UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

TOOLKIT PARA LINUX EMBARCADO

THIAGO WALTRIK

BLUMENAU
2011

2011/2-29

THIAGO WALTRIK

TOOLKIT PARA LINUX EMBARCADO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Miguel Alexandre Wisintainer , Mestre - Orientador

BLUMENAU
2011

2011/2-29

TOOLIT PARA LINUX EMBARCADO

Por

THIAGO WALTRIK

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Miguel Alexandre Wisintainer, Mestre – Orientador, FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB

Blumenau, 13 de dezembro de 2011

AGRADECIMENTOS

À minha namorada, Larissa, pela paciência e apoio nos momentos difíceis.

Ao amigo e professor, Adriano, pelo incentivo e pelos comentários construtivos.

Ao meu orientador, Miguel Alexandre Wisintainer, por ter a motivado e acreditado fortemente na realização deste trabalho.

À Universidade Regional de Blumenau, pela oportunidade concedida.

É melhor a verdade dura do que a fantasia consoladora.

Carl Sagan

RESUMO

Para iniciar o desenvolvimento de aplicativos para sistemas embarcados com Linux é necessário a reunião de vários itens, tais como um modelo de dispositivo embarcado, uma imagem do *kernel* Linux compilada a partir de uma versão customizada do código-fonte para este dispositivo, um sistema de arquivos raiz com *scripts* e bibliotecas necessárias para a execução do sistema proposto e documentação. Este trabalho demonstra o desenvolvimento de um *toolkit* para Linux embarcado, que é a reunião de todos estes itens afim de que o desenvolvedor não gaste uma enorme quantidade de tempo inicial na preparação destes itens. Demonstra a preparação de uma estação de trabalho de desenvolvimento para sistemas embarcados, a customização e geração de uma imagem do *kernel* Linux, a geração de um sistema de arquivos raiz e a implementação de mini aplicativos com interface gráfica do usuário que exploram os principais recursos de hardware do dispositivo alvo utilizado, o Mini2440. Os resultados obtidos podem ser aplicados no desenvolvimento de protótipos de sistemas embarcados como roteadores, interfaces homem-máquina, *thin clients*, *tablet PCs*, *netbooks* e telefones IP.

Palavras-chave: Sistemas embarcados. Linux. ARM. Qt.

ABSTRACT

To start developing applications for embedded systems with Linux is necessary the junction of several items, such as an embedded device, a Linux kernel image compiled from a customized version of the source code for the embedded device, a root file system with scripts and libraries necessary for the implementation of the proposed system and documentation. This monograph describes the development of a embedded Linux toolkit, which is junction all of these items in order that the developer does not spend an excessive amount of time in the initial preparation of these items. This monograph demonstrates the preparation of a workstation for embedded systems development, customization and build of a Linux kernel image, the build of a root file system and the implementation of applications with graphical user interface that exploit the hardware key features of the embedded device used, the Mini2440. The results obtained can be used to develop prototypes of embedded systems such as routers, human-machine interfaces, thin clients, tablet PCs, netbooks and VoIP phones.

Key-words: Embedded systems. Linux. ARM. Qt.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AC97 – *Audio Codec ’97*

ADC – Conversor Analógico/Digital

ALSA – *Advanced Linux Sound Architecture*

AMBA – *Advanced Microcontroller Bus Architecture*

ANVISA – Agência Nacional de Vigilância Sanitária

API – *Application Programming Interface*

ARM – *Advanced RISC Machine*

CISC – *Complex Instruction Set Computer*

CMOS – *Complementary Metal-Oxide-Semiconductor*

CPU – *Central Processing Unit*

DMA – *Direct Memory Access*

DSP – *Digital Signal Processor*

DVD – *Digital Versatile Disc*

EEPROM – *Electrically-Erasable Programmable Read-Only Memory*

EIA – *Eletronic Industries Alliance*

ESSID – *Extended Service Set Identification*

ext2 – *Second EXTended filesystem*

ext3 – *Third EXTended filesystem*

FAT32 – *File Allocation Table*

GITEC – Gerência de Infra-Estrutura e Tecnologia

GNU – *GNU is Not Unix.*

GPIO – *General Purpose Input/Output*

GPRS – *General Packet Radio Service*

GSM – *Global System for Mobile communications*

GUI – *Graphical User Interface*

I2C – *Inter-Integrated Circuit*

IDE – *Integrated Development Environment*

IP – *Internet Protocol*

LCD – *Liquid Crystal Display*

LED – *Light-Emitting Diode*

MIPS – *Microprocessor without Interlocked Pipeline Stages*

MMC – *MultiMediaCard*

MMU – *Memory Management Unit*

MSD – *Market Specific Distribution*

NAND – *Not AND*

NASA – *National Aeronautics and Space Administration*

NOR – *Not OR*

PC – *Personal Computer*

PNG – *Portable Network Graphics*

POSIX – *Portable Operating System Interface*

PowerPC – *Power Optimization With Enhanced RISC Performance Computing*

PPM – *Portable Pixmap Format*

RISC – *Reduced Instruction Set Computer*

RTC – *Real Time Clock*

SD – *Secure Digital*

SDK – *Software Development Kit*

SDRAM – *Synchronous Dynamic Random Access Memory*

SPARC – *Scalable Processor ARChitecture*

TFTP – *Trivial File Transfer Protocol*

UART – *Universal Asynchronous Receiver Transmit*

USB – *Universal Serial Bus*

YAFFS – *Yet Another Flash File System*

SUMÁRIO

1 INTRODUÇÃO

Segundo Weinberg (2002, p. ix), os sistemas embarcados estão emergindo como uma indústria multibilionária. Por todo o mundo, dispositivos inteligentes estão entrando cada vez mais na vida diária. Sloss, Symes e Wright (2004, p. 6) afirmam que sistemas embarcados podem controlar os mais diferentes dispositivos, desde pequenos sensores encontrados em linhas de produção a sistemas de controle de tempo-real utilizados nas sondas espaciais da *National Aeronautics and Space Administration* (NASA). De acordo com Hollabaugh (2002, p. 1), indivíduos e empresas querem dispositivos inteligentes conectados por rede para melhorar suas vidas. Estes dispositivos devem ser simples de operar, confiáveis e pouco dispendiosos.

O componente chave de muitos sistemas embarcados de 32 bits são os processadores com arquitetura *Advanced RISC*[1] *Machine* (ARM) (SLOSS; SYMES; WRIGHT, 2004, p. 1). Os processadores ARM são capazes de executar Linux. Linux é a designação de uma família de sistemas operacionais *unix-like* que compartilham o mesmo *kernel*[2], sendo capaz de ser executado em diferentes plataformas de processadores como Itanium, *Power Optimization With Enhanced RISC Performance Computing* (PowerPC), x86, *Scalable Processor ARChitecture* (SPARC) e *Microprocessor without Interlocked Pipeline Stages* (MIPS). O Linux, quando executado em um sistema embarcado, é denominado Linux embarcado.

Ao iniciar-se o desenvolvimento de aplicativos para sistemas embarcados baseados em Linux é necessário escolher um modelo de dispositivo embarcado (dispositivo alvo), um compilador cruzado para este dispositivo, uma biblioteca para linguagem C e uma imagem do *kernel* Linux compilada a partir de uma versão customizada do código-fonte para este dispositivo. Estes itens, aliados a documentação e uma estação de trabalho de desenvolvimento, formam um *toolkit* para Linux embarcado. Segundo Lombardo (2002, p. 65), a reunião destes itens pode demandar uma enorme quantidade de tempo antes que se possa ter um sistema que simplesmente inicia o processo de *boot*.

Para preencher esta lacuna, desenvolveu-se um *toolkit* para Linux embarcado em um dispositivo que utiliza processador ARM. Este *toolkit* foi desenvolvido para o dispositivo alvo Mini2440 (FRIENDLYARM, 2010a) e fornece uma solução para que possa se iniciar o

---

[1] RISC é o acrônimo de *Reduced Instruction Set Computer*.

[2] *Kernel* é o núcleo do sistema operacional.

desenvolvimento de aplicativos sem que seja necessário dispender uma enorme quantidade de tempo inicial, proporcionando integração entre o hardware e software.

A escolha do Mini2440 deu-se devido ao seu baixo custo e a sua diversidade de periféricos, entre eles um mostrador do tipo *Liquid Crystal Display* (LCD) colorido com tela sensível ao toque, módulo Wireless e interface para cartões de memória *Secure Digital* (SD).

1.1 OBJETIVOS DO TRABALHO

Este trabalho tem como objetivo geral o desenvolvimento de um *toolkit* para Linux embarcado que possibilite o desenvolvimento de aplicativos em tempo reduzido utilizando o Mini2440 como dispositivo alvo.

Os objetivos específicos do trabalho são:

a) disponibilizar documentação passo a passo para a preparação de uma estação de trabalho de desenvolvimento de sistemas com Linux embarcado baseada na distribuição Linux Debian;
b) customizar, compilar e disponibilizar uma versão do *kernel* Linux para execução no Mini2440;
c) disponibilizar mini-aplicativos que explorem os principais recursos de hardware do Mini2440;
d) carregar e executar mini-aplicativos no Mini2440.

1.2 ESTRUTURA DO TRABALHO

Este trabalho divide-se em quatro capítulos. O primeiro capítulo apresenta a introdução, os objetivos e a estrutura do trabalho. A fundamentação teórica é apresentada no segundo capítulo. O terceiro capítulo apresenta os passos realizados para o desenvolvimento deste trabalho juntamente com os resultados e discussão. Por fim, o quarto capítulo apresenta a conclusão e extensões para este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo foi dividido em nove seções. Um breve histórico sobre o sistema operacional Linux é apresentado na seção 2.1. A seção 2.2 trata sobre o sistema Debian. A seção 2.3 apresenta alguns aspectos sobre a arquitetura ARM. A seção 2.4 trata sobre a filosofia da arquitetura RISC. Detalhes técnicos sobre o computador de placa única Mini2440 são abordados na seção 2.5. Na seção 2.6 é apresentada uma definição de sistemas embarcados. A seção 2.7 trata sobre o *framework* Qt. A seção 2.8 define o que é um *toolkit* para Linux embarcado e, por fim, a seção 2.9 apresenta trabalhos correlatos.

2.1 LINUX

O Linux é um sistema operacional *unix-like* cujo marco inicial deu-se em 1991, quando um estudante de graduação chamado Linus Torvalds resolveu desenvolver, por conta própria, um *kernel* para sistema operacional compatível com a norma *Portable Operating System Interface* (POSIX) para que pudesse aprender sobre a nova *Central Processing Unit* (CPU) Intel 80386. A partir da versão 0.2 do *kernel*, Linus recrutou ajuda através da Internet. Em três anos o *kernel* atingiu a versão 1.0 (MAXWELL, 2000, p. 4).

> [...] o Linux não é um sistema operacional na sua totalidade. Quando instala o que comumente é denominado Linux, você está instalando uma enorme quantidade de ferramentas que funcionam em conjunto com um sistema verdadeiramente funcional. O Linux, por si só, é o kernel deste sistema operacional, seu coração, sua mente, seu sistema nervoso. (MAXWELL, 2000, p. 2).

O *kernel* Linux, quando distribuído com um conjunto de ferramentas (softwares), é denominado distribuição. Alguns exemplos de distribuições Linux são Red Hat Linux, Slackware, Ubuntu e Debian.

O Linux foi originalmente desenvolvido para computadores x86 de 32 bits (80386 ou superiores), mas já foi portado para diversas arquiteturas como PowerPC, ARM, SPARC, Itanium e processadores x86 de 64 bits (THE LINUX KERNEL ARCHIVES, 2011a).

> [...] a principal força do Linux está no processo aberto de desenvolvimento. Como o código-fonte está gratuitamente disponível para todos, qualquer um pode criar melhoramentos, que podem então ser gratuitamente disponibilizados para qualquer pessoa. (MAXWELL, 2000, p. 5).

2.2 DEBIAN

O Projeto Debian é uma associação cujos membros compartilham a causa de criar um sistema operacional livre chamado Debian GNU[3]/Linux. O *kernel* utilizado pelo Debian é o *kernel* Linux, desenvolvido inicialmente por Linus Torvalds. Como grande parte das ferramentas básicas são originadas do Projeto GNU, é conhecido como Debian GNU/Linux ou simplesmente Debian. Todas as ferramentas do Projeto GNU também são livres. O Debian possui aproximadamente mil desenvolvedores espalhados pelo mundo que são voluntários em seu tempo livre (DEBIAN, 2011a).

Atualmente o Debian vem sendo utilizado em várias instituições no mundo. No Brasil, pode-se citar a Divisão de Redes do Ministério das Cidades, a Gerência de Infra-Estrutura e Tecnologia (GITEC) da Agência Nacional de Vigilância Sanitária (ANVISA), o Exército Brasileiro e a Gerência de Redes da Eletronorte S/A (DEBIAN, 2011b).

Geralmente esta distribuição é escolhida pelo seu nível de segurança (DEBIAN, 2011c), estabilidade (DEBIAN, 2011d), popularidade (DEBIAN, 2011e) e pelo seu comprometimento com o software livre (DEBIAN, 2011f).

2.3 ARQUITETURA ARM

O processador ARM é o componente chave de muitos sistemas embarcados de 32 bits. O primeiro protótipo, o ARM1, foi concebido em 1985. Na verdade, o núcleo do processador ARM não é um núcleo específico, mas uma família inteira de *designs* que compartilham os mesmos princípios e um conjunto comum de instruções (SLOSS; SYMES; WRIGHT, 2004, p. 3).

O conjunto de instruções ARM é diferente da definição RISC pura para que seja adequado para sistemas embarcados. Entre estas diferenças estão (SLOSS; SYMES; WRIGHT, 2004, p. 6):

a) ciclo variável de execução para certas instruções: nem toda instrução executa em um ciclo de *clock*. Por exemplo, a multiplicação que pode variar o número de

[3] GNU é o acrônimo recursivo de *GNU is Not Unix*.

ciclos de execução dependendo do número de registradores a serem transferidos à memória sequencial ou memória de acesso aleatório;

b) *barrel shifter*[4] de linha: *barrel shifter* de linha é um componente de hardware que pré-processa um dos registradores de entrada antes de ser utilizado por uma instrução;
c) conjunto de instruções Thumb 16 bits: o núcleo do processador ARM possui um conjunto de instruções de 16 bits chamado Thumb que aumenta a densidade do código em cerca de 30% comparado as instruções de 32 bits;
d) execução condicional: uma instrução somente é executada quando uma condição específica é satisfeita. Isso melhora a performance e densidade do código, reduzindo o número de instruções *branch* (*jump*);
e) instruções aprimoradas: instruções para *Digital Signal Processor* (DSP) foram adicionadas ao conjunto de instruções ARM padrão.

Sloss, Symes e Wright (2004, p. 5) afirmam que processadores ARM tem sido especificamente desenhados para serem pequenos e consumirem pouca energia, já que sistemas embarcados portáteis geralmente são alimentados a bateria. Núcleos ARM não são puramente RISC devido às restrições dos sistemas embarcados, que devem ser de baixo custo e ter baixo consumo de energia. Para os sistemas atuais, a performance e o consumo de energia são os fatores-chave na escolha do processador a ser usado.

2.4 ARQUITETURA RISC

O núcleo ARM utiliza arquitetura RISC. A filosofia do design RISC é destinada a fornecer instruções simples que executam em um simples ciclo de *clock*. De acordo com Sloss, Symes e Wright (2004, p. 4), esta filosofia concentra-se em reduzir a complexidade das instruções executadas pelo hardware porque é mais fácil fornecer grande flexibilidade e inteligência a um software do que a um hardware. Já a arquitetura *Complex Instruction Set Computer* (CISC) possui instruções complexas que geralmente levam mais de um ciclo de *clock* para serem executadas.

Sloss, Symes e Wright (2004, p. 4) afirmam que a filosofia da arquitetura RISC é

[4] *Barrel shifter* permite que bits de um registrador sejam rotacionados por um número específico de bits em um único ciclo de *clock*.

baseada em quatro regras principais:

a) instruções: processadores RISC têm um número reduzido de classes de instruções simples que são executadas em um único ciclo de *clock*, sendo o programador ou compilador que sintetizam operações mais complexas, como a divisão de um número inteiro;
b) *pipelines*[5]: o processamento de instruções é feito em pequenas unidades que podem ser executadas em paralelo por *pipelines*, não sendo necessário uma instrução ser executada por um mini-programa (micro-código) como nos processadores CISC;
c) registradores: processadores RISC possuem um grande conjunto de registradores de uso-geral que permitem acesso local rápido a memória;
d) arquitetura *load-store*: o processador opera dados nos seus registradores enquanto instruções separadas de *load-store* fazem a transferência entre o banco de registradores e a memória externa, sendo diferente da arquitetura CISC que efetua operações diretamente na memória.

Isso faz com que os processadores RISC sejam mais simples e possam operar a velocidades de *clock* mais altas que os processadores CISC (SLOSS; SYMES; WRIGHT, 2004, p. 5).

2.5 MINI2440

O Mini2440 é um computador de placa única produzido pela FriendlyARM. Ele é baseado no microprocessador RISC S3C2440A, produzido pela Samsung. Este processador foi desenvolvido com núcleo ARM920T, da arquitetura ARMv4T e pertence a família ARM7TDMI.

O processador S3C2440A implementa *Memory Management Unit* (MMU), barramento *Advanced Microcontroller Bus Architecture*[6] (AMBA) e arquitetura *Harvard*. Possui também um conjunto completo de periféricos integrados *on-chip* que diminuem os custos de desenvolvimento.

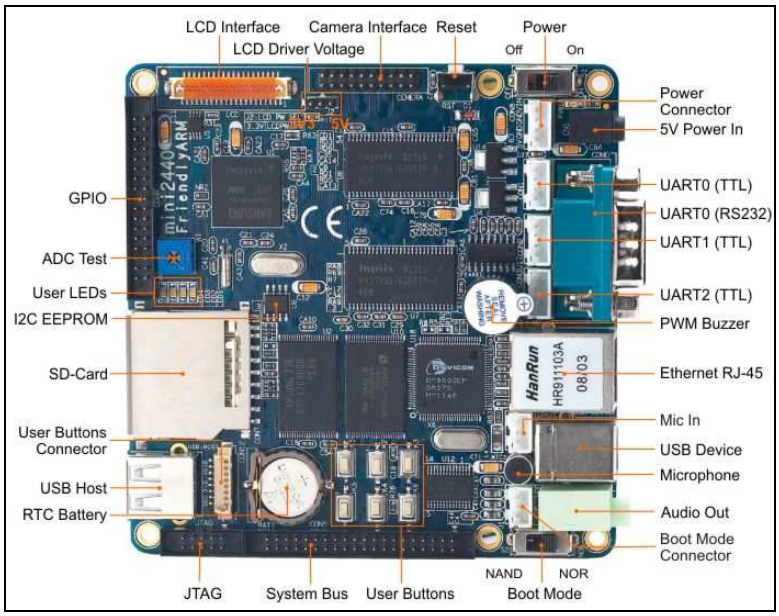
[5] *Pipeline* é uma técnica utilizada para acelerar a velocidade de processamento, carregando uma ou mais instruções para a memória, ao invés de somente a próxima a ser executada.

Dentre estes periféricos *on-chip*, podemos destacar (SAMSUNG, 2005, p. 1-3):

a) controlador de memória externa *Synchronous Dynamic Random Access Memory* (SDRAM);
b) controlador de LCD;
c) controladora *Direct Memory Access* (DMA);
d) controladora *Universal Asynchronous Receiver Transmit* (UART);
e) interface *Audio Codec* ’97 (AC97);
f) interface para cartões de memória SD e *MultiMediaCard* (MMC);
g) *Real Time Clock* (RTC) com função calendário;
h) interface para câmera *Complementary Metal-Oxide-Semiconductor* (CMOS);
i) Conversor Analógico/Digital (ADC);
j) duas portas *Universal Serial Bus* (USB) Host;
k) uma porta USB Device;
l) controlador de memória *Electrically-Erasable Programmable Read-Only Memory* (EEPROM) *Inter-Integrated Circuit* **(**I2C);
m) vinte e quatro portas de interrupção externa;
n) cento e trinta portas de entrada e saída multiplexadas.

O Mini2440 possui seu processador executando a 405 MHz e incorpora 64 MB de SDRAM[7], 64 MB a 1024 MB de *Not AND* (NAND) Flash e várias interfaces e recursos (Figura 1) (FRIENDLYARM, 2010, p. 7).

[6] O barramento AMBA foi introduzido em 1996 e é largamente adotado em processadores ARM. Ele permite que designers de periféricos possam reutilizar o mesmo design em vários projetos.
[7] SDRAM é o acrônimo de *Synchronous Dynamic Random Access Memory*.



Fonte: Friendyarm (2010b, p. 9).

Figura 1– Visão das interfaces e recursos

2.6 SISTEMAS EMBARCADOS

Um sistema embarcado é formado por entradas e saídas e controle lógico armazenados em um *firmware* de sistema. No passado, era simples distinguir um sistema embarcado de um computador de uso-geral. Por exemplo, uma placa T1[8] baseada no 8051 (microcontrolador de 8 bits) era um sistema embarcado de uma estação de trabalho Sun Unix. Hoje, em termos de funcionalidade, é difícil distinguir uma estação de trabalho Sun de uma *set-top box* com *Graphical User Interface* (GUI) que executa um sistema operacional multitarefa capaz de executar vários programas simultaneamente. Os avanços tecnológicos dificultam definir o que é um sistema embarcado (HOLLABAUGH, 2002, p. 8).

Lombardo (2002, p. xv) afirma que a maneira mais fácil de diferenciar um computador de uso-geral de um sistema embarcado é analisando a interface com o usuário. Um computador de uso-geral geralmente possui um teclado, um mouse e um monitor. Pode ser utilizado para navegar na Internet, editar textos e ouvir músicas. Um sistema embarcado pode ou não possuir uma interface com o usuário. Quando possui, geralmente é limitada. As interfaces vão desde simples botões a mostradores do tipo LCD com tela sensível ao toque. Pode ser possível plugar um teclado ou monitor em um sistema embarcado, mas geralmente

isso é utilizado para depuração ou configuração. Estes sistemas são concebidos para realizar operações específicas.

2.7 FRAMEWORK QT

Qt é um *framework* gráfico multiplataforma que permite compilar e executar aplicações em ambientes Windows, Mac OS X, Linux e outros sistemas *unix-like*. Foi desenvolvido originalmente para a linguagem de programação C++, mas hoje possui suporte para outras linguagens como Python, Ruby, PHP e a plataforma .NET (THELIN, 2007, p. 3).

Uma aplicação Qt para Linux embarcado necessita de um servidor de aplicações em execução. Qualquer aplicativo para Linux embarcado pode agir como um servidor. Quando mais de um aplicativo está em execução, as aplicações subseqüentes se conectam ao servidor de aplicações existente. O servidor e o cliente possuem diferentes responsabilidades: o processo servidor gerencia a manipulação do ponteiro do mouse, a entrada de caracteres e a saída de vídeo. O processo cliente executa todas as operações específicas da aplicação (NOKIA, 2010).

A versão anterior da Qt, a 4.6, possui quase 800 classes públicas e mais de um milhão de palavras em sua documentação (SUMMERFIELD, 2010, p. 2). A última versão, a 4.7, possui foco na estabilidade e velocidade. A principal inovação da versão 4.7 é uma nova tecnologia chamada Qt Quick. Esta tecnologia permite criar declarativamente interfaces gráficas de usuário usando uma linguagem semelhante ao Java Script (SUMMERFIELD, 2010, p. 3).

2.8 TOOLKIT PARA LINUX EMBARCADO

A maneira mais simples de embarcar Linux em um dispositivo é fazer *download* da última versão do *kernel* Linux, escolher um compilador-cruzado para a plataforma escolhida, uma biblioteca de linguagem C e alguns aplicativos. O problema é que se consome uma

---

[8] T1 é uma tecnologia que permite transmissão de canais de voz e dados entre dispositivos.

grande quantidade de tempo até ter um sistema que simplesmente efetua o *boot* (LOMBARDO, 2002, p. 65).

Um *toolkit* para Linux embarcado é um produto ou projeto de software livre que, junto com uma estação de trabalho Linux e um dispositivo alvo, fornecem uma solução completa para que seja possível carregar uma imagem com *kernel* Linux para o dispositivo alvo e este execute, pelo menos, um aplicativo simples. Cada *toolkit* tem uma interface diferente com o usuário e estilo próprio. Porém, todos fazem essencialmente a mesma coisa: construir um *kernel* Linux que contém um aplicativo que executa como um *thread* do *kernel* ou construir um sistema de arquivos raiz com o software desenvolvido. O *toolkit* deve ser capaz de instalar novas partes do software desenvolvido em uma imagem binária a partir da qual o dispositivo alvo executa o *boot* (LOMBARDO, 2002, p. 66).

2.9 TRABALHOS CORRELATOS

São encontradas várias ferramentas que auxiliam no desenvolvimento de softwares para sistemas embarcados baseados em Linux. Entre eles está o produto MontaVista Linux (MONTAVISTA, 2009), o *framework* OpenEmbedded Project (HOLGER et al., 2009) e a distribuição Linux uClinux (UCLINUX, 2008a).

2.9.1 MontaVista Linux

O MontaVista Linux 6, desenvolvido pela MontaVista, é um produto que fornece um ambiente de desenvolvimento para sistemas embarcados que inclui um *Market Specific Distribution* (MSD), um *Software Development Kit* (SDK) e suporte especializado (MONTAVISTA, 2009, p. 1).

Os MSDs são distribuições Linux otimizadas para determinados dispositivos. São compostas de um *kernel* Linux e *drivers* apropriados (MONTAVISTA, 2010a). O SDK oferece uma *Integrated Development Environment* (IDE) baseada no Eclipse, compilador C/C++ e bibliotecas de tempo de execução (MONTAVISTA, 2010b). Um dos destaques do MontaVista Linux é o suporte a vários dispositivos embarcados baseados nas arquiteturas

ARM, MIPS, PowerPC, x86, xScale e xTensa (MONTAVISTA, 2010c).

2.9.2 OpenEmbedded Project

O OpenEmbedded Project é um *framework* para Linux embarcado composto por um conjunto de metadados usados para compilação cruzada, empacotamento e instalação de pacotes de software. É utilizado para construir e manter distribuições de Linux embarcado, como a OpenZaurus e Angstrom (HOLGER et al., 2009).

As principais capacidades do OpenEmbedded são (HOLGER et al., 2009):

a) manipular compilação cruzada;
b) manipular dependências entre os pacotes;
c) criar pacotes de software;
d) criar imagens de pacotes;
e) suportar vários dispositivos, distribuições e arquiteturas.

2.9.3 uClinux

O uCLinux, inicialmente, era um projeto derivado do *kernel* Linux versão 2.0 para microcontroladores sem MMU. Hoje é um sistema operacional que inclui *releases* do *kernel* Linux versão 2.0, 2.4 e 2.6, além de aplicativos e bibliotecas (UCLINUX, 2008a). É um projeto designado para sistemas embarcados, tendo cobertura para uma grande variedade de arquiteturas de processadores, entre eles estão os processadores Motorola ColdFire, Motorola DragonBall, Atari 68k e os pertencentes a família ARM7TDMI (UCLINUX, 2008b).

Como o uClinux é desenvolvido para processadores sem MMU, a falta deste implica em algumas limitações como a não implementação de `fork()`[9] e a falta de proteção de memória. Esta falta de proteção de memória permite que qualquer programa acesse a memória de outro programa, fazendo com que o programador tenha um cuidado maior na implementação de softwares embarcados. Embora não implemente `fork()`, o uClinux

[9] O `fork()` é uma chamada de sistema que cria um novo processo filho a partir de um processo pai.

implementa uma outra função chamada `vfork()` que permite que processadores sem MMU forneçam suporte completo a multitarefa (UCLINUX, 2008c).

3 DESENVOLVIMENTO

Para cumprir o desenvolvimento do *toolkit* foram realizados os seguintes passos:

a) levantamento dos principais requisitos (seção 3.1);
b) preparação da estação de trabalho de desenvolvimento (seção 3.2);
c) customização e geração da imagem do *kernel* (seção 3.3);
d) geração do sistema de arquivos raiz (seção 3.4);
e) definição dos mini-aplicativos a serem implementados (seção 3.5);
f) especificação dos mini-aplicativos através de diagrama de casos de uso e diagrama de classes (seção 3.6);
g) implementação dos mini-aplicativos (seção 3.7);
h) documentação dos procedimentos realizados (seção 3.8);
i) testes dos mini-aplicativos (seção 3.9).

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O *toolkit* para Linux embarcado deverá:

a) utilizar sistema operacional Linux Debian para a estação de trabalho de desenvolvimento (Requisito Não-Funcional – RNF);
b) possuir documentação em língua portuguesa (RNF);
c) utilizar o dispositivo alvo Mini2440 (RNF);
d) utilizar linguagem de programação C++ (RNF);
e) utilizar linguagem de *script Shell script* (RNF);
f) criar um sistema de arquivos raiz contendo a distribuição Linux Emdebian (RNF);
g) configurar um *boot-loader* para carga do *kernel* Linux customizado (RNF);
h) carregar imagem do *kernel* e sistema de arquivos raiz para o Mini2440 (RNF);
i) disponibilizar mini-aplicativos que explorem os seguintes recursos de hardware do Mini2440: relógio de tempo real, entradas e saídas digitais de uso-geral, câmera CMOS, módulo Wireless, interface serial RS-232, interface Ethernet, entrada para microfone, saída de áudio estéreo, interface para cartão de memória SD, memória

EEPROM I2C e LCD colorido com tela sensível ao toque (Requisito Funcional – RF).

3.2 ESTAÇÃO DE TRABALHO DE DESENVOLVIMENTO

A versão do Debian utilizada neste trabalho é a última versão estável, a 6.0.3. A Figura 2 mostra a primeira tela da instalação do Debian.

Figura 2 – Primeira tela da instalação do Debian

Durante o processo de instalação, pode-se configurar a interface de rede Ethernet. O Quadro 1 mostra os parâmetros de configuração utilizados. O método de particionamento dos discos utilizado foi o Particionamento Assistido, que utiliza todo espaço disponível do disco rígido para criar o sistema de arquivos e já define automaticamente o tamanho da partição *swap*[10]. Durante o processo de instalação, é questionado quais coleções de software pré-definidas deseja-se instalar. Foi selecionado o Ambiente Gráfico de Trabalho e Utilitários

[10] *Swap* é um tipo de partição utilizado como memória virtual.

Standart de Sistema. Estas opções instalaram o ambiente gráfico Gnome e as ferramentas básicas para o uso do sistema.

Endereço IP:	10.0.0.1
Máscara de rede:	255.255.255.0
Gateway:	<vazio>
DNS:	<vazio>

Quadro 1 – Parâmetros de configuração de rede

O console do Mini2440 pode ser acessado via porta serial, para isto utiliza-se o programa `picocom`. O `picocom` pode ser instalado através do comando `apt-get install picocom`. Para iniciar, utiliza-se o comando `picocom -f n -b 115200 /dev/ttyUSB0`, onde `-f n` refere-se ao desligamento do *flowcontrol*, `-b 115200` ao *baudrate* 115200bps e `/dev/ttyUSB0` ao *device name* da porta serial.

3.3 CUSTOMIZAÇÃO E GERAÇÃO DA IMAGEM DO KERNEL

O Mini2440 já vem embarcado de fábrica com uma imagem do *kernel* Linux versão 2.6.29. O *Bootup logo*[11] possui o logotipo do fabricante FriendlyARM. Além disto, ela não possui alguns recursos essenciais para este trabalho. Entre estes recursos podem-se destacar:

a) suporte aos sistemas de arquivos *Second EXTended filesystem* (ext2) e *Third EXTended filesystem* (ext3): principais sistemas de arquivos utilizados pelas distribuições Linux;
b) suporte a Inotify: Inotify é um sistema de notificação de alteração de arquivos necessário para o funcionamento do `udev`[12];
c) suporte a *Framebuffer Console*: necessário para permitir a rotação (retrato ou paisagem) do console no LCD do Mini2440 e alteração da fonte padrão do console.

A versão do *kernel* utilizada neste trabalho foi a 2.6.32.2 que está disponível para download no site do fabricante. Foi escolhida esta versão específica, pois ela já contém arquivos de configuração pré-ajustados para o hardware do Mini2440 e para outros kits ARM

11 *Bootup logo* é a imagem exibida durante o processo de *boot* do *kernel* Linux.

12 `udev` é um gerenciador de dispositivos dinâmico para Linux (THE LINUX KERNEL ARCHIVES, 2009).

fabricados pela FriendlyARM.

Para iniciar a customização da imagem do *kernel* para uso neste trabalho foi necessário substituir o arquivo de configuração original do *kernel* pelo arquivo de configuração pré-ajustado para o Mini2440. Para tanto, é necessário substituir o arquivo `.config` pelo arquivo `config_mini2440_t35`.

Após a substituição do arquivo configuração, deve-se executar o utilitário de configuração do *kernel* do Linux. O utilitário é executado através do comando `make menuconfig` como usuário `root`. Uma imagem da tela principal do utilitário de configuração do *kernel* Linux pode ser visto na Figura 3.

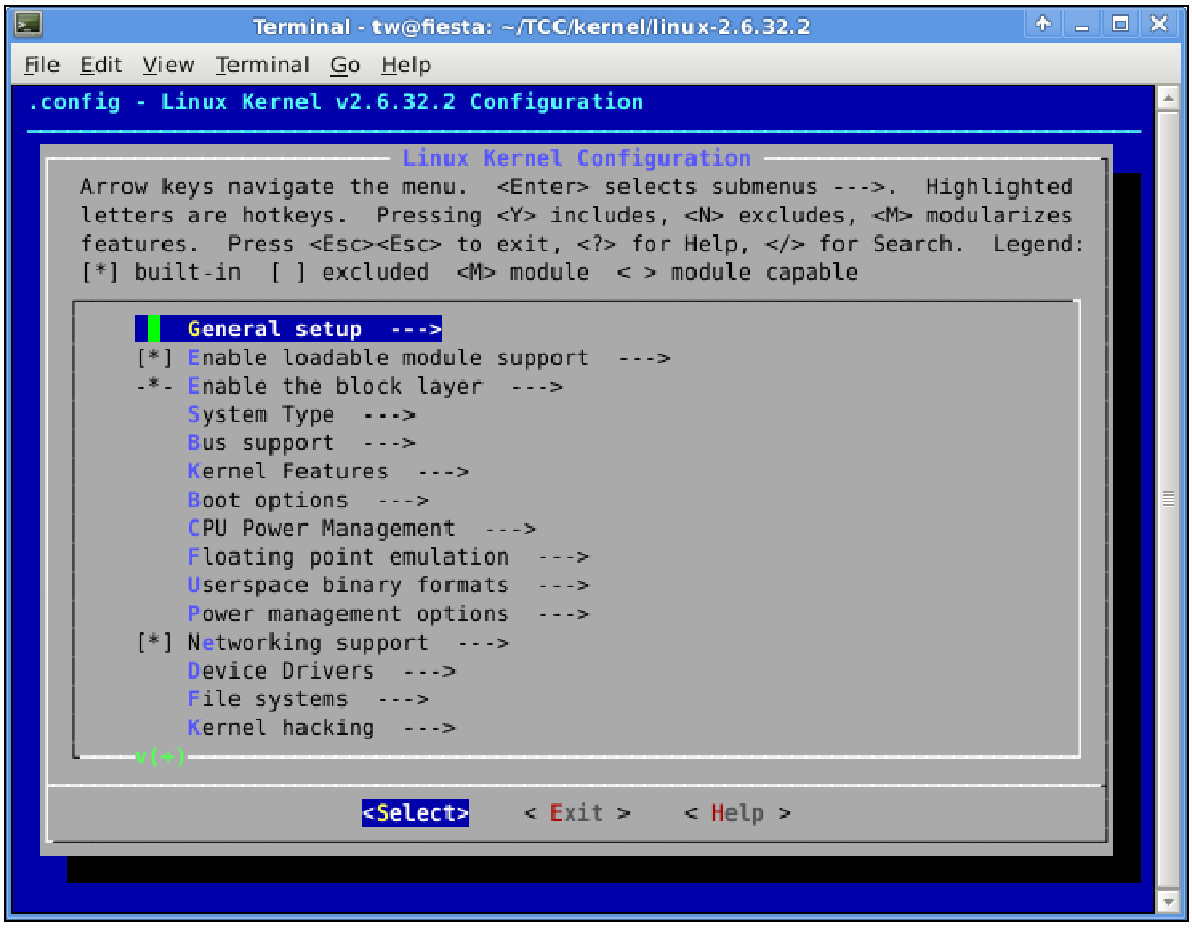


Figura 3– Tela principal do utilitário de configuração do *kernel* Linux

Para ativar o suporte ao ext2, ext3 e Inotify devem-se habilitar as opções descritas no Quadro 2 disponíveis no *submenu File systems*.

```
<*> Second extended fs support
<*> Ext3 journalling file system support
          [*]   Ext3 extended attributes
[*] Inotify file change notification support
[*] Inotify support for userspace
```

Quadro 2 – Suporte ao ext2, ext3 e Inotify

O suporte a *Framebuffer console* é ativado habilitando-se as opções descritas no

Quadro 3. Estas opções são encontradas no *submenu Device Drivers*, seguido pelo *submenu Graphics support* e seguido por último pelo *submenu Console display driver support*.

```
<*> Framebuffer Console support
    [*]   Framebuffer Console Rotation
    [*] Select compiled-in fonts
    [*]   VGA 8x8 font
    [*]   VGA 8x16 font
    [*]   Mac console 6x11 font (not supported by all drivers)
    [*]   console 7x14 font (not supported by all drivers)
    [*]   Pearl (old m68k) console 8x8 font
    [*]   Acorn console 8x8 font
    [*]   Mini 4x6 font
```

Quadro 3 – Suporte a *Framebuffer console*

O *kernel* Linux suporta o uso de *Bootup logos*. O *Bootup logo* pode ser do tipo preto e branco, 16 cores ou 224 cores. Optou-se utilizar o tipo 224 cores pela maior quantidade de cores.

O *Bootup logo* que acompanha os fontes do *kernel* Linux disponibilizado pela FriendlyARM pode ser substituído trocando-se o arquivo `drivers/video/logo/logo_linux_clut224.ppm` por uma imagem do tipo *Portable Pixmap Format* (PPM) de 224 cores.

Um arquivo do tipo *Portable Network Graphics* (PNG) pode ser convertido para o formato PPM de 224 através de uma linha de comandos como visto no Quadro 4.

```
tw@fiesta:~/TCC$ pngtopnm novo-kernel-logo.png | ppmquant 224|
pnmtoplainpnm > logo_linux_clut224.ppm
ppmquant: making histogram...
ppmquant: 200 colors found
ppmquant: choosing 224 colors...
ppmquant: mapping image to new colors...
tw@fiesta:~/TCC$
```

Quadro 4 – Conversão de imagem PNG para PPM

O Quadro 5 compara a *Bootup logo* que acompanha o Mini2440 (esquerda) com a nova imagem (direita) utilizada neste trabalho.

Quadro 5 – Comparação entre imagens

Para compilar a nova imagem do *kernel*, deve-se executar o comando visto no Quadro 6. A compilação em um computador Intel Core 2 Duo de 2.8GHz com 4 GB de RAM leva em média 6 minutos. O resultado do processo é um arquivo chamado `zImage` que é a imagem do *kernel* Linux compactada com o algoritmo da zlib[13].

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
```

Quadro 6 - Como compilar o *kernel*

O U-Boot não é capaz de reconhecer imagens do tipo `zImage`, mas é capaz de reconhecer o tipo `uImage` (ANALOG DEVICES, 2009). Uma imagem do tipo `zImage` pode ser convertida para `uImage` através do utilitário `mkimage`. O Quadro 7 mostra um exemplo de como uma imagem do tipo `zImage` pode ser convertida para `uImage`.

```
fiesta:/home/tw/TCC/kernel/linux-2.6.32.2# mkimage -A arm -O linux -T
kernel -C none -a 0x30008000 -e 0x30008000 -d  arch/arm/boot/zImage
arch/arm/boot/uImage
Image Name:
Created:      Mon Sep 19 10:37:27 2011
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    2472168 Bytes = 2414.23 kB = 2.36 MB
Load Address: 0x30008000
Entry Point:  0x30008000
fiesta:/home/tw/TCC/kernel/linux-2.6.32.2#
```

Quadro 7 - Conversão de imagem do *kernel*

Após a imagem ter sido compilada e convertida, ela deve ser carregada no Mini2440,

[13] zlib é uma biblioteca de compressão de uso geral.

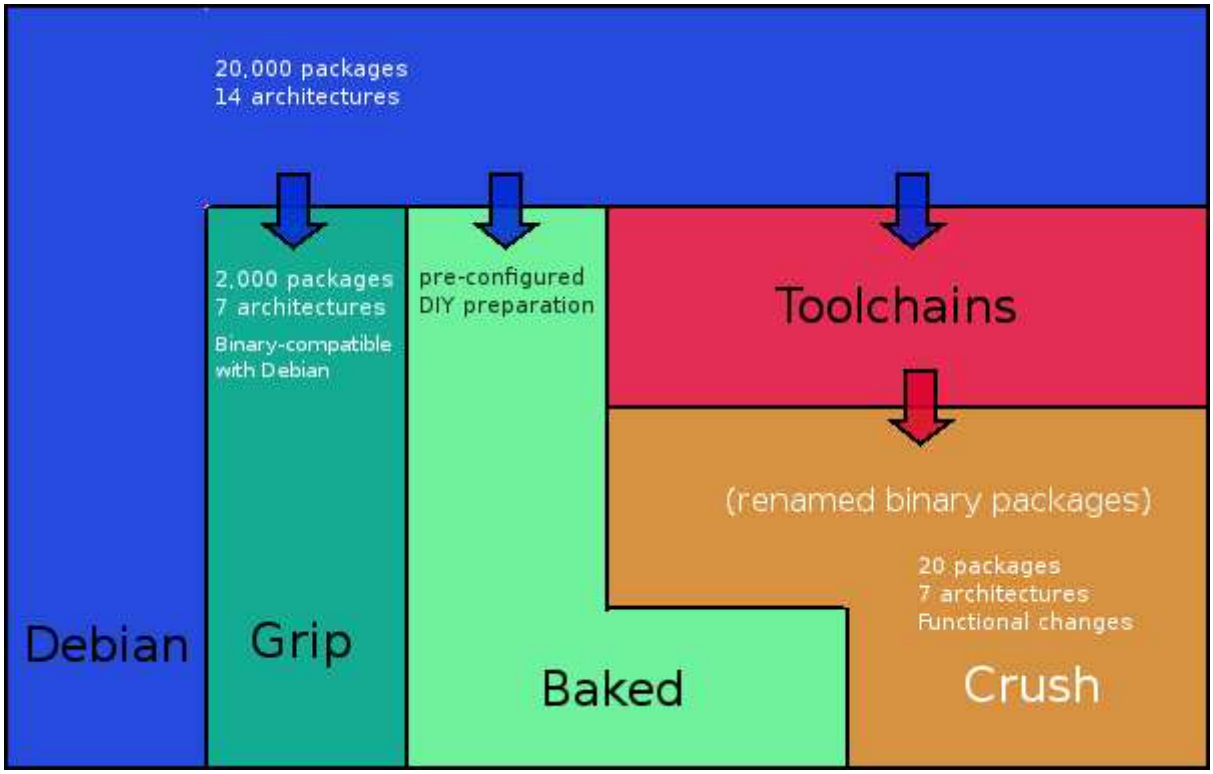
ver Apêndice C.

3.4 GERAÇÃO DO SISTEMA DE ARQUIVOS RAIZ

O Emdebian é uma distribuição Linux baseada no Debian com foco para sistemas embarcados. Existem três possibilidades do Emdebian: Grip, Crush e Baked.

O Emdebian Grip é uma versão mais leve do Debian com poucas mudanças funcionais e o mais alto nível de compatibilidade binária e funcional. Possui a habilidade de misturar e combinar pacotes Debian e Emdebian com o mínimo esforço. O Emdebian Crush possui mudanças funcionais e mudanças em nomes de pacotes. Já o Emdebian Baked é uma versão pré-configurada do Emdebian Grip com os scripts de instalação dos pacotes removidos (EMDEBIAN, 2011).

A Figura 4 compara o Debian com os sabores disponíveis do Emdebian.



Fonte: adaptado de Emdebian (2011).

Figura 4 – Comparação entre Debian e sabores Emdebian

A versão utilizada neste trabalho é a versão Emdebian Grip 2.0 que é baseada na última versão estável do Debian, a versão 6.0. Foi escolhida devido a grande quantidade de pacotes disponíveis e suporte a arquitetura ARM.

`Debootstrap` é uma ferramenta que instala o sistema básico Debian em um subdiretório de outro sistema já instalado. Também pode ser utilizado para criar um sistema de arquivos raiz para uma máquina de arquitetura diferente (DEBIAN WIKI, 2011).

A geração do sistema de arquivos raiz contendo a distribuição Linux Emdebian Grip se dá em dois estágios. No primeiro estágio, os pacotes necessários para se ter um sistema básico funcional são baixados e descompactados em um diretório da estação de trabalho de desenvolvimento. Alguns arquivos também devem ser criados manualmente. No segundo estágio, *scripts* de instalação são executados no dispositivo alvo para finalizar a instalação.

No primeiro estágio, para iniciar a geração do sistema de arquivos raiz com a distribuição Linux Emdebian, deve-se utilizar o comando `debootstrap --arch=armel --foreign squeeze grip/ http://www.emdebian.org/grip/`. Isto gerará um sistema de arquivos raiz para a arquitetura ARM, chamada pela comunidade Debian como `armel` e colocará os arquivos no diretório `grip/`. O Quadro 8 demonstra um trecho da saída para este comando.

```
fiesta:/opt/debian-armel# debootstrap --arch=armel \
--foreign squeeze grip/ http://www.emdebian.org/grip/
I: Retrieving Release
I: Retrieving Packages
I: Validating Packages
.
.
.
I: Extracting tzdata...
I: Extracting util-linux...
I: Extracting xz-utils...
fiesta:/opt/debian-armel#
```

Quadro 8 – Saída do comando `debootstrap`

O diretório `grip/`, que contém o sistema de arquivos raiz, possui aproximadamente 72 MB. Como o Mini2440 possui somente 64 MB de SDRAM e a cópia dos arquivos é feita diretamente para a SDRAM, caso o arquivo que se deseja transferir seja maior que a quantidade de memória SDRAM disponível, o U-Boot reiniciará impedindo a cópia do arquivo. Para reduzir o tamanho total do sistema de arquivos raiz, compactou-se o conteúdo do diretório `usr/`, deixando o sistema de arquivos raiz com aproximadamente 48 MB. Este diretório será descompactado no segundo estágio da instalação do Emdebian no dispositivo alvo. O Quadro 9 demonstra como criar os arquivos essenciais para a primeira inicialização e como reduzir o tamanho total do diretório `usr/`.

```
fiesta:/opt/debian-armel# cd grip
fiesta:/opt/debian-armel/grip# echo "proc /proc proc none 0 0" >>etc/fstab
fiesta:/opt/debian-armel/grip# echo "mini2440" >etc/hostname
fiesta:/opt/debian-armel/grip# mknod dev/ttySAC0 c 204 64
fiesta:/opt/debian-armel/grip# cd usr/
fiesta:/opt/debian-armel/grip/usr# tar -jcf usr.tar.bz2 .
fiesta:/opt/debian-armel/grip/usr# rm -rf bin/ games/ include/ lib/ sbin/
share/ src/
fiesta:/opt/debian-armel/grip/usr# cd ..
fiesta:/opt/debian-armel/grip#
```

Quadro 9 – Finalizando o primeiro estágio

A FriendlyARM disponibiliza uma ferramenta chamada `mkyaffs2image-128M` que possui a função de transformar um diretório em uma imagem de sistemas de arquivos do tipo *Yet Another Flash File System* (YAFFS). O Quadro 10 apresenta um trecho da execução do `mkyaffs2image-128M`.

```
fiesta:/opt/debian-armel# mkyaffs2image-128M grip/ yaffsl.img
mkyaffsimage: image building tool for YAFFS built Apr 29 2008
Processing directory grip/ into image file imagem-yaffs
Object 257, grip//etc is a directory
Object 258, grip//etc/debconf.conf is a file, 6 data chunks written
Object 259, grip//etc/pam.d is a directory
.
.
.
Object 940, grip//lib/libslang.so.2 is a symlink to "libslang.so.2.2.2"
Operation complete.
684 objects in 98 directories
23146 NAND pages
FriendlyARM Computer Technology Inc.
fiesta:/opt/debian-armel#
```

Quadro 10 – Execução do `mkyaffs2image-128M`

Depois de criada a imagem, é necessário transferí-la para o Mini2440 através do U-Boot (ver Apêndice C).

Transferida a imagem e configurado o U-Boot para efetuar o boot automaticamente (Quadro 44), deve-se descompactar o arquivo `usr.tar.bz2` que está dentro do diretório `usr/` do sistema de arquivos raiz. Antes de iniciar o segundo estágio da instalação é necessário criar um novo diretório, caso contrário o segundo estágio abortará. O diretório a ser criado é o `/usr/share/man/man1` que receberá os arquivos de manuais dos programas instalados no segundo estágio. Além disto, é necessário montar o diretório `/proc` e definir a variável `PATH`. O Quadro 11 demonstra como efetuar estes ajustes necessários.

```
sh-3.2# cd /usr
sh-3.2# tar jxf usr.tar.bz2
sh-3.2# rm usr.tar.bz2
sh-3.2# mkdir -p /usr/share/man/man1
sh-3.2# mount /proc /proc -t proc
sh-3.2# export \
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Quadro 11 – Ajustes necessários para o segundo estágio

Para executar o segundo estágio deve-se executar o comando `/debootstrap/debootstrap --second-stage`. Um trecho da saída deste comando pode ser

vista no Quadro 12.

```
# /debootstrap/debootstrap --second-stage
I: Installing core packages...
I: Unpacking required packages...
I: Unpacking base-files...
I: Unpacking base-passwd...
.
.
.
I: Configuring aptitude...
I: Configuring tasksel-data...
I: Configuring tasksel...
I: Base system installed successfully.
save exit: isCheckpointed 1
#
```

Quadro 12 – Saída do `debootstrap` no segundo estágio

Finalizado o `debootstrap`, é necessário habilitar o *prompt* de *login* na porta serial do Mini2440, criar uma senha para o usuário `root` e configurar um *mirror* para o `apt-get`. O *mirror* para `apt-get` permite a instalação dos pacotes que estão no repositório do Emdebian Grip. O Quadro 13 demonstra como fazer estes ajustes finais.

```
sh-3.2# printf "T0:2345:respawn:/sbin/getty -L s3c2410_serial0 115200
vt100" >> /etc/inittab
sh-3.2# passwd # cria senha para o usuario root
sh-3.2# echo 'deb http://www.emdebian.org/grip/ squeeze main contrib non-
free' >>etc/apt/sources.list
```

Quadro 13 – Ajustes finais

Realizados os ajustes finais, deve-se desligar, ajustar o U-Boot conforme Quadro 45 ou Quadro 46 e ligar o Mini2440 para que o Emdebian Linux inicie os serviços necessários para correta execução do sistema operacional.

A configuração da interface de rede do Emdebian é feita da mesma forma que no Debian, através da edição do arquivo `/etc/network/interfaces`. O Quadro 14 possui uma sugestão de configuração para o Emdebian do Mini2440.

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
      address 10.0.0.2
      netmask 255.255.255.0
      network 10.0.0.0
      broadcast 10.0.0.255
      gateway 10.0.0.1
      dns-nameservers 8.8.8.8
```

Quadro 14 – Sugestão de configuração de rede para o Mini2440

3.5 DEFINIÇÃO DOS MINI-APLICATIVOS

Os mini-aplicativos foram definidos com base no recurso de hardware a ser explorado. Cada recurso de hardware será explorado por um mini-aplicativo, com exceção do LCD colorido com tela sensível ao toque. Estes mini-aplicativos, representados por classes no sistema, serão executados a partir de um Lançador de Aplicativos. O Quadro 15 mostra os recursos de hardware e a descrição de cada mini-aplicativo.

Recurso de hardware explorado	**Descrição do mini-aplicativo**
relógio de tempo real	Lê a hora atual do relógio de tempo real
entradas e saídas digitais de uso-geral	Testa os 4 pinos de saída da *General Purpose Input/Output* (GPIO) que estão conectados a *Light-Emitting Diode* (LED) e os 4 pinos de entrada da GPIO que estão conectados a botões
Câmera CMOS	Captura imagens da câmera CMOS e as exibe
módulo Wireless	Lista os *Access Points* que estiverem ao alcance
interface serial RS-232	Envia e recebe caracteres por uma das portas seriais do Mini2440
interface Ethernet	Exibe as principais informações da interface de rede Ethernet como modelo, endereço *Internet Protocol* (IP), endereço físico, número de pacotes enviados e número de pacotes recebidos
entrada para microfone e saída de áudio estéreo	Grava áudio a partir da entrada para microfone e reproduz através da saída de áudio estéreo
interface para cartão de memória SD	Lista arquivos de cartões de memória do tipo SD formatados com o sistema de arquivos *File Allocation Table* (FAT32);
memória EEPROM I2C	Escreve e lê cadeias de bytes da memória EEPROM I2C
LCD colorido com tela sensível ao toque	Todos os mini-aplicativos utilizam a tela de LCD colorido com tela sensível ao toque como interface de entrada e saída padrão

Quadro 15 – Descrição dos mini-aplicativos implementados

3.6 ESPECIFICAÇÃO

Os mini-aplicativos deste toolkit foram especificados utilizando diagramas da UML. Os diagramas de classes e de casos de uso foram elaborados com a ferramenta Enterprise Architect 7.1.

3.6.1 Diagrama de casos de uso

A Figura 5 mostra o diagrama de casos de uso dos mini-aplicativos.

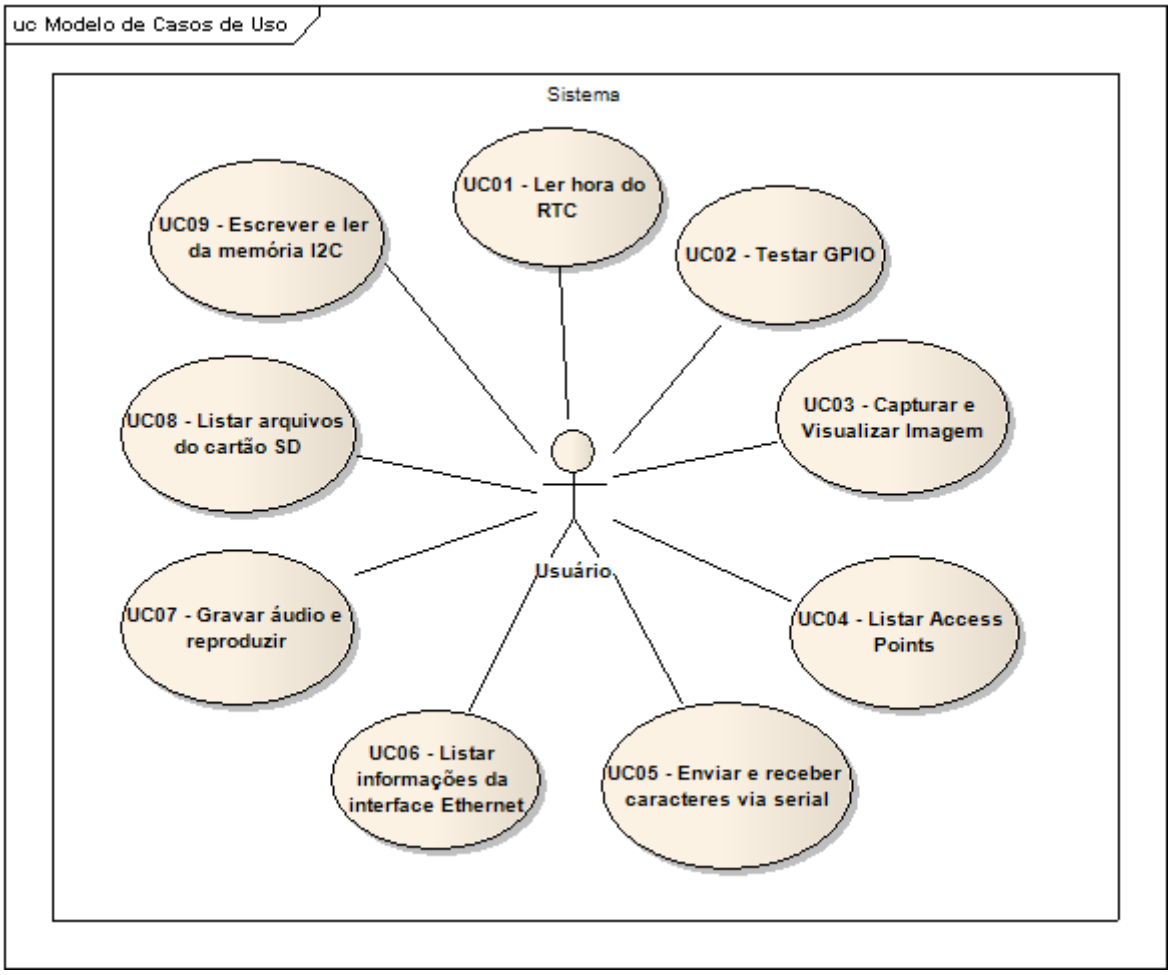


Figura 5 – Diagrama de casos de uso

O Quadro 16 descreve os casos de uso apresentados na Figura 5.

Caso de Uso	**Descrição**
UC01 – Ler hora do RTC	permite o usuário ler a hora do relógio de tempo real
UC02 – Testar GPIO	permite o usuário testar os 4 pinos de saída da GPIO que estão conectados a LEDs e os 6 pinos de entrada da GPIO que estão conectados a botões
UC03 – Capturar e visualiza imagem	permite o usuário capturar imagens da câmera CMOS e vizualizá-las
UC04 – Listar *Access Points*	permite o usuário listar os *Access Points* que estão ao alcance
UC05 – Enviar e receber caracteres via serial	permite o usuário enviar e receber caracteres por uma porta serial
UC06 - Listar informações da interface Ethernet	permite o usuário listar informações da interface Ethernet como modelo, endereço IP, endereço físico, número de pacotes enviados e número de pacotes recebidos
UC07 – Gravar áudio e reproduz	permite o usuário gravar através do microfone e reproduzir o áudio gravado
UC08 – Listar arquivos do cartão SD	permite o usuário listar os arquivos do cartão SD
UC09 – Escrever e ler da memória I2C	permite o usuário escrever e ler cadeias de 256 *bytes* da memória I2C

Quadro 16 – Descrição dos casos de uso

3.6.2 Diagrama de classes

A Figura 6 mostra o diagrama de classes geral do sistema com as funções das classes ocultadas.

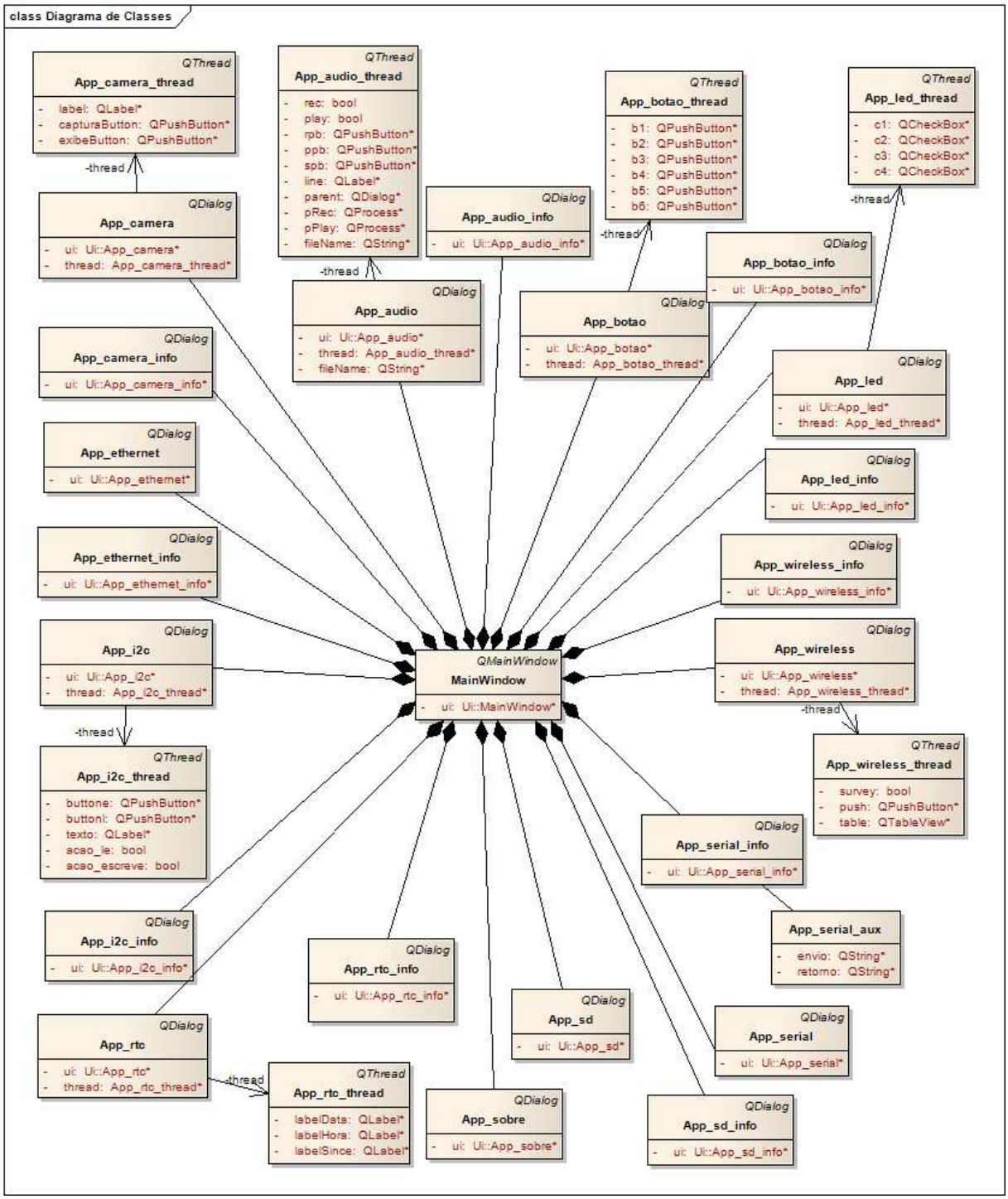


Figura 6 – Diagrama de classes

3.7 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas, ferramentas utilizadas e a operacionalidade da implementação.

3.7.1 Técnicas e ferramentas utilizadas

Para implementação dos mini-aplicativos foi utilizada a linguagem de programação C++. O ambiente de desenvolvimento utilizado foi a IDE QtCreator. Os *scripts* foram implementados em *Shell script.* O funcionamento do Lançador de Mini Aplicativos e dos mini aplicativos e é detalhado nas seções seguintes.

3.7.1.1 Lançador de Mini Aplicativos

O Lançador de Mini Aplicativos é a tela principal do sistema. Ele é executado automaticamente após a inicialização completa do sistema operacional. Consiste em uma tela com botões onde cada botão faz a chamada de um mini aplicativo. O *script* que faz a chamada do Lançador também é responsável por inicializar algumas variáveis de ambiente necessárias para a correta execução do sistema. Devem ser inicializadas as variáveis de ambiente para a `tslib` e para o *framework* Qt. A `tslib` é uma camada de abstração para eventos de painéis *touchscreen* utilizada geralmente em dispositivos embarcados (TSLIB, 2010). Caso não se definam corretamente estas variáveis, o painel *touchscreen* não responderá aos toques e o tamanho da fonte será demasiado pequena, impossibilitando a leitura. O Quadro 17 mostra o *script* `start.sh` de inicialização do sistema.

```
#!/bin/bash
# caso esteja aguardando atualizacao...
if [ -f /root/atualiza.lock ]
then
      echo aguardando atualizacao...
      sleep 2
      exit
fi
echo iniciando...
RUNNING=`pgrep mjpg_streamer`
if [ "$RUNNING" == "" ] ;
then
      echo iniciando daemon de captura de imagens...
      # carrega daemon da camera
      rm -rf /root/cam/*
      export LD_LIBRARY_PATH=/root/mjpg/
      nohup /root/mjpg/mjpg_streamer -b --input "input_s3c2410.so" --output
"output_file.so -f /root/cam/ -d 10 -c /root/mjpg/rename.sh"

      # carrega driver wifi
      insmod /root/vntwusb.ko
      # levanta interface wireless
      ifconfig eth1 up
fi
#limpa tela
clear > /dev/tty1
echo > /dev/tty1
echo > /dev/tty1
echo -n Mais um momento, por favor... > /dev/tty1
#inicia variaveis de ambiente para a Qt e tslib
export TSLIB_TSEVENTTYPE=INPUT
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/event0
export TSLIB_CALIBFILE=/etc/pointercal
export QWS_MOUSE_PROTO=tslib:/dev/input/event0
# ajusta tamanho da fonte
export QWS_DISPLAY=LinuxFB:mmWidth=300:mmHeight=120
# faz cursor parar de piscar
echo 0 > /sys/devices/virtual/graphics/fbcon/cursor_blink
# inicia aplicacao
/root/mini2440 -qws
```

Quadro 17 – *Script* `start.sh`

3.7.1.2 Relógio de Tempo Real

O *kernel* Linux possui dois *frameworks* do espaço de usuário para RTC. O primeiro é o `/dev/rtc`, que é uma *Application Programming Interface* (API) para sistemas *Personal Computer* (PC) compatíveis e a segunda é a `/dev/rtcN` que é parte de um *framework* suportado por uma grande variedade de *chips* RTC. O `N` de `/dev/rtcN` é o número do *chip* RTC, isto é, cada sistema pode ter mais de um RTC. O Mini2440 possui um único RTC que é

compatível com o *framework* `/dev/rtcN`. Este *framework* possui uma interface `sysfs`[14] que fornece acesso a vários atributos do RTC (THE LINUX KERNEL ARCHIVES, 1996). Entre estes atributos, podem-se destacar:

a) `date`: fornece a data no formato AAAA-MM-DD;
b) `since_epoch`: fornece a quantidade de segundos desde 00:00:00 de primeiro de janeiro de 1970, que é uma forma muito comum de representar a data e hora em sistemas *unix-like*;
c) `time`: fornece a hora no formato HH:MM:SS.

Estes atributos são acessados a partir de `/sys/class/rtc/rtc0`. Eles podem ser abertos através da função `open()` e lidos através da função `read()`. O Quadro 18 demonstra um trecho do código que realiza esta leitura.

[14] `sysfs` é um sistema de arquivos virtual do *kernel* que fornece informações sobre dispositivos na área de espaço do usuário.

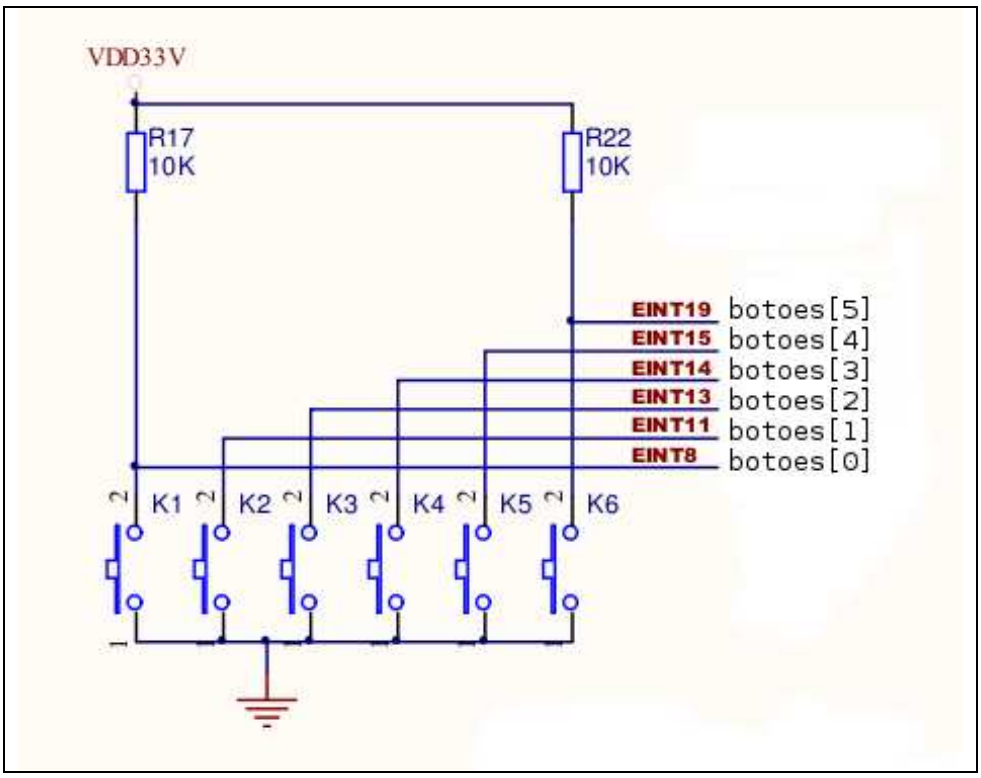
```
// device file da data
int rtc_date_fd;
char date[11]; // 10 bytes + 1 para NULL
char date_br[11]; // 10 bytes + 1 para NULL : para converter data

// device file da hora
int rtc_time_fd;
char time[9]; // 8 bytes + 1 para NULL

// device file do since_epoch
int rtc_since_fd;
char since[11]; // 10 bytes + 1 para NULL

// insere NULL para informar o fim da cadeia de caracteres
date[10]=NULL;
date_br[10]=NULL;
time[8]=NULL;
since[10]=NULL;

forever {
	// realiza a leitura
	// abre fd para leitura
	rtc_date_fd = open("/sys/class/rtc/rtc0/date", 0);
	rtc_time_fd = open("/sys/class/rtc/rtc0/time", 0);
	rtc_since_fd = open("/sys/class/rtc/rtc0/since_epoch", 0);
	// le data
	if (read(rtc_date_fd, date, sizeof date-1) != sizeof date-1) {
		qDebug() << "Erro ao ler date!";
	}
	// le hora
	if (read(rtc_time_fd, time, sizeof time-1) != sizeof time-1) {
		qDebug() << "Erro ao ler time!";
	}
	// le since_epoch
	if (read(rtc_since_fd, since, sizeof since-1) != sizeof since-1) {
		qDebug() << "Erro ao ler since_epoch!";
	}
	// fecha fds
	close(rtc_date_fd);
	close(rtc_time_fd);
	close(rtc_since_fd);

	// aguarda 500ms
	msleep(500);
}
```

Quadro 18 – Leitura do RTC

3.7.1.3 Botões

O Mini2440 possui seis botões do tipo *push-button* que são conectados aos pinos `EINT8`, `EINT11`, `EINT13`, `EINT14`, `EINT15` e `EINT19` do processador S3C2440A. Os estados

destes pinos podem ser acessados através do *device file*[15] `/dev/buttons`. Seis *bytes* podem ser lidos do `/dev/buttons`, cada um representando um botão. Se o valor do *byte* for 1, o botão está pressionado, caso contrário o valor do *byte* será 0. A Figura 7 mostra os seis *bytes* contidos no `/dev/buttons` e seu respectivo botão no Mini2440. O Quadro 19 mostra trecho do código que realiza esta leitura.



Fonte: adaptado de FriendlyARM (2010b).

Figura 7 – Mapeamento `/dev/buttons`

[15] *Device file* é um tipo de arquivo especial que fornece acesso a um periférico (THE SCO GROUP, 2004).

```
// le device file dos botoes
int botao_fd;
char botoes[6];

forever {
	// abre device file dos botoes
	botao_fd = open("/dev/buttons", 0);
	if (read(botao_fd, botoes, sizeof botoes) != sizeof botoes) {
		// se nao conseguir ler, simula botoes pressionados
		botoes[0]='0';botoes[1]='0';botoes[2]='0';
		botoes[3]='1';botoes[4]='1';botoes[5]='1';
	}
	// fecha device file dos botoes
	close(botao_fd);

	// atualiza botoes na tela
	(botoes[5] == '0') ? b1->setText("") : b1->setText("X");
	(botoes[3] == '0') ? b2->setText("") : b2->setText("X");
	(botoes[4] == '0') ? b3->setText("") : b3->setText("X");
	(botoes[2] == '0') ? b4->setText("") : b4->setText("X");
	(botoes[1] == '0') ? b5->setText("") : b5->setText("X");
	(botoes[0] == '0') ? b6->setText("") : b6->setText("X");
	// aguarda 50ms
	msleep(50);
}
```

Quadro 19 – Leitura dos botões do Mini2440

3.7.1.4 LEDs

O Mini2440 possui 4 LEDs conectados aos pinos `GPB5`, `GPB6`, `GPB7` e `GPB8` do processador S3C2440A (INDUSTRIAL ARMWORKS, 2009). Os estados (aceso ou apagado) dos LEDs podem ser acessados através do *device file* `/dev/leds`. Quatro *bytes* podem ser escritos em `/dev/leds`. Se o valor do *byte* for 1, o LED acenderá, se for 0, apagará. O Quadro 20 mostra trecho do código que realiza esta escrita.

```
// file descriptor dos leds
int led_fd;
forever {
      // abre device file dos leds
      led_fd = open("/dev/leds", 0);
      // c1, c2, c3 e c4 são objetos do tipo QCheckBox
      // altera estado do led 0
      (c1->isChecked()) ? ioctl(led_fd, 1, 0) : ioctl(led_fd, 0, 0);
      // altera estado do led 1
      (c2->isChecked()) ? ioctl(led_fd, 1, 1) : ioctl(led_fd, 0, 1);
      // altera estado do led 2
      (c3->isChecked()) ? ioctl(led_fd, 1, 2) : ioctl(led_fd, 0, 2);
      // altera estado do led 3
      (c4->isChecked()) ? ioctl(led_fd, 1, 3) : ioctl(led_fd, 0, 3);
      // fecha fd do led
      close(led_fd);
      msleep(50);
}
```

Quadro 20 – Alteração dos estados dos LEDs

3.7.1.5 Câmera

O Mini2440 pode vir acompanhado de uma câmera CMOS embarcada modelo CAM130. A versão utilizada do *kernel* fornece suporte para este modelo de câmera, mas para captura das imagens foi utilizado o software MJPG-streamer-mini2440 *release* 6 que é uma versão do MJPG-streamer compilado para o Mini2440. O MJPG-streamer é um software que captura imagens e as disponibiliza no formato JPEG. O MJPG-streamer-Mini2440 é iniciado através de linha de comando e captura e salva imagens a um intervalo de tempo definido através do parâmetro `-d`.

Inicialmente foi tentado iniciar o MJPG-streamer-mini2440 para capturar uma imagem cada vez que o botão Captura do mini aplicativo Câmera fosse clicado. O sistema de captura foi testado e demonstrou-se instável. O MJPG-streamer-mini2440 geralmente abortava. Verificando-se a saída da execução, percebeu-se que o mesmo tentava alocar uma página de 4096 KB de memória RAM e, como não era possível, abortava a execução. Isto ocorreu devido à limitação de memória RAM do Mini2440, que é de 64 MB. Aproximadamente 17 MB são utilizados pelo sistema operacional e mais aproximadamente 35 MB pelo sistema desenvolvido, sobrando somente algumas páginas de memória de tamanho inferior a 2048 KB. Quando havia uma página de 4096 KB disponível, o programa executava e finalizava com sucesso, quando não havia, abortava. Para garantir uma página de 4096 KB para o

MJPG-streamer-mini2440, optou-se por iniciá-lo como um *daemon*[16] na inicialização do sistema operacional. Desta forma, ele é iniciado quando ainda há uma ou mais páginas de 4096 KB disponíveis e continua executando até que o Mini2440 seja desligado.

O MJPG-streamer-mini2440 é iniciado com o comando `mjpg_streamer -b --input "input_s3c2410.so" --output "output_file.so -f /root/cam/ -d 10 -c /root/mjpg/rename.sh"`. Para utilizar as imagens capturadas pelo MJPG-streamer-mini2440 foram implementados dois *scripts*. Um *script* é executado cada vez que uma imagem é capturada e outro *script* é executado quando se deseja utilizar uma destas imagens capturadas. O Quadro 21 mostra o *script* `rename.sh` que é executado cada vez que uma imagem é capturada e o Quadro 22 mostra o *script* `captura.sh` que sinaliza que o usuário solicita uma imagem.

```
#!/bin/sh

# verifica se usuario solicitou imagem
# se /root/cam/querofoto nao existe
if [ ! -f /root/cam/querofoto ]
then
      # usuario nao requisitou imagem, apagar atuais
      rm -rf /root/cam/2*
      exit
fi

# usuario solicitou foto, prosseguindo
# verifica quantas imagens foram capturadas
NUM=`ls /root/cam/2*|wc -l`

# aguarda a captura de 5 imagens
if [ ! "$NUM" -gt 5 ]
then
      # se nao possui quantidade suficente de imagens,
      # sai
      exit
fi

# possui quantidade suficiente de imagens, prosseguindo
# apaga imagem anterior
rm -rf /root/cam/cam.jpg
# pega o nome da ultima imagem capturada
ultima=`ls /root/cam/2* -t| head -n 1`
# copia a ultima para cam.jpg
cp $ultima /root/cam/cam.jpg
# apaga a ultima
rm -rf $ultima
# limpa demais imagens
rm -rf /root/cam/2*

# avisa captura.sh que a foto esta disponivel
rm -rf /root/cam/querofoto
```

Quadro 21 – *Script* `rename.sh`

[16] *Daemon* é um programa ou serviço que executa em *background*.

```
#!/bin/sh

# sinaliza para rename.sh que deseja nova imagem
touch /root/cam/querofoto

# enquanto o arquivo nao for excluido...
while [ -f /root/cam/querofoto ] ;
do
      # aguarda foto ficar pronta
      sleep 0.01
done

# apaga foto anterior salva
rm -rf /root/cam/cam_cap.jpg

# copia nova imagem para cam_cap.jpg
cp /root/cam/cam.jpg /root/cam/cam_cap.jpg
```

Quadro 22 – *Script* `captura.sh`

O *script* `rename.sh` aguarda até que 5 imagens sejam capturadas desde o momento em que o *script* `captura.sh` solicita uma imagem. Testes foram realizados a fim de estabelecer este número, já que entre 1 e 4 imagens o MJPG-streamer-mini2440 não exibia a imagem esperada. Ele exibia uma imagem com um grande atraso de tempo. As causas deste comportamento não foram estabelecidas.

3.7.1.6 *Wireless*

A placa USB *Wireless* que acompanha o Mini2440 é a VT6656. A versão compilada do *kernel* utilizada não possui o *driver* para esta placa. Para solucionar este problema, a solução encontrada foi carregar o *driver* da VT6656 na forma de um módulo do *kernel*. Foi utilizado o *driver* para VT6656 versão 1.20.03_x86 que pode ser compilado para o Mini2440 com o comando `make CROSS_COMPILE=arm-linux-gnueabi- -C ../../kernel/linux-2.6.32.2/ M=`pwd``, onde o parâmetro `-C` refere-se ao *path* dos fontes do *kernel* e o parâmetro `M=` refere-se ao local onde será copiado o módulo compilado. O nome *driver* compilado é `vntwusb.ko`. Ele pode ser carregado com comando `insmod vntwusb.ko`. Após carregar o *driver*, é necessário levantar a interface de rede, isto pode ser feito com o comando `ifconfig eth1 up`, onde `eth1` é o nome da interface de rede da placa *Wireless*.

As informações como *Extended Service Set Identification* (ESSID), endereço físico e canal de cada *Access Point* ao alcance são recuperadas através do script `iwlist.sh`, conforme Quadro 23.

```
#!/bin/bash
#define variaveis
filename=/tmp/list.tmp
interface=eth1
# usa nova linha como separador para o for
IFS='
'

# recebe dados
iwlist $interface scan > $filename
grep ESSID $filename > $filename.e
grep Channel: $filename > $filename.c
grep Address $filename > $filename.a
essidlist=`cat $filename.e`

#lista informacoes
i=1;
for linha in $essidlist
do
      # separa linha atual
      essid_linha=`head $filename.e -n $i | tail -n 1`
      channel_linha=`head $filename.c -n $i | tail -n 1`
      address_linha=`head $filename.a -n $i | tail -n 1`
      # retira lixo da linha
      essid=`echo $essid_linha| cut -d ":" -f 2`
      channel=`echo $channel_linha| cut -d ":" -f 2`
      address=`echo $address_linha| cut -d ":" -f 2,3,4,5,6,7`
      # imprime
      echo $essid,$address,$channel
      i=`expr $i + 1`
done
```

Quadro 23 – *Script* `iwlist.sh`

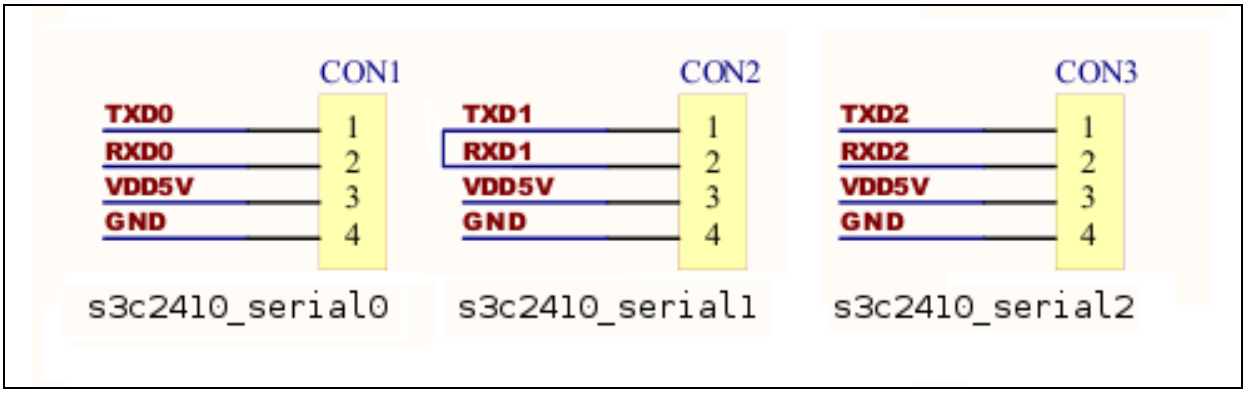
A saída do `iwlist.sh` é processada e seu conteúdo é adicionado a um objeto do tipo `QTableView`.

3.7.1.7 Serial

Segundo Sweet (2005), uma interface RS-232 é um padrão de interface elétrica para comunicações seriais definidos pela *Eletronic Industries Alliance* (EIA).

O Mini2440 possui três interfaces seriais RS-232: `s3c2410_serial0`, `s3c2410_serial1` e `s3c2410_serial2`. Estas interfaces estão disponíveis em `/dev`. A interface `s3c2410_serial0` é reservada para acesso ao console do Mini2440 e as outras duas podem ser utilizadas livremente. Para testar a transmissão e recepção de dados utilizando-se uma mesma porta, pode-se utilizar um *loopback*. Diz-se que uma porta serial está em *loopback* quando o pino TX está conectado ao pino RX. A Figura 8 mostra as três portas acompanhadas dos seus respectivos nomes e o *loopback* feito na porta `s3c2410_serial1` a

fim de testar o mini aplicativo implementado.



Fonte: adaptado de FriendlyARM (2010b).

Figura 8 – Diagrama das portas seriais

O Quadro 24 mostra como a escrita em uma porta serial pode ser implementada.

```
// abre porta
int fd = open("/dev/s3c2410_serial1", O_RDWR | O_NOCTTY | O_NDELAY);

if(fd == -1) // se o open nao tiver sucesso
{
      qDebug() << "Serial: abre_porta: Nao foi possivel abrir a porta.";
      return;
}
else
{
      // define status do arquivo aberto
      fcntl(fd, F_SETFL, 0);
      qDebug() << "Serial: Porta esta aberta.\n";
}

// configura porta
struct termios port_settings;   // estrutura para salvar as
                                // configuracoes da porta

cfsetispeed(&port_settings, B115200);    // define baud rates
cfsetospeed(&port_settings, B115200);

port_settings.c_cflag &= ~PARENB;    // sem paridade,
port_settings.c_cflag &= ~CSTOPB;    // sem bits de parada
port_settings.c_cflag &= ~CSIZE;     // mascara para tamanho dos bits de
dados
port_settings.c_cflag |= CS8;        // 8 bits de dados
port_settings.c_cflag &= ~CRTSCTS; // desabilita hardware flow control

tcsetattr(fd, TCSANOW, &port_settings);    // aplica configuracoes na
porta imediatamente (NOW)

// envia bytes
int n; // n = numero de bytes enviados
QString texto("Texto enviado");
n=write(fd, (const char *)texto->toLatin1() , texto->size());
if(n <= 0)
      qDebug() << "Erro ao enviar bytes.";
else
      qDebug() << "Bytes enviados com sucesso!";
```

Quadro 24 - Como implementar escrita em uma porta serial

3.7.1.8 Ethernet

Segundo Davicom (2006), o DM9000 é um *chip Fast Ethernet* de 10/100Mbps totalmente integrado e de baixo custo. Uma de suas principais características é a de suportar controle de fluxo do tipo *full duplex*. O Mini2440 é embarcado com um DM9000 e a versão do *kernel* utilizada já possui *driver* para este dispositivo.

O comando `ifconfig` exibe várias informações sobre a interface de rede, entre elas podem-se destacar o tipo do encapsulamento, o endereço físico, o endereço IP, o total de pacotes enviados e o total de pacotes recebidos. O Quadro 25 mostra como tratar a saída do comando `ifconfig` utilizando *Shell script*.

```
#!/bin/sh
# define interface
interface=eth0
# le modelo
modelo=`dmesg |grep eth0|head -n 1|cut -d ":" -f 2|cut -d " " -f 2`
# le endereco IP
ip=`ifconfig $interface|grep "inet addr"| cut -d ":" -f 2 | cut -d " " -f
1`
# le MAC Address
mac=`ifconfig $interface|grep "HWaddr"|cut -d " " -f 11`
# le pacotes TX
tx=`ifconfig $interface|grep "TX packets"|cut -d ":" -f 2|cut -d " " -f 1`
# le pacotes RX
rx=`ifconfig $interface|grep "RX packets"|cut -d ":" -f 2|cut -d " " -f 1`

echo Interface: $interface
echo Modelo: $modelo
echo IP: $ip
echo MAC: $mac
echo Pacotes TX: $tx
echo Pacotes RX: $rx
```

Quadro 25 – Exemplo de como tratar a saída do comando `ifconfig`

3.7.1.9 Grava e Reproduz

Segundo NXP (2002), o circuito integrado UDA1341TS é um *codec* econômico para aparelhos de som e aplicações portáteis. Pode-se descrevê-lo também como um Conversor Analógico-Digital e um Conversor Digital-Analógico em um único *chip*. Entre suas características principais, podem-se citar o baixo consumo de energia e alimentação de 3 V. No Mini2440, o UDA1341TS possui uma cápsula de eletreto conectada a uma entrada de áudio e um *jack* J2 estéreo conectado a saída de áudio.

O *kernel* Linux possui um *driver* chamado *Advanced Linux Sound Architecture* (ALSA). Este *driver* é composto de vários módulos, cada módulo com suporte a um ou mais dispositivos de áudio. A ALSA possui um módulo para o UDA1341TS chamado `snd-sa11xx-uda1341` que é carregado automaticamente na inicialização do sistema operacional (THE LINUX KERNEL ARCHIVES, 2011b).

O Emdebian possui um pacote de software chamado `alsa-utils`. Este pacote fornece utilitários para configurar e utilizar o ALSA. Entre os principais utilitários pode-se citar o `arecord` e `aplay`. O `arecord` é um gravador de áudio de linha de comando que suporta vários formatos de arquivo e o `aplay` é um player de áudio de linha de comando. O `arecord` e o `aplay` foram utilizados para implementar um gravador e reprodutor de áudio simples. O Quadro 26 mostra como foi implementado a *thread* do mini aplicativo que grava e reproduz áudio.

```
void App_audio_thread::run() {
	// desativa botoes
	rpb->setEnabled(false);
	ppb->setEnabled(false);

	if(play == true) {
		// toca audio
		QFile file(*fileName);
		if(!file.exists()) {
			// se o arquivo nao existe
			line->setText("Status: Arquivo ainda não gravado.");
		}
		else {
			// se existe, toca
			line->setText("Status: Tocando...");
			pPlay = new QProcess();
			// gera string com comando
			QString comandoPlay("/usr/bin/aplay ");
			comandoPlay.append(fileName);
			qDebug() << "Play: " << comandoPlay ;
			pPlay->start(comandoPlay);
			pPlay->waitForFinished();
			line->setText("Status: Stop");
		}
	}
	if(rec == true) {
		// grava audio
		line->setText("Status: Gravando...");
		// comando para gravar audio:
		// /usr/bin/arecord -d 5 -f cd -t wav /root/file.wav
		// -d = duração
		// -f = qualidade
		// -t = tipo do arquivo
		pRec = new QProcess();
		// gera string com comando
		QString comandoRec("/usr/bin/arecord -d 5 -f cd -t wav ");
		comandoRec.append(fileName);
		qDebug() << "Rec: " << comandoRec ;
		pRec->start(comandoRec);
		pRec->waitForFinished();
		line->setText("Status: Stop");
		spb->setEnabled(true);
	}
	play=false;
	rec=false;
	// ativa botoes
	rpb->setEnabled(true);
	testaPlay();
}
```

Quadro 26 – *Thread* do gravador e reprodutor de áudio

3.7.1.10 Cartão SD

O Mini2440 possui um *slot* para entrada de cartões SD/MMC. O *kernel* Linux possui suporte para estes tipos de cartões, não sendo necessário nenhum *driver* adicional. Quando um

cartão SD é inserido no *slot*, é criado um *device file* chamado `/dev/mmcblkM`, onde `M` é o número seqüencial do dispositivo que sempre começa em 0. Para cada partição do sistema de arquivos do cartão SD, é criado mais um *device file* chamado `/dev/mmcblkMpN`, onde `N` é o número seqüencial da partição que sempre começa em 1. Os arquivos em uma partição do cartão SD não podem ser acessados diretamente através do seu *device file*. Para acessar, é necessário montar o *device file* da partição em algum diretório do sistema de arquivos raiz do sistema operacional. O programa `mount` é capaz de fazer essa montagem. Sua sintaxe básica é `mount /dev/dispositivo /diretorio`. O comando `mount /dev/mmcblk0p1 /sd` monta a primeira partição do cartão SD no diretório `/sd`. A rotina principal do mini aplicativo que exibe os arquivos do cartão SD em formato de árvore pode ser visto no Quadro 27.

```
// verifica se o cartao SD está conectado
QFile *cartao = new QFile("/dev/mmcblk0p1");
if(cartao->exists()) {
      // chama shell script que desmonta o cartao e monta novamente
      QProcess *pSD= new QProcess();
      pSD->start("/root/sdcard.sh");
      pSD->waitForFinished();

      // exibe sistema de arquivos em uma QTreeView
      QFileSystemModel *model = new QFileSystemModel;
      // define path raiz
      model->setRootPath("/sd");
      // define modelo
      ui->treeView->setModel(model);
      // define indice do modelo
      QModelIndex idx = model->index("/sd");
      ui->treeView->setRootIndex(idx);
      ui->treeView->show();
}
else {
      // se nao, limpa treeView
      QFileSystemModel *model = new QFileSystemModel;
      ui->treeView->setModel(model);
      QModelIndex idx;
      ui->treeView->setRootIndex(idx);
      ui->treeView->show();
      // exibe mensagem para usuario
      QMessageBox::information(this,"Cartao SD","Cartao SD nao
conectado.",QMessageBox::Ok);
}
```

Quadro 27 – Rotina que exibe o conteúdo de um cartão SD em formato de árvore

3.7.1.11 Memória I2C

O Mini2440 possui embarcada uma memória EEPROM I2C 24C08. Esta memória é organizada em quatro blocos de 256 bytes cada. Ela suporta um milhão de ciclos de gravação

e possui retenção de 100 anos dos dados gravados (ATMEL, 2009). A memória 24C08 está conectada diretamente aos pinos `I2CSCL` e `I2CSDA` do processador do Mini2440 (FRIENDLYARM, 2010b). A FriendlyARM disponibiliza uma API chamada `24cXX.cpp` que permite a leitura e escrita em memórias do tipo 24C08. Esta memória pode ser acessada através do *device file* `/dev/i2c-0`.

As rotinas implementadas que escrevem e lêem da memória I2C são exibidas no Quadro 28.

```
// le 256 bytes da memoria i2c
void App_i2c_thread::le() {
      struct eeprom e;
      QString saida("");
      // abre device file correspondente a memoria i2c e define que
      // ela possui enderecos de 8 bits
      eeprom_open("/dev/i2c-0", 0x50, EEPROM_TYPE_8BIT_ADDR, &e);
      saida.append("Lido da memoria:\n");
      // realiza a leitura da memoria eeprom a partir do endereco
      // 0 ate contar 256 bytes
      QString *reade = new QString(read_from_eeprom(&e, 0, 256));
      // divive a leitura em linhas de 28 bytes
      // para exibir corretamente na tela do mini2440
      saida.append(split(28,reade));
      // fecha memoria eeprom
      eeprom_close(&e);
      // texto e um objeto do tipo QLabel
      texto->setText(saida);
}

// escreve cadeia de char aleatoria (256 bytes) na memoria eeprom
void App_i2c_thread::escreve() {
      // gera cadeia de char aleatoria
      QString *texto_gerado = new QString("");
      static const char numerico[] = "0123456789";
      for (int i = 0; i < 256; ++i) {
          texto_gerado->append(numerico[rand() % (sizeof(numerico) - 1)]);
      }

      struct eeprom e;
      // abre device file correspondente a memoria i2c e define que
      // ela possui enderecos de 8 bits
      eeprom_open("/dev/i2c-0", 0x50, EEPROM_TYPE_8BIT_ADDR, &e);
      // escreve na memoria eeprom a partir do endereco 0 o texto_gerado
      write_to_eeprom(&e, 0, texto_gerado);
      // fecha memoria eeprom
      eeprom_close(&e);
      QString saida("Escrito na memoria:\n");
      // divive a leitura em linhas de 28 bytes
      saida.append(split(28,texto_gerado));
      // texto e um objeto do tipo QLabel
      texto->setText(saida);
}
```

Quadro 28 – Rotinas de leitura e escrita na memória I2C

3.7.2 Operacionalidade da implementação

O Lançador de Aplicativos, como pode ser visto na Figura 9, é a tela principal do sistema. A partir dele os mini aplicativos são executados e uma descrição sobre eles pode ser vista clicando no botão ?. As sub-seções seguintes apresentam uma breve descrição dos mini aplicativos seguidos de imagens capturadas da tela do Mini2440.

Figura 9 – Tela principal do sistema

3.7.2.1 Relógio de tempo real

A Figura 10 mostra a tela do mini aplicativo que lê os principais parâmetros do RTC e exibe na tela.

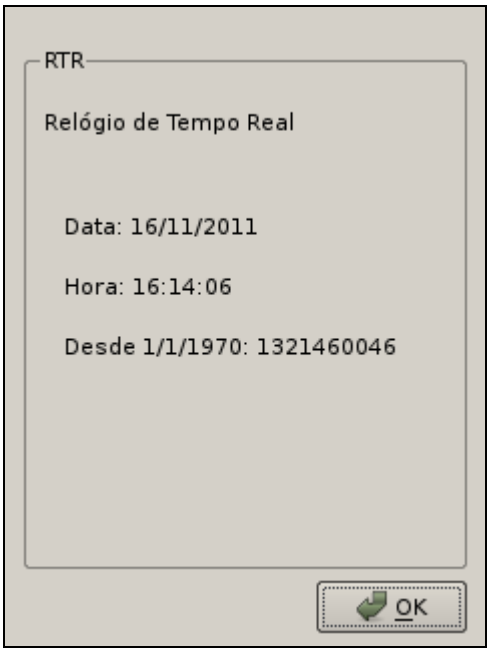


Figura 10 – Tela do relógio de tempo real

3.7.2.2 Botões

A Figura 11 mostra a tela do mini aplicativo que exibe quais botões do Mini2440 estão pressionados no momento.

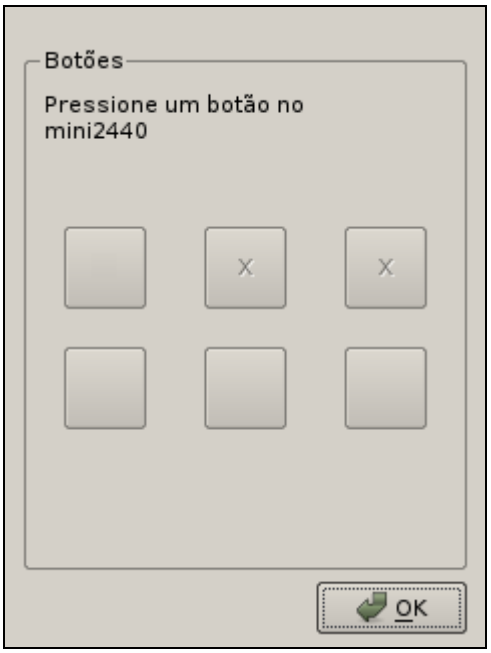


Figura 11 – Tela do mini aplicativo Botões

3.7.2.3 LEDs

A Figura 12 mostra a tela do mini aplicativo LEDs. Os LEDs do Mini2440 acendem de acordo com o estado dos *checkboxes*.

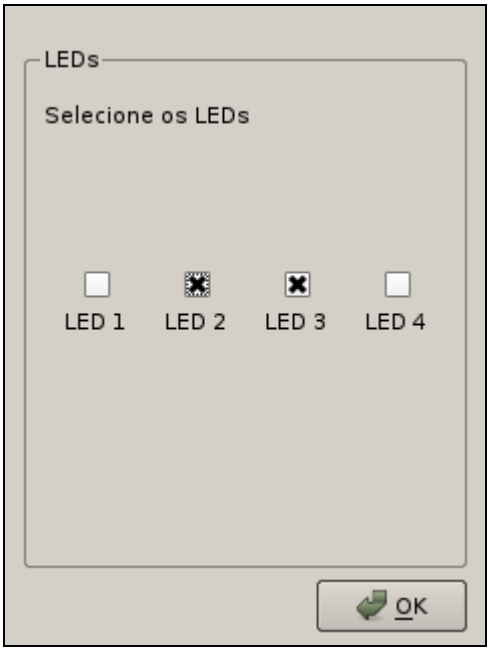


Figura 12 – Tela do mini aplicativo LEDs

3.7.2.4 Câmera

Para capturar uma imagem da câmera, o usuário deve clicar em Captura e aguardar a finalização do processo. Finalizada a captura, o usuário pode visualizar a imagem clicando no botão Exibe, conforme Figura 13.

Figura 13 – Tela do mini aplicativo Câmera

3.7.2.5 *Wireless*

Para listar os *Access Points* ao alcance, o usuário deve clicar no botão *Scan* e aguardar até a lista seja exibida. A Figura 14 mostra a tela deste mini aplicativo.

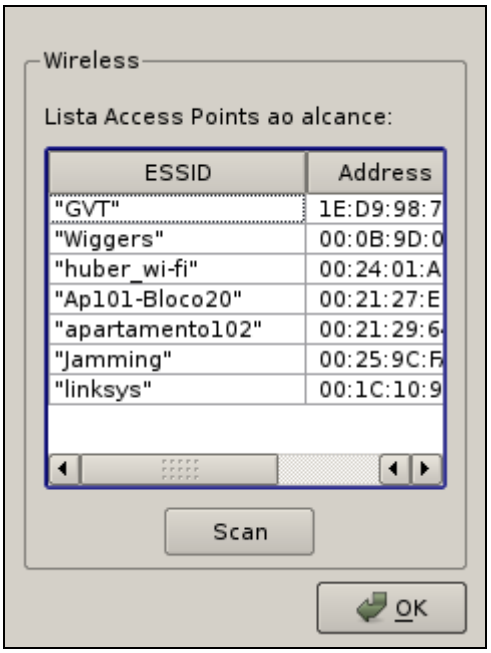


Figura 14 – Tela do mini aplicativo *Wireless*

3.7.2.6 Serial

No mini aplicativo Serial, o usuário pode escolher qual porta serial deseja utilizar: `s3c2410_serial1` ou `s3c2410_serial2`. As configurações da porta são estáticas e foram definidas durante a implementação. Para enviar uma cadeia de *bytes* através da porta serial escolhida, o usuário deve primeiramente gerar um *buffer* de envio através do botão Gera Buffer. Gerado o *buffer*, o usuário pode enviar os *bytes* através do botão Envia. Os *bytes* serão enviados e imediatamente será feita uma leitura na porta selecionada. Se não houver falhas durante a transmissão, o Buffer de Recebimento será igual o Buffer de Envio. A Figura 15 mostra a tela do mini aplicativo Serial.

Figura 15 – Tela do mini aplicativo Serial

3.7.2.7 Ethernet

Este mini aplicativo mostra as principais propriedades do adaptador *Ethernet* do Mini2440, conforme Figura 16.

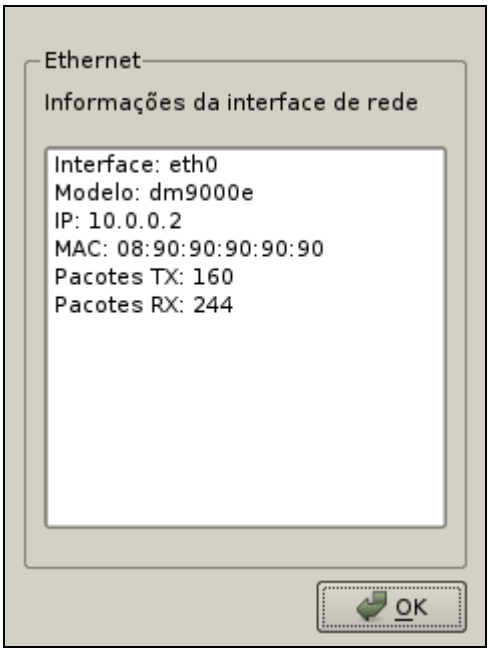


Figura 16 – Tela do mini aplicativo Ethernet

3.7.2.8 Grava e Reproduz

Com este mini aplicativo, o usuário pode gravar uma amostra de áudio de até 5 segundos com taxa de amostragem de 44100 Hz, 16 *bits* e estéreo. Depois de gravado, o áudio pode ser reproduzido. Um botão de *stop* também está disponível caso o usuário prefira interromper a gravação ou a reprodução do áudio. A tela deste mini aplicativo pode ser vista na Figura 17.

Figura 17 – Tela do mini aplicativo Grava e Reproduz

3.7.2.9 Cartão SD

Conforme pode ser visto na Figura 18, o usuário pode exibir o conteúdo cartão SD clicando no botão Listar. Caso nenhum cartão esteja inserido no Mini2440, o mini aplicativo alertará o usuário.

Figura 18 – Tela do mini aplicativo Cartão SD

3.7.2.10 I2C

O mini aplicativo I2C é capaz de escrever e ler na memória 24C08 do Mini2440, conforme Figura 19. Clicando no botão Escrever, uma cadeia aleatória de 256 *bytes* composta por números de 0 a 9 é gerada, exibida na tela e gravada na memória I2C. Clicando no botão Ler, uma cadeia de 256 *bytes* é lida da memória I2C e exibida ao usuário.

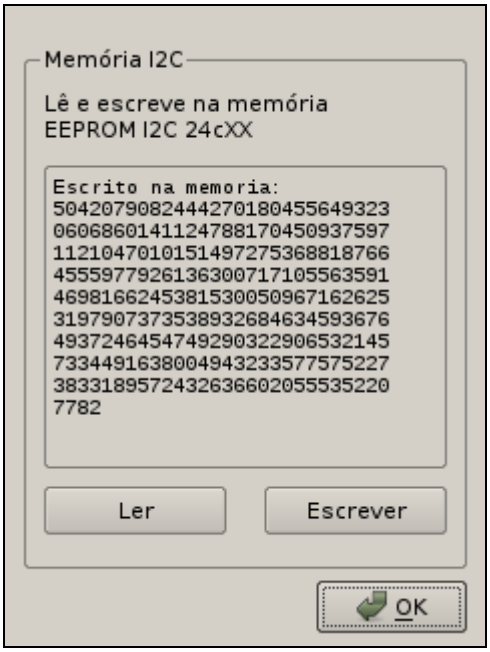


Figura 19 – Tela do mini aplicativo I2C

3.8 DOCUMENTAÇÃO

O procedimento de como instalar um *toolchain* na estação de trabalho de desenvolvimento, necessário para compilar os mini-aplicativos e *kernel*, está descrito no Apêndice A. Para instalar o U-Boot no Mini2440, necessário para efetuar o *boot* do sistema operacional, deve-se seguir o procedimento descrito no Apêndice B. O procedimento de como carregar a imagem do sistema de arquivos e a imagem do *kernel* para o Mini2440 está descrito no Apêndice C.

Todos os mini aplicativos possuem suas principais rotinas comentadas e também possuem um *help* que descreve brevemente o seu funcionamento.

3.9 TESTES

Grande parte dos testes executados durante o desenvolvimento do sistema foram feitos utilizando a classe `QDebug` do *framework* Qt. Esta classe é específica para depuração e rastreamento de informações do programa. Com ela, pode-se imprimir mensagens e informações no console durante a execução do sistema.

A primeira parte dos testes da interface de usuário foram realizados na própria estação de trabalho de desenvolvimento, compilando-se e executando-se o sistema. Isto foi possível devido a grande portabilidade da linguagem C++ e da *framework* Qt. A parte final dos testes foi realizada executando-se o sistema no próprio Mini2440. Para facilitar a transferência do arquivo executável do sistema para o Mini2440, foi implementado o *script* `atualiza.sh`, conforme Quadro 29.

```
#!/bin/sh
# monta sistema de arquivos da estacao de trabalho de desenvolvimento
/root/nfs.sh
# informa ao script start.sh que ocorrera uma atualizacao
touch /root/atualiza.lock
# para script de inicializacao
pkill start.sh
# para aplicacao em executacao
pkill mini2440
# apaga versão antiga da apliacao
rm -rf /root/mini2449
# copia nova versao
cp /nfs/mini2440-ARM/mini2440 /root/mini2440
# avisa start.sh que terminou a atualizacao
rm -rf /root/atualiza.lock
echo atualizado
```

Quadro 29 – *Script* `atualiza.sh`

3.10 RESULTADOS E DISCUSSÃO

Este trabalho apresentou resultados satisfatórios, permitindo o desenvolvimento de aplicativos para Linux embarcado sem que seja necessário gastar uma enorme quantidade de tempo inicial por parte do desenvolvedor. Este *toolkit* disponibiliza os seguintes itens para carga e execução no Mini2440:

a) uma imagem do *kernel* Linux customizada;

b) uma imagem do sistema de arquivos raiz contendo a distribuição Linux Emdebian

combinada com *scripts* e bibliotecas que permitem a inicialização do Lançador de Aplicativos;

c) mini-aplicativos que exploram os principais recursos de hardware do Mini2440.

O tempo para a inicialização completa do sistema, contado desde o momento em que o Mini2440 é ligado até a exibição da tela do Lançador de Aplicativos, é de aproximadamente 45 segundos.

O Mini2440 possui uma grande variedade de periféricos, mas dependendo do sistema que se deseja desenvolver, o hardware pode ser considerado um tanto limitado. Uma das limitações que mais se destaca é a quantidade de memória RAM disponível, que é de 64 MB. Para aplicações com GUI, esta quantidade pode ser insuficiente.

O Quadro 30 apresenta um comparativo entre este trabalho e os trabalhos correlatos.

Função	**Toolkit**	**MonstaVista**	**OpenEmbedded**	**uClinux**
Documentação em Língua Portuguesa	X			
Kernel otimizado para dispositivo alvo	X	X		
Aplicativos que exploram recursos de hardware	X			
Suporte a arquitetura ARM	X	X	X	X
IDE		X		
Suporte a processadores sem MMU				X

Quadro 30 – Comparação com trabalhos correlatos

4 CONCLUSÕES

Sobre as ferramentas utilizadas neste trabalho, pode-se afirmar que o *framework* Qt mostrou-se adequado por possuir uma grande quantidade de classes e ampla documentação disponível, além de ser multiplataforma. A IDE Qt Creator mostrou-se produtiva devido ao seu *designer* de interfaces de usuário ser semelhante do Delphi e a integração com o *framework* Qt.

Embora muitas dificuldades tenham sido encontradas durante a realização deste trabalho, todas foram, de certa forma, superadas cumprindo assim os objetivos iniciais. Dentre as principais dificuldades pode-se citar: limite de memória RAM disponível, documentação esparsa sobre o tema e dificuldades na geração e carga do sistema de arquivos raiz. Outra limitação, que não é tão saliente, é a velocidade do *clock* do processador do Mini2440, que trabalha a 405 MHz. Testes realizados demonstraram grande latência, de aproximadamente 1 segundo, na resposta da interface gráfica do usuário.

Este trabalho possui algumas limitações, tais como:

a) desempenho regular da interface gráfica de usuário devido ao alto *overhead* do *framework* Qt e ao *daemon* MJPG-streamer-mini2440;
b) ausência de mini-aplicativos que explorem outros recursos de hardware do Mini2440 tais como *Pulse-Width Modulation* (PWM) e porta *USB-Device*;
c) geração manual da imagem do *kernel* Linux e do sistema de arquivos raiz.

Como principais vantagens deste trabalho, podem-se destacar:

a) uso de softwares livres para o desenvolvimento, compilação, carga e execução de aplicativos;
b) documentação de procedimentos e códigos-fonte em língua portuguesa;
c) uso de *framework* multiplataforma capaz de executar em diferentes plataformas de hardware e sistemas operacionais;
d) uso de arquitetura ARM que é adequada para sistemas embarcados devido seu baixo custo e baixo consumo de energia;
e) uso de hardware com mostrador do tipo LCD com *touch-screen*, tecnologia amplamente utilizada em *smartphones* e *tablet PCs*;
f) código fonte disponibilizado sob a licença GNU General Public License v3.0 (FREE SOFTWARE FOUNDATION, 2007), que promove a liberdade de execução, estudo, redistribuição e alteração do código fonte por qualquer

indivíduo.

A realização deste trabalho mostrou que é viável o desenvolvimento de aplicações embarcadas utilizando hardware de baixo custo e baixo consumo utilizando softwares livres, sem a necessidade de gastos excessivos com licenciamento de software e pagamento de *royalties*.

4.1 EXTENSÕES

Algumas melhorias podem ser associadas a este trabalho, tais como:

a) desenvolvimento de um *wizard* capaz de gerar imagens do *kernel* Linux e imagens de sistema de arquivos raiz de acordo com as necessidades do usuário;
b) otimização do processo de *boot* do sistema operacional, afim de diminuir o tempo de inicialização completo do sistema;
c) otimização do processo de captura e exibição de imagens;
d) integração com módulos *General Packet Radio Service* (GPRS) e *Global System for Mobile communications* (GSM), tal como o TC65i (CINTERION, 2011), para o desenvolvimento de soluções de monitoramento ambiental automático;
e) desenvolvimento de protótipos baseados neste *toolkit* tais como roteadores, interfaces homem-máquina, *thin clients*, *tablet PCs*, *netbooks* e telefones IP.

REFERÊNCIAS BIBLIOGRÁFICAS

ANALOG DEVICES. **Bootable U-Boot images**. [S.l.], 2009. Disponível em: <https://docs.blackfin.uclinux.org/doku.php?id=bootloaders:u-boot:uimage&do=revisions>. Acesso em: 4 out. 2011.

ATMEL. **Two-wire serial EEPROM**. [S.l], 2009. Disponível em: <http://www.atmel.com/dyn/resources/prod_documents/doc5226.pdf>. Acesso em: 20 set. 2011.

CINTERION. **Cinterion wireless modules GmbH-TC65i**. [S.l.], 2011. Disponível em: <http://www.cinterion.com/products/m2m-evolution/connector/tc65i.html>. Acesso em: 25 out. 2011.

DAVICOM. **DM9000**. Taiwan, 2006. Disponível em: <http://www.davicom.com.tw/userfile/24247/DM9000-DS-F03-041906_1.pdf>. Acesso em: 2 out. 2011.

DEBIAN WIKI. **Debootstrap**. [S.l.], 2011. Disponível em: <http://wiki.debian.org/Debootstrap>. Acesso em: 3 out. 2011.

DEBIAN. **Sobre o Debian**. [S.l.], 2011a. Disponível em: <http://www.debian.org/intro/about>. Acesso em: 15 set. 2011.

_____. **Quem está utilizando o Debian?** [S.l.], 2011b. Disponível em: <http://www.debian.org/users>. Acesso em: 15 set. 2011.

_____. **Quem está utilizando o Debian?** [S.l.], 2011c. Disponível em: <http://www.debian.org/users/gov/cidades>. Acesso em: 15 set. 2011.

_____. **Quem está utilizando o Debian?** [S.l.], 2011d. Disponível em: <http://www.debian.org/users/gov/anvisa>. Acesso em: 15 set. 2011.

_____. **Quem está utilizando o Debian?** [S.l.], 2011e. Disponível em: <http://www.debian.org/users/gov/exercitobrasileiro>. Acesso em: 15 set. 2011.

_____. **Quem está utilizando o Debian?** [S.l.], 2011f. Disponível em: <http://www.debian.org/users/edu/parana_math>. Acesso em: 15 set. 2011.

EMDEBIAN. **Emdebian distributions**. [S.l.], 2011. Disponível em: <http://www.emdebian.org/emdebian/flavours.html>. Acesso em: 5 out. 2011.

FREE SOFTWARE FOUNDATION. **The GNU General Public License v3.0**. Boston, 2007. Disponível em: <http://www.gnu.org/copyleft/gpl.html>. Acesso em: 10 ago. 2011.

FRIENDLYARM. **FriendlyARM Mini2440**. [S.l.], [2010a?]. Disponível em: <http://www.friendlyarm.net/dl.php?file=Mini2440_manual.pdf>. Acesso em: 22 mar. 2011.

_____. **Schematic diagram**: FriendlyARM Mini2440. [S.l.], 2010b. Disponível em <http://www.friendlyarm.net/dl.php?file=mini2440_schematic.zip>. Acesso em: 15 set. 2011.

HOLGER, Hans P. F. et al. **OpenEmbedded user manual**. [S.l.], 2009. Disponível em: <http://docs.openembedded.org/usermanual/usermanual.html>. Acesso em: 21 mar. 2011.

HOLLABAUGH, Craig. **Embedded Linux**: hardware, software, and interfacing. New York: Sams, 2002.

INDUSTRIAL ARMWORKS. **Mini2440 hardware essentials**. [S.l.], 2009. Disponível em: <http://www.andahammer.com/assets/Uploads/All2440/Firmware/Mini2440Essentials.pdf>. Acesso em: 16 ago. 2011.

LOMBARDO, John. **Embedded Linux**. Indianapolis: New Riders, 2002.

MAXWELL, Scot. **Kernel do Linux**. Tradução Carlos Mink. São Paulo: Makron Books, 2000.

MONTAVISTA. **Monta Vista Linux 6**. Santa Clara, 2009. Disponível em: <http://www.mvista.com/download/fetchdoc.php?docid=427>. Acesso em: 20 mar. 2011.

_____. **Monta Vista Linux 6**: market especific distribution. [S.l.], 2010a. Disponível em: <http://www.mvista.com/product_detail_msd.php>. Acesso em: 20 mar. 2011.

_____. **Monta Vista Linux 6**: software development kit. [S.l.], 2010b. Disponível em: <http://www.mvista.com/product_detail_sdk.php>. Acesso em: 20 mar. 2011.

_____. **Platform support for MontaVista Linux**. [S.l.], 2010c. Disponível em: <http://www.mvista.com/boards.php>. Acesso em: 20 mar. 2011.

NETWORK WORKING GROUP. **The TFTP protocol (revision 2)**. Massachusetts: MIT, 1981. Disponível em: <http://www.ietf.org/rfc/rfc783.txt>. Acesso em: 1 out 2011.

NOKIA. **Qt 4.6**: Qt for embedded Linux architecture. [S.l.], 2010. Disponível em: <http://doc.qt.nokia.com/4.6/qt-embedded-architecture.html>. Acesso em: 10 set. 2011.

NXP. **UDA1341TS economy audio CODEC for MiniDisc home stereo and portable applications**. Netherlands, 2002. Disponível em: <http://www.nxp.com/documents/data_sheet/UDA1341TS.pdf>. Acesso em: 20 out. 2011.

SAMSUNG. **S3C2440A application note**. [S.l.], [2005?]. Disponível em: <http://www.samsung.com/global/system/business/semiconductor/product/2010/5/3/919472S3C32440_Application_Note_Rev10.pdf>. Acesso em: 20 mar. 2011.

SLOSS, Andrew N.; SYMES Dominic; WRIGHT Chris. **ARM system developer's guide**: design and optimizing system software. San Francisco: Morgan Haufmann, 2004.

SUMMERFIELD, Mark. **Advanced Qt programming**: creating great software with C++ and Qt 4. Massachusetts: Courier, 2010.

SWEET, Michael R. **Serial programming guide for POSIX operating systems**. [S.l.], 2005. Disponível em: <www.easysw.com/~mike/serial/serial.html>. Acesso em: 10 out. 2011.

THE DENX U-BOOT AND LINUX GUIDE FOR CANYONLANDS. **Abstract**. [S.l.], 2005. Disponível em: <http://www.denx.de/wiki/view/DULG/Manual>. Acesso em: 1 out. 2011.

_____. **Flattened device tree blob**. [S.l.], 2010. Disponível em: <http://www.denx.de/wiki/view/DULG/Manual>. Acesso em: 1 out. 2011.

THE LINUX KERNEL ARCHIVES. **What is Linux?** [S.l.], [2011a?]. Disponível em: <http://www.kernel.org>. Acesso em: 25 mar. 2011.

_____. **Advanced Linux sound architecture**. [S.l.], [2011b?]. Disponível em: <http://www.kernel.org/doc/Documentation/sound/alsa/ALSA-Configuration.txt>. Acesso em: 6 out. 2011.

_____. **Real time clock drivers for Linux**. [S.l.], [1996?]. Disponível em: <http://www.kernel.org/doc/Documentation/rtc.txt>. Acesso em: 6 out. 2011.

_____. **Udev**. [S.l.], [2009?]. Disponível em: <http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html>. Acesso em: 3 out. 2011.

THE SCO GROUP. **Device files**. Utah, 2004. Disponível em: <http://uw714doc.sco.com/en/SHL_using/_Device_files.html>. Acesso em: 1 out. 2011.

THELIN, Johan. **Foundations of Qt development**. New York: Apress, 2007.

TSLIB. **About**. [S.l.], 2010. Disponível em: <http://tslib.berlios.de/>. Acesso em: 20 out. 2011.

UCLINUX. **What is uClinux?** [S.l.], [2008a?]. Disponível em: <http://www.uclinux.org/description/>. Acesso em: 18 mar. 2011.

_____. **uClinux:** ports. [S.l.], [2008b?]. Disponível em: <http://www.uclinux.org/ports/>. Acesso em: 18 mar. 2011.

_____. **uClinux:** FAQ. [S.l.], [2008c?]. Disponível em: <http://www.uclinux.org/pub/uClinux/FAQ.shtml>. Acesso em: 18 mar. 2011.

WEINBERG, William. Foreword. In: HOLLABAUGH, Craig. **Embedded Linux**: hardware, software, and interfacing. New York: Sams, 2002. p. ix.

APÊNDICE A – Instalando *toolchain* no Linux Debian 6.0

Antes de customizar e compilar uma imagem do *kernel* para o Mini2440 é necessário instalar um *toolchain* na estação de trabalho de desenvolvimento.

A distribuição Linux Emdebian possui em seu repositório de pacotes um *toolchain* para a distribuição Linux Debian 6.0. Este *toolchain* é composto de um compilador C, um compilador C++, um *assembler*, um *linker*, uma biblioteca C para dispositivos embarcados e utilitários que permitem a compilação cruzada.

Para instalar, é necessário adicionar a informação contida no Quadro 31 ao final do arquivo `/etc/apt/sources.list` da estação de trabalho de desenvolvimento.

```
deb http://www.emdebian.org/debian/ squeeze main
```

Quadro 31 – Configuração do repositório Emdebian

Depois, é necessário atualizar a lista de pacotes do repositório com comando visto no Quadro 32.

```
fiesta:~# apt-get update
```

Quadro 32 – Comando para atualização da lista de pacotes do repositório

Os pacotes que estão no repositório do Linux Emdebian são assinados digitalmente. Caso tente-se instalar um pacote do repositório, a mensagem de erro que pode ser vista no Quadro 33 é exibida.

```
W: GPG error: http://www.emdebian.org squeeze Release: The following
signatures couldn't be verified because the public key is not available:
NO_PUBKEY B5B7720097BB3B58
```

Quadro 33 – Mensagem de erro ao instalar um pacote

Para evitar esta mensagem de erro, é necessário instalar o pacote `emdebian-archive-keyring`. Este pacote possui as chaves públicas do repositório Emdebian que serão utilizadas para validar a assinatura dos pacotes. Para instalar este pacote deve-se utilizar o comando visto no Quadro 34.

```
fiesta:~# apt-get install emdebian-archive-keyring
```

Quadro 34 – Comando para instalação do *keyring*

Uma mensagem semelhante a do Quadro 35 será exibida.

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  emdebian-archive-keyring
0 upgraded, 1 newly installed, 0 to remove and 33 not upgraded.
Need to get 5,832 B of archives.
After this operation, 57.3 kB of additional disk space will be used.
Get:1 http://debian.pop-sc.rnp.br/debian/ squeeze/main emdebian-archive-
keyring all 2.0.1 [5,832 B]
Fetched 5,832 B in 0s (220 kB/s)
Selecting previously deselected package emdebian-archive-keyring.
(Reading database ... 119666 files and directories currently installed.)
Unpacking emdebian-archive-keyring (from .../emdebian-archive-
keyring_2.0.1_all.deb) ...
Setting up emdebian-archive-keyring (2.0.1) ...
OK
```

Quadro 35 – Instalação do keyring

Para instalar o *toolchain*, deve-se executar o comando visto no Quadro 36.

```
fiesta:~# apt-get install binutils-arm-linux-gnueabi cpp-4.3-arm-linux-
gnueabi gcc-4.3-arm-linux-gnueabi-base gcc-4.4-base-armel-cross libc-bin-
armel-cross libc-dev-bin-armel-cross libc6-armel-cross
```

Quadro 36 – Comando para instalação do toolchain

Uma mensagem semelhante à vista no Quadro 37 deve ser exibida.

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  binutils-doc gcc-4.3-locales libmudflap0-4.3-dev-armel-cross gcc-4.3-doc
libgcc1-dbg-armel-cross libgomp1-dbg-armel-cross libmudflap0-dbg-armel-
cross
The following NEW packages will be installed:
  binutils-arm-linux-gnueabi cpp-4.3-arm-linux-gnueabi gcc-4.3-arm-linux-
gnueabi gcc-4.3-arm-linux-gnueabi-base gcc-4.4-base-armel-cross libc-bin-
armel-cross libc-dev-bin-armel-cross
  libc6-armel-cross libc6-dev-armel-cross libgcc1-armel-cross linux-libc-
dev-armel-cross
0 upgraded, 11 newly installed, 0 to remove and 33 not upgraded.
Need to get 12.6 MB of archives.
After this operation, 23.1 MB of additional disk space will be used.
Do you want to continue [Y/n]?
Get:1 http://www.emdebian.org/debian/ squeeze/main gcc-4.3-arm-linux-
gnueabi-base amd64 4.3.5-4 [110 kB]
Get:2 http://www.emdebian.org/debian/ squeeze/main binutils-arm-linux-
gnueabi amd64 2.20.1-16 [4,670 kB]
.
.
.
Fetched 12.6 MB in 8s (1,543 kB/s)
Selecting previously deselected package gcc-4.3-arm-linux-gnueabi-base.
(Reading database ... 119673 files and directories currently installed.)
Unpacking gcc-4.3-arm-linux-gnueabi-base (from .../gcc-4.3-arm-linux-
gnueabi-base_4.3.5-4_amd64.deb) ...
.
.
.
Processing triggers for man-db ...
Setting up gcc-4.3-arm-linux-gnueabi-base (4.3.5-4) ...
.
.
.
Setting up libc6-dev-armel-cross (2.11.2-10) ...
fiesta:~#
```

Quadro 37 – Instalação do *toolchain*

Se não forem exibidas mensagens de erro durante o processo, o *toolchain* está instalado com sucesso.

APÊNDICE B – Instalação do U-boot no Mini2440

O Das U-Boot, ou somente U-Boot, é um *firmware open source* para dispositivos embarcados (THE DENX U-BOOT AND LINUX GUIDE FOR CANYONLANDS, 2005). O U-Boot também é referenciado como um *bootloader* (THE DENX U-BOOT AND LINUX GUIDE FOR CANYONLANDS, 2010).

O processador do Mini2440, o S3C2440, suporta dois modos de boot: NAND *Flash* ou *Not OR* (NOR) *Flash* (FRIENDLYARM, 2010, p. 8). O Mini2440 possui uma chave chamada *Boot Mode* para escolha do tipo de memória no qual se desejar efetuar o boot. A NOR *Flash* do Mini2440 vem embarcada de fábrica com o *bootloader* Supervivi. Este *bootloader* permite que dados sejam transferidos da estação de trabalho de desenvolvimento para a memória SDRAM do Mini2440 através de conexão USB. Após os dados estarem na memória SDRAM, pode-se executar o programa carregado. O espaço de endereço da memória SDRAM do Mini2440 está compreendido entre `0x30000000` e `0x34000000`, totalizando `0x04000000` *bytes* ou 64 MB.

Antes de iniciar o processo de instalação do U-Boot, o Mini2440 deve estar conectado a porta serial e a porta USB do computador. O Mini2440 é fornecido com os cabos necessários para esta conexão. A chave *Boot Mode* deve estar na posição NOR. A tela principal do Supervivi pode ser vista no Quadro 38.

```
##### FriendlyARM BIOS 2.0 for 2440 #####
[x] format NAND FLASH for Linux
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[a] Absolute User Application
[n] Download Nboot for WinCE
[l] Download WinCE boot-logo
[w] Download WinCE NK.bin
[d] Download & Run
[z] Download zImage into RAM
[g] Boot linux from RAM
[f] Format the nand flash
[b] Boot the system
[s] Set the boot parameters
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
[i] Version: 1026-2K
Enter your selection:
```

Quadro 38 - Tela principal do Supervivi

A imagem a ser gravada na NAND Flash é a `u-boot-256M.bin`, que está no *Digital Versatile Disc* (DVD) que acompanha o Mini2440. Esta imagem possui 242360 *bytes*. Para carregar este arquivo para a memória SDRAM no endereço `0x31000000` através da porta

USB, deve-se utilizar o comando `load ram 0x31000000 242360 u` no *shell* do Supervivi. Após a execução deste comando, o Supervivi irá aguardar o recebimento completo do arquivo. O comando a ser executado na estação de trabalho de desenvolvimento para enviar o arquivo ao Mini2440 é o `./usbpush u-boot-256M.bin`. Estando o U-Boot carregado na memória SDRAM no endereço `0x31000000`, deve-se iniciar a execução do mesmo com o comando `go 0x31000000`. A primeira execução do U-Boot pode ser vista no Quadro 39.

```
U-Boot 1.3.2-mini2440 (Feb 24 2010 - 13:04:49)

I2C:   ready
DRAM:  64 MB
Flash:  2 MB
NAND:  Bad block table not found for chip 0
Bad block table not found for chip 0
1024 MiB
Found Environment offset in OOB..
USB:   S3C2410 USB Deviced
In:    serial
Out:   serial
Err:   serial
MAC: 08:08:11:18:12:27
Hit any key to stop autoboot:  0
MINI2440 #
```

Quadro 39 – *Prompt* do U-Boot

Com o U-Boot em execução, deve-se apagar todo o conteúdo da memória NAND *Flash* com o comando `nand scrub`. As memórias NAND *Flash* podem vir de fábrica com alguns *bad blocks*, por isso após apagar a memória NAND *Flash* deve-se criar a tabela de *bad blocks* com o comando `nand createbbt`. A criação desta tabela em um Mini2440 com 1GB de NAND *Flash* leva em média 18 minutos para ser concluída.

Estando com a memória NAND *Flash* apagada e com a tabela de *bad blocks* criada, deve-se gravar o U-Boot na NAND *Flash* a fim de persisti-lo. O U-Boot pode ser gravado na NAND *Flash* com o comando `nand write.e 0x31000000 0 242360`.

Para testar se o U-Boot está funcionando corretamente, deve-se desligar o Mini2440, mudar a posição da chave *Boot Mode* para NAND *Flash* e ligar novamente o Mini2440. O *prompt* do U-Boot deve ser exibido.

APÊNDICE C – Carregando imagens para o Mini2440.

O U-Boot aceita principalmente transferência de dados para memória SDRAM a partir de cartões de memória MMC, cartões SD e *Trivial File Transfer Protocol* (TFTP). TFTP é um protocolo muito simples utilizado para transferência de arquivos (NETWORK WORKING GROUP, 1981). Neste trabalho optou-se por efetuar as transferências via TFTP devido a facilidade de disponibilização dos dados para o U-Boot sem a necessidade de gravar arquivos em um cartão de memória e inseri-lo fisicamente no Mini2440.

A memória NAND *Flash* é dividida pelo U-Boot em partições. O mapeamento de partições com seus respectivos endereços podem ser visualizados com o comando `mtdparts`. O Quadro 40 mostra como é mapeada a memória NAND *Flash* do Mini2440.

```
MINI2440 # mtdparts

device nand0 <mini2440-nand>, # parts = 4
 #: name                size        offset          mask_flags
 0: u-boot              0x00040000  0x00000000  0
 1: env                 0x00020000  0x00040000  0
 2: kernel              0x00500000  0x00060000  0
 3: root                0x3faa0000  0x00560000  0

active partition: nand0,0 - (u-boot) 0x00040000 @ 0x00000000

defaults:
mtdids  : nand0=mini2440-nand
mtdparts: <NULL>
MINI2440 #
```

Quadro 40 – Lista de partições da NAND Flash do Mini2440

Caso a interface de rede Ethernet da estação de trabalho de desenvolvimento não esteja configurada, é necessário configurá-la. Pode-se configurá-la editando-se o arquivo `/etc/network/interfaces`. O Quadro 41 mostra um exemplo de configuração de rede para comunicação com o Mini2440.

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
     address 10.0.0.1
     network 10.0.0.0
     netmask 255.255.255.0
     broadcast 10.0.0.255
```

Quadro 41 – Sugestão de configuração da interface de rede Ethernet

Para instalar um servidor de TFTP na estação de trabalho de desenvolvimento, deve-se executar o comando `apt-get install tftpd`. Este comando baixará o pacote do servidor TFTP, instalará e iniciará o serviço. Após este passo, é necessário criar o diretório `/srv/tftp/` que é onde o servidor TFTP espera que os arquivos sejam armazenados. A imagem do *kernel* e a imagem do sistema de arquivos raiz devem ser copiadas para o diretório

`/srv/tftp`. Em seguida, é necessário dar permissão de leitura para todos os usuários. Isto se faz necessário para que o servidor de TFTP possa ler os arquivos. Podem-se definir estas permissões com o comando `chmod 644 /srv/tftp/*`.

Para transferir as imagens para o Mini2440 é necessário que ele esteja conectado a estação de trabalho de desenvolvimento via cabo Ethernet e cabo serial. O acesso ao *prompt* do U-Boot é feito através da porta serial.

Para que o Mini2440 e a estação de trabalho de desenvolvimento comuniquem-se, é necessário que seja configurado o endereço IP do U-Boot. Um exemplo de configuração pode ser visto no Quadro 42.

```
MINI2440 # dynenv set 40000
MINI2440 # setenv ipaddr 10.0.0.2
MINI2440 # setenv serverip 10.0.0.1
MINI2440 # saveenv
```

Quadro 42 – Configuração IP do U-Boot

Para transferir a imagem do *kernel* para a memória NAND *Flash* do Mini2440, deve-se executar o comando `tftp 0x31000000 uImageFURB`, onde `0x31000000` refere-se a posição da memória SDRAM onde o arquivo será carregado e `uImageFURB` ao nome do arquivo que contém a imagem compilada do *kernel*. Se a imagem for carregada com sucesso para a memória SDRAM, deve-se apagar o conteúdo da partição *kernel* com o comando `nand erase kernel`. Após apagar o conteúdo da partição *kernel*, deve-se copiar o conteúdo da SDRAM para a memória NAND *Flash*. O comando `nand write.e 0x31000000 kernel 0x22EAF4` efetua esta cópia, onde `0x31000000` é o endereço da memória SDRAM onde estão os dados, `kernel` refere-se a partição do *kernel* e `0x22EAF4` refere-se ao tamanho em *bytes* da imagem a ser copiada.

Para transferir a imagem do sistema de arquivos raiz para a memória NAND *Flash* o procedimento é semelhante, alterando-se somente algumas informações. O Quadro 43 demonstra um exemplo deste procedimento.

```
MINI2440 # tftp 0x30010000 yaffsl.img
MINI2440 # nand erase root
MINI2440 # nand write.yaffs 0x30010000 root 0x2dd71c0
```

Quadro 43 – Gravação da imagem do sistema de arquivos raiz

Mesmo tendo a imagem do *kernel* e a imagem do sistema de arquivos raiz gravadas na NAND *Flash*, o U-Boot não efetua o processo de *boot* automaticamente. É necessário configurá-lo para tal. O U-Boot possui duas variáveis de ambiente responsáveis pelo *boot* do sistema operacional. As variáveis podem ser definidas através do comando `setenv`. Uma das variáveis é a `bootcmd` que informa ao U-Boot que comandos deve utilizar para efetuar o boot. Neste trabalho, a variável `bootcmd` foi definida como `nboot.e kernel ; bootm`. Para definir

o valor da variável desta forma foi utilizado o comando `setenv bootcmd 'nboot.e kernel ; bootm'`.

Além de definir os comandos de boot, é necessário informar ao *kernel* do Linux alguns parâmetros essenciais para a correta inicialização do sistema operacional. Entre estes parâmetros cabe destacar:

a) `root`: define em qual partição está o sistema de arquivos raiz;

b) `init`: define qual programa deve ser iniciado após a inicialização do *kernel*;

c) `rootfstype`: define o tipo do sistema de arquivos utilizado;

d) `console`: define qual dispositivo receberá a saída padrão do sistema operacional.

O Quadro 44 mostra como o `bootargs` deve ser definido para efetuar a segunda parte da geração do sistema de arquivos raiz. O Quadro 45 mostra como pode ser definido após a finalização da segunda parte da instalação do sistema de arquivos raiz utilizando o LCD no formato paisagem como saída padrão com fonte 6x11 *pixels*. Já o Quadro 46 mostra como deve ser definido para o desenvolvimento e testes dos mini-aplicativos.

```
MINI2440 # setenv bootargs 'root=/dev/mtdblock3 noinitrd init=/bin/sh
rootfstype=yaffs console=ttySAC0,115200'
```

Quadro 44 – `bootargs` para segunda da geração do sistema de arquivos raiz

```
MINI2440 # setenv bootargs 'root=/dev/mtdblock3 noinitrd rootfstype=yaffs
consoleblank=0 fbcon=font:ProFont6x11 fbcon=rotate:3'
```

Quadro 45 – `bootargs` para uso do LCD como saída padrão

```
MINI2440 # setenv bootargs 'root=/dev/mtdblock3 noinitrd rootfstype=yaffs
console=ttySAC0,115200 consoleblank=0'
```

Quadro 46 – `bootargs` para o desenvolvimento e testes

Sempre que se altera o valor de alguma variável de ambiente do U-Boot, é necessário persistir a alteração. Para isto, basta utilizar o comando `saveenv`.