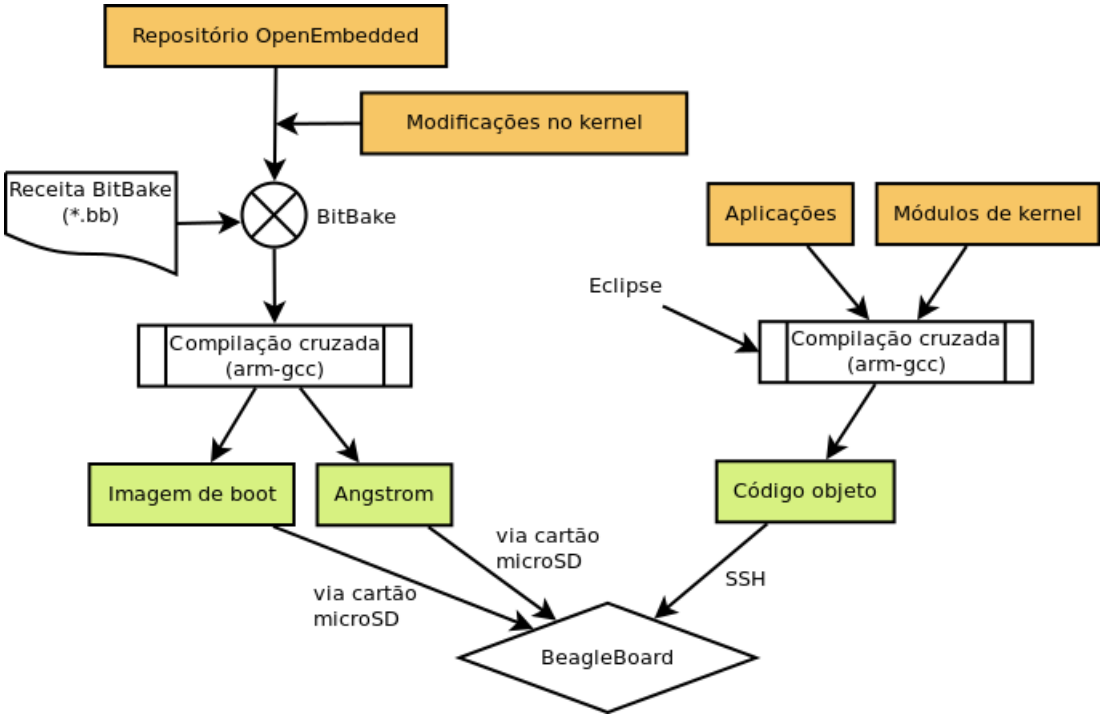


Figura 2.4: O processo de desenvolvimento utilizado no trabalho.

3 SISTEMAS DE TEMPO REAL EM AMBIENTES EMBARCADOS

As aplicações de medição de energia baseiam-se na amostragem de sinais de tensão e de corrente com características de tempo real, tais como a existência de um prazo em um período de amostragem, a importância de não perder amostras para não prejudicar uma medição e a necessidade de efetuar os cálculos de outras grandezas elétricas dependentes de tensão e corrente em intervalos de tempo definidos por normas internacionais. Assim, um sistema operacional que será aplicado em um equipamento multimedidor precisa incorporar aspectos de um sistema operacional de tempo real (RTOS).

Neste capítulo são apresentadas as características e as classificações existentes para RTOSs, e como elas se aplicam em uma distribuição Linux embarcada em um processador ARM. As mudanças a nível de *kernel* em relação a comunicação entre tarefas e escalonamento de processos são igualmente discutidas.

3.1 Introdução

Um RTOS é um sistema projetado para atender as necessidades de uma aplicação de tempo real. Ao contrário de uma aplicação de propósito geral, onde normalmente o sistema operacional irá rodar várias tarefas de usuário e de sistema ao mesmo tempo, e o momento em que essas tarefas estão efetivamente executando na CPU não é muito importante, desde que a tarefa seja concluída em algum momento com o resultado esperado, um RTOS precisa cumprir estas tarefas dentro de um *deadline* pré-estabelecido (Shaw, 2001). Desta forma, para que as tarefas de um RTOS sejam consideradas corretas, os resultados produzidos por elas devem não apenas ser corretos em relação ao seu valor, mas também devem ser gerados em um certo intervalo no tempo.

A partir desta definição, as tarefas de um RTOS possuem critérios de início e fim permitidos para sua execução, de modo a garantir a exatidão da sua operação. A temporização incorreta destas tarefas pode gerar resultados pouco úteis (por exemplo, um reprodutor de Blu-Ray deve decodificar os dados da mídia física em uma taxa pré-estabelecida; se ele for mais rápido ou mais lento, a exibição do vídeo será prejudicada), mas também resultados potencialmente danosos, como por exemplo em um sistema que seja responsável por controlar um sistema *fly-by-wire* de um avião (Abbott, 2006).

3.1.1 Classificação de sistemas de tempo real

Os exemplos citados na introdução permitem a separação de sistemas de tempo real em duas categorias: aqueles que são **estritos** (ou *hard real-time systems*), onde os critérios temporais devem ser respeitados sem exceção, podendo ocasionar um defeito do sistema

no caso contrário, e os sistemas **tolerantes** (ou *soft real-time systems*), cujo cumprimento dos *deadlines* é desejado, mas a falha em cumprir alguns destes prazos é aceitável.

Uma distinção em relação a **criticidade** de um sistema é sugerida por (Shaw, 2001), e diz respeito ao custo de uma falha do sistema: quanto maiores as cifras envolvidas, mais crítico é o sistema. É uma medida diferente da rigorosidade do sistema (estrita ou tolerante), embora normalmente sistemas rigorosos, como o exemplo do controle de uma aeronave, é potencialmente crítico. No caso deste Trabalho de Graduação, as tarefas do RTOS desenvolvido não chegam a ser rigorosas: a perda de uma ou mais medições pode ser facilmente tratada e não causa danos ao sistema, porém a criticidade existe visto que a perda destas medições pode causar prejuízos a uma concessionária de energia que queira cobrar seus clientes, por exemplo.

3.1.2 Características de um RTOS

Um sistema operacional para um sistema em tempo real precisa prover alguns requisitos em termos de serviços para que ele seja capaz de realizar corretamente as tarefas necessárias no prazo previsto. As principais funcionalidades que devem estar presentes na construção de um RTOS são:

- **Suporte à prioridades:** um sistema operacional em tempo real deve possuir suporte à prioridades, já que funcionalidades críticas desse sistema que tenham restrições temporais devem possuir prioridades mais altas.

- **Preempção:** quando uma tarefa de mais alta prioridade entra na fila de tarefas prontas para ser executadas, ela deve ser capaz de tomar o lugar de uma tarefa de mais baixa prioridade em execução.

- **Latência conhecida do escalonador:** a diferença de tempo entre o momento em que uma tarefa crítica se torna pronta para executar e o instante em que sua execução inicia deve ser conhecida (ou seja, essa diferença deve ser determinística).

O último item nesta lista é muito importante e é essencial para o funcionamento correto de sistemas desta natureza, já que as tarefas de medição de grandezas elétricas que irão rodar na BeagleBoard precisam executar suas tarefas na CPU em intervalos de tempo bem definidos e precisos. Assim, o escalonador para o sistema deste trabalho precisa levar em consideração este fato, conhecendo com que frequência estas tarefas devem executar, suas durações e quais os prazos de término associados.

3.2 Tempo real em sistemas Linux

Linux é um sistema operacional de propósito geral, cuja aplicação foi estendida aos poucos para o domínio de sistemas embarcados. Assim, a necessidade de torná-lo um RTOS foi aparecendo, visto que originalmente ele não oferecia suporte algum para sistemas estritos de tempo real. Algumas características do Linux o classificam inicialmente como um sistema sem suporte a recursos de um sistema de tempo real (Raghavan et al., 2006):

- Alta latência devida às interrupções;

- Alta latência devida ao escalonador (o *kernel* não é preemptivo);

- Presença de serviços do sistema operacional com comportamento não-determinístico (memória virtual, alocação dinâmica de memória, entre outros).

A maior contribuição para o indeterminismo do tempo de resposta de uma tarefa em Linux é a latência devida ao escalonador. Como o *kernel* não é preemptivo, processos com prioridade mais alta que não estejam rodando não podem ocupar o lugar de um processo de menor prioridade caso o mesmo esteja no meio de uma execução de uma chamada de sistema, por exemplo. Além do tempo de espera para poder colocar tarefas de mais alta prioridade na CPU, o escalonador do Linux leva um certo tempo para decidir qual o processo que ganha a CPU no momento de efetuar a troca de contexto, e este tempo é proporcional ao número de processos disputando a CPU (o cálculo é efetuado em $O(n)$).

3.2.1 *Real-time kernel patch*

Para adicionar capacidades de um RTOS ao *kernel* do Linux, é necessário fazer a instalação de um *patch* de suporte a características de tempo real. O *real-time kernel patch*, conforme proposto por Ingo Molnár em (RTKernel, 2012), adiciona ao *kernel* do Linux um modo de preempção chamado `PREEMPT_RT`, que permite a preempção de uma tarefa em execução em prol de uma outra tarefa mais urgente a partir de qualquer lugar do código (exceto em regiões realmente críticas de código (*spinlock-protected*), onde a preempção precisa ser desabilitada explicitamente. Modelos antigos de preempção utilizavam uma abordagem mais simples: a preempção era ativada em pontos estratégicos do *kernel*, onde a segurança do sistema em relação a condições de corrida e consistência de dados era garantida (Raghavan et al., 2006).

A Figura 3.1 exibe uma comparação entre uma CPU que não emprega preempção e uma CPU que possui preempção de tempo real. As duas CPUs possuem a tarefa B em execução, até que um evento externo indica que a tarefa A, de prioridade mais alta, está pronta. Com o modo `PREEMPT_RT` ativado, a tarefa A ganha a CPU logo após a tarefa B concluir a sua seção crítica, onde esta não pode ser preemptada.

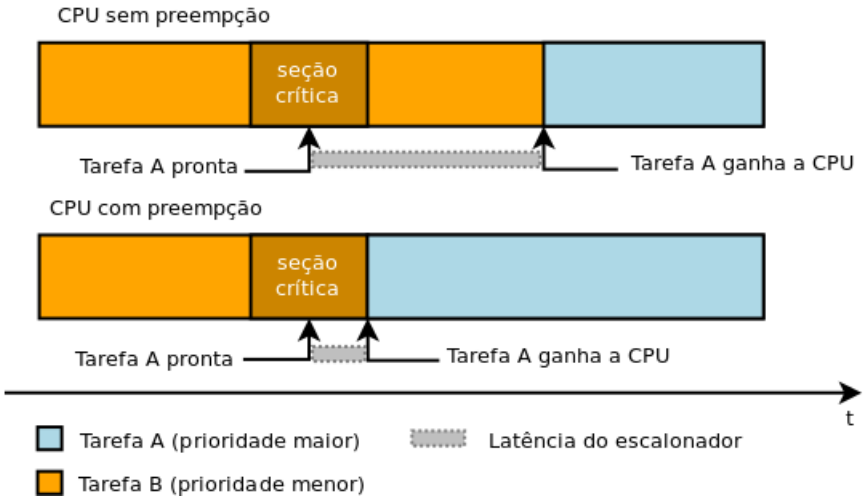


Figura 3.1: Latência do escalonador em CPU sem preempção e com preempção.

Além de implementar este novo modo de preempção, o algoritmo do escalonador utilizado pelo *patch* ajuda a reduzir ainda mais a latência do sistema. Trata-se de um escalonador que decide a próxima tarefa a ocupar a CPU em $O(1)$, ou seja, em um tempo que independe do número de processos disputando o processador.

As funções do arquivo de cabeçalho `sched.h` também agregam algumas funcionalidades úteis para sistemas de tempo real. Elas permitem escolher entre duas políticas de funcionamento do escalonador (através de `sched_setscheduler`):

- **FIFO**: nesta política, as tarefas usam a CPU indeterminadamente, a não ser que a liberem de forma voluntária, sejam preemptadas por outra tarefa com prioridade mais alta, ou tornem-se bloqueadas devido a uma requisição de entrada ou saída. Tarefas mais novas entram "no fim da fila", e aguardam sua vez caso não possam preemptar as tarefas a sua frente.
- **Round Robin**: Similar ao FIFO, exceto que uma tarefa pode ser preemptada caso seu tempo de execução exceda uma certa fatia de tempo pré-definida.

A prioridade de uma tarefa pode ser definida através da função `sched_setparam`. Valores mais altos correspondem a tarefas mais urgentes, do ponto de vista das tarefas (do ponto de vista do *kernel*, tarefas mais prioritárias possuem um valor de prioridade mais baixo).

3.2.2 Sincronização e comunicação entre tarefas

O suporte à comunicação entre tarefas foi implementado com as extensões de tempo real do padrão POSIX 1003.1b para Linux. Este padrão define **filas de mensagens**, um método determinístico e eficiente de comunicação entre processos. As mensagens entre processos passam a ter prioridades definidas, e a fila das mensagens recebidas por um processo é ordenada de acordo com esta prioridade. Além disso, o comportamento determinístico da troca de mensagens é assegurado pela possibilidade de se definir um mecanismo de *timeout* na troca de mensagens.

A comunicação e compartilhamento de dados entre tarefas do sistema operacional incorpora questões de sincronização e consistência dos dados. O uso de **semáforos** para proteção de regiões de código que manipulam dados trocados por mensagens foi efetuado, utilizando a definição de semáforos das POSIX threads (biblioteca *pthread*).

3.3 Aplicação na arquitetura

As características de um RTOS e os detalhes de uma implementação de tempo real em Linux foram utilizadas na implementação de alguns componentes da arquitetura deste trabalho. Conforme descrito no início do capítulo, o processo físico de medir a tensão e a corrente em uma entrada da placa exige, do lado do software que converte o sinal e efetua cálculos sobre uma versão discretizada deste, funcionalidades de um sistema de tempo real crítico. Isto é necessário, uma vez que equipamentos de multimedição são homologados e validados por concessionárias de energia que garantem que os critérios de medição são respeitados. Este aspecto da amostragem do sinal é discutido no próximo capítulo.

Por outro lado, a criação do conceito de tarefas no sistema, de prioridades entre tarefas e de otimizações no escalonador precisam ser utilizados pelas aplicações rodando no sistema operacional, para que o aspecto *real time* do sistema possa ser percebido. A implementação destes aspectos é apresentada junto às aplicações desenvolvidas sobre esta plataforma, no Capítulo 6.

4 AMOSTRAGEM DE TENSÃO E CORRENTE

Neste capítulo, será abordada a teoria da amostragem e sua aplicação à medição de tensão e corrente em equipamentos multimedidores. Em seguida, o mecanismo de medição de tensão e corrente que utiliza a placa de áudio da BeagleBoard é descrito, utilizando os conceitos de tempo real descritos no Capítulo 3.

4.1 Amostragem digital de sinais analógicos

Na concepção de um sistema de medição e análise de qualidade de energia, é preciso encontrar uma solução de processamento sobre sinais analógicos de tensão e corrente, que são contínuos sobre o tempo. Nestes equipamentos é necessário efetuar cálculos sobre os sinais de tensão e corrente vindos da rede elétrica, e estes cálculos são realizados em componentes digitais, como um microprocessador.

Assim, estes sinais devem ser convertidos de sua representação analógica (contínua) para uma representação digital (discreta), sem que haja perda significativa de informação nesta conversão. O componente eletrônico que realiza esta função é o **conversor analógico-digital**, ou simplesmente A/D. Em sua entrada, temos um sinal analógico, e em sua saída, temos uma representação discretizada no tempo e quantizada na amplitude deste sinal (a Figura 4.1 exibe um exemplo desta digitalização: em azul, o sinal analógico, e em vermelho a sua representação digital).

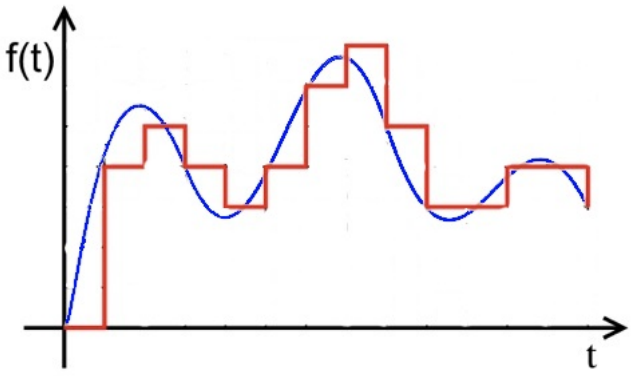


Figura 4.1: Um sinal analógico e sua representação digital.

A discretização de um sinal analógico na amplitude (**quantização**) está ligada à resolução de saída de um A/D: quanto mais bits de saída existirem, mais valores possíveis da grandeza quantizada serão possíveis de serem representados, logo, o valor associado à mudança do bit menos significativo na saída será menor (havendo portanto uma melhor

resolução no sinal digital, e uma representação mais fiel do valor real).

A discretização do sinal em relação ao tempo está diretamente associada à **frequência de amostragem** f_s do conversor, e corresponde ao número de vezes por segundo que um sinal analógico é amostrado. Esta operação, conhecida como **amostragem**, transforma um sinal analógico em uma série de valores discretos que formam um **sinal amostrado** com período $T_s = \frac{1}{f_s}$.

Dependendo da natureza do sinal de entrada, a escolha de f_s influencia na capacidade de reconstrução do sinal analógico a partir das amostras coletadas: um valor baixo de f_s pode não ser capaz de amostrar as frequências mais altas que compõem um sinal analógico. Esta reconstrução é necessária, por exemplo, para reproduzir a forma de onda em um ciclo de tensão a partir de amostras daquele ciclo que foram gravadas no equipamento (oscilografia) ou ainda para realizar os cálculos de flutuação de tensão (seção 1.2). O **Teorema da amostragem de Nyquist-Shannon** aborda esta questão.

4.1.1 Teorema da amostragem de Nyquist-Shannon

O teorema da amostragem de Nyquist-Shannon enuncia:

Teorema 1. (Teorema da amostragem de Nyquist-Shannon)
É possível reconstruir um sinal contínuo $x_C(t)$ no domínio do tempo a partir do sinal discreto $x_D(n)$ de período T_s se:

- *a frequência de amostragem $f_s = \frac{1}{T_s}$ do sinal contínuo $x_C(t)$ verifica*

$$\nu_M < \frac{f_s}{2} \tag{4.1}$$

  onde ν_M é a maior frequência que compõe o sinal $x_C(t)$ e,

- *a transformada de Fourier de $x_C(t)$, $\chi_C(\nu)$, é nula para qualquer valor de ν tal que $|\nu| > \nu_M$.*

Uma vez que a amostragem do sinal contínuo satisfaz as condições do teorema, o sinal analógico $x_C(t)$ pode ser reconstruído através da **fórmula de reconstrução de Shannon**:

$$x_C(t) = \sum_{n=-\infty}^{+\infty} x_D(n) sinc\left[\pi \frac{t - nT_s}{T_s}\right] \tag{4.2}$$

A demonstração do teorema e a obtenção da expressão de reconstrução do sinal podem ser consultadas em (Max et al., 1996).

4.1.2 Amostragem de medição

Um sinal analógico de tensão alternada é modelado idealmente como uma senóide cujo RMS mínimo e máximo e frequência são determinados através de normas que diferem de país para país (Nilsson et al., 2003). No Brasil, a frequência da rede elétrica (frequência nominal) é fixada em 60 Hz, configuração encontrada em boa parte da América; no resto do mundo, a frequência padrão é normalmente igual a 50 Hz.

De acordo com a norma IEC 61000-4-30, a amostragem de um sinal de medição de tensão ou corrente deve conter no mínimo 128 amostras por período (ou ciclo). Assim, a frequência de amostragem f_s que deve ser utilizada pelo circuito de medição de um equipamento de medição que opere no Brasil precisa ser igual a no mínimo $f_s = 128 \times$

$60Hz = 7680Hz$. Para o caso de equipamentos que operem a 50 Hz, este valor cai para $f_s = 6400Hz$. Neste trabalho, foi considerada apenas a primeira situação para a amostragem dos sinais.

Uma frequência de amostragem de 7680 Hz permite reconstruir, de acordo com a equação 4.2, sinais analógicos compostos por frequências de até 3840 Hz. Um sinal de tensão ou de corrente que contenha componentes de frequência desta ordem apresenta harmônicas (conforme seção 1.2) de até 64ª ordem, ou seja, uma amostragem de tal sinal permite detectar distorções harmônicas na rede até a 64ª ordem.

Em relação a quantização do sinal analógico, o conversor A/D da BeagleBoard possui uma resolução na saída de 16 bits, ou seja, capaz de expressar 65536 níveis distintos de tensão ou corrente de entrada. A tensão máxima suportada pelo circuito de amostragem (ver seção 4.2.1) é de 300 V_{RMS} (ou seja, até 850 V_{pp}), resultando em um erro máximo de 0,013 V devido à quantização do sinal.

4.2 Circuito de amostragem

A medição de tensão e corrente é realizada através de uma série de passos, que seguem o esquema da Figura 4.2.

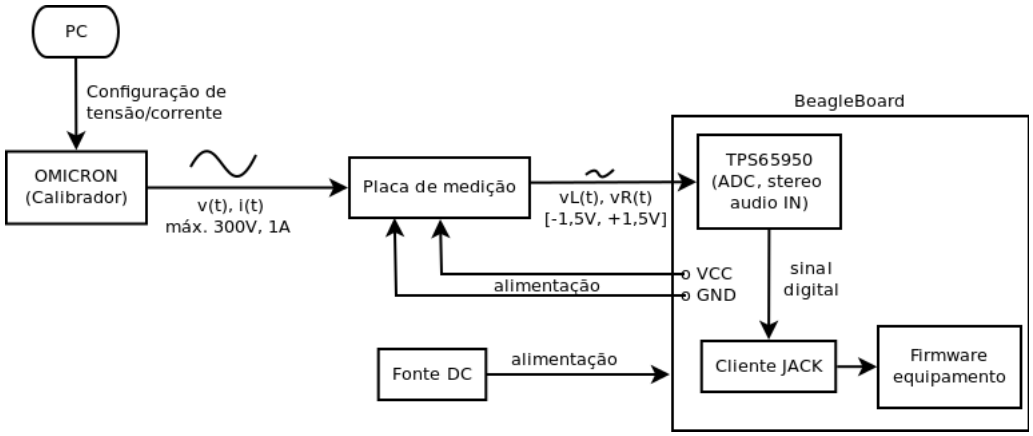


Figura 4.2: Esquemático de medição de corrente e tensão.

Os sinais de tensão a ser medidos podem ser originados diretamente da rede elétrica com uma carga associada, para que haja uma corrente válida, ou a partir de um equipamento de calibração, como o Omicron CMC 256 (Omicron, 2012), cujos sinais gerados podem ser usados como referência para avaliação da precisão dos valores de tensão e corrente medidos. Como estas tensões e correntes podem ser da ordem de centenas de volts e de algumas unidades de amperes, é necessário usar um circuito (placa de medição) que reduza essas grandezas para a ordem de poucos volts, de modo que o conversor A/D não seja danificado. Neste trabalho, os sinais de tensão e corrente foram adaptados para excursionar entre $-1,5V$ e $+1,5V$ (detalhes da placa de medição nas seções 4.2.1 e 4.2.2).

Do lado da BeagleBoard, estes sinais são coletados pela entrada de áudio padrão da placa. Como se trata de um canal estéreo, é possível transmitir pelo plugue do cabo P2 de áudio as informações de tensão e corrente mais a linha cujo potencial é usado como referência (*ground*), conforme a Figura 4.3.

A BeagleBoard é equipada de um TPS85950, um circuito integrado codificador/decodificador (*codec*) de áudio (TPS65950, 2012). Este circuito possui um A/D de 16 bits

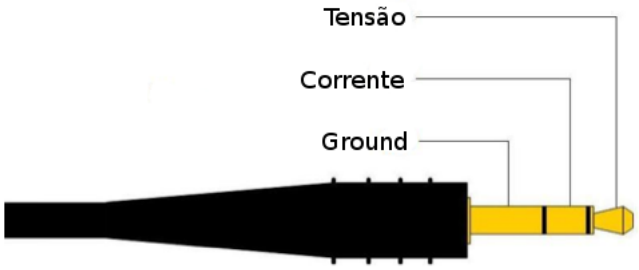


Figura 4.3: Plugue P2 utilizado para transmissão da informação de tensão e corrente.

para conversão de áudio estéreo. Esta interface é usada para fazer a conversão do sinal analógico gerado pela placa de medição para o sinal amostrado que será necessário para o cálculo de tensão e corrente instantâneos da rede.

O suporte à entrada de áudio é oferecido pela ALSA (*Advanced Linux Sound Architecture*), uma arquitetura que disponibiliza drivers a diversas interfaces de áudio para sistemas Linux. A partir desta arquitetura, é possível amostrar o sinal de áudio nas frequências comuns de trabalho com este tipo de sinal (44,1 kHz, 48kHz e alguns múltiplos destas frequências). No caso deste trabalho, a frequência de amostragem escolhida é de 8kHz, suficiente para a obtenção de 133,3 amostras por ciclo (considerando uma frequência nominal de $60Hz$).

No sistema operacional é criado um servidor de áudio JACK (JACK, 2012), que permite a interação com as drivers de audio da ALSA, através da criação de clientes capazes de obter pacotes de áudio convertidos pelo A/D com baixa latência e com suporte a processamento de tempo real (Silva, 2011). Um cliente do servidor JACK é iniciado, e alimenta a aplicação que representa o *firmware* do equipamento com os dados de tensão e corrente processados (mais sobre o servidor JACK na seção 4.2.3).

Finalmente, a alimentação da BeagleBoard é feita por uma fonte que fornece 5 V_{DC} na saída, e até 1 A de carga. Esta fonte é usada para alimentar a BeagleBoard e também o circuito de medição, que é alimentado igualmente a 5 V_{DC}.

4.2.1 Medição de tensão

A placa de medição possui um circuito para gerar os sinais de entrada do conversor A/D para medição de tensão. O propósito deste circuito é converter um sinal de corrente alternada cujo valor RMS atinge algumas centenas de volts (neste caso, até 300 V_{RMS}), em um sinal que tenha a mesma forma e frequência do sinal de entrada, mas cuja amplitude não exceda 1,5 V.

O diagrama funcional deste circuito é similar ao esquemático da Figura 4.4.

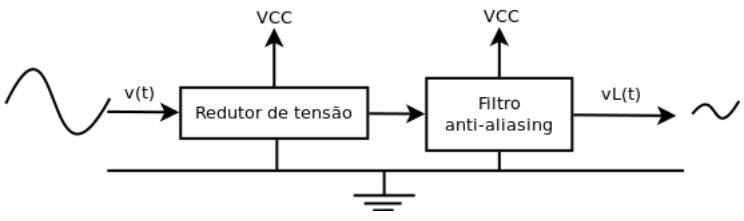


Figura 4.4: Diagrama de blocos do circuito de medição de tensão.

Neste diagrama, identificamos dois blocos fundamentais neste circuito: o primeiro bloco corresponde a um **redutor de tensão** (basicamente, um resistor de alta impedância

na entrada acoplado a um amplificador de alto ganho que faz o sinal excursionar entre os valores de tensão desejados) e um **filtro *anti-aliasing*** que remove as frequências altas inseridas pela conversão que interferem na medição (Sedra et al., 2007).

4.2.2 Medição de corrente

O circuito de medição de corrente, também presente na placa de medição, consiste em um circuito muito similar ao circuito de medição de tensão. No entanto, ele adiciona um bloco de **isolamento** e um bloco de **transdução** de corrente (ver Figura 4.5).

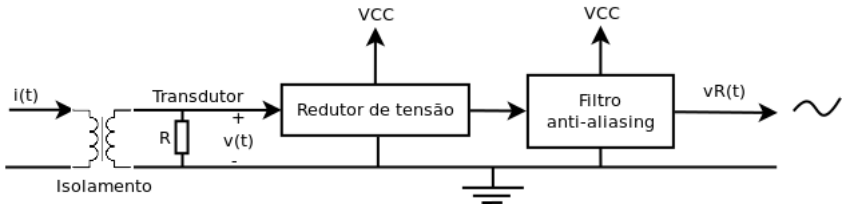


Figura 4.5: Diagrama de blocos do circuito de medição de corrente.

O bloco de isolamento é constituído por um transformador de corrente (TC) que reduz a corrente de entrada para uma corrente com menor intensidade. O circuito de transdução pode ser implementado como um simples resistor, transmitindo a tensão existente sobre o resistor quando este é percorrido pela corrente de saída do TC para o resto do circuito, cuja estrutura é idêntica ao circuito da Figura 4.4.

4.2.3 Processamento das amostras em tempo real

Os sinais vindos da placa de medição chegam à BeagleBoard através da entrada de áudio da placa, conforme a Figura 4.2. A amostragem deste sinal analógico é realizada pelo servidor do JACK, para que então os valores de tensão e corrente possam ser usados pelas aplicações embarcadas no sistema operacional.

Para a utilização do servidor JACK, e para que haja a possibilidade de instalar clientes JACK no sistema, foi necessário adicionar na construção do sistema operacional os pacotes do servidor do JACK, como descrito na seção 2.4. A versão do *daemon* do JACK que atua como servidor de áudio é a 0.118.0.

O servidor JACK é inicializado na placa com o seguinte comando:

```
jackd -R -d alsa -r 8000 -p 128 -S -s
```

Esta linha parametriza o servidor JACK para iniciar no modo tempo real (ou seja, com baixa latência), para realizar 128 amostras por captura do sinal a uma frequência de amostragem de 8000 Hz, a menor frequência possível para o JACK, mas ainda maior que a frequência mínima necessária definida na seção 4.1.2. Assim, cada *buffer* de captura cobre 16 ms em 128 amostras de tensão em um dos canais de áudio e 128 amostras de corrente no segundo canal de áudio.

Um cliente JACK consiste em um programa que se conecta a um ou mais canais de áudio acessíveis através do driver escolhido na inicialização do servidor JACK (neste caso, a ALSA). O cliente define uma função de *callback* que é invocada cada vez que um pacote de amostras for recebido. Estes dados são enviados a aplicação principal, que irá efetuar os cálculos de frequência, potência e energia, além de determinar a corrente e a tensão lidas na fase conectada ao circuito.

4.3 Cálculo da frequência, potência e energia

A determinação da **frequência** do sinal durante a medição é cálculada a partir da análise de *zero-crossing* do sinal de tensão amostrado (Mog et al., 2004). Como a frequência de amostragem f_s do sinal de tensão é conhecida, é feita uma análise sobre a série de valores amostrados e é contado o número de amostras que se encontram entre os extremos de três passagens por zero (a Figura 4.6 apresenta um exemplo de interpolação linear para determinação do *zero-crossing* do sinal). Como normalmente os pontos de cruzamento com zero não serão exatamente zero, devido a quantização do sinal, é feita uma interpolação linear entre duas amostras para que tenhamos um número não-inteiro de amostras n. O valor da frequência da rede f_i passa a ser simplesmente $f_i = n \times f_s$.

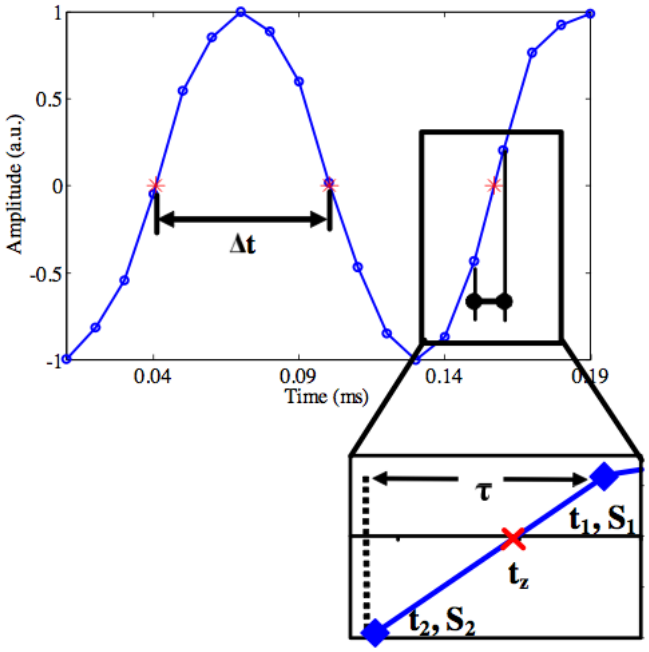


Figura 4.6: Utilizando *zero-crossing* para determinação do período de um sinal discreto (Xu et al., 2008).

O cálculo dos valores RMS da tensão e da corrente é necessário para determinar a potência aparente do sistema. Como este trabalho considera um sistema monofásico, o valor RMS da tensão V_{RMS} e da corrente I_{RMS} em um ciclo pode ser obtido através das expressões 4.3 e 4.4 (onde N é o número de amostras em um ciclo de medição).

$$V_{RMS} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} v_n^2} \tag{4.3}$$

$$I_{RMS} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} i_n^2} \tag{4.4}$$

O valor da **potência aparente** em um ciclo S é obtida pela seguinte expressão:

$$S = V_{RMS} \cdot I_{RMS} \tag{4.5}$$

A **potência ativa** instantânea P é o valor calculado a partir do produto interno do vetor de amostras de tensão $\overrightarrow{v} = \langle v_1, ..., v_n \rangle$ e de corrente $\overrightarrow{i} = \langle i_1, ..., i_n \rangle$ de um ciclo.

$$P = \frac{1}{N} \sum_{n=1}^{N} v_n \cdot i_n \tag{4.6}$$

A **potência reativa** do sistema monofásico Q é calculada através da expressão 4.7. Como essa expressão sempre retorna o valor positivo da raiz quadrada, a determinação do sinal é feita através da análise do sinal do produto interno $\overrightarrow{v} \cdot \overrightarrow{i}^T$.

$$Q = \sqrt{S^2 - P^2} \tag{4.7}$$

O cálculo da **energia ativa** E_a e da **energia reativa** E_r, de acordo com a norma IEC 61000-4-30, deve ser atualizado a cada 200 ms (12 ciclos a 60 Hz). Os acumuladores guardam os valores existentes no equipamento e são incrementados com a energia acumulada de um valor ΔE_a (para a energia ativa) e ΔE_r (para a energia reativa), obtidos com a expressão 4.8 e 4.9, respectivamente.

$$\Delta E_a = \frac{1}{12} \sum_{n=1}^{12} P_n \times \Delta t_{12ciclos} \qquad @60Hz \tag{4.8}$$

$$\Delta E_r = \frac{1}{12} \sum_{n=1}^{12} Q_n \times \Delta t_{12ciclos} \qquad @60Hz \tag{4.9}$$

Estes cálculos são realizados em aritmética de ponto flutuante na aplicação do equipamento, e são efetuados a cada ciclo completo do sinal de tensão (a exceção da energia, que é aplicada sobre uma média dos valores de potência de 12 em 12 ciclos).

4.4 Conclusão

A avaliação de valores de tensão e corrente instantâneas no equipamento, e o cálculo das demais grandezas derivadas destas, encerra uma importante parte da arquitetura. Com estes valores disponíveis, é possível partir para a implementação dos drivers de periféricos da arquitetura, no Capítulo 5, para depois unir estas duas partes no Capítulo 6, que trata das aplicações que rodam no sistema embarcado na BeagleBoard.

5 PERIFÉRICOS

5.1 Módulos de kernel dinâmicos

Drivers para controladores de dispositivos são módulos do *kernel* de um sistema operacional responsáveis por receber os comandos enviados por um dispositivo e enviar ao sistema operacional as respostas apropriadas de acordo com a operação desejada sobre o mesmo, e vice-versa (Tanenbaum, 2010). Para serem utilizados, os drivers precisam ser acoplados de alguma maneira ao sistema operacional, para que possam ser executados em modo núcleo e ter acesso ao hardware.

Na literatura, é possível encontrar três abordagens para esta operação:

- A primeira consiste na associação de um novo driver ao *kernel* do sistema operacional, seguida da reinicialização do sistema para que o driver seja carregado. O Linux funciona em boa parte desta maneira, visto que o código do *kernel* é aberto, e existe a possibilidade de escrever rotinas de software para novos dispositivos diretamente incorporadas ao núcleo do sistema.

- A segunda maneira, utilizada com frequência no Windows, consiste em manter um arquivo com um registro dos drivers necessários para o funcionamento dos periféricos, sendo que o sistema operacional se ocupa de carregar os drivers listados neste arquivo sempre que a máquina é inicializada.

- A terceira abordagem, utilizada neste trabalho, consiste em carregar dinamicamente os drivers sem que haja necessidade de reiniciar todo o sistema. Dispositivos USB utilizam esta abordagem atualmente. No Linux, é possível trabalhar com módulos de kernel dinâmicos, que são carregados e descarregados no kernel conforme a necessidade da aplicação (Salzman et al., 2007).

A estrutura de um módulo carregável dinamicamente é bastante simples (Hallinan, 2012) e (Salzman et al., 2007):

```
1  #include <linux/module.h>
2  static int __init module_init(void)
3  {
4      printk(KERN_INFO "Inicializacao dinamica do modulo.");
5      return 0;
6  }
7  static void __exit module_exit(void)
8  {
9      printk(KERN_INFO "Remocao dinamica do modulo.");
10 }
11 module_init(module_init);
12 module_exit(module_exit);
13
```

```
14 MODULE_AUTHOR("Bruno Guedes");
15 MODULE_DESCRIPTION("Exemplo de modulo.");
16 MODULE_LICENSE("GPL");
```

Para este trabalho, o módulo precisa ser compilado utilizando o *gcc-ARM* para que possa ser instalado na BeagleBoard. A construção de um módulo gera um arquivo com a extensão `.ko` (*kernel object*), que corresponde ao código objeto do módulo.

As funções de inicialização e término do módulo podem ser invocadas a qualquer momento através dos comandos `insmod` e `rmmod` sobre o arquivo `.ko`. No contexto deste trabalho, um módulo será usado como driver de um dispositivo de entrada (o teclado), e pode ser inicializado automaticamente no *boot* do sistema.

5.2 *Polling* e interrupções

Drivers de dispositivos podem atuar tanto através de *polling* ou através de interrupções no que se refere à comunicação com os controladores dos dispositivos de entrada e saída.

A estratégia de *polling* na construção de um driver consiste, após a inicialização do dispositivo, na varredura de um laço infinito que executa testes que verificam a ocorrência de todos os eventos possíveis para aquele dispositivo, um a um, e a execução de rotinas associadas a estes eventos quando o teste for positivo.

A estratégia de interrupções, por outro lado, para o atual fluxo de execução quando um evento é disparado pela controladora do dispositivo e redireciona o fluxo de execução para uma rotina de serviço de interrupção (ISR). Em relação a hardware de entrada e saída, estes eventos são normalmente assíncronos, pois ocorrem externamente ao processador e não são resultado direto da execução de uma instrução. As interrupções que geram estes eventos precisam ser registradas no *kernel*, de modo a associar a um índice do vetor de interrupções do sistema operacional um endereço para a ISR correspondente à interrupção registrada.

Neste trabalho, as duas estratégias são implementadas: no teclado (seção 5.3), a opção de *polling* foi utilizada para detectar eventos de teclas pressionadas, embora as funções disparadas por estes eventos gerem interrupções de software (*traps*). Na seção 6.3, um exemplo de interrupções de hardware é descrito, relativo a comunicação via protocolo Modbus utilizando a interface Ethernet da BeagleBoard.

5.3 Teclado

O teclado que foi utilizado nesta implementação é o mesmo do multimedidor M-500 (Figura 5.1). Trata-se de um teclado matricial de 6 botões dispostos em uma matriz de 2 x 3, com 5 pinos de entrada e saída, três deles correspondentes às colunas e dois correspondentes às linhas.

O teclado é o principal periférico de IHM de um equipamento. Através dele é possível navegar em menus de medições e de executar a programação das diversas funcionalidades do equipamento.

Para este trabalho foram utilizados cinco pinos do banco de GPIOs disponibilizados na BeagleBoard. A BeagleBoard possui 28 pinos de expansão que podem ser usados como pinos de entrada e saída de propósito geral, McBSP, MMC, I2C, entre outros. Como estes pinos podem ser usados para diversos modos, existe um sinal de controle na placa que atua sobre um multiplexador para cada pino de modo a selecionar qual a sua função no banco de expansão. Esta configuração fica gravada no sistema de inicialização da

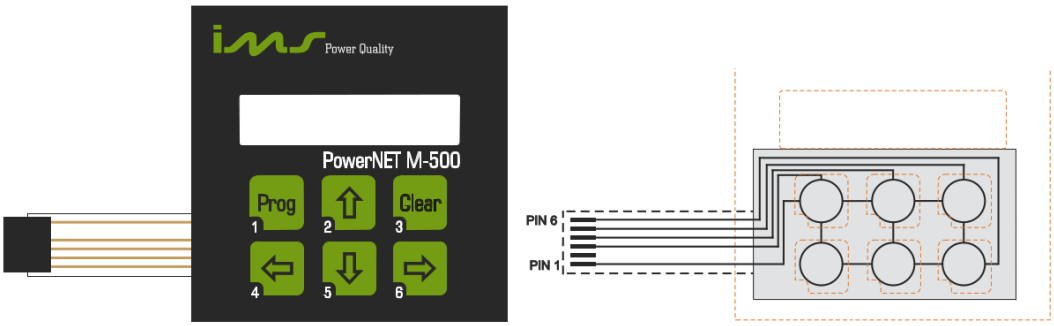


Figura 5.1: Teclado do M-500, usado neste trabalho e o circuito associado a ele.

placa (*bootloader*). A Figura 5.2 mostra as 8 possibilidades de controle oferecidas para cada pino do *slot* de expansão da BeagleBoard ('**X**' representando ausência de sinal, '**Z**' significando um estado de alta impedância e '*****' representando um sinal existente, mas sem utilidade). Os pinos utilizados nesta implementação do teclado foram os pinos 12, 16, 18, 20 e 22 (todos no modo GPIO).

EXP	Processor	0	1	2	3	4	5	6	7
1		VIO_1V8							
2		DC_5V							
3	AE3	MMC2_DAT7	*	*	*	GPIO_139	*	*	Z
4	AB26	UART2_CTS	McBSP3_DX	GPT9_PWMEVT	X	GPIO_144	X	X	Z
5	AF3	MMC2_DAT6	*	*	*	GPIO_138	*	X	Z
6	AA25	UART2_TX	McBSP3_CLKX	GPT11_PWMEVT	X	GPIO_146	X	X	Z
7	AH3	MMC2_DAT5	*	*	*	GPIO_137	*	X	Z
8	AE5	McBSP3_FSX	UART2_RX	X	X	GPIO_143	*	X	Z
9	AE4	MMC2_DAT4	*	X	*	GPIO_136	X	X	Z
10	AB25	UART2_RTS	McBSP3_DR	GPT10_PWMEVT	X	GPIO_145	X	X	Z
11	AF4	MMC2_DAT3	McSPI3_CS0	X	X	GPIO_135	X	X	Z
12	V21	McBSP1_DX	McSPI4_SIMO	McBSP3_DX	X	GPIO_158	X	X	Z
13	AG4	MMC2_DAT2	McSPI3_CS1	X	X	GPIO_134	X	X	Z
14	W21	McBSP1_CLKX	X	McBSP3_CLKX	X	GPIO_162	X	X	Z
15	AH4	MMC2_DAT1	X	X	X	GPIO_133	X	X	Z
16	K26	McBSP1_FSX	McSPI4_CS0	McBSP3_FSX	x	GPIO_161	X	X	Z
17	AH5	MMC2_DAT0	McSPI3_SOMI	X	X	GPIO_132	X	X	Z
18	U21	McBSP1_DR	McSPI4_SOMI	McBSP3_DR	X	GPIO_159	X	X	Z
19	AG5	MMC2_CMD	McSPI3_SIMO	X	X	GPIO_131	X	X	Z
20	Y21	McBSP1_CLKR	McSPI4_CLK	X	X	GPIO_156	X	X	Z
21	AE2	MMC2_CLKO	McSPI3_CLK	X	X	GPIO_130	X	X	Z
22	AA21	McBSP1_FSR	X	*	Z	GPIO_157	X	X	Z
23	AE15	I2C2_SDA	X	X	X	GPIO_183	X	X	Z
24	AF15	I2C2_SCL	X	X	X	GPIO_168	X	X	Z
25	25	REGEN							
26	26	Nreset							
27	27	GND							
28	28	GND							

Figura 5.2: Possibilidades de uso para os 28 pinos de expansão da placa (Coley, 2010).

Fisicamente, o teclado é composto por seis conectores que colocam em curto-circuito uma linha e uma coluna quando a tecla correspondente é pressionada. Como este teclado é um simples circuito sem um controlador de hardware, a verificação do acionamento de uma tecla é feito por *polling*. Assim, um sinal lógico de nível alto é inserido em cada uma das três colunas, alternadamente, enquanto verifica-se o estado das duas linhas do teclado: se um sinal lógico alto é detectado em uma das linhas, houve um curto-circuito

e a tecla foi pressionada e deve ser tratada pelo *kernel*. Esta leitura e escrita de sinais nos pinos de extensão é feita através das funções disponíveis em `linux/gpio.h`.

As teclas acionadas são mapeadas para teclas da interface de *input* padrão do Linux, `linux/input.h`. Um novo dispositivo de entrada é cadastrado no sistema e uma *kernel thread* de varredura é associada ao dispositivo. Ela é responsável por reportar ao sistema operacional que uma tecla foi pressionada e por gerar as interrupções necessárias para tratar este evento.

A inicialização do módulo de teclado é feita automaticamente após o *boot* do sistema. Ela configura os pinos de GPIO usados na BeagleBoard para cada uma das linhas e colunas existentes no barramento do teclado.

5.4 Vídeo

O processador equipado na BeagleBoard possui suporte a interface de vídeo DVI-D com suporte a saída de vídeo com 24 bits de cores RGB, com um conector HDMI do lado da placa. Esta interface é compatível com monitores que suportam o padrão DVI ou o HDMI (neste último caso, apenas o vídeo é transmitido, visto que a implementação na placa não suporta o padrão HDMI completo, que também inclui áudio). Como a distribuição Linux montada através do BitBake não possui o pacote gráfico "X", um monitor conectado a placa exibe a interface de console para o usuário.

Ainda assim, é possível desenvolver aplicações com uma IHM mais avançada mesmo com a interface gráfica limitada ao console, utilizando a biblioteca ***ncurses*** (ncurses, 2012), que cria elementos de interface similares àqueles de ambientes gráficos de UI, porém em modo texto. Por exemplo, neste trabalho a interface de UI de um equipamento existente foi recriada com esta biblioteca para reproduzir as telas existentes no *firmware* do mesmo na saída padrão da BeagleBoard. Na Figura 5.3, três exemplos de telas são exibidos: (a) a tela de seleção de opções principal, (b) a tela de resumo de medições, com tensão, corrente e frequência e (c) a tela de informações do equipamento, com modelo do equipamento, versão do *firmware* e número de série.

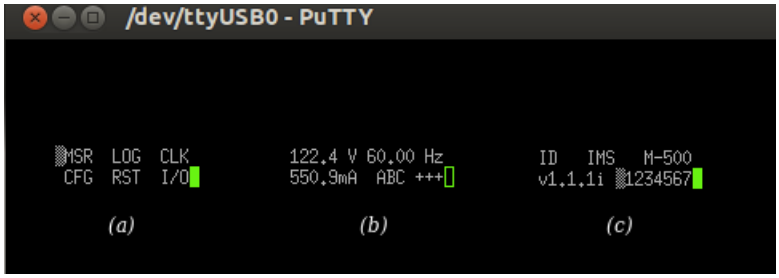


Figura 5.3: Visualização no terminal de telas desenhadas com a biblioteca *ncurses*.

6 APLICAÇÕES

De modo a validar a arquitetura desenvolvida neste trabalho, algumas aplicações foram escritas visando simular as aplicações existentes nos equipamentos atuais. Conforme citado na seção 1.1, o equipamento escolhido para tal foi o PowerNET M-500. Nesta seção, será descrita a camada de abstração do sistema operacional escrita para interagir com o *firmware* do equipamento, e duas aplicações fundamentais em multimedidores: o log em memória (registros das grandezas em arquivos) e a comunicação via protocolo Modbus (interface de troca de informações com softwares de análise de qualidade de energia).

6.1 Camada de abstração do sistema operacional

Uma camada de abstração do sistema operacional (OSAL, do inglês *Operating System Abstraction Layer*) é um artifício usado para encapsular rotinas e serviços comuns a diversos sistemas operacionais de modo a simplificar o desenvolvimento de software quando a portabilidade da aplicação é importante. No caso deste trabalho, a OSAL contém funções que ocultam os detalhes de implementação que variam em sistemas operacionais diferentes ligadas à gerência de tarefas e ao mecanismo de troca de mensagens (*message queues*). Como no *firmware* do M-500, estas funções utilizam diretamente a implementação do sistema operacional embarcado da Keil (Keil, 2012), que é utilizado nestes equipamentos, foi necessário desenvolver a OSAL e modificar o firmware para que ele passasse a usar as funções generalizadas da camada de abstração ao invés das rotinas específicas do sistema operacional embarcado antigo. Finalmente, na implementação Linux da OSAL, foi utilizada a biblioteca *pthread* para assegurar a sincronização das mensagens e a gerência de prioridades entre tarefas. A Figura 6.1 descreve os componentes do sistema que dão suporte às aplicações comuns do equipamento.

Assim, foi possível manter o resto do *firmware* do M-500 e utilizar as aplicações deste diretamente no projeto. O *firmware* é o programa que roda permanentemente no sistema operacional do equipamento. É essencialmente um projeto de *software*, embora o termo *firmware* é utilizado para diferenciá-lo de *software* escrito em linguagens de alto nível. No contexto da arquitetura que está sendo desenvolvida neste trabalho, o firmware interage diretamente com a OSAL, criando e gerenciando as principais tarefas de um equipamento multimedidor, como processar as medições efetuadas, controlar o que é exibido no *display*, gerenciar o relógio, etc. As aplicações são construídas diretamente sobre o *firmware*, podendo utilizar recursos deste, como no caso do log em memória, ou ainda estender as capacidades do *firmware* com o acréscimo de novos serviços, como por exemplo a comunicação por Ethernet.

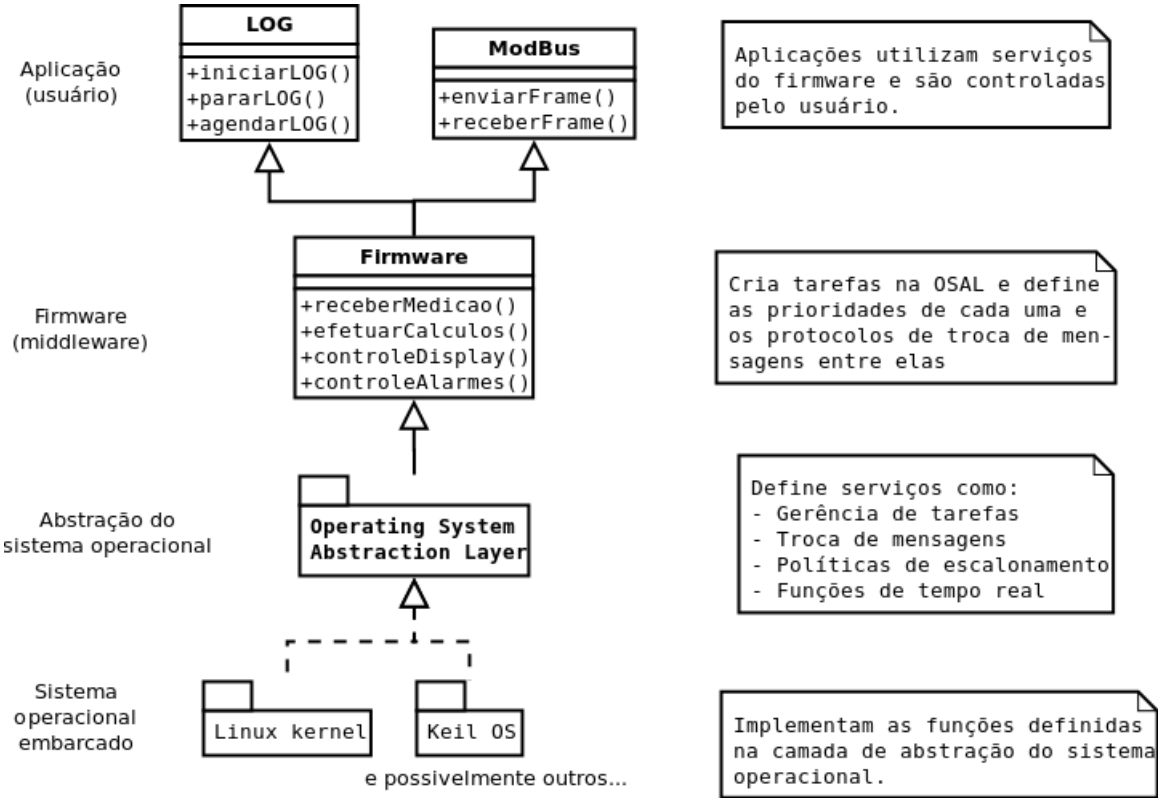


Figura 6.1: Localização da OSAL na hierarquia do sistema.

6.2 Registro de grandezas

A aplicação principal neste trabalho que roda sobre a BeagleBoard é o registro em memória de massa das grandezas medidas pelo equipamento. Este registro é principalmente utilizado para a visualização, por programas analisadores de energia (Figura 6.2), de tabelas e gráficos que sintetizam as informações do que ocorreu com a rede na qual um equipamento esteve ligado por um determinado tempo.

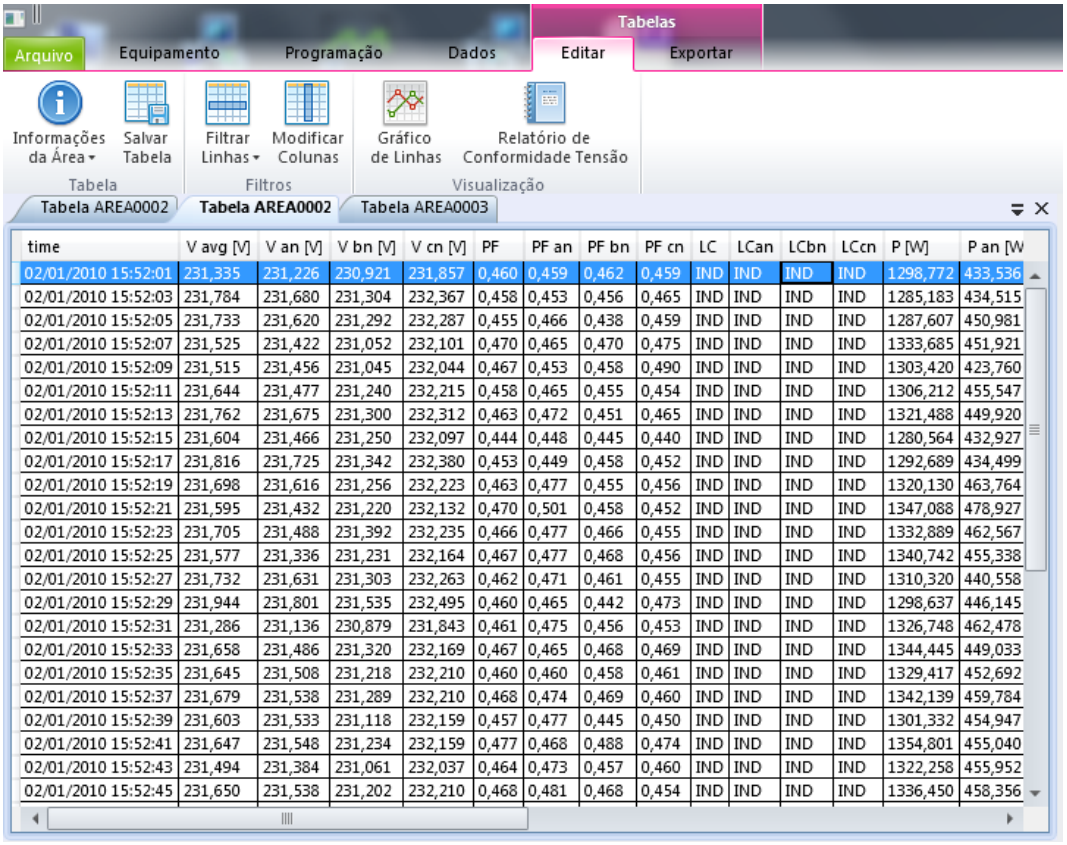
O registro das grandezas pode ser configurado, pelo teclado conectado à placa, em relação ao tempo de gravação de registros ou ao número máximo de entradas registradas, com um certo intervalo de tempo entre uma entrada e outra. Estas informações são os parâmetros da seção de gravação do equipamento.

As informações são salvas pelo equipamento em arquivos de *headers*, que trazem informações sobre as grandezas gravadas (como o tipo de dado que elas representam e a unidade a ser considerada) e em arquivos de dados, que contém os valores em si.

Os arquivos podem ser recuperados por softwares que comuniquem com o equipamento através da troca de mensagens utilizando o protocolo Modbus, implementado nesta arquitetura e que pode ser facilmente adaptado em linguagens de alto nível para softwares de recuperação destes dados.

6.3 Comunicação via protocolo Modbus/TCP

O protocolo Modbus (Modicon, 1996) é um protocolo de comunicação utilizado em redes industriais. Nesta arquitetura, ele utiliza uma conexão Ethernet (Modbus/TCP) como meio físico para seu funcionamento (na maioria dos equipamentos de multimedição, o meio físico utilizado é uma conexão serial RS-232 ou RS-485, cuja velocidade



time	V avg [V]	V an [V]	V bn [V]	V cn [V]	PF	PF an	PF bn	PF cn	LC	LCan	LCbn	LCcn	P [W]	P an [W
02/01/2010 15:52:01	231,335	231,226	230,921	231,857	0,460	0,459	0,462	0,459	IND	IND	IND	IND	1298,772	433,536
02/01/2010 15:52:03	231,784	231,680	231,304	232,367	0,458	0,453	0,456	0,465	IND	IND	IND	IND	1285,183	434,515
02/01/2010 15:52:05	231,733	231,620	231,292	232,287	0,455	0,466	0,438	0,459	IND	IND	IND	IND	1287,607	450,981
02/01/2010 15:52:07	231,525	231,422	231,052	232,101	0,470	0,465	0,470	0,475	IND	IND	IND	IND	1333,685	451,921
02/01/2010 15:52:09	231,515	231,456	231,045	232,044	0,467	0,453	0,458	0,490	IND	IND	IND	IND	1303,420	423,760
02/01/2010 15:52:11	231,644	231,477	231,240	232,215	0,458	0,465	0,455	0,454	IND	IND	IND	IND	1306,212	455,547
02/01/2010 15:52:13	231,762	231,675	231,300	232,312	0,463	0,472	0,451	0,465	IND	IND	IND	IND	1321,488	449,920
02/01/2010 15:52:15	231,604	231,466	231,250	232,097	0,444	0,448	0,445	0,440	IND	IND	IND	IND	1280,564	432,927
02/01/2010 15:52:17	231,816	231,725	231,342	232,380	0,453	0,449	0,458	0,452	IND	IND	IND	IND	1292,689	434,499
02/01/2010 15:52:19	231,698	231,616	231,256	232,223	0,463	0,477	0,455	0,456	IND	IND	IND	IND	1320,130	463,764
02/01/2010 15:52:21	231,595	231,432	231,220	232,132	0,470	0,501	0,458	0,452	IND	IND	IND	IND	1347,088	478,927
02/01/2010 15:52:23	231,705	231,488	231,392	232,235	0,466	0,477	0,466	0,455	IND	IND	IND	IND	1332,889	462,567
02/01/2010 15:52:25	231,577	231,336	231,231	232,164	0,467	0,477	0,468	0,456	IND	IND	IND	IND	1340,742	455,338
02/01/2010 15:52:27	231,732	231,631	231,303	232,263	0,462	0,471	0,461	0,455	IND	IND	IND	IND	1310,320	440,558
02/01/2010 15:52:29	231,944	231,801	231,535	232,495	0,460	0,465	0,442	0,473	IND	IND	IND	IND	1298,637	446,145
02/01/2010 15:52:31	231,286	231,136	230,879	231,843	0,461	0,475	0,456	0,453	IND	IND	IND	IND	1326,748	462,478
02/01/2010 15:52:33	231,658	231,486	231,320	232,169	0,467	0,465	0,468	0,469	IND	IND	IND	IND	1344,445	449,033
02/01/2010 15:52:35	231,645	231,508	231,218	232,210	0,460	0,460	0,458	0,461	IND	IND	IND	IND	1329,417	452,692
02/01/2010 15:52:37	231,679	231,538	231,289	232,210	0,468	0,474	0,469	0,460	IND	IND	IND	IND	1342,139	459,784
02/01/2010 15:52:39	231,603	231,533	231,118	232,159	0,457	0,477	0,445	0,450	IND	IND	IND	IND	1301,332	454,947
02/01/2010 15:52:41	231,647	231,548	231,234	232,159	0,477	0,468	0,488	0,474	IND	IND	IND	IND	1354,801	455,040
02/01/2010 15:52:43	231,494	231,384	231,061	232,037	0,464	0,473	0,457	0,460	IND	IND	IND	IND	1322,258	455,952
02/01/2010 15:52:45	231,650	231,538	231,202	232,210	0,468	0,481	0,468	0,454	IND	IND	IND	IND	1336,450	458,356

Figura 6.2: Exemplo de visualização de dados no software PowerMANAGER 2.

de transmissão é menor quando comparada ao padrão Ethernet). O protocolo Modbus traz ainda códigos de verificação de erros (CRCs) em cada mensagem.

A comunicação é efetuada através da troca de mensagens entre um mestre (neste caso, o software analisador) e um escravo (o equipamento): o software solicita ao equipamento informações sobre suas medições, por exemplo.

A implementação do escravo consiste em uma rotina que recebe um buffer Modbus do mestre e decide o que fazer com a informação adquirida; o recebimento da mensagem é acusado através de uma interrupção levantada pelo driver Ethernet que roda na placa. A informação recebida é geralmente o pedido de uma medição ou então uma escrita de um valor em um registro de parametrização do equipamento.

A versão do protocolo Modbus/TCP utilizado nesta arquitetura é a mesma utilizada na biblioteca FreeMODBUS (FreeMODBUS, 2012), que contém a implementação do protocolo para escravos e é voltada para sistemas embarcados. Como esta biblioteca suporta, além do protocolo Modbus serial, a versão TCP do protocolo, é a escolha ideal para este trabalho.

6.4 Modelo final da arquitetura

Com as aplicações desenvolvidas, é possível incorporá-las à arquitetura final deste trabalho, que oferece um *framework* de ferramentas básico para a construção e o funcionamento de um equipamento multimedidor (Figura 6.3).

Na arquitetura, identifica-se primeiramente os componentes externos à placa de prototipação: o teclado é a interface de entrada principal do usuário, que se comunica com o

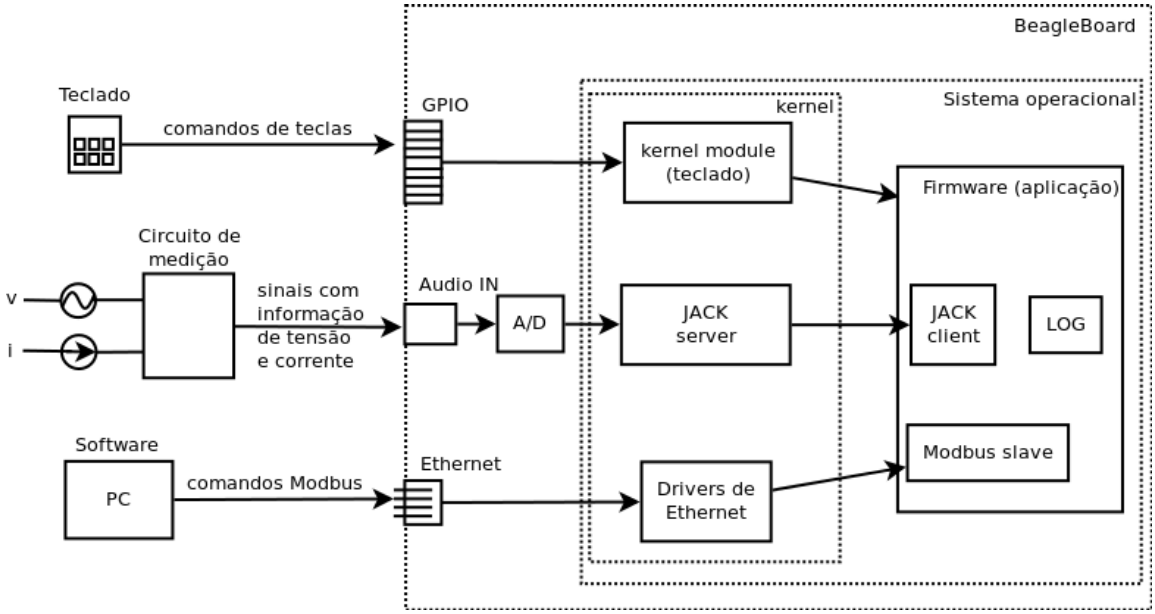


Figura 6.3: Arquitetura desenvolvida neste trabalho de graduação (diagrama).

sistema operacional através de um módulo de *kernel* desenvolvido para este fim capaz de interpretar os sinais recebidos nos pinos de GPIO. A placa de medição é o componente capaz de enviar os sinais de entrada do A/D da placa (pela entrada de áudio) visando a determinação da corrente e da tensão da rede. Finalmente, um software instalado em um computador capaz de se comunicar utilizando o protocolo Modbus pode utilizar a entrada Ethernet da placa para se comunicar com o *firmware*.

Do lado do ambiente de desenvolvimento, o sistema operacional é construído com o BitBake sobre o código aberto disponibilizado pelo OpenEmbedded. O BitBake agrega pacotes, extensões e aplicações necessárias a esta arquitetura, como o Jack e o *patch* de tempo real, e remove aqueles não necessários neste trabalho. Com estas modificações, uma imagem do sistema operacional passou de 330 MB para apenas 45 MB.

No sistema operacional, um servidor do JACK deve ser iniciado para a interpretação da saída do conversor A/D. Uma aplicação de LOG pode rodar ao mesmo tempo no sistema operacional, visando registrar estas medições em arquivos. Estas aplicações fazem parte do *firmware* do equipamento, que ainda gerencia a comunicação via Modbus pela porta Ethernet.

A Figura 6.4 exibe uma foto do sistema em funcionamento, com os componentes citados nessa seção. Os itens 1 e 2 na foto correspondem a saída em vídeo, usando a biblioteca *ncurses*, da aplicação do *firmware* e a saída de vídeo da placa. O item 3 é a entrada de áudio da BeagleBoard (neste exemplo, ela está conectada à saída de aúdio de um computador gerando sinais de áudio que são convertidos pela aplicação. O plugue 4 corresponde à alimentação da placa. O item 5 corresponde à entrada Ethernet da placa, ligada na rede local, sendo usada para comunicação via protocolo Modbus com softwares compatíveis. A entrada do item 6 corresponde aos pinos de GPIO da BeagleBoard, ligados ao teclado utilizado neste trabalho. A placa correspondente ao item 7 é a placa de medição, com seus cabos de alimentação e a saída padrão P2 que carrega a informação de tensão e corrente para a BeagleBoard.

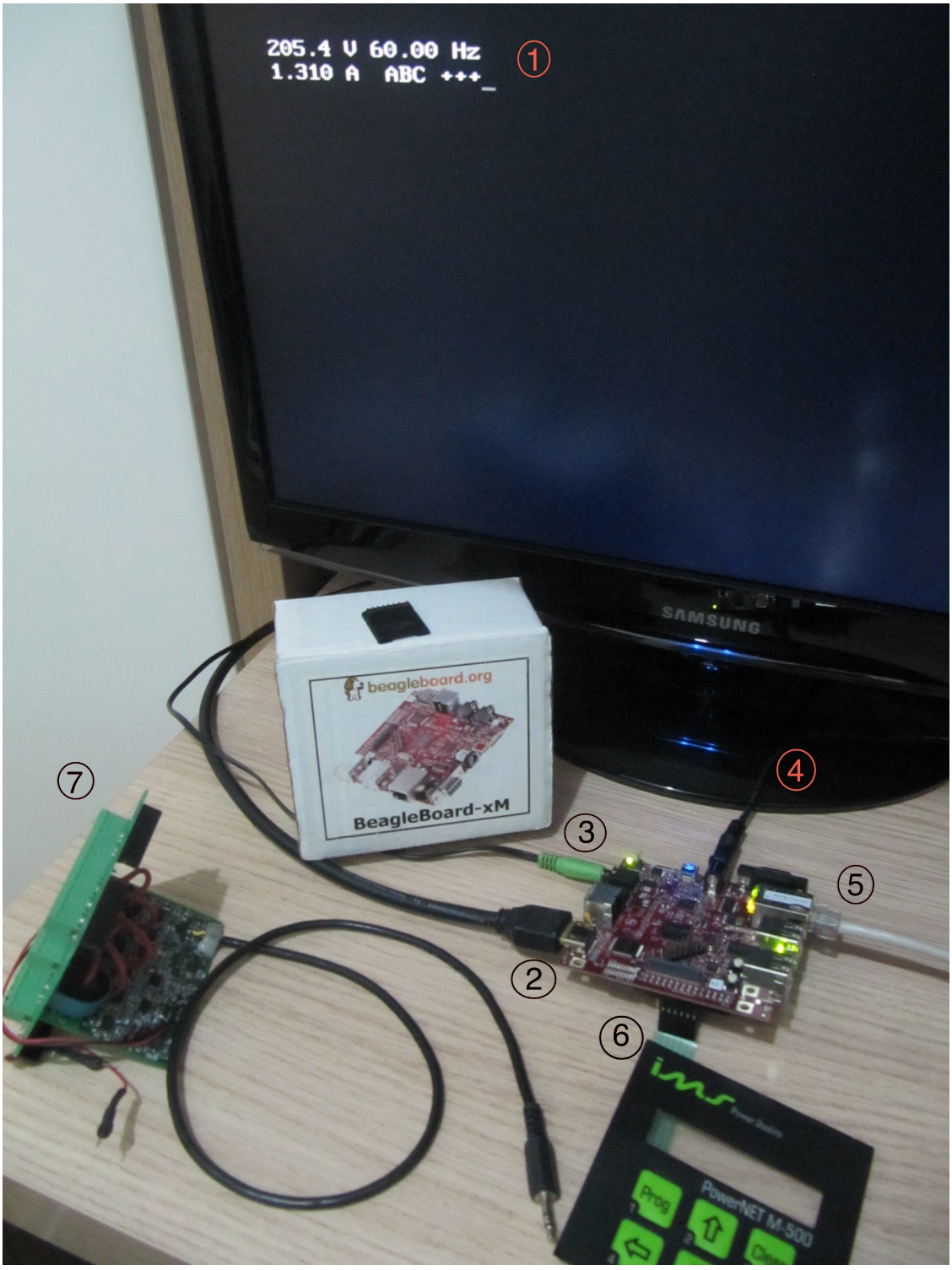


Figura 6.4: Arquitetura desenvolvida neste trabalho de graduação com seus componentes.

7 CONCLUSÃO

Este trabalho abordou diversos pontos da construção de uma arquitetura para um equipamento multimedidor de energia. Inicialmente, foi necessário construir uma distribuição Linux personalizada de acordo com as necessidades do equipamento, retirando todos os módulos que não são necessários (como Bluetooth, Wi-fi, etc.) e instalando as adições requeridas (como o JACK server).

Em seguida, o aspecto tempo real do Linux foi analisado. Como o Linux não é um RTOS originalmente, algumas extensões e alguns pacotes são exigidos para as tarefas existentes em um equipamento de análise de qualidade de energia. O trabalho utilizou uma revisão teórica sobre o processo de amostragem de tensão e corrente de modo a ilustrar a necessidade destas ferramentas de *real-time*.

Como um equipamento de multimedição é um sistema embarcado, é natural que ele tenha dispositivos de entrada e saída e aplicações de usuário que rodam nele. A construção de drivers para Linux foi abordada no trabalho, sendo que a construção do driver para um modelo de teclado utilizado na IMS foi feita. A camada de abstração do sistema operacional que separa os detalhes do *kernel* das aplicações foi revista, e alguns exemplos de aplicação foram citados.

Uma arquitetura Linux se mostrou bastante versátil para a construção destes equipamentos. O fato de se poder construir uma distribuição a partir da seleção de uma grande variedade de pacotes, com o BitBake (apesar deste processo levar algumas horas caso o número de alterações seja muito grande), se mostrou bastante adequado, visto que os pacotes estão frequentemente atualizados e a existência de muito material online sobre este tema.

Por outro lado, a placa de prototipação utilizada, a BeagleBoard-xM, sofre com a falta de informação na Internet. Apesar de ser uma placa nova, ela acabou sendo substituída pelo público por placas mais baratas, como a Raspberry Pi, ou pela sua recente sucessora, a BeagleBone. Isto não desmerece as qualidades do DM3730, um processador com consumo relativamente baixo, para sua alta frequência de operação e o fato de possuir um núcleo ARM e um núcleo DSP.

Como prova de conceito, o resultado obtido neste Trabalho de Graduação demonstra diversas possibilidades com esta arquitetura. Porém, muito ainda deve ser feito visando a construção de um novo equipamento que seja superior aos equipamentos atuais. Por exemplo, pode-se pensar no uso de um display de LCD usado em um equipamento real conectado a placa ao invés de um monitor, com a possibilidade de traçar gráficos de formas de onda a partir das medições efetuadas. Além disso, pode ser interessante utilizar o núcleo DSP para efetuar os cálculos de energia e potência, liberando o núcleo ARM para executar mais aplicações.

Uma vez que estas duas melhorias forem efetuadas, será interessante sair da Beagle-

Board e desenvolver um projeto mais voltado a *hardware*, com o objetivo de projetar uma placa de um futuro equipamento montada com um DM3730 e com suporte a medição trifásica de tensão e corrente (a BeagleBoard limita a medição a apenas uma fase). O planejamento dos periféricos que deverão ser montados na nova placa também deve ser executado.

Este trabalho é de grande importância para a IMS no médio prazo, quando os equipamentos em desenvolvimento atualmente estiverem concluídos e maduros comercialmente e uma nova família de equipamentos se tornar necessária: uma nova série de equipamentos poderá ser construída sobre esta nova arquitetura, agregando funcionalidades não existentes nos equipamentos atuais graças às novas possibilidades oferecidas por ela.

REFERÊNCIAS

ABBOTT, D. **Linux for Embedded and Real-Time Applications**, 2nd edition, Elsevier, 2006. 300p.n

AXELSON, J. **Using Eclipse to Cross-compile Applications for Embedded Systems**, disponível para consulta em <http://www.lvr.com/eclipse1.htm>, acesso em setembro de 2012.

BitBake team. **BitBake**, disponível para consulta em <http://bitbake.berlios.de/manual/>, acesso em julho de 2012.

BUSATTO, T. **Análise dos Requisitos Mínimos Necessários para Medição da Qualidade da Energia em Atendimento aos Procedimentos de Distribuição (PRODIST)**. IX Conferência Brasileira sobre Qualidade de Energia, 2011, páginas 664-668.

COLEY, G. . **BeagleBoard-xM System Reference Manual**, disponível para consulta em <http://beagleboard.org/static/BBxMSRM_latest.pdf>, acesso em maio de 2012. 164p.

Texas Instruments. **DM3730 Datasheet**, disponível para consulta em <http://www.ti.com/litv/pdf/sprs685d>, acesso em novembro de 2012.

DNP3 team. **Distributed Network Protocol**, disponível para consulta em <http://www.dnp.org/>, acesso em dezembro de 2012.

FreeMODBUS. **FreeMODBUS - A Modbus ASCII/RTU and TCP implementation**, disponível em <http://www.freemodbus.org>, acesso em dezembro de 2012.

HALLINAN, C. **Embedded Linux Primer**, 2nd edition, Pearson Education, Inc, 2011. 592p.

IEC. **Testing and measurement techniques – General guide on harmonics and interharmonics measurements and instrumentation, for power supply systems and equipment connected thereto**, edition 2.1, 2010.

IEC. **Electromagnetic Compatibility (EMC) – Part 4-15: Testing and Measurement techniques – Flickermeter – Functional and design specifications**, edition 2.0, 2010.

IEC. **Electromagnetic Compatibility (EMC) – Part 4-30: Testing and Measurement techniques – Power quality measurement methods**, edition 2.0, 2008.

IMS Power Quality **IMS Gerenciamento de energia elétrica, multimedidores, controladores, analisadores**, disponível em <http://www.ims.ind.br/br>, acesso em novembro de 2012.

JACK Audio Team. **JACK | connecting a world of audio**, disponível em <http://www.jackaudio.org>, acesso em outubro de 2012.

Keil. **Keil embedded development tools**, disponível em <http://www.keil.com>, acesso em maio de 2012.

MAX, J., LACOUME, J-L. **Méthodes et techniques de traitement du signal et applications aux mesures physiques (Tome 1 - Principes généraux et méthodes classiques)**, 5ème edition, Masson, 1996. 355p.

MODICON, Inc. **Modicon Modbus Protocol Reference Guide**, rev. J, 1996. 115p.

MOG, G.E., RIBEIRO, E.P. **Zero Crossing Determination by Linear Interpolation of Sampled Sinusoidal Signals**. 2004 IEEE/PES Transmission and Distribution Conference and Exposition: Latin America, páginas 799-802.

Ncurses. **ncurses 5.9**, disponível para consulta em <http://www.gnu.org/software/ncurses>, acesso em dezembro de 2012.

NILSSON, J., RIEDEL. S. **Circuitos Elétricos**, 6ª edição, LTC, 2003. 656p.

OpenEmbedded collaborators. **OpenEmbedded Wiki**, disponível para consulta em <http://www.openembedded.org/wiki/Main_Page>, acesso em julho de 2012.

Omicron. **CMC256plus - OMICRON**, disponível para consulta em <http://www.omicron.at/en/products/pro/secondary-testing-calibration/cmc-256plus/>

PandaBoard website. **PandaBoard References**, disponível para consulta em <http://pandaboard.org/content/resources/references>, acesso em novembro de 2012.

ANEEL. **Procedimentos de Distribuição (PRODIST), Módulo 8 – Qualidade da Energia Elétrica, Agência Nacional de Energia Elétrica**, 2010.

Raspberry Pi website. **Raspberry Pi**, disponível para consulta em <http://www.raspberrypi.org/>, acesso em novembro de 2012.

RAGHAVAN, P., LAD, A., NEELAKANDAN, S. **Embedded Linux System Design and Development**, Auerbach Publications, 2006. 400p.

MOLNAR, I. and contributors. **Real-time kernel Project**, disponível para consulta em <http://www.kernel.org/pub/linux/kernel/projects/rt/>, acesso em novembro de 2012.

SALZMAN, P. J., BURIAN, M., POMERANTZ, O. . **The Linux Kernel Module Programming Guide**, 2ª edição, 2007, 82p.

SEDRA, A.S., SMITH, K.C. **Microeletrônica**, Pearson do Brasil, 5ª edição, 2007. 848p.

SHAW, A. C. **Real-Time Systems and Software**. John Wiley & Sons, Inc, 1st edition. 224p.

SILVA, B. D. B. **Localização de som com sinais binaurais**, Projeto de Diplomação em Engenharia Elétrica, UFRGS. 72p.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**, 3a. Edição. Pearson do Brasil, 2010. 653p.

Texas Instruments. **OMAP Mobile Processors**, disponível para consulta em <http://www.ti.com/omap>, acesso em maio de 2012.

Texas Instruments. **TPS65950 Data Manual (Rev. E)**, disponível para consulta em <http://www.ti.com/lit/ds/symlink/tps65950.pdf>, acesso em novembro de 2012.

XU, Z., CARRION, L., MACIEJKO, R. **A zero-crossing detection method applied to Doppler OCT**. Optics Express, Vol. 16, Issue 7, 2008, páginas 4394-4412.

ANEXO A: TRABALHO DE GRADUAÇÃO 1

Linux embarcado na BeagleBoard para Power Quality

Bruno Silva Guedes[1], João César Netto[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{bsguedes,netto}@inf.ufrgs.br

Resumo. *Este artigo descreve a especificação de uma nova arquitetura de um sistema em tempo real baseado em uma distribuição Linux e rodando sobre a plataforma de prototipação BeagleBoard, voltada para aplicações de medição e análise de grandezas físicas relacionadas à qualidade da energia elétrica.*

Abstract. *This paper describes the specification of a new architecture for a real-time system based on a Linux distribution that runs over the BeagleBoard prototyping board, for applications of measure and analysis of physical quantities related to electrical Power Quality.*

1. Introdução

Este Trabalho de Graduação está sendo elaborado a partir da necessidade da IMS Power Quality, empresa da cidade de Porto Alegre que desenvolve equipamentos de medição, análise e controle da energia elétrica, de construir equipamentos de Power Quality (monitoração de qualidade de energia) de alto desempenho que sejam capazes de medir e registrar grandezas fundamentais para a análise da qualidade da energia.

Exemplos de grandezas de Power Quality para essa aplicação incluem distorção harmônica de tensão e corrente, *sags* e *swells* (afundamento e elevação do valor eficaz da tensão) e *flickers* (efeito causado por flutuações de tensão em frequências baixas). O ponto crítico nessa aplicação consiste na capacidade de efetuar essas tarefas com alto grau de precisão, sendo que essas medidas e cálculos ocorrem várias vezes por segundo, em tempo real.

Os equipamentos atuais de Power Quality da IMS já são capazes de efetuar estas análises. No entanto, limitações no hardware usado atualmente impedem que novas funcionalidades sejam implementadas, como um sistema robusto em tempo real para processamento das medições, um sistema de arquivos estável e a implementação de drivers de dispositivos como teclado e conversor analógico-digital.

Este artigo apresenta uma proposta de arquitetura embarcada sobre a qual um framework de aplicações e serviços será construído visando validar a possibilidade de utilização de um sistema embarcado em tempo real nos equipamentos da empresa. No Capítulo 2, esta proposta é descrita com mais detalhes. O Capítulo 3 descreve as nuances da concepção de sistemas operacionais em tempo real. No Capítulo 4, a plataforma de prototipação que será utilizada é apresentada. O artigo é concluído com um esquemático da proposta de arquitetura, seguido do cronograma para guiar a segunda parte deste Trabalho de Graduação e das conclusões retiradas da primeira parte desta atividade.

2. Objetivo

O objetivo deste trabalho é desenvolver um sistema embarcado para rodar sobre uma plataforma de prototipação chamada BeagleBoard, com a tarefa de suprir os requisitos de funcionamento de equipamentos de Power Quality descritos na Seção 1. Esta solução consiste em integrar no sistema embarcado da placa (uma distribuição Linux para sistemas embarcados), os drivers dos periféricos necessários para a operação dos equipamentos de Power Quality, incluindo os básicos, como teclado e display, aos mais avançados e específicos, como memória compartilhada, sistema operacional de tempo real e comunicação com DSP.

Além disso, equipamentos de Power Quality devem atender alguns requisitos operacionais regulados pelas normas IEC 61000-4, seções 7, 15 e 30. Essas normas determinam classes de qualidade para cada equipamento em função da precisão de suas medições. Para que seja possível desenvolver equipamentos que atendam melhor os requisitos destas normas (logo, que possam agregar mais valor de mercado aos produtos da IMS), a solução passa por desenvolver um novo framework que sirva de base para a construção de novos equipamentos.

A arquitetura desenvolvida consiste em um prova de conceito (POC – *proof of concept*) para a empresa. A ideia é conseguir demonstrar que o uso de um sistema embarcado rodando uma distribuição Linux é viável para os projetos da IMS e que é possível agregar novas funcionalidades a esta arquitetura. Assim, o modo de demonstrar que o objetivo foi alcançado, será reconstruir, a partir do framework desenvolvido (conjunto de funções usadas em aplicações de Power Quality), um dos equipamentos já existentes da IMS sobre a arquitetura proposta (por exemplo, o M-500), demonstrando a equivalência de ambos em operação.

3. Sistemas em tempo real em ambientes embarcados

Os sistemas convencionais de computação, sejam eles embarcados ou de propósito geral, como PCs, são considerados corretos quando a sua funcionalidade é realizada do modo previsto na sua construção. Normalmente, estes sistemas rodam várias aplicações de usuário e de sistema operacional ao mesmo tempo, e o momento em que essas aplicações efetivamente estão executando na CPU não é muito importante, desde que a tarefa seja concluída em algum momento com o resultado esperado [Shaw, 2001]. Porém, existem situações em que a correção de um sistema não depende apenas da produção de uma saída adequada, mas também do instante em que essa saída foi produzida. Por exemplo, um decodificador de um fluxo de vídeo não deve ser capaz apenas de converter dados multimídia utilizando um determinado algoritmo de decodificação, mas deve também realizar essa operação utilizando uma taxa de quadros por segundo específica, correndo o risco de reproduzir um filme com travamentos [Raghavan *et al*, 2006]. A categoria de **sistemas em tempo real** compreende esse tipo de sistema, no qual existem requisitos temporais em relação à precisão e à duração de suas tarefas que são necessários na adequação do resultado final.

Sendo assim, as tarefas de um sistema em tempo real possuem critérios de início e fim permitidos para sua execução, de modo a garantir a correção da sua operação. A temporização incorreta de um sistema em tempo real pode gerar resultados pouco úteis, como no exemplo citado no parágrafo acima, mas também resultados que são, além de inúteis, danosos, como por exemplo em um sistema que seja responsável por controlar o

resfriamento de um reator nuclear [Lages, 2008]. Por isso, esses sistemas são classificados de acordo com a sua **criticidade** [Shaw, 2001]: quanto maior o custo de uma falha do sistema, mais crítico é o sistema.

De acordo com [Raghavan *et al*, 2006], um sistema operacional para um sistema em tempo real precisa prover alguns requisitos em termos de serviços para que ele seja capaz de realizar corretamente as tarefas necessárias. As principais funcionalidades são:

- **Suporte à prioridades:** um sistema operacional em tempo real deve possuir suporte à prioridades, já que funcionalidades críticas desse sistema que tenham restrições temporais devem possuir prioridades mais altas.
- **Preempção:** quando uma tarefa de mais alta prioridade entra na fila de tarefas prontas para ser executadas, ela deve ser capaz de tomar o lugar de uma tarefa de mais baixa prioridade em execução.
- **Latência conhecida do escalonador:** a diferença de tempo entre o momento em que uma tarefa crítica se torna pronta para executar e o instante em que sua execução inicia não excede um valor conhecido.

O último item nesta lista é muito importante e é crítico para o funcionamento correto de sistemas desta natureza, já que os processos que irão rodar na placa precisam executar suas tarefas na CPU em intervalos de tempo bem definidos e precisos. Assim, o escalonador para o sistema deste Trabalho de Graduação precisa levar em consideração este fato, conhecendo com que frequência cada processo deve executar, qual sua duração e qual o seu prazo de término.

3.1. Algoritmos de escalonamento para sistemas operacionais em tempo real

Dois algoritmos de escalonamento que empregam preempção merecem ser analisados com mais detalhes [Tanenbaum, 2010]. O primeiro, o **escalonamento por taxa monotônica**, que consiste na atribuição, a cada processo, uma prioridade proporcional à frequência com a qual este processo (evento) ocorre. Em tempo de execução, o escalonador escolhe, entre os processos prontos, aquele que possui prioridade mais alta (ou seja, o mais frequente), removendo o processo atual do processador caso seja necessário. Este algoritmo é considerado estático visto que a prioridade de cada processo é fixa e é definida no projeto das tarefas que este sistema irá desempenhar.

O segundo algoritmo, de **escalonamento por prazo mais curto**, entra na classe de algoritmos dinâmicos, já que a prioridade atribuída aos processos é uma medida que depende de quão próximo um processo pronto está de seu prazo de término. O escalonador prioriza processo cuja faixa de tempo está expirando, podendo em situações mais drásticas fazer a preempção do processo em execução.

A escolha do algoritmo de escalonamento depende das características das tarefas que o sistema precisa executar. Embora a implementação do primeiro seja mais simples e o escalonamento leve menos tempo (já que ele é estático), [Liu e Layland, 1973] mostra que o primeiro algoritmo funciona apenas se a ocupação do processador não for muito grande, com o risco de o sistema não atender algumas tarefas antes de seu vencimento (por exemplo, para 10 processos a taxa de utilização da CPU não pode exceder 72% [Tanenbaum, 2010]). O segundo algoritmo, embora mais complexo,

permite que a CPU fique ocupada 100% do tempo, considerando que as tarefas sejam projetadas de modo a possibilitar o atendimento a todas elas com a CPU disponível (sendo possível, neste caso, realizar um teste de escalonabilidade de modo a verificar a capacidade do algoritmo por prazo mais curto de atender todas as tarefas).

4. Linux embarcado na BeagleBoard

Os equipamentos atuais da IMS são equipados por uma placa embarcada desenvolvida pela Embedded Artists. Estas placas são controladas por um sistema operacional embarcado proprietário da Keil [Keil, 2012], que possuiu um ambiente de desenvolvimento elaborado com diversas ferramentas para o desenvolvimento das aplicações de medição e análise necessárias nos equipamentos. No entanto, este ambiente é fechado e suas licenças de utilização constituem um custo elevado no orçamento de desenvolvimento da empresa.

Assim, a IMS começou um processo de transição para a utilização de uma plataforma embarcada aberta, que permita que uma distribuição Linux seja instalada e customizada, de acordo com as funcionalidades necessárias aos equipamentos de Power Quality. A escolha para este trabalho foi a BeagleBoard, na sua última revisão (no início de 2012), a BeagleBoard-xM revC. A escolha desta plataforma se deve ao fato de ela possuir design aberto altamente extensível [Coley, 2010], permitindo no futuro a personalização desta placa para qualquer aplicação visada pela IMS.

4.1. Características da BeagleBoard

A BeagleBoard é uma pequena placa de aproximadamente 8 cm x 8 cm, como mostrado na Figura 1. Construída por um time de engenheiros da Texas Instruments, ela é voltada para aplicações educacionais, já que suas características a classificam como um pequeno computador com baixo consumo de energia, e também como um modo de demonstrar as capacidades dos chips da família OMAP da TI, que são encontrados hoje em diversas aplicações como celulares e *tablets* [TI, 2012].

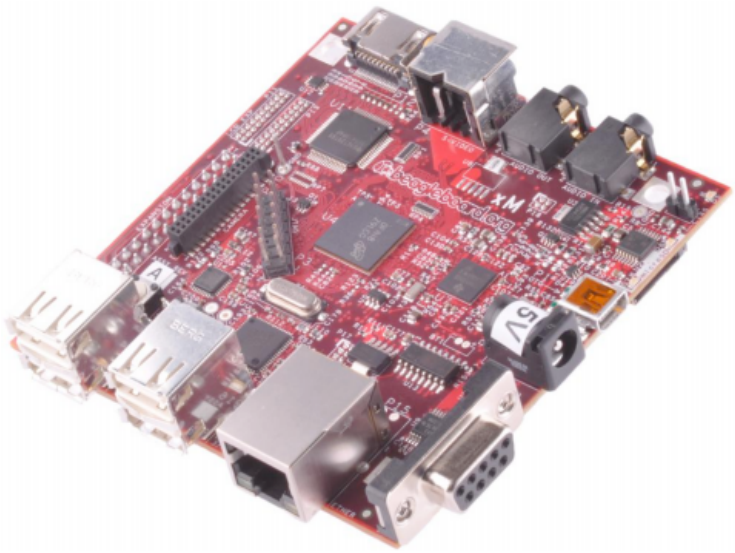


Figura 1. A placa BeagleBoard-xM revC. [Coley, 2010]

Algumas das principais funcionalidades da BeagleBoard na revisão utilizada neste Trabalho de Graduação são [Coley, 2010]:

- Processador TI Cortex A8, com *clock* de 1GHz (chip OMAP3730 com núcleos ARM e DSP integrados);
- Memória RAM de 512 MB (@ 166 MHz);
- 4 portas USB de alta velocidade (USB 2.0);
- Armazenamento via slot microSD;
- Saídas de vídeo DVI-D e S-video;
- Entrada/saída de áudio estéreo;
- Conexões Ethernet, JTAG, Câmera e outras.

A alimentação é fornecida através de uma fonte que forneça até 3A a uma tensão de 5V DC, ou ainda via uma entrada mini USB.

A BeagleBoard é capaz de rodar diversos sistemas operacionais, como Android, Windows Embedded CE, RISC OS e também diversas distribuições de Linux, como Fedora e Ubuntu (uma lista completa de projetos existentes, incluindo a documentação dos mesmos é constantemente atualizada em [BeagleBoard, 2012]). A placa suporta boot diretamente do cartão microSD de 4GB fornecido com a placa, através do *bootloader* que já vem gravado na sua memória ROM. Para fins de teste com a BeagleBoard, foi instalada na placa a distribuição Angstrom do Linux, utilizando o procedimento descrito em [Silva, 2011], onde uma imagem para esta distribuição foi criada utilizando a plataforma Narcissus [Narcissus, 2012] e instalada na placa com o *kernel* e os utilitários padrão do Linux já compilados.

A metodologia de trabalho com a BeagleBoard será similar à efetuada por [Silva, 2011], porém voltada às aplicações de Power Quality: inicialmente, deve-se buscar remover do sistema operacional suporte aos componentes da placa que não serão utilizados, e que consomem recursos como memória dinâmica, CPU e espaço em cartão SD desnecessariamente. Em seguida, será preciso instalar o ambiente de desenvolvimento em uma máquina de trabalho, que consiste nos softwares necessários para desenvolvimento (editores de texto, controle de versão e documentação). Faz parte ainda deste ambiente a instalação dos compiladores capazes de gerar código para o processador da BeagleBoard, neste caso, a biblioteca C6EZRun [C6EZRun, 2012], que permite a geração de código DSP a partir do compilador que roda no núcleo ARM, chamado *arm-gcc* (“*ARM GNU C Compiler”)*.

Para a implementação das tarefas próprias do escopo deste trabalho, serão utilizadas igualmente algumas bibliotecas prontas de uso extensivo e bastante documentadas, cujo trabalho não precisa ser refeito (como por exemplo suporte à *Posix Threads* e suporte ao núcleo de processamento digital de sinais – DSP – do chip da Texas que roda na placa).

5. Descrição da arquitetura e do framework

Além das tarefas de modificar o *kernel* do Linux para implementar as funções que podem ser usadas para construir aplicações próprias aos equipamentos que serão

desenvolvidos sobre o framework que está sendo criado neste trabalho, será preciso realizar algumas modificações de hardware e de interface na placa para que ela se adapte aos produtos criados pela IMS.

O esquemático da Figura 2 consiste no diagrama de blocos de alto nível da BeagleBoard, somado a algumas das adições necessárias neste diagrama para o desenvolvimento deste trabalho. É preciso criar um módulo de software que implemente o protocolo Modbus [Modbus, 2012], protocolo de comunicação industrial largamente utilizado que utiliza uma porta Serial RS-232 e que é utilizado pela IMS para realizar a transmissão de dados entre os equipamentos e os softwares de gerenciamento dos mesmos. Será necessário também criar um suporte ao barramento I2C [I2C, 2012], cujo protocolo será utilizado para criar a interface entre a placa e o teclado nos equipamentos da empresa.

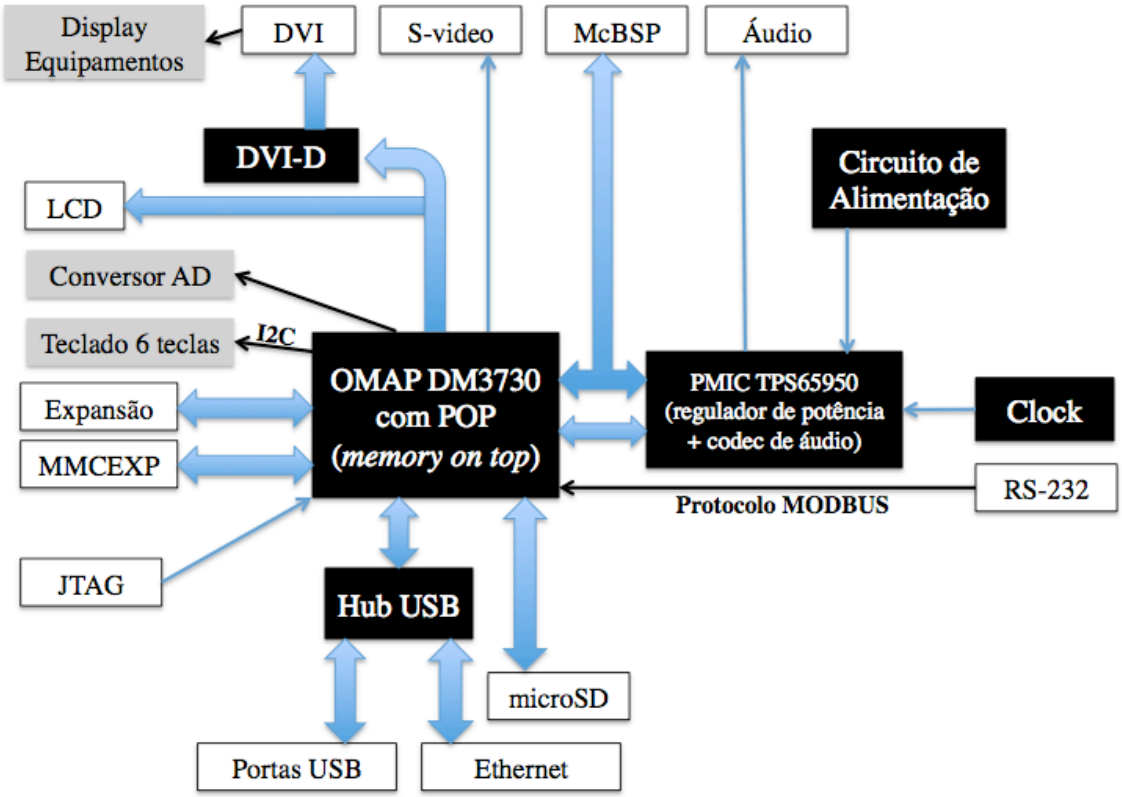


Figura 2. Diagrama de blocos de alto nível da BeagleBoard - adaptado de [Coley, 2010]. Com fundo preto: circuitos da placa; em branco: componentes de E/S; em cinza claro: adições aportadas por este Trabalho.

Para realizar as medições das grandezas elétricas, um conversor AD deve ser construído de forma a constituir a interface entre as grandezas analógicas e os valores digitais recebidos pela placa. Inicialmente, neste POC, será utilizada a interface de áudio para transmitir estes dados para o processador. Como o áudio é transmitido por dois canais (som estéreo), teremos acessos às informações de uma fase da rede elétrica (tensão e corrente). Com esses dois valores, é possível determinar o fator de potência de uma rede monofásica, as potências ativa, reativa e aparente, a frequência e realizar a análise de harmônicos da rede.

Finalmente, um display gráfico faz parte da interface dos equipamentos da IMS com o mundo real. Neste trabalho, deverá ser criado suporte a pelo menos um dos displays utilizados hoje em dia nos equipamentos da empresa, devendo ser implementado um driver que interaja com esse dispositivo de vídeo e criada uma aplicação gráfica simples que valide a utilização deste display.

6. Conclusões

Este artigo descreve o resultado da pesquisa realizada durante a primeira parte do Trabalho de Graduação, cujo objetivo era (a) modelar a especificação do problema, que era bastante abstrata no início do projeto; (b) solidificar a base teórica em sistemas de tempo real e escalonamento de processos em tempo real, necessária para o desenvolvimento das aplicações de qualidade de energia elétrica e (c) fornecer alternativas de implementação para os diversos componentes da arquitetura e do framework descritos neste artigo, que deverão ser elaborados no projeto. Com esses três objetivos alcançados, é possível elaborar um plano de trabalho para a segunda parte, que consiste no seguinte cronograma de atividades.

6.1. Cronograma

A Tabela 1 descreve as principais atividades que devem ser realizadas na segunda etapa do Trabalho de Graduação. Outras tarefas podem ser integradas nesta tabela, mas a princípio o projeto não deve divergir muito deste plano já elaborado. A escrita da monografia e a consulta a outras referências são consideradas atividades transversais que se estendem ao longo de todo o resto do trabalho.

Tarefa/Mês	Jun	Jul	Ago	Set	Out	Nov
Atualização bibliográfica contínua nos tópicos de desenvolvimento de hardware e de software	X	X	X	X	X	X
Redação da monografia	X	X	X	X	X	X
Operação com a BeagleBoard	X					
Instalação do Ambiente de Desenvolvimento	X	X				
Levantamento dos requisitos de hardware		X				
Teclado via I2C		X	X			
Display via DVI			X			
Controle da frequência de amostragem			X	X		
Construção do conversor AD				X		
Implementação do protocolo Modbus				X		
Avaliação do comportamento temporal do protótipo				X		
Ajuste dos parâmetros do escalonador (prioridades, filas)					X	

Co-processamento e memória compartilhada					X	
Interface de usuário					X	X
Implementação do LOG de grandezas						X
Apresentação do Trabalho de Graduação						X

Tabela 1. Cronograma para a segunda parte do Trabalho de Graduação

Referências

BeagleBoard (2012) "BeagleBoard.org – Projects", disponível para consulta em <http://beagleboard.org/project> (acesso em junho de 2012).

C6EZRun (2012) "C6EZRun – TI Embedded Processors Wiki", disponível para consulta em <http://processors.wiki.ti.com/index.php?title=C6EZRun> (acesso em maio de 2012).

Coley, G. (2010) "BeagleBoard-xM System Reference Manual", páginas 21-47, disponível para consulta em <http://beagleboard.org/static/BBxMSRM_latest.pdf> (acesso em maio de 2012).

Keil (2012) "Keil Embedded Development Tools", disponível para consulta em <http://www.keil.com> (acesso em maio de 2012).

Lages, W.F. (2008) "ENG04008 Sistemas de Tempo-real", disponível para consulta em <http://www.ece.ufrgs.br/~fetter/eng04008> (acesso em maio de 2012).

Liu, C. L.; Layland, J. W. (1973) "Scheduling Algorithms for multiprogramming in a hard real-time environment". J. of the ACM, v. 20, páginas 46-61.

Modbus (2012) "The Modbus Organization", disponível em <http://www.modbus.com> (acesso em maio de 2012).

Narcissus (2012). "Online image builder for the angstrom distribution", disponível para consulta em <http://narcissus.angstrom-distribution.org> (acesso em maio de 2012).

Raghavan, P.; Lad, A.; Neelakandan, S. "Embedded Linux System Design and Development", páginas 201-202. Auerbach Publications.

Shaw, A. C. (2001). "Real-Time Systems and Software", páginas 1-4. John Wiley & Sons, Inc.

Silva, B. D. B. (2011) "Localização de som com sinais binaurais", páginas 38-39, Projeto de Diplomação em Engenharia Elétrica, UFRGS.

Tanenbaum, A. S. (2010) "Sistemas Operacionais Modernos", 3ª. Edição, páginas 302-305, Pearson do Brasil.

TI (2012). "OMAP Mobile Processors", disponível para consulta em <http://www.ti.com/omap>, acesso em maio de 2012.