UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA

Albert Nissimoff
Kátia Regina Akemi Sasaki
Rodrigo Bettini Alonso

Dispositivo seguro e de baixo consumo capaz de transferir dados por internet: Servidor Pessoal (SP)

SÃO PAULO

2010

Albert Nissimoff
Kátia Regina Akemi Sasaki
Rodrigo Bettini Alonso

Dispositivo seguro e de baixo consumo capaz de transferir dados por internet: Servidor Pessoal (SP)

Monografia de Projeto de Conclusão de Curso apresentada à Escola Politécnica da USP
Disciplina: Projeto de Formatura II – PSI2594

SÃO PAULO
2010

Albert Nissimoff
Kátia Regina Akemi Sasaki
Rodrigo Bettini Alonso

Dispositivo seguro e de baixo consumo capaz de transferir dados por internet: Servidor Pessoal (SP)

Monografia de Projeto de Conclusão de Curso apresentada à Escola Politécnica da USP
Disciplina: Projeto de Formatura II – PSI2594

Área de concentração: Sistemas Eletrônicos

Orientador: Profº Dr. Jorge Kinoshita

SÃO PAULO
2010

Agradecimentos

Ao Prof. Dr. Jorge Kinoshita pela orientação e pelo constante estímulo transmitido durante todo o trabalho.

Ao senhor Joe Bungo, ARM Holdings, e ao senhor Ricado Guiraldelli, engenheiro da Dixtal, pelas informações técnicas fornecidas sobre ARM/Linux

Ao Sr. Jair Pereira de Souza que nos auxiliou muito na soldagem da placa.

À Sra. Paula Schnaider pela ajuda na compra dos componentes do projeto.

Ao Sr. Ygor Amadeo Sartori Regados, pela ajuda fornecida em relação ao sistema operacional Linux.

Resumo

Hoje em dia, as pessoas precisam cada vez mais estar conectadas à internet, seja por razões profissionais ou simplesmente para se comunicar. Porém, embora a necessidade de acesso à rede a partir dos mais variados lugares já tenha sido praticamente atendida com as redes celulares e tantas outras tecnologias, o problema do acesso remoto a dados pessoais ou confidenciais continua pendente. Já existem diversas soluções para este problema, como o "*cloud computing*" (computação em nuvem), *pendrives*, entre outras. Embora a idéia resolva o problema da mobilidade, cria, por sua vez, uma série de outros. Desta forma, propõe-se desenvolver um dispositivo capaz de transferir dados pela internet, de forma segura e com baixo consumo e custo, ou seja, um servidor pessoal, tendo como público alvo as pessoas com necessidade de deslocamento e acesso aos seus dados. O dispositivo a ser criado deve ser concebido primeiramente para atender às expectativas dos usuários. Assim, os requisitos que precisa cumprir são: *disponibilidade*, ou seja, o sistema deve ser estável e capaz de realizar todas as funções a que se propõe, sempre que solicitado; *segurança*, para que o usuário não se preocupe com possíveis ameaças aos seus dados (tanto ameaças internas, que existiriam se os seus dados fossem confiados a uma empresa onde algum funcionário poderia vender ou corromper os dados dos usuários, quanto ameaças externas, como "*hackers*"); *consumo de energia*, pois problemas ambientais são uma realidade moderna, onde os usuários estão cada vez mais conscientes da necessidade de se utilizar dispositivos eficientes e de baixo consumo; *baixo custo*, para que seja possivel uma ampla adoção; e *tamanho*, para que o espaço seja otimizado. Quanto à criação do dispositivo, o projeto foi dividido em duas etapas: uma mais voltada à parte de *hardware* e outra à parte de *software*. Na primeira fase, foi desenvolvido pelo grupo o esquemático e o *layout* do dispositivo para um microprocessador ARM, seguido da confecção da placa, montagem e soldagem dos componentes. Na segunda fase, foi desenvolvido o *software*, contemplando o estudo e adaptação do *kernel* do Linux, além de alguns aplicativos selecionados.

Palavras-chave: Servidores de Rede. ARM-Microprocessadores. Sistemas Embutidos. Linux. ARM-Linux-Microprocessadores. Consumo de Energia Elétrica.

Abstract

Nowadays, people connect to the internet more than ever, for professional reasons or simply to communicate. However, although the necessity to access the internet from varied places has been mostly catered to by the use of cell phone networks and many other technologies, the problem of remote access to personal or confidential data still remains unanswered. There are many solutions to this problem, such as cloud computing, flash drives, among many others. Although some ideas solve the mobility issue, they create, on the other hand, many others. Hence, this project proposes the development of a device capable of securely transferring data through the internet, with low power consumption and cost, i.e. a Personal Server targeted at people on the move that still need to access their data. The device that has been created was conceived to fulfill users' expectations. Therefore, the requisites it should encompass are: availability, i.e. the system has to be stable and capable of executing all proposed functions, whenever needed; security, so that the user will not have to worry about possible threats to his data (both internal, that would exist if data were trusted to a company where employees could possibly sell or corrupt it, and external, such as hackers); energy consumption, because environmental concerns are a modern reality, and users are increasingly more aware of the necessity of using efficient and low power devices; low cost, thus permitting widespread adoption; and size, so space can be optimized. As for the device's implementation, the project has been divided into two stages: one more hardware-oriented and the other software-oriented. In the first stage, the group has developed the schematic diagrams and printed circuit board layout for an ARM microcontroller, followed by the board assembly and soldering. In the second stage, the software was developed, encompassing a study and adaptation of the Linux kernel as well as some selected applications.

Keywords: Network Servers. ARM-Microprocessors. Embedded Systems. Linux. ARM-Linux-Microprocessors. Power Consumption.

SUMÁRIO

1 Introdução 9

1.1 Descrição do Problema 9

1.2 Contexto do Projeto 10

1.3 Organização do Trabalho 11

2 Projeto Conceitual 12

2.1 Motivação e Justificativas Técnicas 12

2.2 Necessidades do Usuário 13

2.3 Objetivos do Projeto 14

2.3.1 Requisitos de *Marketing* 14

2.3.2 Requisitos de Engenharia 15

2.3.3 Matriz de Requisitos Engenharia – *Marketing* 15

2.3.4 Matriz de Requisitos Engenharia – Engenharia 16

2.4 Metodologia 17

2.5 Infraestrutura e Meios Necessários 18

2.6 Plano de Trabalho e *Deliverables* 19

2.7 Disciplinas da EPUSP relacionadas ao projeto 21

2.8 Análise de Viabilidade do Projeto 22

2.8.1 Critérios de Escolha do Processador 23

2.8.2 Diagrama em Blocos 26

2.8.3 Primeira Estimativa de Potência Consumida 28

2.8.4 Critérios de Escolha dos Outros Componentes 30

2.8.5 Viabilidade Técnica 41

2.8.6 Viabilidade Econômica 41

2.9 Riscos de Projeto 42

2.9.1 Componentes Eletrônicos 42

2.9.2 Placa de Circuito Impresso 43
2.9.3 Sistema Operacional 44
2.9.4 Prazo 45
3 Projeto Detalhado 46
3.1 Esquemático do Dispositivo 46
3.2 *Layout* do Dispositivo 47
3.3 Compra dos Componentes Importados 48
3.4 Compra dos Outros Componentes 50
3.5 Sistema Operacional: Linux 51
3.6 Ambiente de Desenvolvimento do *Software* 52
3.7 Desenvolvimento do *Toolchain* 53
3.8 Programa SAM-BA 55
3.9 Processo de inicialização e *Boot* 56
3.9.1 Processador 57
3.9.2 Inicialização do *Hardware* 58
3.9.3 Inicialização do *Software* 58
3.9.4 Ciclo de Inicialização Completo 59
3.9.5 AT91Bootstrap 60
3.9.6 U-Boot 62
3.9.7 *Kernel* 63
3.10 Projeto de *Software* 65
3.10.1 Sistema de Arquivos Raiz 66
3.11 Seleção dos tipos de aplicativos 73
3.12 UBIFS - Sistema de Arquivos UBI 76
3.13 Interface Web 78
3.13.1 *Front-End* 79
3.13.2 *Back-End* 83

4 Implementação, Testes e Validação 87
4.1 Fabricação das placas de circuito impresso 87
4.2 Etapas de Montagem 87
4.3 Implementação e Testes do Dispositivo 88
4.4 Validação do Dispositivo 89
4.5 Validação da Interface Web 90
4.5.1 W3C Validator 91
4.6 Programação do Dispositivo através do SAM-BA 92
4.7 FMEA 93
5 Resultados 97
6 Discussão 100
7 Conclusão 101
8 Recomendações para Trabalhos Futuros 102
Referências 103
Bibliografia 110
APÊNDICE 112
APÊNDICE A – Esquemático do Dispositivo 113
APÊNDICE B – *Layout* do Dispositivo 117
APÊNDICE C – Tabela das fases de montagem 118
APÊNDICE D – Programa AT91Bootstrap 122
APÊNDICE E – Resultado do *boot*: AT91Bootstrap, U-Boot e *Kernel* 124

1 Introdução

1.1 Descrição do Problema

Atualmente, a necessidade de se estar conectado à internet a todo o momento é cada vez maior. Entretanto, mesmo as pessoas podendo acessar a rede pelos mais diferentes meios (banda larga, 3G, EDGE, GPRS e assim sucessivamente) existe uma limitação no que diz respeito ao acesso a dados pessoais, por exemplo, um arquivo em que se trabalhou em casa e que se gostaria de ler do escritório.

Assim, surgiu naturalmente o conceito de "*cloud computing*" [1], ou computação em nuvem. Embora a idéia resolva o problema da mobilidade, cria uma série de outros, como por exemplo, o de segurança e a necessidade de se transmitir o arquivo a um servidor para depois poder recuperá-lo.

As grandes empresas têm acesso a servidores dedicados, que permitem todos esses tipos de acesso, mas que por outro lado apresentam grande consumo de energia elétrica e alto preço.

Desta forma, propõe-se desenvolver um dispositivo capaz de transferir dados pela internet, de forma segura, com baixo custo e com baixo consumo, ou seja, um servidor pessoal.

Figura 1 - Logotipo do projeto

1.2 Contexto do Projeto

O contexto em que se insere este projeto é o de uma sociedade cada vez mais móvel e ao mesmo tempo com maior volume de dados. Além disso, a preocupação com o meio ambiente é grande, e procura-se reduzir o consumo de energia e tornar os equipamentos mais energeticamente eficientes. Finalmente, busca-se um produto com capacidade de adaptação e de expansão frente às novas tecnologias.

Logo, pode-se definir este projeto como sendo do tipo pesquisa aplicada, combinando *market pull*, no sentido de preencher uma lacuna do mercado, e *technology push*, visando a conceber um produto inovador e com tecnologia de ponta.

O público alvo primário seria o de pessoas que precisam se deslocar para vários locais, mas que querem ter acesso aos seus dados. Neste caso, o dispositivo ficaria em suas residências. Outro público alvo seriam as pequenas empresas, o comércio ou indústrias que têm empregados que se deslocam o dia todo e precisam de dados que estão nos seus locais de trabalho.

1.3 Organização do Trabalho

Este relatório está dividido da seguinte maneira:

No capítulo 1, é apresentada a introdução ao projeto, com o problema a ser solucionado e a sua contextualização;

No capítulo 2, são apresentados os conceitos e os motivos (as motivações do grupo, as necessidades do usuário, os objetivos, os requisitos, o cronograma, as análises de viabilidade e os riscos) que resultaram nas decisões de cada parte do projeto;

No capítulo 3 tem-se uma descrição detalhada do processo de definição da solução escolhida e seu desenvolvimento;

No capítulo 4 encontram-se descritas as metodologias de implementação, testes e validação do projeto;

Finalizando com os resultados parciais e finais (capítulo 5), discussão (capítulo 6), conclusão (capítulo 7) e trabalhos futuros (capítulo 8).

2 Projeto Conceitual

2.1 Motivação e Justificativas Técnicas

O principal motivo pelo qual foi escolhido este projeto é a impossibilidade de acesso a dados pessoais de qualquer lugar em que as pessoas se encontrem. Com a vida cotidiana cada vez mais permeada pela necessidade de mobilidade, isso se torna um problema cada vez mais presente no dia-a-dia.

Além disso, a atual solução para esse problema, o "*cloud computing*", em que o armazenamento dos dados é realizado por meio de servidores e seu acesso é feito pela internet [1], não apresenta uma boa garantia de segurança dos dados. Para segurança externa, por exemplo contra *hackers*, já existem processos específicos de criptografia, entretanto, não se pode confiar plenamente nos funcionários dessas empresas que também têm acesso aos dados e podem comprometê-los [2].

Outro problema apresentado por essa solução é quanto à localização geográfica desses servidores. É difícil realizar uma avaliação exata dos riscos incorridos sem saber em que países os dados estão sendo armazenados. Por exemplo, em países onde uma revolução é iminente o risco de perda ou quebra de sigilo dos dados é muito grande.

Por fim, há uma demanda de mobilidade dos dados, sem a necessidade de carregar um objeto (por exemplo, *pen drive*) que pode ser esquecido ou não conter o arquivo de que se precisa, e, por enquanto, esta demanda ainda não foi atendida pela indústria. Este novo dispositivo apresentaria baixo custo e baixo consumo e, além de uma boa capacidade de processamento, teria a possibilidade de realizar tarefas complexas, mesmo que o dispositivo seja de pequeno porte.

2.2 Necessidades do Usuário

O dispositivo a ser criado deve ser concebido primeiramente para atender às expectativas dos usuários. Assim, cita-se em primeiro lugar a disponibilidade, ou seja, o sistema deve ser estável e capaz de realizar todas as funções a que se propõe, sempre que solicitado.

Além disso, a segurança deve estar sempre presente. O usuário, por possuir o sistema, não deve se preocupar com ameaças internas (por exemplo, algum funcionário de uma empresa de computação em nuvem resolve vender os dados dos usuários) e, ao mesmo tempo, também não deve temer as ameaças externas, como "*hackers*" [2].

Os problemas ambientais são uma realidade moderna, pois os usuários estão cada vez mais conscientes da necessidade de se utilizar dispositivos eficientes e de baixo consumo. Assim, esta é uma prioridade.

Finalmente, para que seja possível uma ampla adoção, deve-se garantir um baixo custo do sistema (quando comparado a um servidor normal).

2.3 Objetivos do Projeto

Uma vez definido o problema e as características esperadas pelos usuários para uma possível solução, são definidos então os objetivos almejados pelo projeto.

2.3.1 Requisitos de *Marketing*

Primeiramente se fez um resumo dos objetivos na concepção e criação de um dispositivo, ou seja, um *hardware* que atenda aos seguintes requisitos:

- permitir o acesso a dados por meio da internet com alta disponibilidade;
- ecologicamente correto, ou seja, alta eficiência energética;
- baixo preço (quando comparado a um servidor atual);
- capacidade de expansão (é interessante notar que ao se criar um dispositivo complexo que pode ser conectado à internet, abre-se automaticamente um leque de possibilidades de expansão que não devem ser menosprezadas: automação residencial, automação industrial, M2M, etc.);
- realizar todos os itens citados com excelente nível de segurança;
- tamanho reduzido.

2.3.2 Requisitos de Engenharia

Podem-se definir, através dos requisitos anteriores, as especificações técnicas primárias do projeto:

- permitir o acesso a dados por meio da internet com alta disponibilidade;
- consumo muito menor que um servidor atual, ou seja, consumo menor que 40W;
- preço inferior a um servidor atual, ou seja, preço menor que R$600,00;
- o dispositivo deve conter um sistema operacional, capaz de executar a função de reconhecer um novo dispositivo e permitir que o mesmo seja configurado facilmente;
- a transmissão de dados deve ser criptografada, através do protocolo SSH (*Secure Shell*) que criará uma VPN (*Virtual Private Network*) que permitirá uma comunicação segura entre o usuário e o servidor pessoal;
- dimensões reduzidas, inferiores a 300 cm^2.

2.3.3 Matriz de Requisitos Engenharia – *Marketing*

A Figura 2 mostra a relação existente entre os requisitos de engenharia e de *marketing*.

		Disponibilidade	Consumo	Custo	Fácil Configuração	Criptografia	Dimensões
		+	-	-	+	+	-
Disponibilidade	+	↑↑	↓	↓		↑↑	
Eficiência	+	↓	↑↑	↓↓	↓		↑
Custo	-		↓	↑↑	↑	↓	↓
Expansibilidade	+			↓	↑↑	↑	
Segurança	+	↑↑			↓	↑↑	
Tamanho	-		↑	↓			↑↑

Figura 2 – Matriz de requisitos de engenharia - *marketing*

2.3.4 Matriz de Requisitos Engenharia – Engenharia

Na Figura 3 encontra-se a relação dos requisitos de engenharia entre si.

		Disponibilidade	Consumo	Custo	Fácil Configuração	Criptografia	Dimensões
		+	-	-	+	+	-
Disponibilidade	+		↓	↓		↑↑	
Consumo	+			↓↓	↓		↑
Custo	-				↑	↓	↓
Fácil Configuração	+					↑	
Criptografia	+						
Dimensões	-						

Figura 3 – Matriz de requisitos de engenharia – engenharia

2.4 Metodologia

A metodologia usada na fase de *hardware* é a chamada "metodologia *bottom-up*", em que o sistema como um todo é dividido em vários subsistemas que são muito bem detalhados e, após a especificação de cada subsistema, eles serão interligados para formar o sistema inteiro [3] [4] [5].

Seguindo essa metodologia, primeiramente os requisitos (do sistema operacional e da memória) serão determinados, depois os componentes (memórias, processador, alimentação e os periféricos) e o *software* de trabalho, seguido do esquemático (memórias, processador, alimentação e os periféricos), do *layout* e do *software*.

Já para a etapa de *software*, bem como a integração das duas partes, foi utilizada a metodologia espiral. Essa metodologia consiste em uma seqüência de 4 passos principais que se repetem, havendo uma progressão a cada ciclo da espiral (Figura 4). [6]

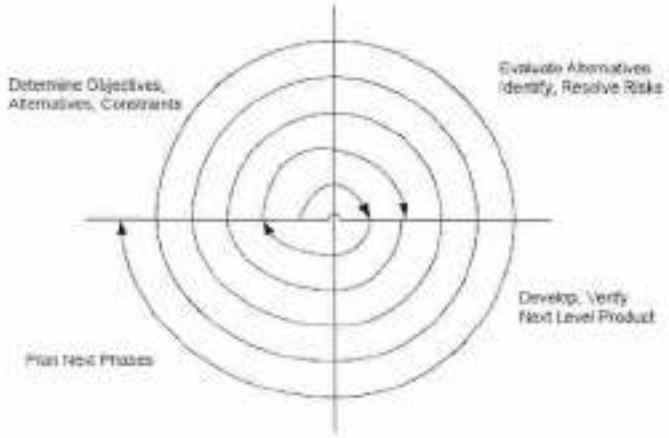

Figura 4 - Representação da metodologia espiral. [6]

Cada um desses ciclos representa um nível de elaboração do projeto, compreendendo os seguintes passos [6]:

1. determinação dos objetivos, alternativas e restrições do ciclo;
2. avaliação de alternativas, identificação e resolução de riscos;
3. desenvolvimento e verificação dos resultados do ciclo;
4. planejamento do próximo ciclo.

Dessa forma, a familiarização com o Linux, a comunicação com o SAM-BA, a adaptação do U-Boot, do *kernel*, a seleção dos aplicativos, a geração da imagem final e finalmente, os testes do conjunto compreendem cada ciclo dessa espiral no presente projeto.

2.5 Infraestrutura e Meios Necessários

Para a parte de *hardware* contou-se com o auxílio do Sr. Jair Pereira de Souza, técnico do LME (Laboratório de Microeletrônica), que utilizou equipamentos como uma *Heat Gun* similar à da Figura 5 e um microscópio estéreo para realizar a soldagem dos componentes, etapa mais detalhada na seção Etapas de Montagem e Implementação e Testes do Dispositivo.

Figura 5 - *Heat Gun*

No desenvolvimento do *software* utilizou-se uma máquina virtual, o Virtualbox, detalhado mais adiante, na seção Ambiente de Desenvolvimento do *Software*, e para os testes usou-se o programa SAM-BA, também detalhado mais a frente, na seção Programa SAM-BA.

2.6 Plano de Trabalho e *Deliverables*

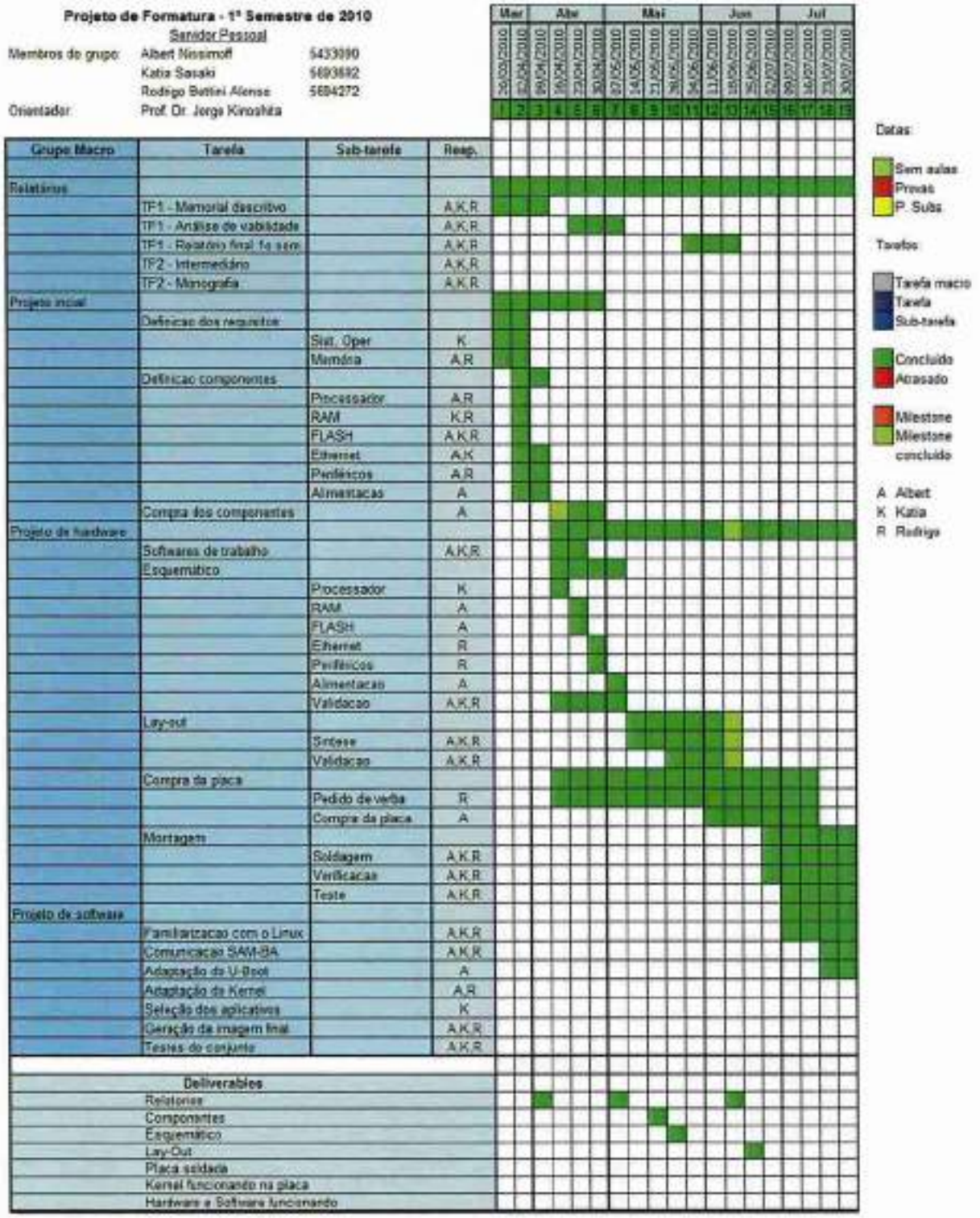

Figura 6 - Cronograma do 1º semestre.

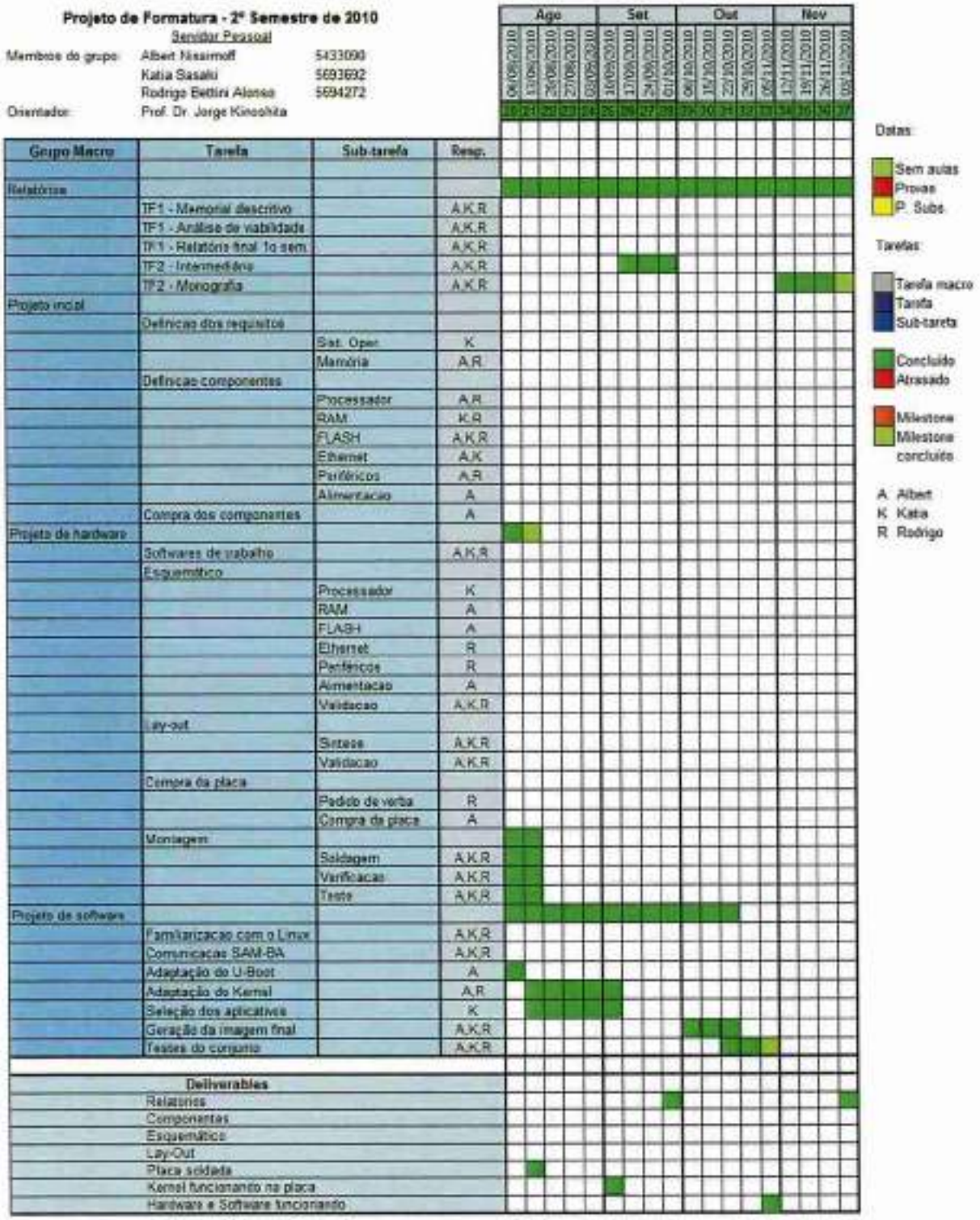

Figura 7 - Cronograma do 2º semestre.

2.7 Disciplinas da EPUSP relacionadas ao projeto

Dentro da ênfase "Sistemas Eletrônicos", a principal disciplina com que este projeto se relaciona é "Projeto de Sistemas Integrados", cuja sigla é **PSI2553**. Ela apresenta alguns conceitos de projeto estruturado de sistemas integrados *hardware-software*, de microprocessadores dedicados, de uso específico e de uso geral, focando num processador MIPS (*Microprocessor without Interlocked Pipeline Stages*) e em sua linguagem *Assembly*. Além disso, mostra alguns conceitos básicos de interfaceamento (endereçamento, interrupções, DMA e arbitragem) [7] [8]. Como já mencionado anteriormente, o projeto consiste num dispositivo digital que disponibiliza dados pessoais e executa tarefas pré-determinadas. Deste modo, a utilização de um processador e de memórias e, portanto, de conhecimentos de como eles funcionam e se comunicam, além de todos os seus periféricos e de sua programação, foram necessários.

Outras disciplinas necessárias são as de eletrônica, **PSI2223**, "Introdução à eletrônica" e **PSI2324**, "Eletrônica I", que dão a base para a análise de circuitos eletrônicos com diodos e transistores, na primeira, e de circuitos discretos mais gerais na segunda [9] [10]. Além disso, a parte de amplificadores e diagrama de Bode se fazem necessários, uma vez que o circuito também possui parte analógica e depende de seu funcionamento em função da freqüência de operação (alimentação, comunicação, periféricos).

A característica interdisciplinar do projeto se dá, principalmente, com **PCS2477**, "Arquitetura e organização de computadores", que também apresenta os principais conceitos relacionados com a organização e projeto de computadores, enfatizando, além dos aspectos de interface *software-hardware*, o seu impacto no desempenho. O escopo dessa disciplina inclui: programação em linguagem *Assembly*, a conversão de uma linguagem de alto nível para uma linguagem de máquina, a estrutura geral dos computadores, interrupções, *caches*, translação de endereços e hierarquia de memória [11]. Todos esses tópicos são essenciais para a quase totalidade do projeto, uma vez que ele consiste num sistema digital, com uma parte

de *hardware* e outra de *software*, além da comunicação com o usuário e a interligação e desempenho de todos eles.

A disciplina **PCS2304**, "Projeto Lógico Digital" também discute aspectos de eletrônica digital (as tecnologias TTL, *Transistor-Transistor Logic*, e CMOS, *Complementary Metal-Oxide Semiconductor*, e suas características), necessárias a esse projeto. Ela apresenta, de uma forma mais geral, os blocos combinatórios lógicos (decodificadores, transcodificadores, multiplexadores), seqüenciais (*flip-flops*, registradores, deslocadores, contadores) e aritméticos (somadores, subtratores, comparadores, unidade lógica aritmética), bem como as memórias (tipos e respectivas características) e, novamente, a composição de um projeto estruturado (conceito de unidade lógica e fluxo de dados) [12]. Todos esses blocos juntos compõem o projeto.

Para o acesso aos dados e a comunicação, a disciplina **PCS2476**, "Fundamentos de redes de computadores", é essencial, uma vez que envolvem os conceitos do modelo de camadas de um sistema de comunicação, do protocolo TCP/IP, as características da tecnologia *Ethernet* e criptografia [13], já que o projeto também visa à segurança dos dados.

Por fim, foi utilizada a linguagem C na configuração do sistema operacional utilizado, matéria vista em **MAC2166**, "Introdução à computação para engenharia", e a linguagem C++ na implementação dos aplicativos gerais, apresentada em **PCS2478**, "Tópicos de programação" [14] [15].

2.8 Análise de Viabilidade do Projeto

A análise de viabilidade e riscos é uma etapa fundamental no início de um projeto, primeiramente, para que se possa avaliar a viabilidade técnica e econômica de se concretizar um dado escopo. Entretanto, por se tratar de uma etapa inicial, é muito complicado atingir resultados expressivos sem que se tenha um bom conhecimento das etapas envolvidas no projeto.

Desta forma, a seção 2.8 deste relatório está dividida em duas partes: inicialmente serão apresentados os resultados das análises realizadas (2.8.1 a 2.8.4) e, em seguida, serão estudadas a viabilidade técnica e econômica das etapas identificadas (2.8.5 a 2.8.6).

2.8.1 Critérios de Escolha do Processador

Esta talvez seja a etapa mais crítica de um projeto digital, a escolha do principal processador da placa. Desta escolha decorrem todas as definições sobre periféricos, recursos de *hardware* necessários ou disponíveis, consumo de energia, capacidade de processamento e muitos outros.

Assim, é natural que esta escolha seja feita ainda no início do projeto, pois sua definição permite o estudo de todos os blocos e componentes auxiliares, que não necessariamente são compatíveis com diferentes plataformas.

Além de todos os requisitos de engenharia e de *marketing* já listados, por se tratar de um projeto de formatura, algumas outras características são importantes. A possibilidade de se estudar a fundo um sistema durante um ano, com a ajuda de toda uma equipe de professores e de um orientador, deve ser aproveitada ao máximo. Desta forma, é importante escolher uma plataforma já presente no mercado e que seja conhecida pelos profissionais do setor. Ainda no intuito de facilitar ou até mesmo viabilizar o desenvolvimento, a literatura disponível para uma plataforma deve ser extensa e facilmente acessível.

Sob uma perspectiva de *hardware*, é muito interessante que o tamanho dos componentes seja suficiente para serem soldados a mão, ou por meio de instrumentos simples como *heat guns*. Componentes do tipo BGA, uBGA, FPBGA ou qualquer outro tipo de encapsulamento cujos pinos se encontrem abaixo da pastilha inviabilizam imediatamente a soldagem manual.

Quanto ao *software*, os requisitos de *marketing* levam o grupo a escolher um sistema operacional completo e facilmente customizável. Embora a escolha deste sistema não deva ser obrigatoriamente realizada no projeto conceitual, é interessante conhecer as necessidades dos sistemas usuais, principalmente aqueles que têm grandes chances de serem implementados na plataforma em desenvolvimento: Linux e Windows CE.

Estes dois sistemas operacionais necessitam obrigatoriamente de um processador de 32 bits além de uma unidade de gerenciamento de memória (MMU – *Memory Management Unit*). Assim, só são aceitáveis os processadores que possuem estas características.

Para concluir esta introdução aos critérios utilizados, vale lembrar que, por se tratar de um projeto de último ano do curso de Engenharia Elétrica com ênfase em Sistemas Eletrônicos, é interessante que a plataforma escolhida seja utilizada amplamente no mercado ou ainda que seja uma tendência para o futuro.

2.8.1.1 Microcontroladores ARM

Em julho de 2009, o grupo Gartner Inc publicou um estudo que indicava que a arquitetura ARM detinha a maior parcela do mercado de microcontroladores de 32 bits e era também a arquitetura que mais crescia em participação [16]. Outro ponto a se ressaltar é que se considerassem exclusivamente os processadores RISC (*Reduced Instruction Set Computing*) de 32 bits, a fatia de mercado sobe para 90% [16]. Como último dado, vale lembrar que 98% dos celulares vendidos hoje em dia possuem pelo menos um processador ARM (*Advanced* RISC *Machine*) [17].

Além disso, por ser uma arquitetura facilmente licenciável por qualquer empresa, a oferta de processadores no mercado é enorme, permitindo a escolha precisa do mais adequado para uma determinada aplicação. Empresas que vendem processadores ARM incluem: Alcatel-Lucent, Apple Inc., Atmel, Broadcom, Cirrus Logic, Digital Equipment Corporation, Freescale, Intel, LG, Marvell Technology

Group, NEC, NVIDIA, NXP, Oki, Qualcomm, Samsung, Sharp, STMicroelectronics, Symbios Logic, Texas Instruments, VLSI Technology, Yamaha, ZiILABS e muitas outras.

Desta forma, ficou decidido pelos membros do grupo e pelo orientador que a arquitetura ARM seria a ideal para o projeto, além de ser facilmente aplicável à indústria.

2.8.1.2 Seleção Final do Processador

Uma vez definida a arquitetura, restava selecionar um microcontrolador entre as várias opções dentro da família ARM, são elas: ARM7, ARM9, ARM11 e Cortex. A Figura 8 mostra uma representação gráfica destas famílias e suas funções particulares.

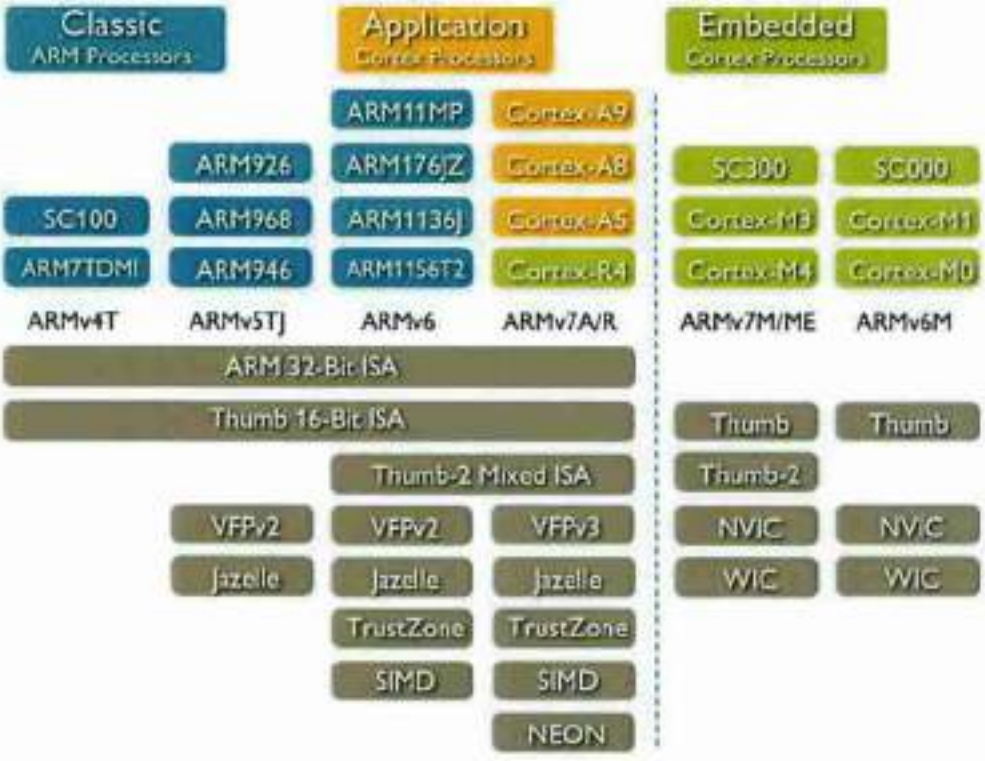

Figura 8 - Família de processadores ARM. [18]

Embora muitos desses processadores sejam capazes de executar um sistema operacional completo (muitos de forma comparável a um *Desktop*), a maioria não é vendida, por nenhum fabricante, em um encapsulamento com pinos salientes (ou seja, são todos do tipo BGA que dificilmente poderiam ser soldados de forma manual). Assim, a escolha ficou restrita às famílias ARM7, ARM9 e Cortex-M1 e M3.

Como requisito já indicado, a presença de uma MMU é fundamental, reduzindo a escolha à família ARM9, que atende facilmente a todos os requisitos de engenharia e de *marketing*.

Uma vez definida a arquitetura ARM9, passou-se à próxima etapa, a seleção do processador com base na oferta do mercado. Foram selecionados três fabricantes para a fase final do processo: Atmel, Freescale e ST. Vale notar que a NXP não passou à fase final em razão da indisponibilidade de processadores com saídas para barramento externo sem recorrer ao encapsulamento BGA. Finalmente, elaborou-se a seguinte matriz de decisão (Tabela 1):

Tabela 1 - Matriz de decisão para a escolha do processador.

		Facilidade de soldagem	Qtd. de pinos de Entrada/Saída	Clock Máximo	Consumo	Qualidade da documentação	Ferramentas de debugging	NOTA FINAL
	Peso da coluna:	1	1	2	1	3	3	
COMPONENTE	ATMEL (AT91SAM9)	5	10	5,25	7,34	10	10	8,27
	Freescale (i.MX23)	8	4,85	10	5,62	2	7	5,90
	ST Micro (STR9)	10	4,85	2,5	10	5	7	6,60

2.8.2 Diagrama em Blocos

Com o processador definido, foi possível passar para a próxima etapa, visando identificar todos os blocos ou componentes necessários para se atender aos requisitos de engenharia e de *marketing*. Assim, chegou-se ao diagrama em blocos do sistema (Figura 9).

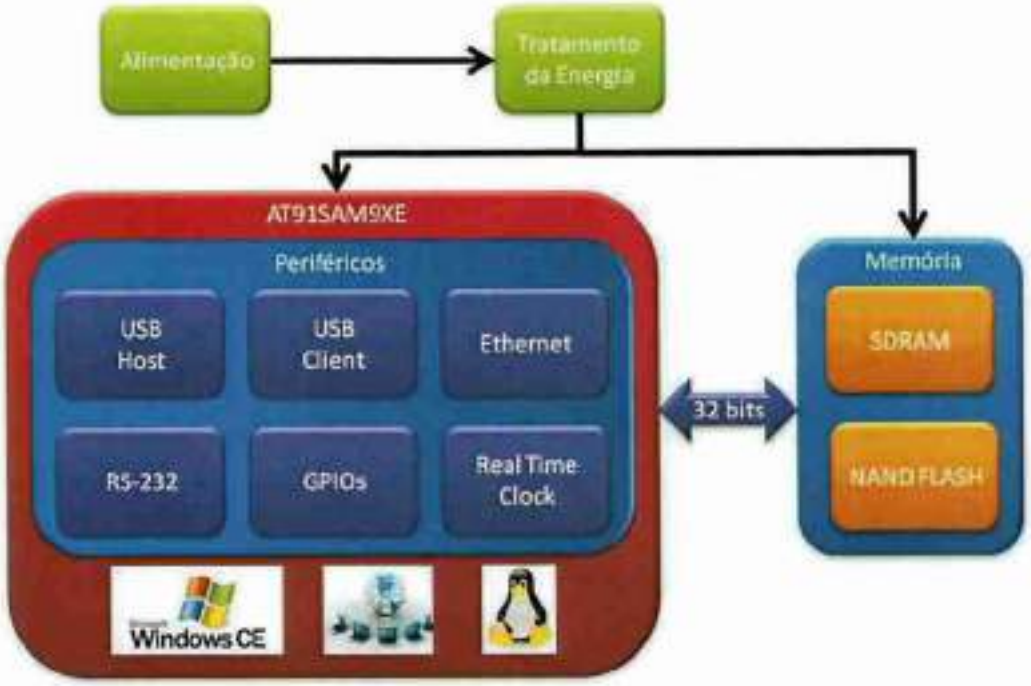

Figura 9 - Diagrama em blocos do sistema

Neste diagrama, identifica-se o bloco em vermelho representando o microcontrolador AT91SAM9XE, um verdadeiro SoC (*System on Chip*). Ele é capaz de executar Linux ou Windows CE além de incorporar uma série de periféricos (indicadas pelos seis pequenos blocos). Por se tratar de um processador de 32 bits, deve-se prever um barramento adequado para a comunicação com as memórias voláteis (SDRAM) e não-voláteis (*Flash*).

Finalmente, um dos pontos chave do projeto é a gestão inteligente da energia consumida, almejando sempre um baixo consumo e bom desempenho. Assim, um bloco fundamental é o de tratamento de energia, responsável por gerar todas as tensões necessárias dentro do circuito a partir de uma única fonte externa.

2.8.3 Primeira Estimativa de Potência Consumida

Como uma das maiores preocupações do projeto é com o baixo consumo e a alta eficiência energética do dispositivo, foi decido desde cedo elaborar um diagrama de fluxo de potência (Figura 10) para se estimar com mais pertinência estas duas grandezas. O procedimento para esta etapa foi:

1. listar todas as cargas tentando identificar da forma mais precisa possível a sua tensão de operação e corrente média consumida;
2. propagar os valores de potência consumida até os respectivos reguladores (3,3V e 1,8V);
3. os reguladores serão do tipo chaveado para aumentar a eficiência na conversão DC / DC. Inicialmente se supôs uma eficiência de 80%, mas acredita-se que seja possível atingir valores melhores desde que se possa contar com componentes de qualidade como capacitores com baixo ESR (*Equivalent Series Resistance*) ou indutores com alta corrente de saturação além de baixa resistência série;
4. considerando-se as respectivas eficiências, propagou-se os valores de potência consumida até a fonte externa;
5. mais uma vez se supôs uma determinada eficiência, mas desta vez para a fonte externa;
6. finalmente, foi propagada esta potência total até a tomada, onde se encontra uma potência vista pela rede de 3,43W. Este valor parece bem aceitável como primeira estimativa.

A Figura 10 se trata de uma planilha Excel facilmente editável dando ao grupo a oportunidade de atualizar os valores e de verificar, a todo instante, onde está sendo desperdiçada energia ou onde a eficiência está baixa.

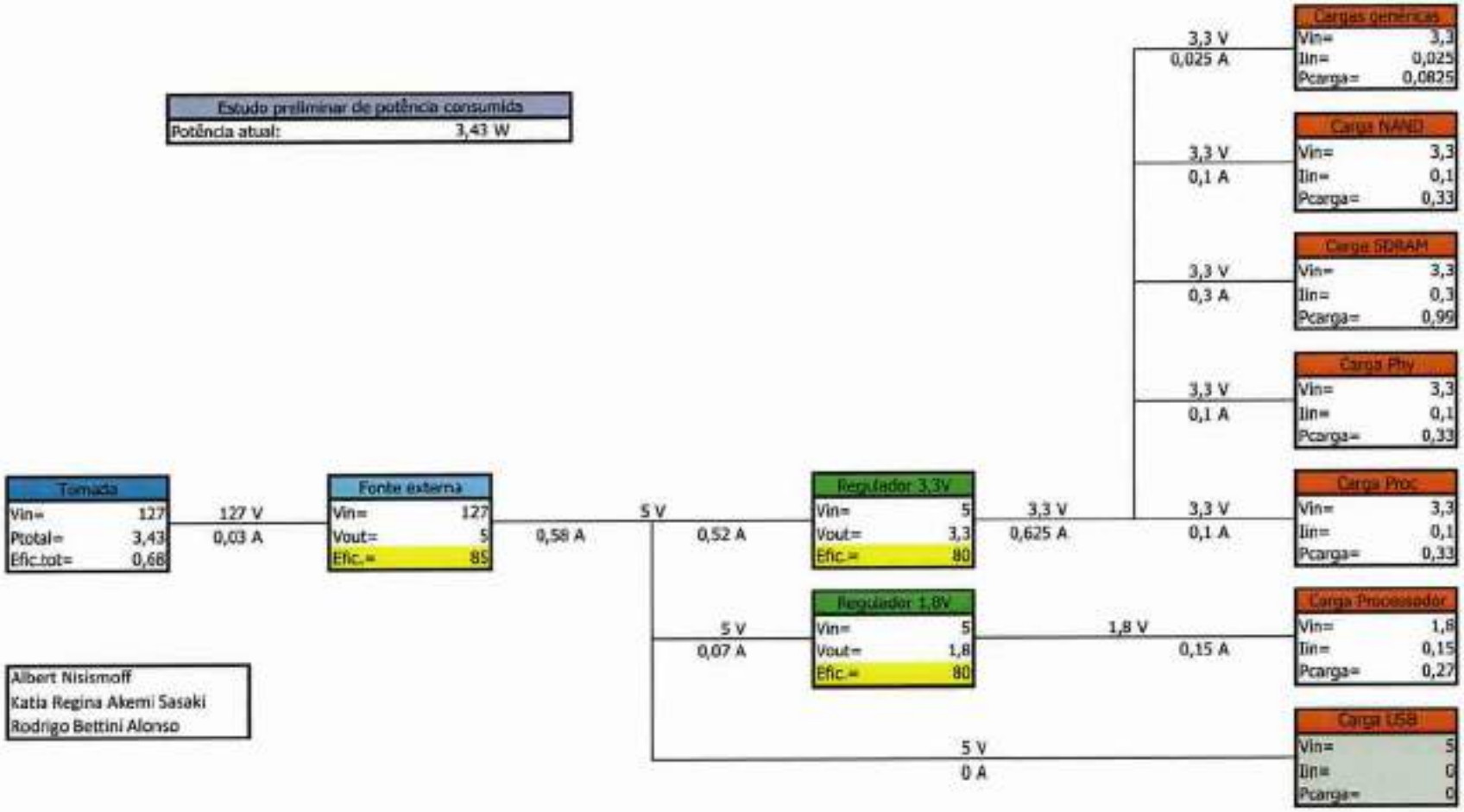

Figura 10 - Estimativa de potência consumida

2.8.4 Critérios de Escolha dos Outros Componentes

2.8.4.1 Memórias

2.8.4.1.1 Memória SDRAM

Definido o processador, seguiu-se para a escolha das memórias. Dentre todos os tipos de memórias de acesso aleatório (RAM), existem:

- SRAM (RAM estática): é um tipo de memória de acesso aleatório que mantém os dados armazenados desde que seja mantida sua alimentação, não precisando que as células que armazenam os bits sejam atualizadas de tempo em tempo, como é o caso das memórias DRAM [19];
- DRAM (RAM dinâmica): é um tipo de memória RAM de acesso direto que armazena cada bit de dados num condensador ou capacitor. O número de elétrons armazenados no condensador determina se o bit é considerado 1 ou 0. Como vai havendo fuga de elétrons do condensador, a informação acaba por se perder, a não ser que a carga seja atualizada periodicamente [20].

A escolha da DRAM pelo grupo foi natural porque as memórias SRAM são mais caras, ocupam mais espaço, consomem mais energia e aquecem mais que as DRAM. Mesmo que as memórias DRAM sejam mais lentas que as SRAM e que precisem do *refresh* pelo fenômeno da perda de carga, suas vantagens compensam.

Porém há ainda a escolha dentre as muitas variações da DRAM, em que se destacam duas principais:

- DRAM assíncrona: é aquela em que não há uma sincronização com um sinal externo (*clock*), então um período mínimo de tempo é requerido e determinado para que uma operação se complete. Para cada uma das operações internas de um chip DRAM assíncrono é determinado um tempo

mínimo, de modo que a resposta da DRAM seja menor que um ciclo de *clock* [21];

- DRAM síncrona (SDRAM): é uma memória em que sua operação é sincronizada com um sinal externo (*clock*). Com a evolução dos computadores, os barramentos conectados precisavam rodar a mais de 66MHz, precisou-se então encontrar uma maneira de sobrepujar os problemas de latência que existiam nas DRAM assíncronas. Implementou-se uma interface síncrona: a DRAM obtém os endereços, os dados e os sinais de controle e a CPU espera até que a DRAM complete suas operações internas para receber os dados. Após um número específico de ciclos, os dados são disponibilizados e a CPU pode lê-los das linhas de saída. Na SDRAM, as entradas de dados são também simplificadas, pois os sinais de controle, endereços e dados podem todos ser obtidos sem necessidade de monitoramento pela CPU de tempos de preparação e manutenção. Os mesmos benefícios são aplicados à saída de dados [21].

Como o barramento das memórias no processador é de 100 MHz, então é necessário que se escolha a SDRAM.

Estudando o processador vê-se que ele possui 32 pinos de dados [22] limitando o barramento da memória RAM a ser utilizada para 32 bits. Para atender o requisito de se poder soldar os circuitos à mão, foram excluídos aqueles com encapsulamento BGA (*Ball Grid Array*).

Detalhado mais a frente, o sistema operacional a ser utilizado será o Linux. Por meio de pesquisas e de uma conversa com Joe Bungo, um engenheiro sênior que trabalha na ARM e que também trabalha com Linux, indicado pelo orientador, concluiu-se que um total de 32MB de RAM é suficiente para se rodar um ARM-Linux com uma razoável margem para os periféricos.

Por fim, ainda para atender ao mesmo tempo às especificações de baixo custo e de dimensões reduzidas, optou-se por duas memórias de 128Mbits (16MB), ao invés de uma de 256Mbits (32MB) ou quatro de 64Mbits (8MB). Com duas memórias de 128Mbits e, a partir da limitação do barramento dado pelo processador, define-se,

então, duas memórias de 128Mbits com barramento de 16bits. Também foi definida uma tensão de alimentação padrão de 3,3V.

Da lista de memórias com essas especificações, escolheu-se a de menor preço que atende aos requisitos de projeto definidos anteriormente, visto que o preço das outras poderia inviabilizar economicamente o projeto.

2.8.4.1.2 Memória NAND *Flash*

Considerando que o dispositivo precisa de uma memória que armazene os dados mesmo que sem alimentação, as memórias *Flash* são mais interessantes que outros tipos de memórias, pois o processador disponibiliza uma série de ferramentas que auxiliam no controle deste tipo de memória.

Existem dois tipos de memória *Flash*:

- memórias NOR *Flash*: cada célula tem uma extremidade conectada diretamente ao terra, e a outra extremidade conectada diretamente a uma linha de bit. Este arranjo é chamado de "*flash NOR*", pois ele age como uma porta NOR: quando uma das linhas de palavra é colocada em nível alto, o correspondente transistor "puxa" a linha de bits de saída para nível baixo [23]. A NOR *Flash* tem uma interface de acesso aleatório à memória totalmente semelhante a uma RAM, com linhas dedicadas de endereço e linhas dedicadas de dados;

- memórias NAND *Flash*: utiliza transistores de porta flutuante, mas estão ligados de uma forma que lembra uma *porta NAND*: *diversos* transistores são ligados em série, e somente se todas as linhas de palavras são postas em nível alto a linha de saída do bit é posta em nível baixo [23]. Diferentemente das memórias NOR *Flash*, esse tipo de memória possui um acesso seqüencial, já que não apresenta um barramento dedicado aos endereços, resultando num tempo maior para um acesso aleatório, porém, com um tempo

menor para se programar e apagar seus dados. Além disso, por ter uma menor área por bit, esse tipo de memória possui maior densidade de células [24].

A seguir são apresentadas as diferenças entre duas memórias *Flash* semelhantes, uma NAND e outra NOR, presente em um dos *datasheets* do processador fornecidos pela Atmel [25] (Tabela 2):

Tabela 2 - Comparação entre duas memórias *Flash* semelhantes: NAND *Flash* e NOR *Flash*. [25]

Characteristics	NAND Flash K9F2G08U0M	NOR Flash AT49BV16x4-90
Random access read	25 µs (first byte) 30 ns each for remaining 2111 bytes	0.09 µs
Sustained read speed (sector basis)	37 Mbytes/s	11 Mbytes/s
Random write speed	300 µs/2,112 bytes	20 µs / bytes
Sustained write speed (sector basis)	5 Mbytes/s	0.05 Mbytes/s
Erase block size	128 Kbytes	64 Kbytes
Erase cycles	100,000 to 1,000,000	10,000 to 100,000
Erase time per block	2 ms	200 ms

Comparação final das vantagens e desvantagens entre as memórias *Flash* NAND e NOR [25]: (Tabela 3)

Tabela 3 - Comparação final entre as memórias NAND *Flash* e NOR *Flash*. [25]

	NAND	NOR
Advantages	Fast writes Fast erases Lower bit cost Higher density	Random access Byte writes possible
Disadvantages	Slow random access Byte writes difficult Bad blocks management and ECC required	Slow writes Slow erase
Applications	File (disk) applications Voice, data, video recorder Any large sequential data	Execute directly from non volatile memory

Pela Tabela 3, vê-se que a memória NAND *Flash* é a que possui melhores características, portanto, optou-se por ela.

Também estudando o *datasheet* do processador, tem-se que o tamanho mínimo para o barramento da memória é de 8 bits e o tamanho mínimo avaliado razoável para o uso do projeto foi de 1Gbit ou 2Gbit.

Considerando novamente a especificação da soldabilidade e disponibilidade para pequenas quantidades, chegamos a preços que inviabilizariam o projeto para memórias NAND *Flash* com barramento de 16 bits, limitando o barramento em 8 bits.

Da lista de memórias restantes, o preço das memórias de 4Gbit (512MB) também inviabilizaria o projeto, definindo assim o tamanho total da memória NAND *Flash* escolhida de 2Gbit (256MB) com barramento do 8 bits. Inicialmente, essas memórias foram importadas, porém, devido a um erro na especificação da tensão de operação, decidiu-se utilizar memórias de *pen drives* que, além de terem um custo menor, apresentam maior capacidade sem aumento na dimensão do dispositivo.

2.8.4.2 Módulo PHY - Ethernet

Com as memórias já determinadas, partiu-se para a definição do módulo *Ethernet*, o componente mais importante depois do processador e das memórias, pois é por ele que se dará a comunicação do dispositivo com a internet.

Da mesma forma que nas memórias, foram definidas as especificações de 3,3V de alimentação, a necessidade do encapsulamento soldável (excluindo, portanto, as famílias BGA, QFP (*Quad Flat Package*) e QFN (*Quad Flat No-Lead Plastic Package*)) e disponibilidade para pedidos em pequenas quantidades, que foram os quesitos para filtrar os muitos CIs para os módulos PHY (*PHYsical layer*) existentes.

Como características específicas do funcionamento desse módulo, tem-se a necessidade do suporte à norma IEEE 802 (*Institute of Electric and Electronic Engineers*) e aos padrões 10baseT e 100baseTx como mínimo.

MII (*Media Independent Interface*) e RMII (*Reduced Media Independent Interface*) são dois tipos de interface entre a camada física (*physical layer*) e a camada de enlace (MAC – *Media Access Control*). A interface MII gerencia os dados recebidos/transmitidos de forma a deixá-los, no caso deste projeto, no padrão IEEE 802 e consiste em um canal para transmissão, e separadamente, um canal para a recepção, em que cada canal possui seus próprios sinais de controle, de dados e de *clock*, apresentando quatro bits de transmissão e quatro de recepção. Dessa forma, na camada MAC são tratados sinais digitais e na camada PHY, sinais analógicos [26].

A partir da análise do funcionamento desse chip, verifica-se que a velocidade dos bits na camada física é maior que na de enlace, facilitando, portanto, seu processamento.

No modo de transmissão MII, o PHY envia o sinal do *clock* de transmissão para o controlador que sincroniza os dados transmitidos com a borda de subida desse *clock* e habilita a transmissão [27].

Na recepção, o PHY envia os dados recebidos mais o sinal de *clock* de referência de recepção para o controlador de memória que os sincroniza com a borda de subida do sinal de *clock* e habilita a recepção dos dados [27].

A Figura 11 mostra um esquema simplificado desse módulo:

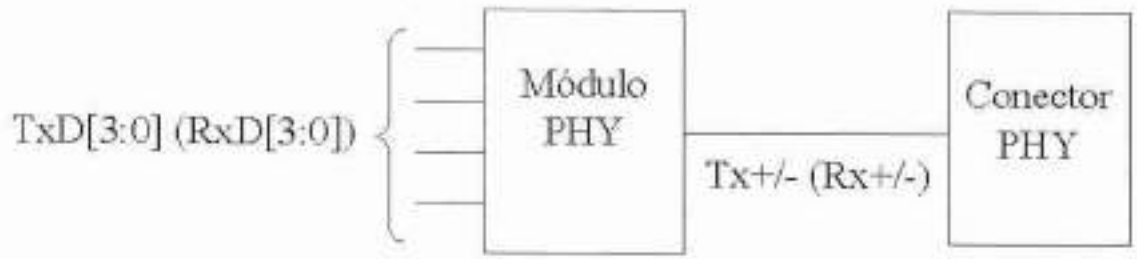

Figura 11 - Esquema simplificado do Módulo PHY.

A interface RMII nada mais é que um "aprimoramento" da MII, no sentido que a quantidade de pinos utilizada é reduzida, suprimindo-se alguns de seus sinais. Por exemplo, nesse modo, há apenas dois bits de dados transmitidos e dois de dados recebidos [27].

Na hora de se escolher o módulo PHY, selecionaram-se os que tinham os dois tipos de interface para dar flexibilidade ao projeto, pois a diferença de preço com relação aos que tinham um único módulo era pequena, compensando o ganho em flexibilidade. E finalmente, da mesma forma que nas memórias, das opções que restaram, optou-se por aquele de menor preço.

2.8.4.3 Módulo de Interface Serial

A saída serial é a saída usual do processador para ser programado pelo desenvolvedor, pois a Atmel oferece, entre as várias ferramentas para o auxílio do desenvolvimento do projeto, o SAM-BA (*SAM Boot Assistant*), que é um *software* que provê meios que facilitam a programação e o *boot* dos processadores Atmel AT91SAM.

Para se fazer essa comunicação entre a saída serial e o processador, é necessário um módulo que converta as tensões de 12V do padrão serial para 3,3V que é o padrão de tensão do processador. Para isso é usado o módulo de interface serial.

As suas especificações seguem como as dos componentes anteriores: 3,3V de alimentação, a necessidade do encapsulamento soldável (excluindo, portanto, as famílias BGA, QFP e QFN) e disponibilidade para pedidos em pequenas quantidades. Por fim, foi escolhido o módulo mais barato que atendesse todos esses requisitos.

2.8.4.4 Extensões (GPIO)

Uma das especificações do projeto é a possibilidade de expansão da funcionalidade do dispositivo, então foi incluída uma saída a mais no dispositivo para se conectar outros circuitos periféricos que utilizariam o processamento do servidor pessoal para ampliarem as suas funcionalidades. Algumas áreas que poderiam se beneficiar dessa expansão são: automação residencial, registro de consumo de energia de equipamentos ou da residência inteira e M2M.

2.8.4.5 Alimentação

2.8.4.5.1 Regulador Chaveado

Um regulador chaveado trabalha pegando pequenos pedaços de energia, pouco a pouco, a partir da tensão de entrada, e movendo-os para a saída. Isso é conseguido com a ajuda de uma chave elétrica (transistor) e um controlador que regula a taxa na qual energia é transferida para a saída (daí o termo regulador "chaveado"). Enquanto que um regulador linear funciona dissipando a diferença entre a tensão de entrada e a de saída na forma de calor. Quanto maior a diferença entre a entrada e a saída, mais calor é dissipado. Na maioria dos casos, um regulador linear consome mais energia baixando a tensão do que realmente acaba entregando para o dispositivo [28].

Reguladores chaveados oferecem principalmente três vantagens em relação aos reguladores lineares. Primeiro, a eficiência do chaveado pode ser muito melhor que a do linear. Em segundo lugar, porque menos energia é perdida na transferência, então se podem fazer componentes menores e que esquentam menos. Em terceiro lugar, a energia armazenada por um indutor em um regulador de comutação pode ser transformada em tensões de saída que podem ser maior do que a de entrada

(*boost*), negativas (conversor), ou pode mesmo ser transferido através de um transformador para fornecer isolamento elétrico em relação ao de entrada [29].

Há, sem dúvida, desvantagens com os reguladores de comutação. Eles podem ser barulhentos e exigem uma gestão de energia na forma de uma malha de controle. Felizmente, a solução para estes problemas de controle encontra-se nos modernos controladores integrados [29].

Como alguns dos requisitos do projeto são o baixo consumo e dimensões reduzidas, escolheu-se, para a maioria das partes do dispositivo, o regulador chaveado, porque apresenta menor dissipação de potência, menor tamanho e maior rendimento do que os outros tipos de reguladores.

Também se deve considerar a necessidade do encapsulamento soldável (excluindo, portanto, as famílias BGA, QFP e QFN) e a disponibilidade para pedidos em pequenas quantidades. Por fim, foi escolhido o regulador mais barato que atendesse todos esses requisitos.

2.8.4.5.2 Regulador Linear

Além da alimentação principal dos componentes, o processador disponibiliza uma entrada de alimentação de 1,8V dedicada à manutenção do oscilador de baixa freqüência (32,768 kHz) cujo objetivo é permitir a medida de tempo mesmo quando o sistema estiver desconectado da alimentação. Assim, na próxima vez em que for ligado, será possível saber a hora atual exata. Desta forma, é de se esperar que este sistema seja projetado para operar com baterias e que necessite de pouquíssima potência, que é o caso, pois a corrente consumida pelo processador ARM para esse circuito é de 1μA.

Como as baterias em forma de "moeda" usuais para este tipo de aplicação são de 3V, é necessário diminuir a tensão até 1,8V. Para tanto, pode-se utilizar tanto um regulador linear quanto um chaveado.

A eficiência dos reguladores chaveados é muito boa para circuitos que consomem muita corrente (da ordem de centenas de mA ou mais). Entretanto, para baixas correntes, sua eficiência começa a piorar bastante em função da corrente quiescente necessária para alimentar toda a complexa lógica de chaveamento interna ao circuito integrado.

Por exemplo, se um regulador consumir 100µA de corrente quiescente (Iq) e estiver regulando 1A, sua eficiência será praticamente independente de Iq, pois: $1A/(100\mu A + 1A) = 0{,}9999$. No entanto, se a carga consumir 1µA, a eficiência máxima teórica cai drasticamente para $1\mu A/(100\mu A + 1\mu A) = 0{,}0099$.

Por outro lado, os reguladores lineares têm uma corrente quiescente muito pequena quando comparada à dos chaveados. Assim, sua eficiência máxima teórica ainda será limitada por $1{,}8V / 3V = 0{,}6$. Pode-se supor ainda que o regulador linear consuma 4µA de Iq (caso do regulador TPS79718 escolhido), vê-se que a eficiência do circuito implementado é de: $(1\mu A * 1{,}8) / (5\mu A * 3{,}3V) = 0{,}1091$, ou seja, dez vezes superior ao regulador chaveado.

Assim, para conseguir um maior tempo de operação do circuito a partir de uma pequena bateria, foi decidido utilizar um regulador linear, pois, embora a eficiência seja baixa, ela ainda é muito superior à do regulador chaveado para baixas correntes.

2.8.4.6 Conectores

2.8.4.6.1 USB

Conector muito popular atualmente e é o padrão usado por *pendrives* e *HD* externos. Este conector é útil para que o usuário possa expandir a capacidade de armazenar dados no servidor pessoal. No projeto foram utilizados um conector USB A e outro USB B, ambos no padrão USB 2.0.

2.8.4.6.2 Ethernet

Para uma melhor recepção e transmissão dos dados, o módulo *Ethernet* necessita de um módulo de isolação [26]. Escolheu-se, assim, um conector com "núcleo magnético" (*Mag Jack*) no lugar do conector RJ-45 convencional. Esse "núcleo magnético", na verdade são alguns transformadores que isolam o meio externo do interno. O conector também possui LEDs (*Light Emitting Diode*) que podem ser usados, entre outras coisas, para verificação de sinal, facilitando a fase de testes, e confirmação da transmissão do dispositivo.

2.8.4.6.3 Debugging (JTAG)

Este conector é o último recurso caso os programas colocados pela serial e/ou pela USB não funcionem. Utilizaram-se dez e não vinte pinos para a sua construção, como é o normal, para se economizar área na placa de circuito impresso do dispositivo, mesmo assim se manteve a compatibilidade com os conectores convencionais, pois os pinos retirados eram de terra (GND).

Outro motivo para a escolha de se utilizar esse conector foi que o orientador do projeto, o Prof. Dr. Jorge Kinoshita tem o circuito programador pela porta JTAG, por utilizar muitas placas com processadores ARM.

2.8.4.6.4 Serial

Foi escolhido o conector RS-232, que é um padrão de conector comum de serial. Além de ser o mais barato, o envio da programação para o processador por meio dele não demanda circuitos muito complexos.

2.8.5 Viabilidade Técnica

Quanto à viabilidade técnica, o grupo acredita que seu esforço continuo em pesquisa na fase inicial do projeto permitiu desenvolver um bom conhecimento do funcionamento de todas as partes integrantes do projeto.

Além disso, por meio dos estudos iniciais de viabilidade, foi possivel confirmar que os requisitos de engenharia são alcançáveis.

Evidentemente, é possível e provável encontrar algum tipo de dificuldade com o projeto ainda não identificada quando se começar efetivamente a montar e a testar o sistema, mas se acredita que nenhum deles coloque em dúvida a solidez deste estudo inicial e as premissas utilizadas. Assim, concluiu-se que o projeto é realizável dentro das limitações já citadas em todo este relatório.

2.8.6 Viabilidade Econômica

No que diz respeito à viabilidade econômica, têm-se pleno conhecimento que os preços dos componentes e das placas que se necessita são relativamente elevados. Dessa forma, o grupo está ciente que parte dos recursos financeiros deste projeto inevitavelmente terá de ser disponibilizada pelo próprio grupo. Houve a intenção de se utilizar a verba disponibilizada da USP, porém, devido ao risco de se não conseguir cumprir com os prazos, o grupo acabou por arcar com todas as despesas.

A estimativa inicial de custo para cinco protótipos encontra-se na Tabela 4.

Tabela 4 - Estimativa de custo para 5 protótipos.

Ativos	Memória Flash	R$ 100,00
	Memória SDRAM	R$ 50,00
	Processador	R$ 200,00
	Alimentação (Reguladores de tensão, indutores ...)	R$ 75,00
	Chips de interface (Ethernet e RS-232)	R$ 100,00
Passivos	Resistores	R$ 50,00
	Capacitores	R$ 50,00
	Conectores	R$ 100,00
Geral	Misc (Diodos, Cristais, etc.)	R$ 100,00
	Placa de Circuito Impresso	R$ 400,00

TOTAL: R$ 1.225,00

2.9 Riscos de Projeto

Nesta parte do relatório será tratado dos riscos presentes nas diferentes etapas do projeto com base no estudo preliminar já realizado.

É importante levantar os riscos desde cedo para que se possa seguir mais de perto as etapas críticas e possibilitar a consecução dos objetivos e metas no prazo e dentro do orçamento.

2.9.1 Componentes Eletrônicos

Um grande problema em todo projeto de eletrônico é a compra dos componentes necessários. Muitas vezes, os componentes disponíveis durante o projeto param de ser produzidos uma vez o projeto concluído ou, em razão de uma forte procura, passam a custar mais caro ou até mesmo a ficarem indisponíveis.

Levando-se em conta o fato que o projeto será desenvolvido no Brasil, as dificuldades para obtenção de componentes são ainda maiores em razão dos

impostos de importação e trâmites demorados e nem sempre certos de desembaraço.

Sendo assim, optou-se por importar os componentes chave para o projeto (processador, memória e módulo PHY principalmente) logo no começo. Assim, caso houvesse um problema, o grupo não seria surpreendido no segundo semestre, quando provavelmente já seria tarde demais.

2.9.2 Placa de Circuito Impresso

Depois de se mandar e-mails para as 30 empresas nacionais produtoras de placas de circuito impressos associadas à ABRACI, Associação Brasileira de Circuitos Impressos, o grupo esperou as respostas para consolidar uma tabela de preços. Foram recebidas 14 respostas, sendo que uma não envia orçamentos sem o arquivo Gerber e outra não fabrica menos de mil unidades.

A partir dos requisitos do projeto, definiram-se as características para a placa a ser fabricada. A seguir tem-se uma lista com essas características:

1. quantidade: 5;
2. material: FR4 ou superior;
3. tamanho: 110mm por 70mm;
4. cobre: 1 oz;
5. número de camadas: 2;
6. número de vias/furos: aproximadamente 200;
7. traço mínimo: 8mils;
8. espaçamento mínimo: 8mils;
9. *soldermask* em um ou dois lados;
10. opção de tratamento para *heat gun;*
11. *layout* fornecido em formato Gerber padrão.

Essas especificações foram escolhidas para que se tenha o menor custo com um mínimo de folga para se projetar a placa com segurança (traço e espaçamento mínimos, tamanho, nº de camadas). Como os dispositivos são do tipo SMD, é necessário utilizar uma máscara de solda (*soldermask*) e, como será utilizado uma *heat gun* para soldar esse tipo de componente, é necessário um tratamento da placa para isso. Quanto aos outros itens da lista, são estimações do que também será necessário na placa.

A princípio, o maior risco do projeto está justamente nesta etapa. Por ser um processo lento e caro, a produção do circuito impresso obriga o grupo praticamente a conceber um projeto do tipo RFT ("*Right First Time*"). Além disso, a compra da placa está no caminho crítico do cronograma, impossibilitando qualquer avanço sem que ela esteja correta. Finalmente, ainda existe o risco do fabricante não atender às expectativas em relação à qualidade, principalmente em função dos pequenos traços exigidos pelos componentes SMD.

2.9.3 Sistema Operacional

Analisando-se a parte de controle de todo o sistema, esse projeto necessita de um sistema operacional com baixo consumo de memória, fácil implementação e configuração. O sistema operacional que melhor atende aos propósitos especificados é o Linux, pois é um sistema operacional não pago, com um grande acervo de dados disponível, podendo-se alterar o seu *kernel* para melhor adaptá-lo ao dispositivo. Além disso, este é um dos sistemas operacionais mais usados do mundo, ou seja, é mais prático para o público alvo do projeto evitando-se assim um período de adaptação ou até mesmo uma barreira à compra.

No entanto, por se tratar de um sistema complexo, como é a maioria dos sistemas operacionais atuais, ele pode ser considerado outro risco do projeto, uma vez que a adaptação do *kernel* não é uma tarefa fácil e configurar todas as aplicações requeridas exige um tempo considerável. A explicação para a escolha do Linux será dada na seção Projeto Detalhado: Sistema Operacional: Linux.

2.9.4 Prazo

Dado o escopo do projeto, fica claro que existem muitas tarefas de alta complexidade. Primeiramente, foi necessária muita pesquisa para conhecer a fundo os sistemas e componentes envolvidos, além de muito tempo para que se execute a soldagem e montagem.

Além disso, praticamente todas as tarefas estavam no caminho crítico, ou seja, se uma atrasar acabará atrasando todo o projeto. Finalmente, o tempo limitado em um ano para o projeto, devendo ser compartilhado com outras disciplinas, foi também um risco.

3 Projeto Detalhado

Nesta parte do relatório, serão abordadas as etapas do projeto relativas ao detalhamento da solução escolhida além dos documentos gerados, como o diagrama esquemático e o *layout*.

3.1 Esquemático do Dispositivo

Primeiramente foi necessário escolher um ambiente de desenvolvimento de placas de circuito impresso, para auxiliar e facilitar a criação do esquemático e do *layout*. O grupo escolheu o Eagle, por ser um *software* já conhecido, por ter sido usado em outras disciplinas da Poli e por ser gratuito.

Em seguida, criaram-se as bibliotecas dos componentes que não estavam inclusas na biblioteca do Eagle, compreendendo o *package*, que é, em termos gerais, o "desenho" que ele terá no *layout* e, conseqüentemente, na placa final; o *symbol*, que é o "desenho" que ele terá no esquemático e o *device*, onde se determina quais *pads* do *package* se referem a quais pinos do *symbol*.

Iniciou-se, então, o processo de acrescentar um componente por vez e ligá-lo ao resto do esquemático já feito. Começou-se com a inserção do processador, colocando sua alimentação, capacitores de desacoplamento e "linhas de conexão" (*nets*), depois se adicionou o circuito de alimentação, as memórias e os periféricos, resultando nos esquemáticos que estão no apêndice A.

3.2 *Layout* do Dispositivo

Com o esquemático já terminado e revisado, passou-se para a fase de *layout*. Primeiro se gerou, a partir do esquemático, um *layout* inicial, onde se aparecia todos os componentes, porém sem o roteamento das conexões. Então os componentes foram colocados nas posições mais adequadas e foram roteadas uma a uma todas as mais de 700 conexões existentes.

O Eagle tem um comando que faz todo o roteamento das conexões, o que economizaria muito trabalho, porém esse modo automático não considera uma série de fatores que são críticos no projeto: impedância; caminhos de mesmo tamanho; sem curvas muito abruptas, por exemplo, de 90°, para não ter reflexões; as análises de campo magnético e de dissipação de calor, influenciando nas larguras das linhas conforme as diferentes correntes que passam nelas, as influências que um componente ou sinal pode gerar nos outros.

A etapa mais crítica da confecção do *layout* foi a ligação das memórias SDRAM, já que se tem a necessidade de se ligar mais pinos, de tempos iguais de chegada de todos os dados/endereços/sinais ao seu destino (processador ou memória), além da maior sensibilidade a problemas de impedância. Alguns dos motivos deste problema com a impedância é não se ter um plano contínuo único de terra, levando a problemas de reflexão e de diferentes tempos de chegada dos dados/endereços no seu destino (processador ou memória). Para minimizar esses problemas, foi necessário fazer as conexões entre as memórias e o processador de modo que os caminhos fossem os mais curtos possíveis e de mesmo tamanho.

Após a SDRAM, foram roteados, paralelamente, os conectores USB, o conector serial e seu respectivo módulo de interface serial, por ambos conectores estarem localizados distantes o suficiente do processador para que as conexões de um não interfiram nas de outros componentes. O conector serial foi razoavelmente simples de ser roteado, porém para o conector USB, por possuir sinais diferenciais, ou seja, o que importa mais é a diferença dos níveis entre esses dois sinais, necessitou-se de muito cuidado com o plano de terra e com os diferentes caminhos desses dois

sinais, pois as duas trilhas têm que ter o mesmo comprimento, não se esquecendo de deixar espaço para o roteamento do resto da placa.

Outro bloco importante e crítico do *layout* foi o módulo PHY e seu conector. Essas duas partes também apresentam pares de sinais diferenciais, portanto, o mesmo cuidado tomado com o conector USB foi necessário para se ligar o módulo PHY e o conector *Ethernet*, sendo que neste caso, havia mais pinos a serem ligados que no conector USB.

Tomou-se o cuidado também de posicionar os capacitores de desacoplamento próximo ao respectivo bloco de alimentação. E, quanto aos cristais, fez-se um anel de terra ao seu redor como proteção para diminuir a influência do resto do circuito nesse componente que é muito sensível, além disso, também se posicionou esses cristais próximos aos seus respectivos pinos de atuação.

Para a fase de verificação do *layout*, o Eagle possui uma ferramenta chamada DRC (*Design Rules Check*), em que se definem alguns parâmetros limite como a máxima distância entre linhas, *pads*, vias, componentes e limites da placa (umas com as outras e entre elas mesmas), a espessura da camada de isolação, as distâncias da máscara de solda (*soldermask*), do plano de terra, entre outros. Utilizou-se esta ferramenta como uma das formas de fazer a verificação, porém não se excluiu a análise visual minuciosa de cada membro do grupo que, mesmo com essa ferramenta, detectou outros tipos de erros que o programa não detecta, como a sobreposição de componentes. O resultado final do *layout* pode ser visto no apêndice B.

3.3 Compra dos Componentes Importados

Os componentes, listados na Tabela 5 com seus respectivos preços foram comprados através de uma distribuidora de componentes eletrônicos americana chamada *Digikey* em 20/04/2010:

Tabela 5 - Lista dos componentes importados.

Índice	Quantidade por protótipo	Quantidade total	Número da peça	Descrição	Preço unitário (USD)	Total (USD)
1	1	5	NAND02GR3B2DN6E-ND	IC FLASH 2GBIT 48TSOP	7,56	37,80
2	1	5	KSZ8721CL	IC TXRX PHY 10/100 3.3V 48-LQFP	3,85	19,25
3	1	5	AT91SAM9XE256-QU	MCU ARM9 256K FLASH 208-PQFP	19,14	95,70
4	2	10	LTC3564ES5#TRMPBFCT-ND	IC REG SYNC 1.25A TSOT23-5	4,76	47,60
5	2	10	PCD2230CT-ND	COIL CHOKE 2.2UH SHIELDED SMD	0,64	6,36
6	10	50	445-5223-1-ND	FERRITE CHIP 1000 OHM 1.5A 0805	0,06	2,97
7	1	5	887-1061-1-ND	CRYSTAL 18.432 MHZ 18PF SMD	0,53	2,65
8	1	5	535-9883-1-ND	CRYSTAL 25.0000 MHZ 18PF SMD	0,66	3,30
9	1	5	BS-7-ND	HOLDER COINCELL 2032 RETAINRCLIP	0,59	2,95
10	1	5	296-11988-1-ND	IC LDO VOLT REG SC70-5	1,29	6,45
11	4	20	BAT54CCT-ND	DIODE SCHOTTKY DUAL CC SOT23-3	0,10	1,98
12	2	10	W9812G6IH-6-ND	IC SDRAM 128Mbits 166MHZ 54-TSOPII	2,20	21,97
13	1	5	380-1104-ND	CONN MAGJACK 1PORT SHLD 10/100BT	4,38	21,90
14	1	5	ICL3232CBNZ-ND	IC TXRX DUAL RS232 3-5.5V 16SOIC	2,04	10,20
15	1	5	535-9033-1-ND	CRYSTAL 32.768KHZ 12.5PF CYL SMD	0,64	3,20

Número de Protótipos:	5

Total (USD):	284,28

3.4 Compra dos Outros Componentes

A Tabela 6 mostra os componentes não importados com suas quantidades e preços. Esta compra foi realizada em 19/05/2010.

Tabela 6 - Lista dos componentes nacionais.

Componente	Quantidade por Protótipo	Quantidade Total Comprada	Preço Unitário (R$)	Preço Total (R$)
Capacitor 100nF	50	500	0,06	30,00
Capacitor 10µF	3	50	0,50	25,00
Capacitor 470µF	1	5	0,50	2,50
Capacitor 22µF	4	20	1,20	24,00
Resistor 100kΩ	6	100	0,04	4,00
Resistor 10kΩ	1	100	0,04	4,00
Resistor 15kΩ	1	10	0,10	1,00
Resistor 1kΩ	4	100	0,04	4,00
Resistor 33kΩ	1	10	0,10	1,00
Resistor 4,7kΩ	1	10	0,10	1,00
Resistor 6,49kΩ	1	10	0,10	1,00
Resistor 68kΩ	2	10	0,10	1,00
Botão	4	20	0,20	4,00
Conector USB Host	1	5	0,90	4,50
Conector USB Client	1	5	1,50	7,50
Conector Serial RS-232	1	5	1,10	5,50
Conector de Alimentação Jack DC-005	1	5	0,50	2,50
			Total (R$)	122,50

3.5 Sistema Operacional: Linux

Criado em 1969 por uma equipe de desenvolvedores dos laboratórios Bell Labs, o UNIX surgiu como uma solução para o problema do uso de diferentes sistemas operacionais para cada computador da época, que eram muito caros e consumiam muita potência. Além disso, esses sistemas operacionais utilizavam a linguagem *Assembly* em seu código, de difícil manutenção e expansão. Para resolver esse problema de compatibilidade, o UNIX foi desenvolvido como um sistema operacional simples e elegante, escrito na linguagem C e com um código possível de ser atualizado. O Linux nada mais é que um sistema operacional baseado em UNIX apto para o uso em estações de trabalho, em servidores medianos e também nos mais sofisticados. [30]

Esse sistema operacional apresenta as vantagens de ser gratuito; uma vez que seu código é aberto, tem uma maior facilidade de se encontrar informações e soluções para eventuais problemas; é portátil, podendo ser utilizado em várias plataformas de *hardware*; foi feito para continuar funcionando sem ser reinicializado a todo momento; é seguro e versátil; sua instalação é feito por pacotes, permitindo que se possa escolher os recursos desejados, e, portanto, ocupando menor espaço de memória; além de possuir menor tempo de depuração, já que, por ser um código aberto, várias pessoas já desenvolveram e testaram esse sistema operacional. [30] [31]

No entanto, por ser um sistema de código aberto, ou seja, os programadores podem ler, modificar e distribuir o código, várias distribuições surgiram, cada um sendo melhor para um diferente tipo de aplicação; além disso, ele não é muito amigável para iniciantes. [30] [31]

Outra alternativa para o sistema operacional do dispositivo é o Windows CE que apresenta as vantagens de possuir uma interface gráfica mais amigável com o usuário; seu desenvolvimento é mais rápido para o mercado, porém seu código é fechado e é pago, chegando a custar quase US$1000,00 apenas a ferramenta de desenvolvimento, sem contar a licença, o que inviabilizaria o orçamento e o requisito de baixo custo. [32] [31]

Dessa forma, além do Windows CE ser muito caro, as desvantagens do Linux são facilmente contornáveis, visto que a maior parte das dificuldades encontradas ficaria para o desenvolvedor e não para o usuário final.

3.6 Ambiente de Desenvolvimento do *Software*

Para que seja possível rodar dois sistemas diferentes é necessária uma camada de virtualização que provê a compatibilidade entre eles. Uma máquina virtual é a interface vista por meio dessa camada de virtualização, permitindo o acoplamento entre as interfaces desses dois sistemas, de forma que um programa desenvolvido para uma plataforma possa executar sobre outra. [33]

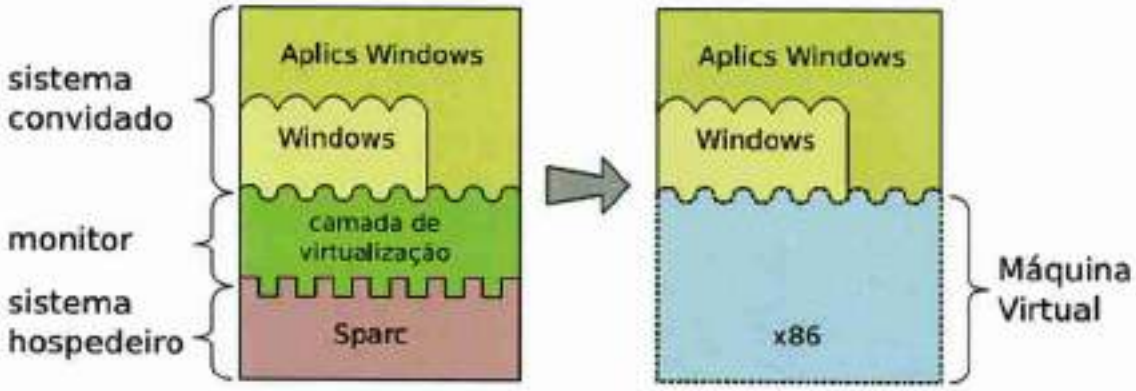

Figura 12 - Uma Máquina Virtual. [33]

Uma máquina virtual consiste de três partes básicas (Figura 12): [33]

1. o sistema real, nativo ou hospedeiro, que contém os recursos reais de *hardware* e de *software* do sistema;
2. o sistema virtual, ou convidado, que executa sobre o sistema virtualizado;
3. a camada de virtualização, ou monitor, que constrói as interfaces virtuais a partir da interface real.

De acordo com o tipo de sistema convidado suportado, os ambientes de máquinas virtuais podem ser divididos em duas grandes famílias: [33]

- máquinas virtuais de aplicação: são ambientes de máquinas virtuais destinados a suportar apenas um processo ou aplicação convidada especifica;
- máquinas virtuais de sistema: são ambientes de máquinas virtuais que suportam sistemas operacionais convidados completos, com aplicações convidadas executando sobre eles.

Para que todos do grupo tenham as mesmas especificações do ambiente de desenvolvimento do *software* e, além disso, não se necessite configurar o ambiente para cada um do grupo, optou-se por realizar todo o desenvolvimento por meio de uma máquina virtual de sistema, o Virtualbox, rodando o sistema operacional Linux, com a distribuição Ubuntu, devido à maior familiaridade dos membros do grupo com esta distribuição.

3.7 Desenvolvimento do *Toolchain*

Antes de se poder instalar o sistema operacional na placa, é necessário gerar algumas ferramentas de programação para que seja possível compilar programas para ela. Esse conjunto de ferramentas de programação é o chamado *toolchain*. Essas ferramentas estão ligadas umas às outras de forma que a entrada de um programa é a saída de outro. São exemplos simples dessas ferramentas um compilador, um *linker* para gerar o executável a partir desse código fonte, bibliotecas para prover a interface com o sistema operacional, um depurador, entre outros. [34] [35]

Para a compilação do projeto, foi necessária a realização de uma compilação cruzada. Esse tipo de compilação é muito utilizado para sistemas embarcados, uma vez que esse tipo de sistema não possui recursos suficientes para rodar um compilador nativo [36].

A compilação cruzada consiste na criação de códigos executáveis para outra plataforma que não é aquela em que a compilação nativa está rodando. [36]

Quando a compilação cruzada é realizada entre máquinas diferentes, existe também a chamada compilação cruz-canadense. Dado 3 máquinas A, B e C, a máquina A é usada para fazer a compilação cruzada que roda sobre a máquina B para criar executáveis para a máquina C. Quando se usa a compilação cruz-canadense com o GCC, pode haver 4 compiladores envolvidos (Figura 13): [36]

1. o compilador nativo para a máquina A (1) é usado para construir o compilador GCC nativo para a máquina A (2);
2. o compilador GCC nativo para a máquiina A (2) é usado para construir o compilador GCC cruzado da máquina A para a máquina B (3);
3. o compilador GCC cruzado da máquina A para a máquina B (3) é usado para construir o compilador GCC cruzado da máquina B para a máquina C (4).

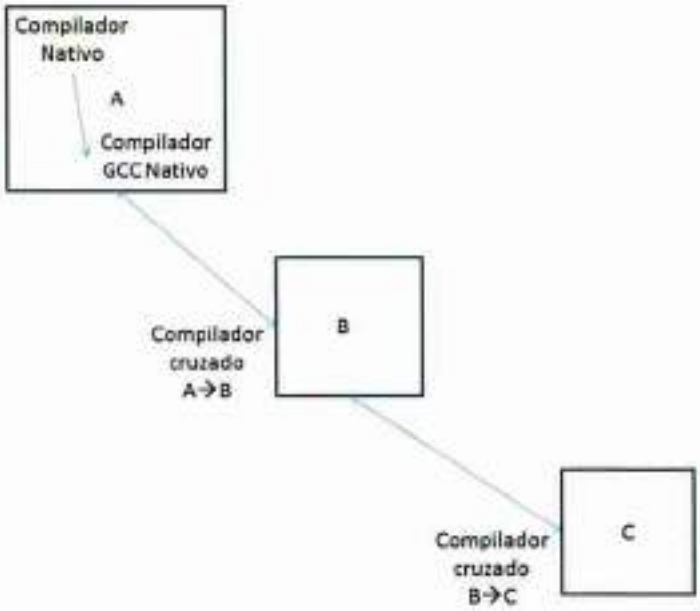

Figura 13 - Esquema de um exemplo de compilação cruz-canadense.

O resultado final da compilação cruzada não será capaz de rodar sobre a máquina A, ao invés disso, seria melhor usá-lo sobre a máquina B para compilar uma aplicação a partir de seu código executável que então seria copiado para a máquina C e executado na máquina C. [36]

No caso, a máquina virtual de desenvolvimento seria a máquina A, que geraria código para a máquina B, ou seja, a placa do Servidor Pessoal. Como a placa não gera código, não haverá uma cruz-canadense, mas simplesmente uma compilação cruzada.

3.8 Programa SAM-BA

O *SAM Boot Assistant* (SAM-BA) é um *software* que fornece uma maneira fácil de inicializar e programar dispositivos com processadores Atmel da família AT91SAM. Ele é baseado em uma biblioteca com ligação dinâmica (DLL), a SAMBA_DLL.DLL. [37] [38]

Principais Características do *software* SAM-BA:

- executa a programação no sistema, através das interfaces JTAG, RS232 ou USB;
- permite a programação tanto na memória *Flash* incorporada no processador quanto na memória externa do dispositivo;
- pode ser utilizado através de uma interface gráfica (GUI) ou iniciado por uma janela DOS;
- roda em Windows 2000 e XP, e também em Linux.

O SAM-BA pode operar no modo gráfico ou pode ser iniciado em linha de comando. Ambos os modos podem ser combinados para obter facilmente uma solução poderosa para programar dispositivos com processadores AT91SAM customizado para qualquer projeto. No modo gráfico, a primeira coisa que se deve fazer é identificar a porta para a conexão com o dispositivo e seu *design* como mostra a Figura 14.

Figura 14 - Janela de escolha de protocolo. [37]

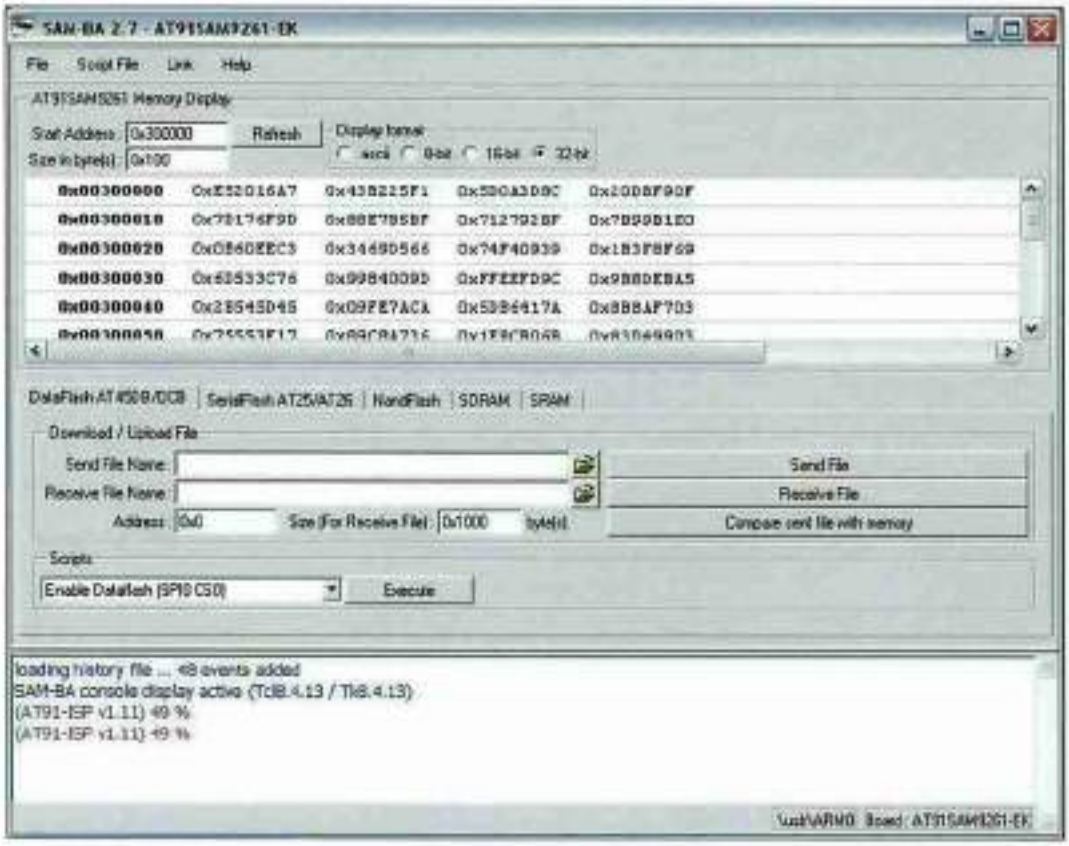

Figura 15 - Janela principal do SAM-BA. [37]

Na Figura 15 vê-se a interface principal do usuário que permite a programação do processador AT91SAM9XE.

3.9 Processo de inicialização e *Boot*

Da mesma forma que um computador pessoal (PC) moderno necessita da BIOS (*Basic Input/Output* System) para sua correta inicialização e posterior transferência do controle ao sistema operacional, o dispositivo desenvolvido, um servidor pessoal, também precisa de um código especialmente adaptado para sua inicialização.

3.9.1 Processador

O processador escolhido para o projeto é o AT91SAM9XE256, que permite o *boot* (primeira instrução executada) de dois blocos distintos de memória:

- memória ROM: Nesta memória do tipo somente-leitura fica armazenado o programa da Atmel capaz de se comunicar com o PC e realizar todas as tarefas do *software* SAM-BA;
- memória *FLASH*: Esta memória não-volátil de 256 kB contém um programa específico para o projeto, capaz de inicializar a placa e transferir o controle ao sistema operacional.

A escolha entre um modo e outro se dá por meio de um bit especial armazenado em uma posição especial de um pequeno bloco de memória *Flash* de configuração. Assim, por meio do *software* SAM-BA é possível carregar um programa na memória *Flash* do processador e, em seguida, ativar o bit de configuração responsável por forçar o carregamento do *software* específico ao invés daquele fornecido pela Atmel. O processo está representado na Figura 16.

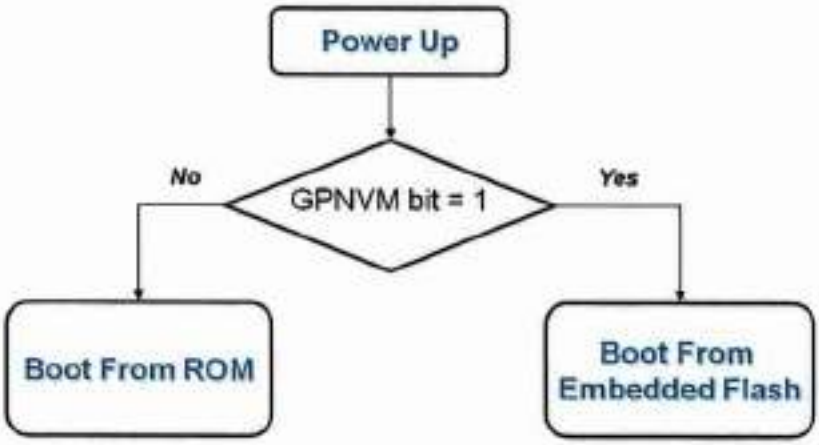

Figura 16 - Inicialização do Processador. [39]

Entretanto, é possível que se queira reprogramar a memória interna do processador e, para tanto, será necessário executar o *software* da Atmel armazenado na ROM. Assim, por meio de um pino específico do processador, devidamente ligado a um *jumper* na placa desenvolvida, é possível limpar toda a memória *Flash*, inclusive os bits de configuração e, assim, permitir que a ROM seja executada e sua memória reprogramada na próxima vez em que o processador for ligado.

3.9.2 Inicialização do *Hardware*

Ao alimentar-se a placa, a primeira tarefa a ser realizada é a inicialização do *hardware* da placa, como PLLs (*clocks*) e principalmente a memória SDRAM (*Synchronous Dynamic Random Access Memory*) de 32MB de capacidade, uma vez que a memória RAM interna ao processador (de tipo SRAM, *Static Random Access Memory*) é de meros 32 kB. Além disso, é interessante indicar de alguma forma que o processador começou a executar código, por exemplo, pelo acendimento de um LED ou até mesmo o envio de uma seqüência de caracteres pela porta serial de protocolo RS-232.

Assim, para realizar esta tarefa, foi escrito um *software* baseado no programa AT91Bootstrap em sua versão 1.16, descrito neste relatório e fornecido com código-fonte pela Atmel (que é mostrado na seção Programação do Dispositivo através do SAM-BA).

Por meio do *toolchain* criado, é possível adaptar e compilar uma nova versão do AT91Bootstrap capaz de inicializar a placa e indicar seu funcionamento.

3.9.3 Inicialização do *Software*

Com os elementos de *hardware* básicos inicializados, é possível passar à inicialização dos elementos de *software* necessários à transferência do controle para o Sistema Operacional.

Desta forma, é necessário que se implemente um programa que carregue o *kernel* do Sistema Operacional na memória RAM, ou seja, carregue este arquivo de um sistema de arquivos e execute-o com o cuidado de transferir os comandos de inicialização corretos a ele.

Uma característica desejável na fase de desenvolvimento é a possibilidade de se inicializar já nesta fase o controlador de Ethernet para que se possa carregar o *kernel* de uma pasta compartilhada na rede, evitando-se transferir um arquivo a cada

revisão e ganhando-se sensivelmente em tempo. Também é muito interessante a possibilidade de se utilizar um sistema de arquivos montado por rede (NFS – *Network File System*) como raiz para o Linux, permitindo que não só se evite transferir o *kernel* a cada revisão, mas também que não se transfira o sistema de arquivos a ser instalado na placa.

A tarefa de simplesmente procurar e ler um arquivo em um sistema de arquivos completo, como ext3 ou JFFS2, já seria desafiadora do ponto de vista de *software*. Porém, oferecer ainda todas as possibilidades desejadas na fase de desenvolvimento necessitaria de um esforço ainda maior.

Felizmente, uma série de desenvolvedores de plataforma embarcadas (sejam elas baseadas em ARM, MIPS, PowerPC ou tantos outros) detectaram estas mesmas necessidades. Assim, nasceu um projeto de código aberto chamado U-Boot.

Por meio de uma série de adaptações relativamente complexas (mapeamentos de IOs (*Input/Output's*), *drivers* para o controlador de Ethernet em modo MII, *drivers* para memórias NAND *Flash* e muitas outras), foi possível compilar o U-Boot para a placa com o *toolchain*.

Finalmente, como desafio adicional realizado com sucesso, foi possível adicionar o código do U-Boot ao AT91Bootstrap de forma que todo este novo programa coubesse na pequena memória *flash* de 256 kB do processador.

3.9.4 Ciclo de Inicialização Completo

Com base nos itens a seguir, podemos resumir todo o processo de inicialização da placa na Figura 17.

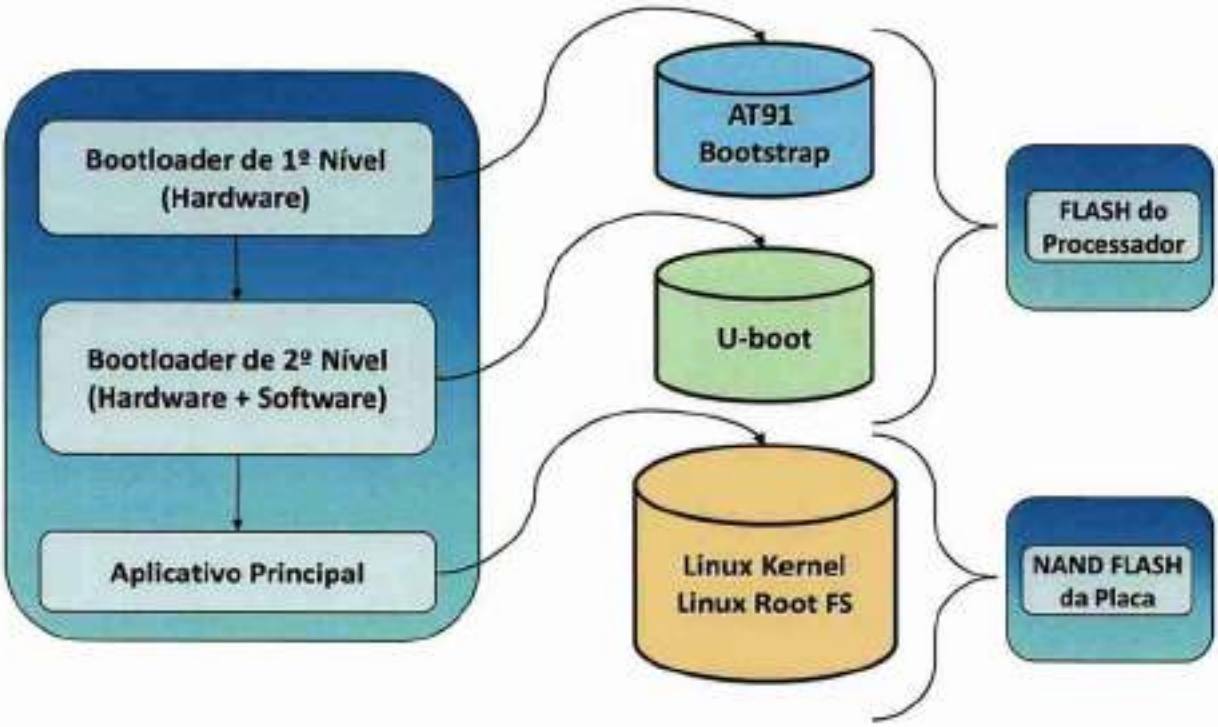

Figura 17 - Estágios de Inicialização e *Bootloaders*. [39]

3.9.5 AT91Bootstrap

AT91Bootstrap é um *bootloader*, o primeiro passo para fornecer um conjunto de algoritmos para gerenciar a inicialização de *hardware* (GPIO, Relógio, SDRAM, etc), para baixar o aplicativo principal da memória *Flash* para a memória principal e iniciá-lo [40].

O programa AT91Bootstrap é o primeiro nível de gerenciamento da inicialização para microprocessadores da família AT91SAM9 da Atmel. Esse programa pode facilmente ser usado para personalizar uma estratégia de implementação da programação requerida do processador, pois ele trabalha em limguagem C.

O AT91Bootstrap também fornece exemplos claros de como realizar configurações básicas de vários módulos do processador, como PMC (*Power Management Controller*), SDRAMC (*SDRAM controller*), DBGU (*Debug unit*), GPIO (*general purpose input output*), entre outros módulos, sendo que alguns não serão usados como o *DataFlash controller*.

A seguir será explicado um pouco mais sobre cada um dos módulos usados durante o projeto:

- PMC (*Power management controller*): é usado para otimizar o consumo de energia controlando os relógios em cada parte do processador. O PMC habilita ou desabilita as entradas dos relógios em muitos módulos do processador;
- SDRAMC (*SDRAM controller*): é usado para estender a capacidade de memória interfaciando com uma memória SDRAM externa. Suporta memórias com páginas de tamanho de 2048 até 8192 bytes e o número de colunas de 256 até 2048. Suporta acessos por byte (8-bit), meia palavra (16-bit) e palavra (32-bits);
- DBGU (*Debug unit*): é usado para se poder verificar erros na execução dos programas e como terminal para o Linux;
- GPIO (*general purpose input output*): é usado como entrada ou saída de sinais necessários para o funcionamento da placa, como alguns sinais do módulo de Ethernet, para a saída de expansão e para os botões e LEDs de teste da placa;
- *NAND Flash controller*: é usado para controlar a memória NAND *Flash* presente na placa.

Além disso, o AT91Bootstrap integra vários conjuntos de algoritmos para:

- inicialização do dispositivo;
- configuração da velocidade do relógio;
- controladores de periféricos, tais como GPIO, PMC, SDRAMC;
- *drivers* para *DataFlash* e NAND *Flash*;
- algoritmos de compressão e codificação.

Portanto, utilizando este conjunto de algoritmos, é possível obter um *bootloader* básico que será colocado na memória *Flash* interna, a partir de um computador pelo programa SAM-BA. O *bootloader* executa a inicialização do processador e de seus módulos (PLLs, PIOs, SDRAMC, SPI), carrega o próximo programa de inicialização (o segundo passo da inicialização), o U-Boot, e depois salta para ele.

3.9.6 U-Boot

Novos dispositivos embarcados Linux estão aparecendo no mercado a uma taxa surpreendente. Desde aparelhos pequenos de 3 polegadas como o "Gumstix" até *tablets*, PDAs e smartphones estão recebendo Linux embarcado. A instalação e inicialização do Linux sobre estas placas extremamente diferentes entre si é uma tarefa complexa. [41]

Um *bootloader* é como um programa monitor da inicialização: um pequeno pedaço de *software* que é executado logo após se ligar o computador. Por exemplo, num computador *desktop* Linux é um programa como o LILO ou GRUB, que reside no *master boot record* (MBR) do disco rígido que faz a função de *bootloader*. Depois que a BIOS do computador faz a primeira inicialização do *hardware* (no caso deste projeto essa parte é feita pelo AT91Bootstrap), ele executa o *bootloader* localizado no MBR. Então ele passa as informações do sistema para o *kernel* e o executa. [41]

No mínimo, um *bootloader* embutido oferece os seguintes recursos: [41]

- inicialização do *hardware*, especialmente o controlador de memória;
- fornecendo parâmetros de inicialização para o *kernel* Linux;
- passa o controle para o *kernel* do Linux.

O U-Boot é um *bootloader* GPL multi plataforma responsável pelas mesmas funções dos programas do MBR mencionados anteriormente. O U-Boot fornece suporte para centenas de placas e uma grande variedade de processadores, incluindo PowerPC, ARM, XScale, MIPS, Coldfire, NIOS, MicroBlaze e x86. Pode-se facilmente configurar U-Boot para encontrar o equilíbrio certo entre o conjunto de recursos necessários e um tamanho reduzido de programa. [41]

O U-Boot tem suas origens no projeto 8xxROM, um carregador de *boot* para sistemas PowerPC 8xx por Magnus Damm. No ano 2000 o projeto foi trazido para o *Sourceforge*, onde o atual líder do projeto, Wolfgang Denk, renomeou o projeto para PPCBoot. [41]

No projeto, ele é responsável por configurar interfaces principais e lançar o sistema Linux. No entanto, é possível evitar este passo e iniciar o Linux diretamente do AT91Bootstrap, em fase de produção, por exemplo [40].

Com o U-Boot é possível saltar logo no começo do projeto diretamente para o Linux embarcado, deixando-o cuidar das inicializações de baixo nível. Se surgir a necessidade de se adicionar algo ao projeto, pode-se fazer o *download* de *drivers* para novos *hardwares* ou adicionar recursos especiais, como foi feito no caso deste projeto. [41]

A seguir são apresentados alguns comandos do U-Boot: [42]

- flinfo – imprime na tela a informação sobre a memória *Flash*;
- go 'addr' – começa um aplicativo de um determinado endereço;
- ls – lista os arquivos do diretório atual;
- ping 'address' – envia um pacote de PING no protocolo TCP/IP para um endereço da rede local;
- setenv 'variable' 'value' – configura alguma variável do sistema.

3.9.7 *Kernel*

O *kernel* é o núcleo de um sistema operacional, é ele quem controla todo o *hardware*. Constitui-se de vários arquivos em linguagem C e em *Assembly* que fazem a interface entre os programas e o *hardware* do computador, permitindo que todos os processos sejam executados pela CPU e que estes consigam compartilhar a memória do computador. [43]

No início da década de 90, Linus Torvalds desenvolveu o Linux a partir de um sistema operacional baseado em UNIX, na verdade ele criou o primeiro *kernel* do Linux e, a partir de então, bastava adicionar os aplicativos a esse *kernel* e adaptá-lo para cada aplicação para que um sistema operacional surgisse. Portanto, é a partir dessa personalização que surgem as diferentes distribuições do Linux. [43]

Optou-se por personalizar esse *kernel* do Linux, ao invés de usar uma distribuição já existente, para o presente projeto por se tratar de um sistema embarcado, e que, portanto, quanto menor o tamanho do núcleo, mais se economiza em memória e, assim, a adaptação é mais efetiva, uma vez que se consideraria apenas a parte do núcleo que é de interesse para o projeto.

Com o tempo, novas versões do Linux são lançadas para corrigir vulnerabilidades, prover melhorias em determinadas funções, adicionar e/ou melhorar a compatibilidade dos recursos sobre as versões anteriores. Dessa forma, decidiu-se utilizar a versão do *kernel* mais atualizada que seja estável, ou seja, a versão 2.6.33.2. [43]

A seguir temos a seqüência de seu funcionamento para um PC: [43]

- na inicialização do sistema, quando são lidas as instruções contidas na MBR (*Master Boot Record)* responsáveis por indicar ao BIOS do computador como e onde carregar o sistema operacional:
    - o *kernel* detecta os dispositivos de *hardware* essenciais do computador;
    - a imagem do *kernel* é carregada;
    - o *kernel* confirma a presença da memória e a prepara para o uso por meio de uma função de paginação;
    - as interrupções, os discos, a memória-cache, entre outros, são acionados em seguida.
- com o sistema operacional pronto para funcionar:
    - o *kernel* carrega as funções responsáveis por verificar o que deve ser inicializado em nível de *software* e processos, normalmente o que é carregado é a tela de *login* do usuário.
- com o usuário logado:
    - o *kernel* executa suas funções, como a de controlar o uso da memória pelos programas ou a de atender a chamada de uma interrupção.

3.10 Projeto de *Software*

Com base nos requisitos estabelecidos para o projeto e o tipo de plataforma a ser desenvolvida, foi definida uma forma de se trabalhar que permitisse o desenvolvimento de *software* otimizado para a arquitetura utilizada e que ao mesmo tempo fosse prática.

Assim, de um ponto de vista de *software*, foram identificados os seguintes pontos como cruciais para que o projeto pudesse realizar seus objetivos:

1. compilação cruzada e depuração dos *Bootloaders* de 1° e 2° nível;
2. compilação cruzada e depuração do *kernel* do Linux;
3. criação e manutenção de um sistema raiz de arquivos (*Root File System*) básicos;
4. compilação cruzada de aplicativos para a plataforma.

Para que se pudessem atender todos estes pontos de forma efetiva e eficiente, prosseguiu-se com uma análise de uma série de ferramentas disponíveis, comercialmente ou de código aberto, para que se pudesse realizar da melhor forma possível os requisitos específicos citados acima.

Assim, para que se possa realizar a compilação cruzada de qualquer programa, seja ele o *kernel* ou um *Bootloader* de 1° nível, é necessário que se tenha um *toolchain*, ou seja, um conjunto de ferramentas (compiladores) capazes de gerar código para a plataforma.

Uma vez de posse do *toolchain*, a geração do sistema de raiz de arquivos pode ser feita de forma bastante facilitada, embora não simples, pelo programa *Buildroot*, de código aberto.

Finalmente, a compilação cruzada de aplicativos para a plataforma pode ser feita de muitas formas, entretanto, uma tendência que vem se solidificando no mercado é o uso de *recipes* (receitas) que permitem realizar esta tarefa. Assim, optou-se pelo uso da solução *OpenEmbedded* juntamente com *BitBake*, dois *softwares* extremamente poderosos e também de código aberto.

A Figura 18 representa os principais programas escolhidos para o projeto de *software*, além de suas interdependências:

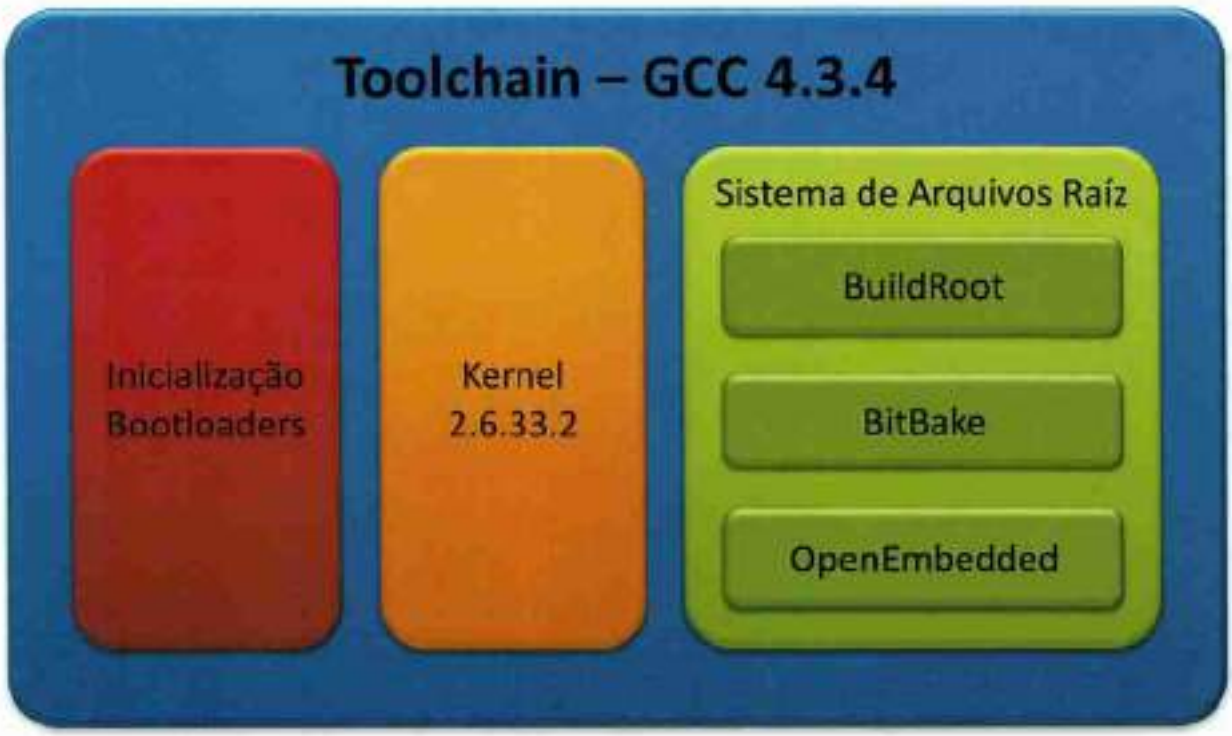

Figura 18 - Principais programas e suas interdependências

Vale salientar que as diferentes ferramentas escolhidas, assim como as características que levaram à sua escolha, serão apresentadas a seguir.

3.10.1 Sistema de Arquivos Raiz

Conforme [44], a criação de um Sistema de Arquivos Raiz (*Root File System*, ou *RootFS*) corresponde à seleção e disponibilização de um mínimo de arquivos necessários para que o sistema possa funcionar.

Assim, no caso do sistema operacional Linux, deve-se fornecer os seguintes itens:

- um sistema de arquivos básico (ext3, ext4, nfs, JFFS2);
- alguns diretórios obrigatórios, como `/dev`, `/proc`, `/bin`, `/etc`, `/lib`, `/usr`, `/tmp`;
- alguns programas obrigatórios: `sh`, `ls`, `cp`, `mv`, etc;

- arquivos de configuração: `rc`, `inittab`, `fstab`, etc;
- os dispositivos presentes: `/dev/hd*`, `/dev/tty*`, `/dev/fd0`, etc;
- bibliotecas de tempo de execução (neste caso, glibc).

Existe uma série de especificações do que é necessário, recomendável e também do que não é permitido para um sistema de arquivos raiz de Linux. Estas especificações são concentradas no documento *Filesystem Hierarchy Standard*, desenvolvido pelo *Filesystem Hierarchy Standard Group* e disponível em sua versão atual (2.3) [45].

3.10.1.1 Criação do Sistema de Arquivos Raiz - Buildroot

A criação de um Sistema de Arquivos Raiz é uma tarefa bastante complexa, mas por outro lado também bastante comum quando do desenvolvimento de distribuições (sejam elas muito famosas como Ubuntu ou Debian, ou menores como Angstrom).

Por isto, existem algumas ferramentas capazes de auxiliar o desenvolvedor a criar um sistema de arquivos básico, já com os aplicativos necessários e para qualquer arquitetura. Uma destas ferramentas é o *Buildroot* (BR), cujo símbolo é mostrado na Figura 19.

Figura 19 - Logotipo do *BuildRoot*

O *Buildroot* consiste em um conjunto de *Makefiles* e *patches* que facilitam a geração de um Sistema de Arquivos Raiz para um sistema Linux embarcado. Em sua versão atual, possibilita a compilação de um *toolchain*, a criação de um *RootFS* (função para a qual foi utilizado), a compilação do *Kernel* e a compilação de um *bootloader*.

Assim, ele é útil principalmente para pessoas trabalhando com pequenos dispositivos ou dispositivos embarcados, que utilizem uma arquitetura como x86, ARM, MIPS, PowerPC entre outras.

Algumas possibilidades muito interessantes oferecidas pelo *Buildroot* são:

- fácil configuração por meio de menu gráfico (*menuconfig*, *gconfig* ou *xconfig*), com interfaces familiares para os desenvolvedores. De forma geral, uma imagem básica leva de 15 a 30 minutos para ser gerada. Neste caso, entretanto, como o *toolchain* foi compilado externamente e o *BuildRoot* só gera o *RootFS*, o tempo é de menos de 5 minutos;
- suporte a mais de uma centena de pacotes de aplicativos, como X.org, Gtk2, Qt, SDL, Gstreamer e outros mais simples relacionados à rede e sistema. Novamente, esta opção não foi usada por não oferecer tanta flexibilidade quanto o *OpenEmbedded* e *BitBake* descritos mais a frente;
- permite criar a imagem final em diversos formatos: JFFS2, UBIFS, tarballs, romfs, cramfs, squashfs e outros. No caso deste projeto, foi usado o *tarball*, por permitir fácil integração com o *OpenEmbedded* e *BitBake*;
- permite gerar um *toolchain* baseado em uClibc, ou reutilizar um existente baseado em glibc, eglibc ou uclibc. Foi utilizado um *toolchain* baseado em glibc, a mais completa das bibliotecas, embora seja a maior em tamanho;
- depende exclusivamente da linguagem do aplicativo *make*, tornando muito simples sua adaptação e extensão.

Desta forma, é fácil concluir que o *BuildRoot* atende e muitas vezes até excede os requisitos estabelecidos para a construção do Sistema de Arquivos Raiz. Um exemplo de tela de configuração deste poderoso *software* é mostrado na Figura 20.

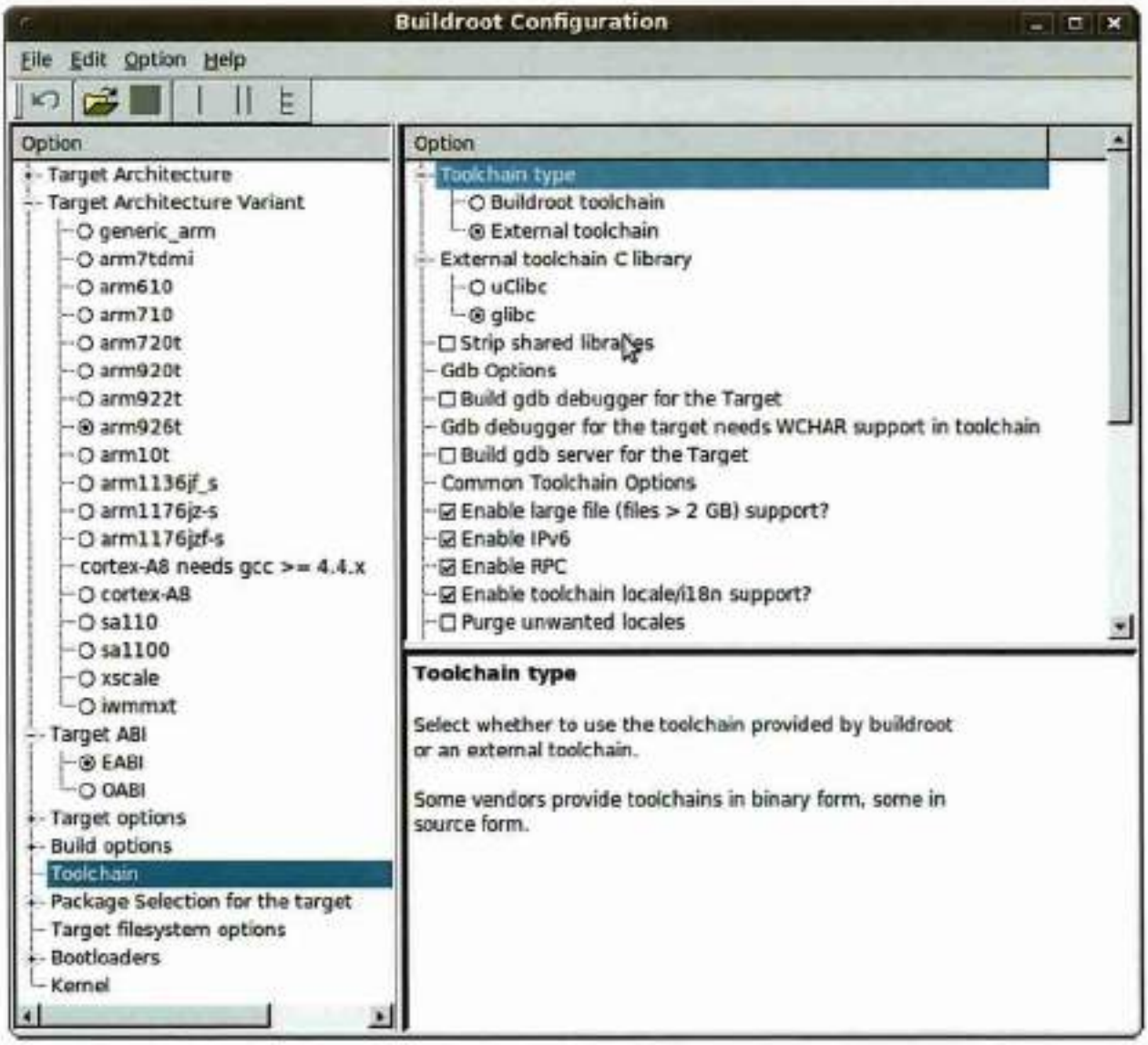

Figura 20 - Tela de configuração do *BuildRoot*

3.10.1.2 Compilação dos Aplicativos

Uma vez o Sistema de Arquivos Raiz devidamente criado e testado por meio do *BuildRoot*, foi possível passar à etapa posterior, a compilação de aplicativos para a plataforma.

Uma grande parcela dos *softwares* disponíveis para Linux são fornecidos com código aberto, de forma que todos poderiam compilar este código para a plataforma de sua escolha. Assim, muitos *softwares* mais complexos, ou seja, que tem muitos

arquivos fonte ou que "se adaptam" à arquitetura para qual serão compilados possuem *scripts* de configuração automática, por exemplo, os famosos *configure*, *autoconf* e *automake*.

Entretanto, muitos grupos de desenvolvedores têm por alvo uma determinada arquitetura, normalmente x86 (o padrão para *desktops*, *notebooks* e até servidores). Desta forma, são feitas otimizações que normalmente se comportam muito mal durante compilações cruzadas, inviabilizando sua realização direta.

Sendo assim, é muitas vezes necessário alterar o código-fonte original do aplicativo para torná-lo compatível com os procedimentos de compilação cruzada. Independentemente da plataforma visada, as adaptações são bastante similares, pois consistem numa generalização de algumas partes do código.

Assim, pode-se ver que o processo de analisar o código fonte de cada um dos aplicativos que se deseja executar no dispositivo pode rapidamente se tornar uma tarefa que toma muito tempo e seja até inviável.

Felizmente, este é um problema que também é enfrentado diretamente por muitos desenvolvedores de plataformas embarcadas. Por isto, existem muitas soluções para este tipo de problema.

A distribuição Debian, por exemplo, oferece pacotes pré-compilados de muitos aplicativos para a arquitetura armv4t, compatível com este projeto, mas um pouco menos eficiente que a armv5te implementada por eles. Já o Ubuntu só fornece pacotes para armv7, ou seja, processadores muito mais complexos (e de maior consumo) que o utilizado.

Um aplicativo relativamente simples que permite compilar aplicativos inteiros de forma cruzada é o *ScratchBox*. Entretanto, este *software* requer que para cada um dos pacotes o usuário tenha o trabalho de fazer a generalização e adaptação dos arquivos fonte para a plataforma de destino, ou seja, o trabalho é facilitado, mas é basicamente o mesmo.

É justamente para resolver este problema bastante comum, mas de grandes dimensões que surgiu uma ferramenta chamada *BitBake*.

3.10.1.3 BitBake

Oficialmente criado em 7 de Dezembro de 2004, o *BitBake* (BB) é fruto do trabalho de desenvolvimento do *OpenEmbedded*. Por ser um componente altamente flexível deste, decidiu-se oferecê-lo por si só, sem necessariamente estar ligado a quem lhe deu origem, o *OpenEmbedded*.

De forma muito simplificada, o *BitBake* é uma ferramenta para executar tarefas e gerir metadados. Assim, suas semelhanças com a ferramenta *make* e outras de compilação são evidentes. Ele foi muito inspirado pelo *Portage*, o gestor de pacotes usado pela distribuição Gentoo de Linux. Ainda hoje, o *BitBake* é a base do projeto *OpenEmbedded*, que é usado para compilar e atualizar uma série de distribuições embarcadas de Linux, incluindo a OpenZaurus, que lhe deu origem.

Antes da criação do *BitBake*, nenhuma outra ferramenta atendia de forma adequada as necessidades das distribuições Linux embarcadas que surgiam [46]. Todas as ferramentas de compilação presentes nas distribuições Linux para *desktop* não tinham funcionalidades importantes e, por sua vez, as ferramentas específicas não podiam ser expandidas e nem mantidas facilmente.

Assim, alguns objetivos estabelecidos e atendidos em sua totalidade pelo *BitBake* são:

- suporte total à compilação cruzada;
- suporte total às dependências entre pacotes, tanto em tempo de execução quanto de compilação e tanto no sistema usado para compilar os pacotes quanto no alvo;
- suporte à execução de um número ilimitado de tarefas, como baixar código fonte, descompactação, adaptação (*patching*), configuração e muitas outras;
- independente da distribuição Linux;
- independente da arquitetura;
- compacto, ou seja, todos os arquivos estão dentro dele ou de suas pastas, e não em uma árvore específica do sistema anfitrião;

- facilidade de adaptação;
- herança de dependências.

Esta ferramenta atendeu todos estes objetivos e comprovou que, de fato, flexibilidade e capacidade sempre foram suas prioridades, sendo também muito extensível. A título de curiosidade, o *BitBake* é desenvolvido e distribuído em *Python*.

Pode-se ver então que ele se adapta muito bem ao modelo de *recipes* (receitas), onde se especifica qual o código fonte que se deseja compilar, as adaptações que precisam ser feitas (*patches*) e quais otimizações são necessárias usar e habilitar na hora de compilar. Assim, um exemplo de utilização é o comando:

```
bitbake openssl
```

que permite a compilação do pacote *openssl* de forma muito mais simples do que se feita manualmente.

Finalmente, resta ainda a pergunta de qual a receita que explica como compilar um dado pacote (por exemplo, o *openssl* como mostrado no exemplo). É justamente para isso que se utilizou o *OpenEmbedded*, **um "livro de receitas"**.

3.10.1.4 OpenEmbedded

O *OpenEmbedded* (OE) é um projeto de código aberto que permite a desenvolvedores de sistemas embarcados baseados em Linux criar uma distribuição completa. Algumas de suas caracteristicas principais são:

Figura 21 - Logotipo da OE

- suporte a uma série de arquiteturas;
- ferramentas que evitam a recompilação de muitos pacotes, quando se altera algo em apenas um deles;
- fácil customização;
- pode ser executado em qualquer distribuição Linux;

- executa a compilação cruzada de milhares de aplicativos, contendo mais de 7000 receitas.

De forma simplificada, o *OpenEmbedded* trabalha com uma estrutura de diretórios em árvore com as seguintes 4 pastas principais:

- *build*: Nesta pasta ficam armazenados os pacotes finais gerados além do código fonte em forma descompactada utilizado para compilá-los;
- *overlay*: Esta pasta permite que o desenvolvedor sobreponha (do inglês, *overlay*) receitas próprias às fornecidas pelo *OpenEmbedded*, permitindo customizar absolutamente todas as receitas;
- *bitbake*: Aqui ficam guardados os arquivos executáveis e gerais referentes ao *BitBake*, como exposto no item anterior;
- *openEmbedded*: Aqui ficam guardadas as receitas oferecidas pelo *OpenEmbedded* para possibilitar a compilação de mais de 7000 pacotes.

Assim, pode-se ver que esta ferramenta, embora bastante complexa em primeira análise, é muito poderosa e atende aos requisitos estabelecidos para a compilação dos aplicativos necessários ao projeto.

3.11 Seleção dos tipos de aplicativos

A primeira etapa na seleção dos aplicativos foi da escolha de quais tipos de aplicativos seriam necessários para atender, pelo menos, aos requisitos de *marketing* e de engenharia levantados no início do projeto.

Para permitir o acesso aos dados é necessário um servidor SSH (*Secure Shell*). Esse servidor permite a conexão com outro computador na rede que executaria comandos nesse servidor remotamente. Com essa conexão o usuário poderia acessar seus arquivos e, sabendo que tudo em um computador nada mais são que arquivos [30], sejam eles de dados, de configuração ou ligações a outros arquivos, o servidor SSH permitiria também que o usuário pudesse alterar as configurações do

dispositivo, ou até mesmo instalar outros programas, precisando apenas que o usuário tenha permissão para tal.

A implementação de uma interface mais amigável com o usuário também é um requisito. A melhor alternativa no caso de uma aplicação embarcada é por meio de um servidor web e, conseqüentemente, a escolha de uma linguagem de programação apropriada também é necessária.

Quanto ao requisito de baixo custo, baixo consumo e tamanho reduzido, a escolha de um aplicativo influenciaria no fato de haver alguns servidores que são pagos e/ou muito grandes para uma aplicação embarcada.

Dessa forma, será necessário fazer a escolha dos seguintes aplicativos principais: servidor SSH, servidor web e uma linguagem de programação para a geração dos *scripts*.

Para a escolha do servidor SSH, primeiramente, fez-se um levantamento geral dos servidores SSH existentes: CopSSH, Dropbear, FreeSSHD, KpyM, LSH, OpenSSH, Tectia, VShell entre outros. Com essa lista em mãos, separou-se aqueles que não rodam em Linux (por exemplo, o CopSSH e o KpyM) e aqueles com muito pouca informação como uma forma de minimizar os riscos (como o FreeSSHD). Chegando-se a seguinte lista inicial de servidores SSH:

- dropbear;
- LSH;
- openSSH.

O OpenSSH também possui a versão para sistemas embarcados, porém, foi lançada muito recentemente, o que reduziria as informações sobre este servidor para uma aplicação nesse momento.

Da mesma forma que para o servidor SSH, também se fez um levantamento dos servidores web existentes: APACHE, BadBlue, Boa, Lighttdp, thttpd, Nginx, entre outros. Após se obter essa lista, foram escolhidos aqueles com mais informações, os mais citados e com boas indicações, além de atender ao requisito de rodar no sistema operacional utilizado, resultando nos seguintes servidores web:

- APACHE HTTP Server;
- boa;
- light HTTP Server (Lighttpd);
- thttpd.

Para as linguagens de programação, foram estudadas: HTML, JavaScript, PHP, PERL, JSP e Shell (Ash, Bash, Dash), sendo que HTML é para sites estáticos e as mais conhecidas pelo grupo são: Shell, PHP e PERL.

A partir das listas preliminares dos aplicativos levantadas anteriormente, passou-se então à escolha final dos aplicativos a serem utilizados.

Para o servidor SSH, optou-se pelo OpenSSH por atender todos os requisitos, além de possuir o maior número de informações tanto com relação aos estudos realizados como em relação à familiaridade do grupo com ele.

Para o servidor web, o Lighttpd foi o eleito para a presente aplicação. Além da melhor familiaridade pelo grupo, esse servidor é comparável ao Apache, sem dúvida com maior quantidade de informações e indicações, porém a memória utilizada por esse último é extremamente maior. Já os servidores web Boa e Thttpd não possuem o suporte ao protocolo HTTPS, sendo assim descartados.

Para a interface web decidiu-se pelo HTML, devido, novamente, à familiaridade de maior parte dos integrantes do grupo. Além disso, pela linguagem produzir interfaces estáticas, necessitou-se da escolha de outra linguagem para dar dinamismo à interface, o que é indispensável na presente aplicação, elegendo o JavaScript em conjunto com o CGI em linguagem *Shell* e o CSS. O primeiro foi utilizado principalmente pela grande relação com o HTML e por não ser muito complexo; já o segundo, pela vantagem de estar presente em qualquer distribuição Linux, fazendo com que a página possa rodar no servidor e se comunicando com outros programas. A necessidade do terceiro veio da compatibilidade com os atuais navegadores existentes, resultando em uma melhor organização da página. Utilizou-se também a linguagem CSS para as formatações das páginas. Esse assunto será melhor abordado em "Interface Web".

3.12 UBIFS - Sistema de Arquivos UBI

O UBIFS (*Unsorted Block Image File System*) é um novo sistema de arquivos para memórias *Flash* desenvolvido por engenheiros da Nokia com a ajuda da Universidade de Szeged.

Ele é muito diferente de qualquer sistema de arquivos tradicional, pois não trabalha sobre dispositivos de bloco, como discos rígidos, cartões MMC / SD, *drives flash* USB, discos SSD, etc. O UBIFS foi projetado para trabalhar sobre *chips* de memória *Flash*.

De certa forma, o UBIFS pode ser considerado como a próxima geração do sistema de arquivos JFFS2 (*Journalling Flash File System version 2*), que também é usado em memórias *Flash*. Porém, o sistema de arquivos JFFS2 funciona sobre o MTD (*Memory Technology Device*), enquanto que o UBIFS funciona sobre volumes UBI, uma camada acima do MTD. Em outras palavras, existem três subsistemas envolvidos [47]:

- subsistema de MTD, que fornece uma interface uniforme para acessar os chips de memória *Flash*;
- subsistema UBI, que é um sistema de gerenciamento de volume para as memórias *Flash*. O UBI funciona em cima de dispositivos MTD, ou seja, é uma entidade de nível superior do que os dispositivos MTD e que cuida do gerenciamento dos arquivos em si e não com os problemas do gerenciamento dos arquivos nas memórias *Flash*, como desgaste e *bad blocks*;
- UBIFS é o sistema de arquivos que trabalha com o subsistema UBI.

A seguir está uma lista de algumas das características UBIFS [47]:

- escalabilidade – UBIFS funciona bem independente do tamanho da memória, ou seja, tempo de montar a memória, o consumo de memória e a velocidade de entrada e saída dos dados praticamente não dependem do tamanho da memória *Flash*. Assim, o UBIFS deveria funcionar bem para memórias *Flash* de centenas de *Gigabytes*; no entanto, como depende do UBI, acaba tendo algumas

limitações de escala (o UBI é linear com relação ao tamanho da memória), ainda assim a escalabilidade do UBI / UBIFS é muito melhor do que a do JFFS2;

- montagem rápida - ao contrário do JFFS2, o UBIFS não tem que verificar toda a memória *Flash* quando faz a montagem, o que leva milissegundos, e não depende do tamanho da memória *Flash*. No entanto, existe um pequeno tempo de inicialização no UBI que depende do tamanho da memória e tem que ser levado em consideração;
- suporte para *write-back* - isso melhora significativamente a capacidade do sistema de arquivos em comparação com JFFS2, pois as mudanças nos arquivos na memória *Flash* não ocorrem imediatamente, pois eles são armazenados em cachê e colocados na memória *Flash* somente quando é absolutamente necessário;
- tolerância para *unclean reboots* – O UBIFS é um sistema de arquivos com uma tabela de paginação, portanto tolera falhas súbitas e reinicializações inesperadas (*unclean reboots*); ele apenas recupera a tabela de paginação da última inicialização do sistema, sendo a montagem neste caso um pouco mais lenta que o normal devido à necessidade de recuperação da tabela, mas o UBIFS não precisa fazer a varredura de toda memória, de modo que mesmo assim leva frações de segundo para montar;
- rápida Entrada/Saída (E/S) - Mesmo com o *write-back* desabilitado, o UBIFS mostra bom desempenho, próximo ao do JFFS2. É extremamente difícil competir com JFFS2 em termos de E/S síncrona, porque o JFFS2 não mantém estruturas de indexação de dados na memória *Flash*. Porém, o UBIFS ainda é rápido por causa da maneira que ele usa a tabela de paginação: não se move os dados fisicamente de um lugar para outro, mas em vez disso, ele só adiciona informação correspondente no índice do sistema de arquivo e escolhe um setor vazio para colocar uma nova tabela (ou seja, a tabela muda constantemente de posição);
- compressão *on-the-fly* - os dados são armazenados em formato compactado na memória *Flash*, o que torna possível colocar ainda mais dados na memória *Flash*, o que é muito semelhante ao que tem o JFFS2. O UBIFS também permite ligar ou desligar a compressão para cada *inode*. Por exemplo, pode-se desativar a compressão e permitir a compressão somente para determinados arquivos, ou

pode-se ter a compressão por padrão, mas desativá-la para dados que não podem ser comprimidos como arquivos multimídia. Atualmente o UBIFS oferece suporte apenas para compressores zlib e LZO;

- *recoverability* – O UBIFS pode ser totalmente recuperado se for corrompida a indexação dos arquivos, pois cada pedaço de informação em UBIFS tem um cabeçalho e é possível reconstruir integralmente o índice de sistema de arquivos verificando toda a memória *Flash*, ou seja, imagine que se perca a tabela de paginação do seu sistema de arquivos FAT, para o FATFS isto seria fatal, mas se acontecer a mesma coisa com o UBIFS, ainda se pode voltar a construí-la;
- integridade – O UBIFS (assim como UBI) verifica tudo o que escreve na memória *Flash* para garantir a integridade dos dados, de forma que o UBIFS não deixa dados corrompidos despercebidos.

Assim, vê-se que o UBIFS é o sistema de arquivos mais adequado para o projeto, por ser o mais novo sistema de arquivos para memórias *Flash*, que é a memória do projeto, e por possuir muitas vantagens em relação a outro famoso sistema de arquivos, o JFFS2. Caso se decidisse ainda assim utilizar o JFFS2 teria-se que disponibilizar 88MB de RAM somente para montar os 2GB de memória *Flash*, ou seja, claramente inviável já que o dispositivo possui 32 MB para todo o sistema, enquanto que utilizando-se o UBIFS para montar uma memória *Flash* de 2GB de capacidade necessita-se somente de 2MB de RAM.

3.13 Interface Web

Para facilitar a utilização do servidor pessoal e tornar a interface mais amigável, foi criada uma interface web que roda em qualquer navegador disponível no mercado.

A implementação dessa interface foi dividida em duas partes: a primeira é a aparência que a página terá para o usuário (*Front-End*) com toda a interface necessária para controlar as funcionalidades do servidor, onde será usada a linguagem HTML em conjunto com CSS, JavaScript e AJAX; a segunda é integrar a parte anterior com os comandos do servidor (*Back-End*) fazendo com que os comandos dados pela interface sejam realizados pelo servidor.

3.13.1 *Front-End*

Das duas partes essa é a visivel pelo usuário, ou seja, é propriamente a interface com que o usuário vai lidar para se comunicar e controlar o Servidor Pessoal. Através de botões, *links* e caixas de textos inseridos na interface o usuário poderá transferir arquivos do servidor para outros computadores, tocar música guardada no servidor ou uma estação de rádio da internet, controlar os LEDs, configurar o servidor e organizar os seus arquivos. Para isso foram usadas as linguagens HTML, CSS, JavaScript e AJAX que serão explicadas a seguir.

3.13.1.1 HTML

HTML (*HyperText Markup Language*) é uma linguagem utilizada para criar páginas na web, sendo que os arquivos em HTML são interpretados por navegadores, como o Internet Explorer, Firefox, entre outros. Essa tecnologia é fruto da união dos padrões *HyTime* e SGML (*Standard Generalized Markup Language*).

HyTime é um padrão para a representação estruturada de hipermídia e conteúdo baseado em tempo. Um documento é visto como um conjunto de eventos concorrentes dependentes de tempo (como áudio, vídeo, etc.), conectados por hiper-ligações. O padrão é independente de outros padrões de processamento de texto em geral [48].

SGML é um padrão de formatação de textos. Não foi desenvolvido para hipertexto, mas tornou-se conveniente para transformar documentos em hiper-objetos e para descrever as ligações [48].

As primeiras versões do HTML foram definidas com regras sintáticas flexíveis, o que ajudou aqueles sem familiaridade com a publicação na web. Atualmente a sintaxe do HTML se tornou mais rígida, permitindo códigos mais precisos. Com o passar dos anos a utilização da linguagem HTML e suas ferramentas aumentou muito, sendo usada na maior parte dos sites na internet, tornando necessário tornar a sintaxe

cada vez mais rígida. Apesar disso, os navegadores de hoje ainda conseguem interpretar páginas web que já não tem um código HTML válido para se manter a compatibilidade com sites que ainda usam as codificações anteriores.

Todo documento HTML apresenta etiquetas, elementos entre parênteses angulares (*chevron*) (< e >); esses elementos são os comandos da linguagem, que em sua maioria tem etiquetas de abertura e de fechamento, como por exemplo: <BODY> </BODY>.

Isso é necessário para se definir a formatação de uma porção do documento, e assim marcar onde começa e termina o texto com essa formatação específica. Alguns elementos são chamados "vazios", pois não marcam uma região de texto, apenas inserem algum elemento no documento, como por exemplo: <IMG SRC="imagem">

De uma maneira geral o HTML é um poderoso recurso, sendo uma linguagem de marcação muito simples e acessível voltada para a produção e compartilhamento de documentos e imagens [48].

3.13.1.2 CSS

CSS (*Cascading Style Sheets*) é uma linguagem utilizada em conjunto com HTML e XML para definir a apresentação e o estilo de páginas na web. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

Ao invés de colocar a formatação da página web dentro do documento HTML, o desenvolvedor cria um *link* para um documento que contém os estilos que serão utilizados. Então quando for necessário alterar a aparência da página basta modificar apenas um arquivo [49].

O suporte do CSS varia de navegador para navegador. O Internet Explorer 6, por exemplo, tem suporte total a CSS1 (primeira versão) e praticamente nulo a CSS2

(segunda versão). Já navegadores mais modernos como Opera, Internet Explorer 8 e Mozilla Firefox tem suporte maior, inclusive até a CSS3 [49].

Um documento em CSS que define o estilo de uma página consiste de uma lista de elementos com opções de formatação. Cada elemento é formado de uma ou mais opções de formatação de cada aspecto da página web em questão. Cada opção em si é uma propriedade, dois pontos (:), um valor, então um ponto e vírgula (;). Por exemplo, em:

```
"h1{
color: #FF00FF;
}"
```

onde se tem a propriedade "`color`" e o valor "`#FF00FF`".

3.13.1.3 JavaScript

JavaScript é uma linguagem de programação criada por Brendan Eich para a Netscape em 1995, que a princípio se chamava LiveScript. Após o sucesso inicial desta linguagem, a Netscape recebeu uma colaboração considerável da Sun, mudando o nome da linguagem para JavaScript, porém as semelhanças entre a linguagem da Sun, denominada Java [50].

Essa linguagem foi criada para a validação de formulários no lado cliente (navegador) e para melhorar a interação dos usuários com a página web, sendo desenvolvida como uma linguagem de *script*. O JavaScript tem sintaxe semelhante à do Java, mas é totalmente diferente no conceito e no uso.

As principais características do JavaScript são: é uma linguagem interpretada e não compilada; permite a criação de variáveis que podem mudar de tipo durante sua execução, portanto são declaradas sem tipo; e oferece bom suporte de ferramentas para padronizar suas listagens. Sua união com o CSS é conhecida como DHTML.

Usando o Javascript, é possível modificar dinamicamente os estilos dos elementos da página em HTML.

Além do uso em páginas HTML dinâmicas, o JavaScript é também usado na construção do navegador Mozilla, o qual oferece todo um conjunto de ferramentas para auxiliar a criação de sistemas de interface GUI (*Graphical User Interface*), que incluem um interpretador de JavaScript, um comunicador de JavaScript com C++ entre outros.

3.13.1.4 AJAX

AJAX (*Asynchronous Javascript And XML*) é uma técnica que utiliza em conjunto das linguagens de programação de páginas web como JavaScript e XML, para torná-las mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações. Foi inicialmente desenvolvida por Jessé James Garret e mais tarde por diversas associações [51].

Os navegadores o implementam desde o ano 2000, porém sua popularização só ocorreu nos últimos anos devido as muitas melhorias feitas na web, o que tem estimulado a construção de aplicações web mais dinâmicas e criativas. AJAX não é uma única tecnologia, mas um conjunto de tecnologias conhecidas trabalhando em conjunto, oferecendo novas funcionalidades.

As principais características do AJAX são: exposição e interação dinâmica; intercâmbio e manipulação de dados usando XML e XSLT; recuperação assíncrona de dados; e utilizando JavaScript fazendo a junção entre os elementos.

O modelo clássico de aplicação web trabalha da seguinte maneira: a maioria das ações do usuário na interface dispara uma solicitação HTTP para o servidor web. O servidor processa a solicitação, recuperando dados, realizando cálculos, conversando com vários sistemas, e então retorna uma página HTML para o cliente [51].

Com a popularização de sistemas que funcionam inteiramente na web e também com o aumento do tamanho das páginas web e suas aplicações, o tempo da espera pelo envio da página inteira se tornou muito mais evidente para o usuário. A principal vantagem das aplicações que utilizam AJAX é que os dados trafegados pela rede são reduzidos e o usuário não precisa aguardar a página ser recarregada totalmente a cada interação com o servidor.

Apesar de não possuir nada inovador em sua essência, o uso de AJAX revolucionou a web inteira trazendo a tona muitos conceitos importantes para o desenvolvimento web [51].

3.13.2 *Back-End*

O *Back-End* é a parte responsável por fazer a parte lógica da interface, ou seja, a ligação entre os comandos e/ou dados do usuário pela interface web e os programas propriamente ditos do servidor. É essa parte que vai verificar qual programa será chamado para realizar as tarefas requisitadas. Neste trabalho, utilizou-se um método denominado CGI e a linguagem *Shell* para esta função.

3.13.2.1 CGI

CGI, ou *Common Gateway Interface* é um método que permite a intermediação entre o servidor web e os outros programas executados no sistema (Figura 22) [52]. Possibilita a geração de páginas dinâmicas pela passagem dos parâmetros necessários para que o programa, alojado no servidor web, rode [53].

Denominam-se *scripts* CGI os programas que interpretam esses parâmetros e geram a página [53].

Figura 22 - Funcionamento do CGI [52]

A utilização do *script* pode acontecer nos seguintes modos [54]:

1. como um *link* para um programa que gerará uma página HTML;
2. para mostrar a data em que o arquivo HTML foi referenciado pela última vez, isso pode ser usado para a contagem de visitas;
3. na opção ACTION da tag <form> de um arquivo HTML. Nesse caso, essa opção contém o local em que está o *script* a ser executado e os dados nos campos preenchidos pelo usuário serão repassados para o servidor;
4. o *script* do cliente pode referenciar diretamente o *script* do servidor.

Scripts do cliente podem ser em JavaScript ou em VB, por exemplo, já os do servidor podem ser qualquer linguagem que produza código interpretável ou executável desde que o servidor web esteja configurado para suportá-lo, é o caso, por exemplo, das linguagens C, PERL e *Shell*.

Para o acesso aos dados, o CGI possui duas formas: por meio da sua entrada padrão, nesse caso deve ser passada o comprimento do conteúdo (variável CONTENT_LENGTH) para que o programa saiba como ler os dados; ou por meio da variável de ambiente QUERY_STRING que possui um comprimento limitado.

O formato da saída gerada pelo programa CGI é composto de [52]:

- cabeçalho de saída:
  - *content-type*: refere-se ao tipo de arquivo que está sendo enviado (Tabela 7);

Tabela 7 - Formatos comuns [52]

Formato	Content-type
HTML	text/html
text	text/plain
.gif	image/gif
.jpeg	image/jpeg
.mpeg	video/mpeg

  - *location*: permite redirecionar o usuário para outra página, se necessário;
  - *status*: é um código de *status* ao servidor e ao cliente. Por exemplo: "*status: 204 No Response*", nesse caso o programa CGI não possui saída e o documento que estava na tela deve continuar a ser exibido.

Ao final, deve-se enviar uma linha em branco como indicador de final de cabeçalho.

- dados de saída:
  - é o que realmente vai ser enviado ao cliente, este deverá estar relacionado com o *content-type* especificado no cabeçalho.

Quando esse cabeçalho é gerado pelo programa *script* do servidor, o servidor web proverá o resto dos cabeçalhos HTTP. Por exemplo, um programa *script* pode ser escrito para gerar um arquivo HTML padrão capaz de ser interpretado diretamente pelo navegador do usuário.

Para o desenvolvimento dos *scripts* CGI, utilizou-se a linguagem *Shell*, pela vantagem de ser compatível com praticamente qualquer plataforma Linux.

3.13.2.2 Ash

Ash (*Almquist Shell*) é um dos tipos de interpretadores de comandos usado em algumas distribuições Linux. As linguagens *Shell* do Unix podem ser divididas em 4 categorias: *Bourne-Like*, *C-Shell like*, *nontraditional* e *historical*. Neste projeto utilizou-se uma das linguagens do tipo *Bourne-Like* chamada Ash. A linguagem Ash também é compatível com a *Shell*, que também é do tipo *Bourne Shell*, devido ao seu criador Steve Bourne. [55].

As derivações do Ash são comumente utilizadas como o *Shell* padrão em sistemas como FreeBSD, NetBSD, DragonFly BSD e Minix. Também é bastante usado em sistemas que tenham Linux embarcado. Seu código foi incorporado no executável genérico do *BusyBox*, geralmente empregado nessa área. A versão do Ash do Debian é conhecida como dash (*Debian Almquist Shell*). No nosso caso, também foi usado o *BusyBox* e, consequentemente, o Ash. [56]

Algumas distribuições Linux também usam uma variação do Ash como *Shell* padrão, embora o Bash (*Bourne Again Shell*) seja mais popular. No Ubuntu há um redirecionamento para o dash *Shell* para uma execução mais rápida do *script*, porém essa distribuição ainda mantém o Bash como *Shell* padrão. [56]

O seguinte trecho foi extraído das informações sobre o pacote Ash do SlackWare: [56]

> ash (ash *Shell* de Kenneth Almquist)
>
> Um *shell* leve (92K) compatível com o *Bourne shell*. Muito bom para máquinas com pouca memória, mas não provê todos os extras de *shells* como bash, tcsh e zsh. Roda a maioria dos *scripts Shell* que são compatíveis com o *Bourne Shell*. Note que sob o Linux, a maioria dos *scripts* parecem que usam pelo menos algumas sintaxes específicas. Os *scripts* das configurações do Slackware são uma exceção, uma vez que o ash é o *Shell* usado em instalações de discos.(...)

4 Implementação, Testes e Validação

4.1 Fabricação das placas de circuito impresso

Com a criação do *layout* da placa, a etapa seguitne foi a fabricação da mesma para se implementar o servidor pessoal. Então se procurou uma empresa brasileira que fizesse placas de circuito impresso de duas camadas e que atendessem os requisitos estabelecidos na seção de riscos de projeto: placas de circuito impresso.

A empresa que atendia os requisitos (com alguma margem no comprimento das trilhas para que a placa não viesse com problemas de fabricação dessa natureza) foi a Lauquen, que cobrou R$450,00 para fabricar as cinco placas. O processo de requisição foi o seguinte, primeiramente se requisitou um orçamento a partir das especificações da seção de riscos de projeto: placa de circuito impresso, depois de se verificar que a empresa atendia os requisitos e que tinha um preço dentro das expectativas, se enviou um arquivo gerber com o "desenho" do *layout* da placa e se pediu o orçamento do mesmo.

Novamente se verificou que o preço estava de acordo com o apresentado anteriormente então se requisitou a fabricação das cinco placas de circuito impresso. A fabricação e a entrega à domicilio demoraram 12 dias. As placas já vieram testadas eletricamente.

4.2 Etapas de Montagem

Com as placas já fabricadas, a próxima etapa no desenvolvimento dos dispositivos foi a soldagem dos componentes nas placas. Para que a soldagem fosse realizada de forma organizada, sistemática e que permita a realização dos testes nos dispositivos, criou-se uma tabela em que a soldagem foi dividida em seis etapas: alimentação, processador, memória RAM, memória NAND *Flash*, módulo Ethernet, e

conector de expansão. Com isso, ao fim de cada etapa é possivel testar os componentes soldados a fim de se verificar se eles estão funcionando corretamente.

A tabela com as fases e os respectivos componentes para serem soldados estão no apêndice C.

4.3 Implementação e Testes do Dispositivo

A soldagem, por se tratar de componentes SMD ou com encapsulamento com pinos pequenos, requer uma aparelhagem mais profissional, portanto o grupo pediu ajuda ao Sr. Jair Pereira de Souza do LME para nos assistir na soldagem dos componentes.

Durante a soldagem, verificou-se que a escolha de pinos salientes facilitou muito a soldagem dos componentes mesmo eles sendo pequenos, mesmo devendo-se soldar e dessoldar algumas memórias NAND *Flash* para testar se elas funcionariam corretamente no projeto. Outro fato para se levar em conta é que o tempo de soldagem dos componentes foi de aproximadamente 25 horas.

Primeiro teste realizado – Alimentação: colocou-se 5V na entrada e verificou-se que na saída dos reguladores de tensão tinham 3,3V e 1,8V.

Segundo teste realizado - Processador: conectou-se o dispositivo a um computador através de um cabo USB e verificou-se que o programa SAM-BA identificou o processador.

Terceiro teste realizado – Memória RAM: conectou-se o dispositivo a um computador através de um cabo serial e verificou-se que o programa SAM-BA identificou a memória RAM e conseguiu ler e escrever em todos os bytes de sua memória.

Quarto teste realizado – Memória NAND: conectou-se o dispositivo a um computador através de um cabo serial e verificou-se que o programa SAM-BA identificou a memória NAND *Flash* e conseguiu ler e escrever em todos os bytes de sua memória.

Quinto teste realizado – Módulo Ethernet: o teste deste módulo não é uma tarefa que requer somente *hardware*, pois para verificar que o mesmo está funcionando é necessário estabelecer a comunicação entre o dispositivo e um computador (ou outro dispositivo eletrônico que se comunique por TCP/IP), e como o protocolo é dependente de *software* então é necessário programá-lo. Essa programação será discutida na seção: Programação do Dispositivo através do SAM-BA.

Sexto teste realizado: conectou-se o dispositivo a um computador através de um cabo serial e verificou-se que o programa SAM-BA identificou o dispositivo e através de um programa criado, que será discutido mais tarde conseguiu acender e apagar os LEDs da placa, que estão conectados ao GPIO do processador.

4.4 Validação do Dispositivo

Com base nos resultados apresentados, é possível perceber que todas as premissas e objetivos estabelecidos para o projeto foram atendidas.

- consumo de Energia Elétrica: Com base nas medições realizadas, a placa consome aproximadamente 1W de potência quando o processador funciona a 100%, é feita uma conexão *Ethernet* de 100Mbps e não há nenhum acessório conectado, ou seja, bem abaixo do objetivo máximo proposto de 40W. O consumo de 1W durante 7 dias por semana e 30 dias por mês corresponde a aproximadamente R$ 0,36 em energia, em São Paulo, no ano do projeto (2010). Assim, vê-se que o consumo é, de fato, muito baixo;
- expansibilidade: Desde o momento em que se conseguiu executar o sistema operacional Linux na plataforma desenvolvida neste projeto, procurou-se garantir a expansibilidade do projeto, seja por meio do barramento a isso dedicado ou pela porta USB. Como era desejado, o sistema reconhece qualquer dispositivo USB desde que haja um driver aberto para Linux, ou seja, o sistema consegue ser tão flexível quanto uma distribuição Linux para *desktop*;

- disponibilidade: Embora os testes finais ainda não tenham sido realizados, pode-se desde já constatar que a plataforma criada é muito estável e, mesmo ligada durante horas, não apresenta qualquer sintoma de instabilidade ou aquecimento;
- baixo custo: Conseguiu-se produzir cada protótipo a um preço inferior a R$300,00, ou seja, considerando-se que o primeiro lote é sempre mais caro e que, ainda por cima, a produção em pequena escala costuma ser muito mais cara que a produção em série de produtos eletrônicos, pode-se ver facilmente que o objetivo de baixo custo foi atingido com sucesso;
- segurança: Este ponto depende fundamentalmente dos aplicativos que executaremos no sistema. Assim, para o conjunto de aplicativos fornecidos com o sistema, pôde-se validar a segurança do Servidor Pessoal, pois os aplicativos escolhidos atendem integralmente os requisitos de segurança especificados.

Desta forma, vê-se que o projeto atende ou excede todos os requisitos especificados em seu início.

4.5 Validação da Interface Web

Uma vez desenvolvida a interface *web* foi possível validá-la com base nos requisitos de *marketing* e de engenharia estabelecidos no início do projeto.

Este processo, juntamente com a metodologia em espiral adotada para a parte de *software*, permitiu uma série de iterações sobre o *software* embarcado culminando com uma versão que não somente atendesse às necessidades apresentadas, mas também fosse simples de usar e elegante.

A tecnologia AJAX, por exemplo, foi fundamental para permitir uma boa interação com o usuário e um dinamismo muito interessante, por exemplo, na parte de *upload* e *download* de arquivos para / do Servidor Pessoal.

Ainda na fase de validação, verificou-se que os algoritmos de criptografia utilizados (RSA 1024 bits e RC4_MD5 de 128 bits) permitem conciliar bom desempenho (*upload* e *download* a aproximadamente 1MB/s) a um excelente nível de segurança.

Finalmente, para garantir que os atuais navegadores realmente consigam interpretar os *scripts* desenvolvidos, decidiu-se submetê-los a um site, o W3C, que os valide conforme alguns padrões que os principais navegadores utilizam. O W3C, *World Wide Web Consortium*, é uma comunidade internacional que desenvolve padrões para assegurar o crescimento a longo prazo da *web*. [57]

4.5.1 W3C Validator

O W3C desenvolve especificações técnicas e orientações por meio de um processo destinado a maximizar o consenso sobre o conteúdo de um relatório técnico; garantir uma qualidade técnica e editorial alta; promover consistência entre especificações; e ganhar a aprovação da comunidade em geral. O W3C provê um número de visões de suas especificações, incluindo [58]:

- por tópico tecnológico (como todas as especificações relacionadas ao HTML);
- por estado (com publicações recentes no topo e, em seguida, todas as especificações agrupadas por status, a partir de recomendações aos projetos);
- por data (mais recentes no início);
- por grupo (que está trabalhando na especificação).

A seguir serão mencionadas algumas atividades do W3C:

- projetos e aplicações web - que envolvem os padrões de construção e renderização de páginas web, incluindo HTML5, CSS, SVG, AJAX e outras tecnologias para aplicações web ("WebApps");
- arquitetura web - que enfoca nas tecnologias e princípios das fundações que sustentam a web, incluindo URIs e HTTP;

- desenvolvimento das tecnologias XML - que incluem XML, XQuery, XML Schema, XSLT, XSL-FO, Efficient XML Interchange (EXI), e outros padrões relacionados;
- rede de serviços - que se refere à concepção baseada em mensagens frequentemente encontradas na internet e em *software* empresarial. A rede de serviços é baseada em tecnologias como HTTP, XML, SOAP, WSDL, SPARQL e outros;
- rede de dispositivos - W3C tem seu foco em tecnologias que permitem o acesso a web de qualquer lugar, a qualquer hora, usando qualquer dispositivo. Isso inclui o acesso a web de celulares e outros dispositivos móveis bem como o uso de tecnologias da web em eletrônicos de consumo, impressoras, televisões interativas, e mesmo automóveis.

Para o projeto, o grupo utilizou as ferramentas HTML Validator e CSS Validator para testar as páginas web.

4.6 Programação do Dispositivo através do SAM-BA

Para começar a testar o funcionamento do processador é necessário inicializar alguns módulos obrigatórios, portanto, foi criado um programa que inicializa o PMC, GPIO, entre outros módulos e verifica que o programa está rodando por meio de uma função que permite acender e apagar os LEDs da placa a partir dos botões da mesma. O programa utilizado é apresentado no apêndice D. Vale lembrar que a Atmel fornece parte do código utilizado, tendo o grupo adaptado o código para a placa construída e para a funcionalidade requisitada.

Esse programa foi colocado por meio do SAM-BA primeiramente na memória *Flash* interna e depois na NAND externa para verificar o funcionamento de ambas e para confirmar que a inicialização da placa era possível pelas duas memórias.

Como dito anteriormente, o teste do módulo Ethernet não pode ser feito sem o protocolo TCP/IP, então se utilizou o AT91Bootstrap e o U-Boot para configurar o módulo Ethernet e instalar o protocolo TCP/IP na placa. Através do cabo serial e

utilizando-se o *Shell* da máquina virtual de desenvolvimento para visualizar o *Shell* do dispositivo, utilizou-se o comando ping 'endereço do computador' para se verificar a comunicação do computador e do dispositivo pelo cabo Ethernet. Para isso foram utilizados os comandos a seguir:

- setenv ipaddr 192.168.0.200;
- setenv ethaddr 08:00:27:e0:03:46;
- setenv serverip 192.168.0.151;
- ping 192.168.0.151.

4.7 FMEA

Para que o usuário consiga usar o servidor pessoal sem problemas, é necessário prever algumas possíveis falhas e fazer indicações de que atitudes tomar na ocorrência deles. Dessa forma, a Tabela 8 apresenta as possíveis falhas levantadas, as respectivas ações já tomadas para minimizá-las e as indicações das ações a tomar, caso ela ocorra. [59]

Os valores quantitativos foram estimados da seguinte forma:

- probabilidade de ocorrência: praticamente impossível corresponde ao número 1 e praticamente certo ao número 10;
- severidade: o número 1 corresponde a um cenário onde o usuário não sofre com o problema. Já o número 10 representa a impossibilidade total de utilização do dispositivo ou perda de um recurso fundamental;
- probabilidade de detecção: o número 1 corresponde a um caso praticamente indetectável e o número 10 representa a detecção certa.

Tabela 8: FMEA (continua)

Possíveis Falhas	Causas	Probabilidade de ocorrência (*)	Efeitos	Severidade (**)	Detecção	Probabilidade de detecção (***)	Ações preventivas tomadas	Ações corretivas a tomar
Interferência eletromagnética	Trilhas muito próximas	1	Pode alterar o funcionamento ou danificar o aparelho	6	Falha do Sistema	10	Cuidado no roteamento do layout	Colocar o dispositivo dentro de uma caixa metálica
Problemas com a internet	Indisponibilidade do servidor ou problemas na rede do usuário	5	Internet lenta ou sem conexão	6	Led do módulo PHY	3	Escolha de um bom servidor web. No caso de problemas na rede do usuário, falar com o administrador da rede.	Chamar o técnico
Falha de segurança	Perda da chave/senha, esquecer o servidor logado	1	Perda da privacidade dos dados e outras pessoas poderem alterar o arquivo	10	Os arquivos serem publicados ou alguém comunicar a falha	1	Escolha de um bom algoritmo de criptografia	Trocar a chave/senha
Não reconhecimento de um novo dispositivo	Ausência do driver, problemas na comunicação (barramento ou no próprio dispositivo)	3	Não reconhecimento desse dispositivo ou problemas na comunicação com o mesmo	3	Não reconhecimento desse dispositivo ou problemas na comunicação com o mesmo	10	Inclusão dos drivers responsáveis mais utilizados	Instalar o driver do novo dispositivo
Problemas na interface	Navegador que não consegue interpretar os *scripts*	4	Má renderização, não atendimento dos comandos do usuário	3	Má renderização, não atendimento dos comandos do usuário	2	Desenvolvimento de uma interface web de fácil utilização e *script* validado com base nos navegadores mais utilizados	Tentar com outro navegador (IE8, Firefox, Opera, Safari, Chrome)
Corte na alimentação	Falta de energia	5	Servidor não funciona	8	O servidor para de funcionar	10	Nenhuma	Esperar a energia voltar

Tabela 8: FMEA (continuação)

Possíveis Falhas	Causas	Probabilidade de ocorrência (*)	Efeitos	Severidade (**)	Detecção	Probabilidade de detecção (***)	Ações preventivas tomadas	Ações corretivas a tomar
Na volta do servidor depois do corte de energia	Interrupção do servidor de forma inadequada	5	Perda eventual de algum dado não salvo, mas não do sistema.	3	Interrupção da energia	1	O servidor volta a funcionar normalmente	Nada, ele volta a funcionar normalmente.
Problemas na transmissão/ recepção	Problemas na internet do usuário	4	Não possibilidade de transmissão/recepção ou realizada de forma lenta ou arquivo corrompido	5	Led da transmissão/ recepção	5	Escolha do módulo PHY, escolha de um bom servidor SSH e WEB.	Chamar o técnico
Problemas de SW (no Linux)	Possíveis bugs da versão do *kernel* do Linux utilizado.	3	Não funcionamento ou comprometimento de algumas funções	5	Não funcionamento ou comprometimento de algumas funções	3	Escolha de um bom SO, em sua última versão estável	Reinicializar
Problemas na configuração	Dispositivo configurado de forma errada	3	Pode ir desde funcionamento em desacordo com a preferência do usuário até o não funcionamento do dispositivo	6	Funcionamento em desacordo com a preferência do usuário até o não funcionamento do dispositivo	3	Interface web que permita uma fácil configuração e a validação desta interface, além de uma configuração padrão "vinda de fábrica"	Se o usuário conseguir, configurar corretamente via interface web, caso não, chamar um técnico
Problemas na memória SDRAM	Não reconhecimento da memória ou problemas na sua leitura	1	Não reconhecimento, o produto não funciona, problemas na leitura.	10	O Linux não é carregado	10	Testes a cada memória SDRAM soldada	Testes de memória como prevenção e detecção de memória que tende a falhar.

Tabela 8: FMEA (conclusão)

Possíveis Falhas	Causas	Probabilidade de ocorrência (*)	Efeitos	Severidade (**)	Detecção	Probabilidade de detecção (***)	Ações preventivas tomadas	Ações corretivas a tomar
Problemas na memória NAND	Não reconhecimento da memória ou problemas na leitura	1	Não reconhecimento ou problemas na leitura da inicialização, o produto não inicializa, depende do que não foi possível ler	10	O Linux não é carregado	10	Testes a cada memória NAND soldada.	Testes de memória como prevenção

(*) Probabilidade de ocorrência: probabilidade de a falha ocorrer [59].

(**) Severidade: gravidade do efeito da falha para o funcionamento do sistema [59].

(***) Probabilidade de detecção: probabilidade dos sistemas de qualidade existentes apontarem a falha antes de ela ocorrer [59].

5 Resultados

Após vários meses e etapas do projeto, o grupo conseguiu finalizar as atividades da etapa de *hardware*, que envolviam: desenvolver o esquemático e o *layout* da placa (apresentados nos apêndices A e B respectivamente), realizar a compra dos componentes necessários (a lista foi apresentada nas seções Compra dos Componentes Importados e Compra de Outras Peças), requisitar a fabricação e compra das placas, fazer a soldagem dos componentes nos cinco protótipos e realizar os testes necessários para a verificação do funcionamento dos mesmos. A seguir se apresenta algumas imagens dos componentes e das placas:

Figura 23 - Componentes Comprados

Figura 24 - Imagem dos cinco protótipos

Figura 25 - Vista superior da placa

Figura 26 - Vista inferior da placa

Na etapa de *software* o grupo desenvolveu um ambiente virtual para a criação e implementação das ferramentas para a compilação dos programas para as placas, criou a ferramenta de compilação cruzada, desenvolveu as três etapas de inicialização do dispositivo: AT91Bootstrap (mostrado no apêndice D), U-Boot e o *kernel* do Linux; compilou e instalou os aplicativos necessários para a implementação do servidor pessoal, e criou a interface web entre o Servidor Pessoal e o usuário. No apêndice E são mostradas as mensagens de inicialização do AT91Bootstrap, U-Boot e do *kernel* do Linux.

6 Discussão

Pode-se dizer que os resultados obtidos são sólidos e mostram que o projeto foi concluído com êxito.

Percebe-se também que a metodologia usada pelo grupo para a etapa de *hardware*, que é a *bottom-up*, foi bem escolhida e decisiva para o cumprimento de todas as etapas realizadas, pois com a definição do processador e do sistema operacional, os outros itens necessários foram sendo decididos "naturalmente" pelo grupo e levaram a uma rápida definição de esquemático, que por sua vez levou à conclusão do *layout* para fabricação.

Quanto à parte de *software*, pode-se concluir que ela foi realizada dentro do prazo estabelecido pelo grupo e que a escolha da metodologia em espiral também influenciou de forma positiva o desenvolvimento e implementação da parte de inicialização e escolha dos aplicativos. Só foi possível testar os programas desenvolvidos a partir do momento em que se tinha um dos protótipos funcionando, portanto, toda a etapa de *hardware* pode ser considerada um sucesso, pois permitiu que a etapa de *software* fosse executada tranquilamente.

O problema mais grave enfrentado pelo grupo foi um erro na compra da memória NAND *Flash*, pois a compra de um *pen drive* com uma memória compatível que pudesse ser dessoldada e ressoldada no protótipo atrasou o projeto em uma semana, o que felizmente pôde ser corrigido em outras etapas do projeto e com uma placa "completa", conseguiu-se seguir com o projeto sem maiores transtornos.

Outro problema que ocorreu foi no *layout* da placa, onde se esqueceu de se ligar um dos pinos do conector Ethernet na tensão de 2,5V. Esse é um erro fácil de corrigir, pois é só ligar um fio do 2,5V do módulo Ethernet ao conector, porém foi um erro difícil de se encontrar porque o Servidor Pessoal simplesmente não se conectava com o computador.

7 Conclusão

Com base em todo o trabalho executado ao longo deste ano e nos resultados apresentados neste relatório, pode-se dizer que foi de fato possível desenvolver um dispositivo que alie boa capacidade de processamento e baixo custo ao uso inteligente de recursos e baixo consumo.

O projeto conceitual inicial, sendo bastante sólido, permitiu uma transição suave e tranqüila à fase de detalhamento onde foram elaborados tanto a parte de *hardware* quanto a parte de *software*.

Além disso, em diversos momentos do projeto foram tomadas ações visando à mitigação de riscos, como a compra dos componentes, ainda em uma fase inicial do projeto para garantir sua disponibilidade, e o uso de ferramentas de *software* conhecidas e testadas. É possível apreciar agora a significância que tiveram tais ações, permitindo que o projeto se mantivesse no prazo, no orçamento e respeitando os objetivos e requisitos propostos. E, assim, com base nos resultados apresentados, vê-se que o projeto encontra-se concluído com sucesso.

Vale salientar que o sucesso do projeto é devido também a uma grande preocupação com a testabilidade do projeto como um todo. As metodologias estabelecidas para as partes de *hardware* (*bottom-up*) e de *software* (espiral) foram devidamente trabalhadas para incorporar pontos de teste e validação intermediários, facilitando e acelerando o desenvolvimento, ou seja, um DFT (*Design for Testability*).

Portanto se conclui que o trabalho de formatura atingiu todos os objetivos propostos (permitir o acesso a dados por meio da internet com alta disponibilidade, consumo de 1W, transmissão criptografada dos dados, senha para o acesso à interface do servidor, capacidade de expansão por meio dos conectores GPIO, USB e serial, tamanho reduzido, facilidade na configuração e reconhecimento de dispositivos), dentro do prazo estabelecido e dentro do orçamento proposto (menos de R$1500,00 para 5 protótipos).

8 Recomendações para Trabalhos Futuros

Algumas atividades que o grupo imagina que podem ser desenvolvidas em cima do Servidor Pessoal são:

- colocar um aplicativo que guarda imagens diretamente de uma *Webcam*, assim se poderia colocar câmeras na casa e guardar os vídeos diretamente no Servidor Pessoal;
- utilizar o servidor para acompanhar o consumo de luz ou de água numa residência, sendo que para isso é necessário criar um circuito elétrico de interface entre o servidor e o relógio de luz ou água;
- criar uma maneira de se comunicar diretamente com o servidor na rede local sem a necessidade de se utilizar um navegador, da mesma maneira que se compartilha informação entre computadores em uma rede do Windows®, diretamente pelo Windows Explorer® (Windows® e Internet Explorer® são marcas registradas da Microsoft Corporation).
- verificar se o projeto atende as normas internacionais como FCC e CE para produtos de uso doméstico e/ou em escritório.

Referências

1. WEISS, A. **Computing in the clouds**. [S.l.]: NetWorker, v. 11, 2007. 16-25 p.

2. NEWS, B. Phishing attack targets Hotmail, 05 outubro 2009. Disponivel em: <http://news.bbc.co.uk/2/hi/technology/8291268.stm>. Acesso em: 08 maio 2010.

3. BRESSER PEREIRA, L. C. J. M. M. A. A. P. **Economic reforms in new democracies**. [S.l.]: Cambridge University Press, 1993.

4. ENCICLOPÉDIA Britannica. **Merriam-Webster Dictionary:** bottom-up. Disponivel em: <http://www.britannica.com/bps/dictionary?query=bottom-up&header_go=>. Acesso em: 8 maio 2010.

5. XIE, F. . Y. H. . S. X. **Component-based hardware/software co-verification for building trustworthy embedded systems**. [S.l.]: The journal of systems and software, 2006. 643-654 p.

6. SERVICES, C. –. C. F. M. &. M. Selecting a development approach, março 2008. Disponivel em: <http://www.cms.hhs.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf>. Acesso em: 25 set. 2010.

7. JUPITERWEB, S. Disciplina PSI2553 – Projeto de Sistemas Integrados. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=PSI2553&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

8. CHAU, W. J. Programa da disciplina PSI2553 – Projeto de Sistemas Integrados. Disponivel em: <http://moodle.redealuno.usp.br/moodle/file.php/572/Recurso/programa.pdf>. Acesso em: 31 março 2010.

9. JUPITERWEB, S. Disciplina PSI2223 – Introdução à Eletrônica. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=PSI2223&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

10. JUPITERWEB, S. Disciplina PSI2324 – Eletrônica I. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=PSI2324&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

11. JUPITERWEB, S. Disciplina PCS2477 – Arquitetura e Organização de Computadores. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=PCS2477&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

12. JUPITERWEB, S. Disciplina PCS2304 – Projeto Lógico Digital. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=PCS2304&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

13. JUPITERWEB, S. Disciplina PCS2476 – Fundamentos de Redes de Computadores. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=PCS2476&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

14. JUPITERWEB, S. Disciplina MAC2166 – Introdução à Computação para Engenharia. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=MAC2166&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

15. JUPITERWEB, S. Disciplina PCS2478 – Tópicos de Programação. Disponivel em: <http://sistemas2.usp.br/jupiterweb/obterDisciplina?sgldis=PCS2478&codcur=3031&codhab=180>. Acesso em: 31 março 2010.

16. ARM. General Purpose MCUs. Disponivel em: <http://www.arm.com/markets/embedded/mcu.php>. Acesso em: 9 maio 2010.

17. EXTREMETECH. ARM Architecture. Disponivel em: <http://www.extremetech.com/article2/0,3973,633095,00.asp?kc=ETTH102099TX1K0100486>. Acesso em: 9 maio 2010.

18. ARM. Processor_Overview_Architecture. **ARM - The Architecture for the Digital World**, 2010. Disponivel em: <http://www.arm.com/images/roadmap/Processor_Overview_Architecture.jpg >. Acesso em: 26 set. 2010.

19. SKOROBOGATOV, S. **Low temperature data remanence in static RAM**. University of Cambridge, Computer Laboratory. [S.l.]. 2002.

20. COPELAND B. JACK, A. O. **Colossus:** The Secrets of Bletchley Park's Codebreaking Computers Oxford. [S.l.]: Oxford University Press, 2006.

21. REOCITIES. Memórias DRAM Tipos e Tecnologia. Disponivel em: <http://reocities.com/ResearchTriangle/4480/academic/academic-files/dram.html>. Acesso em: 17 junho 2010.

22. ATMEL. AT91 ARM Thumb Microcontrollers - AT91SAM9XE128, AT91SAM9XE256, AT91SAM9XE512 Preliminary. Disponivel em: <http://www.atmel.com/dyn/resources/prod_documents/6254s.pdf>. Acesso em: 19 junho 2010.

23. PRODUCTS, E. NAND vs. NOR flash technology. Disponivel em: <http://www2.electronicproducts.com/NAND_vs_NOR_flash_technology-article-FEBMSY1-feb2002-html.aspx>. Acesso em: 17 junho 2010.

24. ALTERA. **Application Note - NAND Flash Memory Interface with MAX II CPLDs**, Dezembro 2007. Disponivel em: <http://www.altera.com/literature/an/an500.pdf>. Acesso em: 21 junho 2010.

25. ATMEL. NAND Flash Support in AT91SAM9 Microcontrollers. Disponivel em: <http://www.atmel.com/dyn/resources/prod_documents/doc6255.pdf>. Acesso em: 19 junho 2010.

26. WILLIAM WONG, E. E. Network Design FAQs – Ethernet MAC and PHY. Disponivel em: <http://archive.electronicdesign.com/files/29/9177/9177_01.pdf>. Acesso em: 21 junho 2010.

27. FLEXPWR™, S. –. S. B. D. Datasheet - 15kV ESD Protected MII/RMII 10/100 ETHERNET Transceiver with HP Auto-MDIX and flexPWR™ Technology in a small footprint, maio 2007. Disponivel em: <http://www.ethernut.de/en/hardware/enut5/datasheets/lan8700i.pdf>. Acesso em: 21 junho 2010.

28. ENGINERING, D. A beginner's guide to switchingregulators. Disponivel em: <http://www.dimensionengineering.com/switchingregulators.htm>. Acesso em: 22 junho 2010.

29. MAXIM. DC-DC Converter Tutorial. Disponivel em: <http://www.maxim-ic.com/app-notes/index.mvp/id/2031>. Acesso em: 22 junho 2010.

30. GARRELS, M. Introduction to Linux - A hands on guide, 2008. Disponivel em: <http://tldp.org/LDP/intro-linux/intro-linux.pdf>. Acesso em: 26 set. 2010.

31. SYSTEMS, B. Embedded Technology: Embedded Software. Disponivel em: <http://www.bluewatersys.com/core-technologies/embedded-software>. Acesso em: 26 set. 2010.

32. MICROSOFT. Windows Embedded CE Overview, 2010. Disponivel em: <http://www.microsoft.com/windowsembedded/en-us/products/windowsce/default.mspx>. Acesso em: 26 set. 2010.

33. LAUREANO, M. A. P. . M. C. A. "Virtualização: Conceitos e Aplicações em Segurança" Mini-curso do VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2008. Disponivel em: <http://www.ppgia.pucpr.br/~maziero/lib/exe/fetch.php/research:2008-sbseg-mc.pdf>. Acesso em: 26 set. 2010.

34. GNUARM. GNUARM, 2010. Disponivel em: <http://www.gnuarm.com/>. Acesso em: 26 set. 2010.

35. YAGHMOUR, K. M. J. B.-Y. G. **Building Embedded Linux Systems**. 2nd. ed. [S.l.]: O'Reilly Media, 2008.

36. VMLINUX. How to configure, build, and install GCC as a cross-compiler, 2010. Disponivel em: <http://vmlinux.org/crash/mirror/www.objsw.com/CrossGCC/FAQ-4.html>. Acesso em: 26 set. 2010.

37. ATMEL. AT91 ISPSAM-BA User Guide. Disponivel em: <http://www.atmel.com/dyn/resources/prod_documents/6421B.pdf>. Acesso em: 23 Setembro 2010.

38. ATMEL. Sam-ba_2.10_release_note.txt. Disponivel em: <http://www.atmel.com/dyn/resources/prod_documents/sam-ba_2.10_release_note.txt>. Acesso em: 23 Setembro 2010.

39. BOYER, F. AT91SAM Boot Strategies Application Deployment, 2010. Disponivel em: <http://www.at91.com/linux4sam/pub/Linux4SAM/GettingStarted/SAM9_Boot_Strategies.ppt>. Acesso em: 26 set. 2010.

40. **Linux & Open Source related information for AT91 Smart ARM Microcontrollers**. Disponivel em: <http://www.at91.com/linux4sam/bin/view/Linux4SAM/WebHome>. Acesso em: 20 Setembro 2010.

41. BRUNE, C. Introduction to Das U-Boot, the universal open source bootloader, 2004. Disponivel em: <http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Introduction-to-Das-UBoot-the-universal-open-source-bootloader/>. Acesso em: 26 set. 2010.

42. BASIC Command Set. **DENX software engineering**. Disponivel em: <http://www.denx.de/wiki/view/U-Bootdoc/BasicCommandSet>. Acesso em: 25 Setembro 2010.

43. ALECRIM, E. O kernel do Linux, 2007. Disponivel em: <http://www.infowester.com/linuxkernel.php>. Acesso em: 26 set. 2010.

44. BUILDING a root filesystem. **Linux Online**. Disponivel em: <http://www.linux.org/docs/ldp/howto/Bootdisk-HOWTO/buildroot.html>. Acesso em: 23 Setembro 2010.

45. FILESYSTEM Hierarchy Standard. Disponivel em: <http://www.pathname.com/fhs/pub/fhs-2.3.pdf>. Acesso em: 23 Setembro 2010.

46. BITBAKE User Manual. **Background and Goals**. Disponivel em: <http://bitbake.berlios.de/manual/ch01s02.html>. Acesso em: 23 Setembro 2010.

47. LINUX-MTD. UBIFS - UBI File-System. Disponivel em: <http://www.linux-mtd.infradead.org/doc/ubifs.html>. Acesso em: 10 Novembro 2010.

48. W3. HTML 4.0 Specification. Disponivel em: <http://www.w3.org/TR/1998/REC-html40-19980424>. Acesso em: 6 Novembro 2010.

49. W3. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Disponivel em: <http://www.w3.org/TR/CSS21/cover.html>. Acesso em: 6 Novembro 2010.

50. COMPUTERWORLD. The A-Z of Programming Languages: JavaScript. Disponivel em: <http://www.computerworld.com.au/article/255293/a-z_programming_languages_javascript/>. Acesso em: 6 Novembro 2010.

51. ADAPTIVEPATH. Ajax: A New Approach to Web Applications. Disponivel em: <http://www.adaptivepath.com/ideas/essays/archives/000385.php>. Acesso em: 10 Novembro 2010.

52. OTSUKA, J. L. O que é CGI. Disponivel em: <http://penta.ufrgs.br/edu/forms/cgi.html>. Acesso em: 2 Novembro 2010.

53. W3. CGI: Common Gateway Interface. Disponivel em: <http://www.w3.org/CGI/>. Acesso em: 4 Novembro 2010.

54. COMPTECH. Common Gateway Interface. Disponivel em: <http://www.comptechdoc.org/independent/web/cgi/cgimanual/cgi.html>. Acesso em: 28 Outubro 2010.

55. SOFTPANORAMA. Introduction to the Unix shell history, 2010. Disponivel em: <http://www.softpanorama.org/People/Shell_giants/introduction.shtml>. Acesso em: abr. Novembro.

56. ULM, I. Ash (Almquist Shell) Variants. Disponivel em: <http://www.in-ulm.de/~mascheck/various/ash/>. Acesso em: 6 Novembro 2010.

57. W3. W3C. Disponivel em: <http://www.w3.org>. Acesso em: 10 novembro 2010.

58. W3. W3C Standards. Disponivel em: <http://www.w3.org/standards/>. Acesso em: 10 Novembro 2010.

59. KOFUJI, S. T. DFMEA - ANÁLISE DE MODOS DE FALHA E EFEITOS. Disponivel em: <http://moodle.stoa.usp.br/file.php/760/FMEA-23092010.pdf>. Acesso em: 11 Novembro 2010.

Bibliografia

- ATMEL, **"ARM-based Solutions: AT91SAM ARM-based MCUs and eMPUs"**, encontrado em: <http://www.atmel.com/products/at91/default.asp?family_id=605>, acessado em 31/03/2010.

- MICRON, **"Micron Technology"**, encontrado em: <http://www.micron.com>, acessado em 31/03/2010.

- MICREL, **"Micrel Inovation Throught Technology"**, encontrado em: <http://www.micrel.com>, acessado em 31/03/2010.

- ARROW, **"Arrow Electronics"**, encontrado em: <http://www.arrow.com>, acessado em 31/03/2010.

- DIGIKEY, **"Digikey Corp."**, encontrado em: <http://www.digikey.com>, acessado em 31/03/2010.

- MOUSER, **"Mouser Electronics"**, encontrado em: <http://www.mouser.com>, acessado em 31/03/2010.

- JAMECO, **"Jameco Electronics"**, encontrado em: <http://www.jameco.com>, acessado em 31/03/2010.

- SLOSS, A, SYMES, D, WRIGHT, C, **"ARM System Developer's Guide: Designing and Optimizing System Software"**, Morgan Kaffmann, Elsevier, 2004.

- FURBER, S, **"ARM System on Chip Architecture"**, Addison Wesly Pubblish, 2000.

- BARR, M, MASSA, A, **"Programming Embedded Systems, Second Edition: With C and GNU Development Tools"**, O'Reilly Media, Inc., 2006.

- FORD, R, COULSTON, C, **"Design for Electrical and Computer Engineering: Theory, Concepts and Practice"**, McGRAW-HILL, 2008.

- HALL, S. H., HALL, G. W., MACCALL, J. A., **"High-Speed Digital System Design—A Handbook of Interconnect Theory and Design Practices."**, Wiley-Interscience Publication JOHN WILEY & SONS, INC., 2000.

- Linux-MTD, **"UBI – Unsorted Block Images"**, encontrado em: <http://www.linux-mtd.infradead.org/doc/ubi.html>, acessado em: 10/11/2010

- Kerneltrap, **"UBI File System"**, encontrado em: <http://kerneltrap.org/Linux/UBI_File_System>, acessado em 11/11/2010

- Codigofonte, **"Dicas - Códigos"**, encontrado em: <http://www.codigofonte.net/dicas/html>, acessado em 12/11/2010

- W3, **"Markup Validation Service"**, encontrado em: <http://validator.w3.org/docs/help.html#validation_basics>, acessado em 10/11/2010.

- NUMA - Núcleo de Manufatura Avançada, **"FMEA (Failure Model and Effect Analysis)"**, encontrado em: <http://www.numa.org.br/conhecimentos/conhecimentos_port/pag_conhec/FMEAv2.html>, acessado em 09/11/2010

APÊNDICE

Estão inseridos os seguintes itens no apêndice:

- esquemático do dispositivo;
- *layout* do dispositivo;
- tabela das fases de montagem;
- programa AT91Bootstrap;
- resultado do boot: AT91Bootstrap, U-Boot e *Kernel*.

APÊNDICE A – Esquemático do Dispositivo

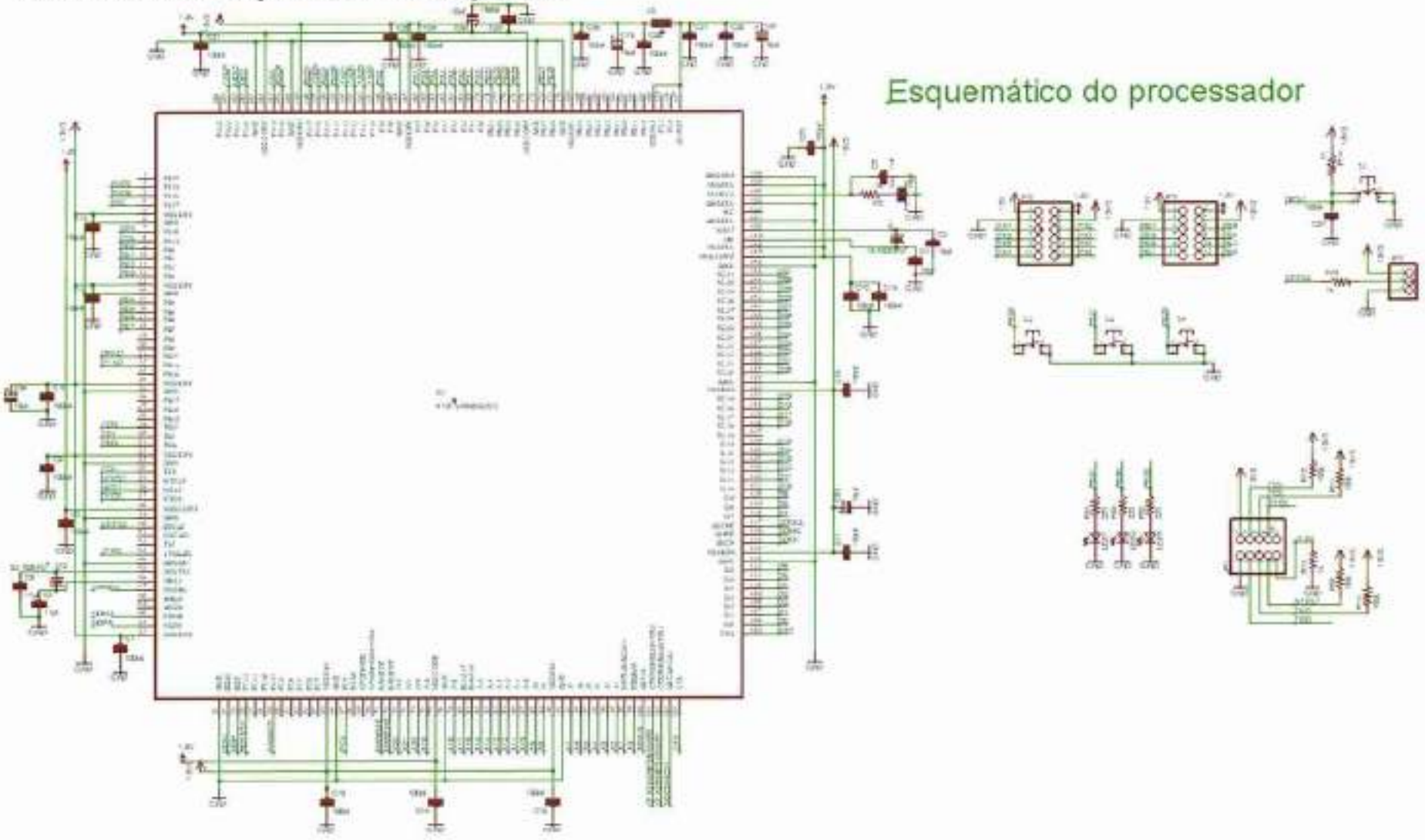

Figura 27 - Esquemático do processador.

Esquemático da memória

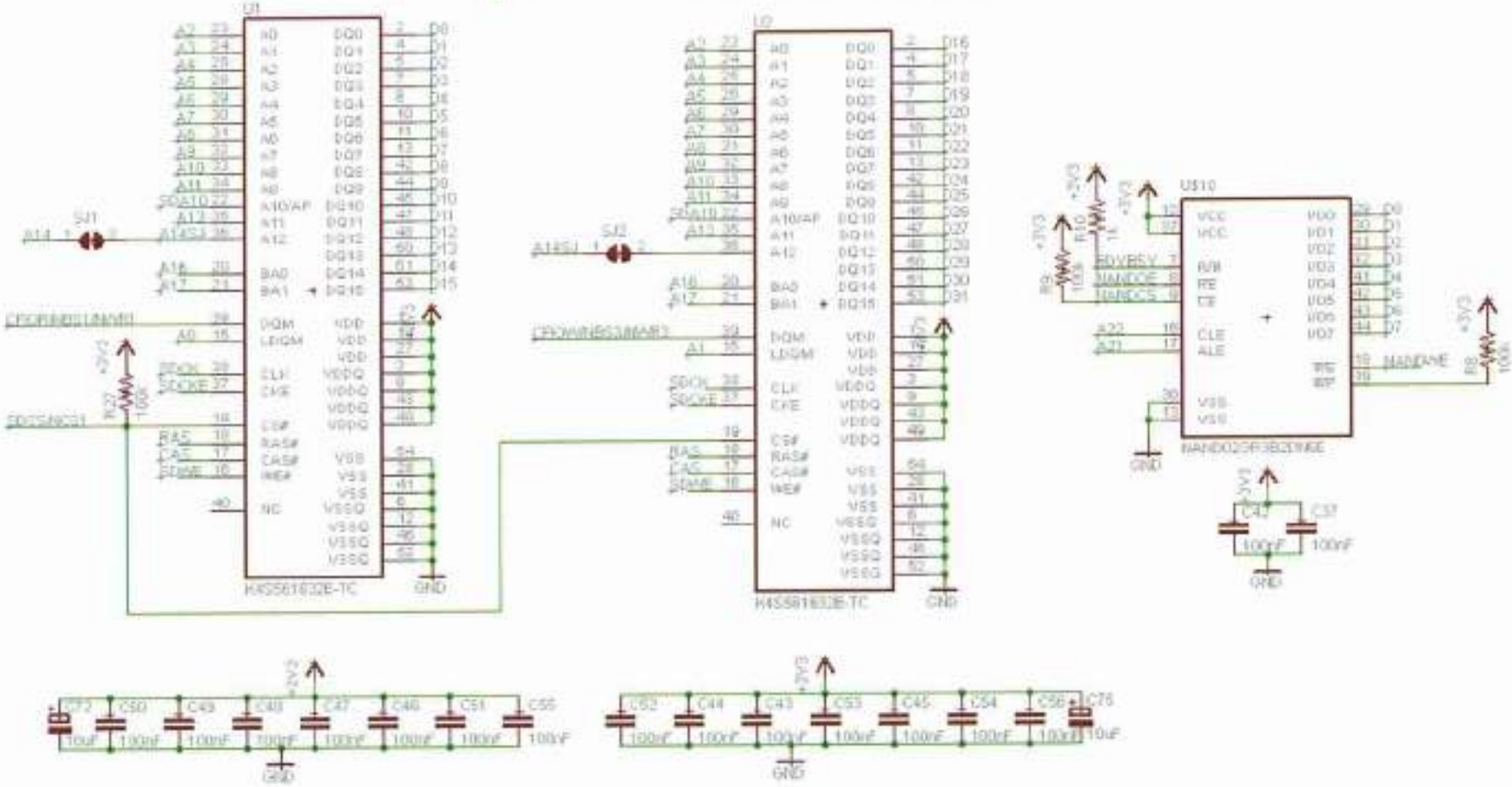

Figura 28 - Esquemático das memórias.

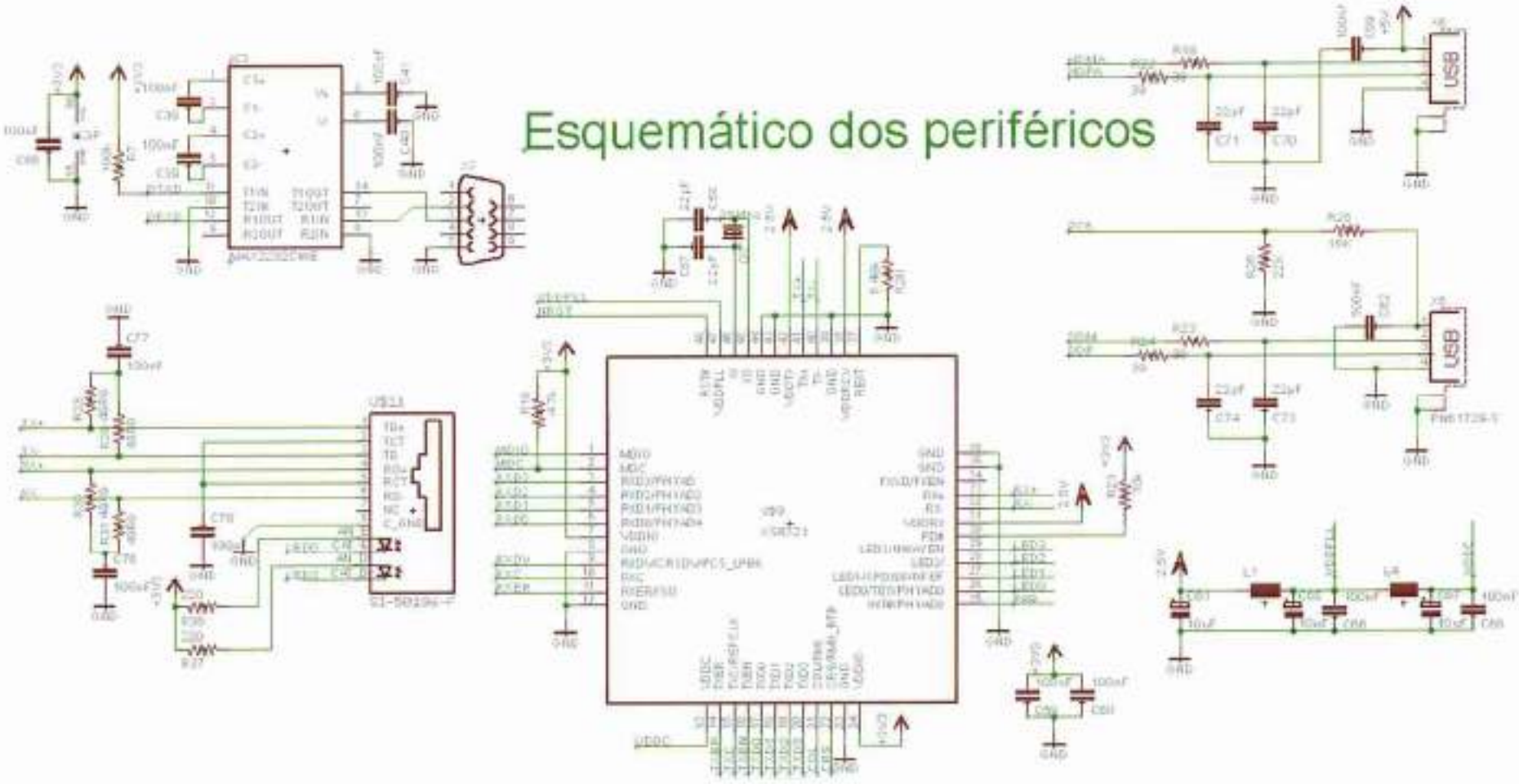

Figura 29 - Esquemático dos periféricos.

Esquemático da alimentaçao

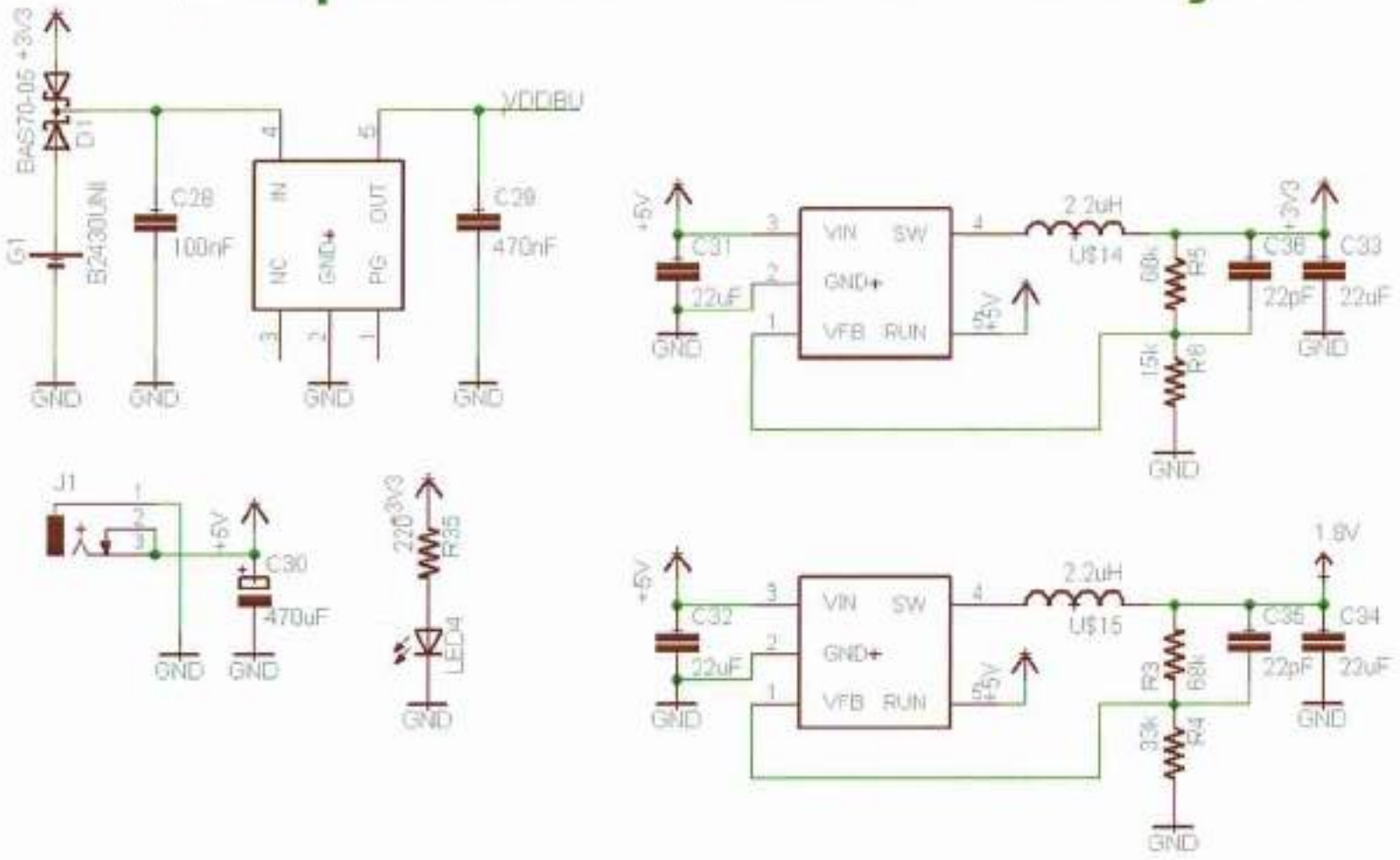

Figura 30 - Esquemático da alimentação.

APÊNDICE B – *Layout* do Dispositivo

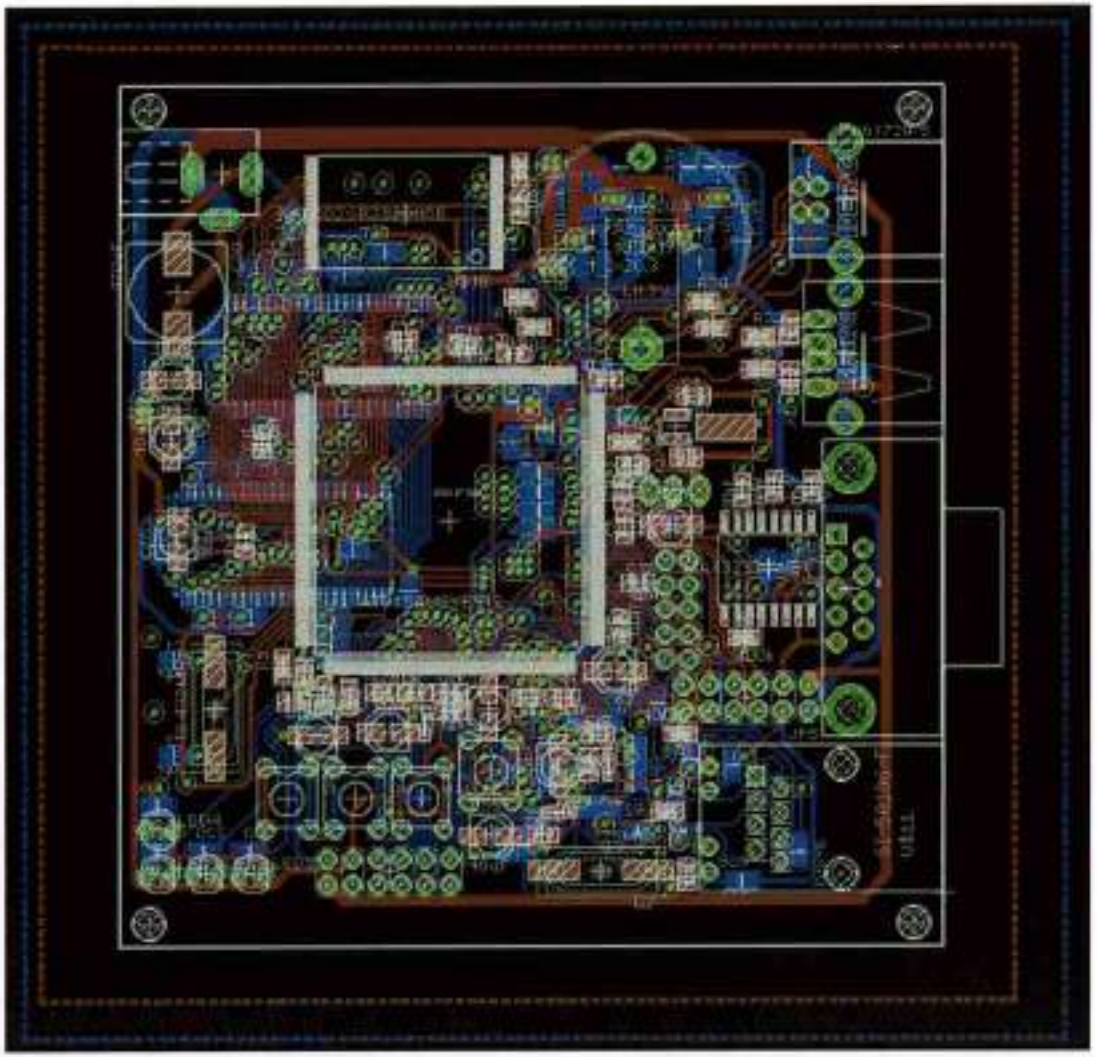

Figura 31 - *Layout* do dispositivo.

APÊNDICE C – Tabela das fases de montagem

Tabela 9 - Metodologia da soldagem (continua).

COMPONENTES POR FASE DO PROJETO			
1)	2 reguladores chaveados	**4)**	NAND Flash
	1 regulador linear		Conector USB Host
	Conector alimentação		Soquete da bateria de backup
2)	Processador	**5)**	Ethernet
	Capacitores de desacoplamento / filtro	**6)**	Conector de expansão
	USB device (conector de impressora USB B)		
3)	Serial / LEDs / Botões	●	Lado do processador
	SDRAM	○	Lado inferior da placa

CÓDIGO	VALOR	PACOTE	X(cm)	Y(cm)	LADO	FASE	PCB1	PCB2	PCB3	PCB4
C28	100nF	C0805K	4,05	5,26	○	1				
C35	22pF	C0805K	6,28	8,10	○	1				
C36	22pF	C0805K	4,91	8,05	○	1				
C31	22uF	C0805K	5,32	7,71	○	1				
C32	22uF	C0805K	6,42	7,71	○	1				
C33	22uF	C0805K	5,32	7,35	○	1				
C34	22uF	C0805K	6,42	7,35	○	1				
C29	470nF	C0805K	4,66	5,39	○	1				
C30	470uF	ELETROLÍTICO	0,61	6,79	●	1				
D2	1N4004	PACOTE SMB	1,86	7,96	○	1				
D1	BAS70-05	SOT23	4,92	5,91	○	1				
J1	ALIMENTAÇÃO	CONECTOR	1,03	7,96	●	1				
LED4		LED3MM	0,45	1,27	●	1				
R35	220	R0805	0,37	1,51	○	1				
R6	15k	R0805	4,48	7,85	○	1				
R4	33k	R0805	5,87	8,10	○	1				
R3	68k	R0805	6,08	8,10	○	1				
R5	68k	R0805	4,70	8,05	○	1				
U$14	2.2uH	INDUTOR	2,27	7,05	●	1				
U$15	2.2uH	INDUTOR	0,84	7,05	●	1				
U$6	LTC3564	SOT23-5	2,35	7,55	●	1				
U$7	LTC3564	SOT23-5	0,89	7,58	●	1				
U$5	TPS79718SC-70	SC70	1,85	5,42	●	1				
C10	100nF	C0805K	5,35	3,80	●	2				
C11	100nF	C0805K	5,20	4,74	●	2				
C12	100nF	C0805K	5,02	5,91	●	2				
C13	100nF	C0805K	4,07	6,17	●	2				
C14	100nF	C0805K	3,59	6,25	●	2				
C15	100nF	C0805K	2,90	6,31	●	2				
C16	100nF	C0805K	1,28	4,24	●	2				
C17	100nF	C0805K	1,46	5,27	●	2				
C18	100nF	C0805K	2,22	3,47	○	2				
C19	100nF	C0805K	2,22	3,28	○	2				
C20	100nF	C0805K	2,22	3,09	○	2				
C21	100nF	C0805K	4,58	2,62	●	2				
C22	100nF	C0805K	3,05	2,68	●	2				
C23	100nF	C0805K	2,69	2,68	●	2				
C24	100nF	C0805K	3,41	2,68	●	2				
C25	100nF	C0805K	4,22	2,63	●	2				
C26	100nF	C0805K	2,35	2,63	●	2				
C27	100nF	C0805K	2,09	2,66	●	2				
C62	100nF	C0805K	7,11	8,11	○	2				
C7	100nF	C0805K	5,22	4,38	●	2				
C8	100nF	C0805K	5,29	5,02	●	2				
C80	100nF	C0805K	3,41	2,49	●	2				

Tabela 9 - Metodologia da soldagem (continuação).

COMPONENTES POR FASE DO PROJETO										
1)	2 reguladores chaveados 1 regulador linear Conector alimentação		**4)**	NAND Flash Conector USB Host Soquete da bateria de backup						
2)	Processador Capacitores de desacoplamento / filtro USB device (conector de impressora USB B)		**5)**	Ethernet						
			6)	Conector de expansão						
3)	Serial / LEDs / Botões SDRAM			● Lado do processador ○ Lado inferior da placa						
CÓDIGO	VALOR	PACOTE	X(cm)	Y(cm)	LADO	FASE	PCB1	PCB2	PCB3	PCB4
C9	100nF	C0805K	5,25	3,33	●	2				
C5	10nF	C0805K	1,79	2,59	●	2				
C1	10pF	C0805K	0,56	3,00	○	2				
C2	10pF	C0805K	0,56	2,03	○	2				
C61	10uF	ELETROLÍTICO	4,53	1,92	●	2				
C65	10uF	ELETROLÍTICO	3,84	1,19	●	2				
C67	10uF	ELETROLÍTICO	5,12	2,89	●	2				
C72	10uF	ELETROLÍTICO	0,52	5,91	●	2				
C75	10uF	ELETROLÍTICO	0,58	4,27	●	2				
C79	10uF	ELETROLÍTICO	3,81	2,50	●	2				
C81	10uF	ELETROLÍTICO	2,77	2,28	●	2				
C82	10uF	ELETROLÍTICO	2,02	2,24	●	2				
C83	10uF	ELETROLÍTICO	0,52	5,31	●	2				
C84	10uF	ELETROLÍTICO	5,69	4,39	●	2				
C3	18pF	C0805K	5,85	5,11	●	2				
C4	18pF	C0805K	5,92	5,73	●	2				
C6	1nF	C0805K	1,89	2,86	●	2				
C73	22pF	C0805K	7,00	7,36	○	2				
C74	22pF	C0805K	7,20	7,36	○	2				
IC1	AT91SAM9XE512	208 pinos	3,40	4,45	●	2				
JP2	PIN HEADER	1X03	5,72	4,76	●	2				
L5	FERRITE BEAD	INDUTOR 805	3,16	2,35	●	2				
Q1	18.432MHz	CRYSTAL	0,97	2,51	●	2				
Q3	32.768kHz	CRYSTAL-32KHZ-SMD	5,75	5,42	●	2				
R23	39	R0805	6,72	6,66	●	2				
R24	39	R0805	6,61	6,97	●	2				
R11	100k	R0805	4,33	4,74	○	2				
R12	100k	R0805	4,33	4,54	○	2				
R13	100k	R0805	4,33	4,34	○	2				
R38	100k	R0805	4,33	4,95	○	2				
R14	1K	R0805	4,19	1,09	○	2				
R15	1k	R0805	5,61	4,43	○	2				
R16	1k	R0805	5,21	5,26	●	2				
R17	1K	R0805	5,16	5,52	○	2				
R2	1k	R0805	1,67	2,95	●	2				
S1	Chave pequena		3,83	1,87	●	2				
X5	USB B - de impressora	CONECTOR	7,48	7,77	●	2				
C43	100nF	C0805K	1,22	5,01	○	3				
C44	100nF	C0805K	0,87	5,05	○	3				
C45	100nF	C0805K	1,33	3,45	○	3				
C46	100nF	C0805K	1,52	5,41	○	3				
C47	100nF	C0805K	1,94	5,41	○	3				
C48	100nF	C0805K	0,93	6,30	○	3				
C49	100nF	C0805K	1,41	7,07	○	3				
C50	100nF	C0805K	1,90	7,07	○	3				

Tabela 9 - Metodologia da soldagem (continuação).

COMPONENTES POR FASE DO PROJETO										
1)	2 reguladores chaveados			**4)**	NAND Flash					
	1 regulador linear				Conector USB Host					
	Conector alimentação				Soquete da bateria de backup					
2)	Processador			**5)**	Ethernet					
	Capacitores de desacoplamento / filtro			**6)**	Conector de expansão					
	USB device (conector de impressora USB B)									
3)	Serial / LEDs / Botões				● Lado do processador					
	SDRAM				○ Lado inferior da placa					
CÓDIGO	VALOR	PACOTE	X(cm)	Y(cm)	LADO	FASE	PCB1	PCB2	PCB3	PCB4
C51	100nF	C0805K	2,30	7,07	○	3				
C52	100nF	C0805K	0,39	4,19	○	3				
C53	100nF	C0805K	2,98	3,71	○	3				
C54	100nF	C0805K	1,60	5,01	○	3				
C55	100nF	C0805K	3,48	6,35	○	3				
C56	100nF	C0805K	0,95	3,45	○	3				
C38	100nF	C0805K	7,13	4,80	●	3				
C39	100nF	C0805K	6,75	4,80	●	3				
C40	100nF	C0805K	6,43	4,76	●	3				
C41	100nF	C0805K	6,99	4,02	○	3				
C86	100nF	C0805K	6,68	3,97	○	3				
IC3	MAX3232CWE	SO16L	6,69	3,95	●	3				
LED1		LED3MM	0,46	0,81	●	3				
LED2		LED3MM	0,91	0,81	●	3				
LED3		LED3MM	1,36	0,81	●	3				
R32	220	R0805	1,33	1,04	○	3				
R33	220	R0805	0,85	1,04	○	3				
R34	220	R0805	0,37	1,04	○	3				
R27	100k	R0805	2,72	7,24	○	3				
R7	100k	R0805	6,44	3,23	●	3				
S2	Chave pequena		2,42	1,59	●	3				
S3	Chave pequena		3,11	1,59	●	3				
S4	Chave pequena		1,72	1,59	●	3				
U1	SDRAM	TSOP54	2,19	6,16	○	3				
U2	SDRAM	TSOP54	1,69	4,21	○	3				
X7	Serial	CONECTOR	7,54	3,74	●	3				
C69	100nF	C0805K	7,58	5,97	○	4				
C37	100nF	C0805K	4,13	7,71	●	4				
C42	100nF	C0805K	1,77	7,54	○	4				
C70	22pF	C0805K	6,91	5,91	●	4				
C71	22pF	C0805K	6,91	6,43	●	4				
G1	BATERIA DE BACKUP	SOQUETE	0,32	7,38	●	4				
R19	39	R0805	6,59	6,01	●	4				
R22	39	R0805	6,59	6,33	●	4				
R8	100k	R0805	4,13	8,09	●	4				
R9	100k	R0805	4,28	6,45	●	4				
R10	1k	R0805	4,13	6,76	●	4				
U$10	NANDFLASH	TSOP48	0,39	7,64	●	4				
X6	USB HOST (PC)	CONECTOR	7,48	6,17	●	4				
C59	100nF	C0805K	4,01	1,95	○	5				
C60	100nF	C0805K	5,26	2,65	○	5				
C66	100nF	C0805K	4,85	1,93	●	5				
C68	100nF	C0805K	4,94	2,25	●	5				
C76	100nF	C0805	5,79	2,25	●	5				
C77	100nF	C0805	6,33	0,99	○	5				

Tabela 9 - Metodologia da soldagem (conclusão).

COMPONENTES POR FASE DO PROJETO										
1)	2 reguladores chaveados 1 regulador linear Conector alimentação			**4)**	NAND Flash Conector USB Host Soquete da bateria de backup					
2)	Processador Capacitores de desacoplamento / filtro USB device (conector de impressora USB B)			**5)**	Ethernet					
3)	Serial / LEDs / Botões SDRAM			**6)**	Conector de expansão					
				●	Lado do processador					
				○	Lado inferior da placa					
CÓDIGO	VALOR	PACOTE	X(cm)	Y(cm)	LADO	FASE	PCB1	PCB2	PCB3	PCB4
C78	100nF	C0805	7,01	1,03	○	5				
C57	22pF	C0805K	4,37	1,20	●	5				
C58	22pF	C0805K	5,85	0,79	●	5				
L1	FERRITE BEAD	INDUTOR 805	4,54	1,45	●	5				
L4	FERRITE BEAD	INDUTOR 805	5,04	1,94	●	5				
Q2	25Mhz	SM49	4,97	0,83	●	5				
R36	220	R0805	6,44	0,68	○	5				
R37	220	R0805	6,32	1,91	○	5				
R21	10k	R0805	5,55	2,60	○	5				
R18	4.7k	R0805	4,34	1,40	○	5				
R28	49R9	R0805	6,33	1,34	○	5				
R29	49R9	R0805	5,83	1,57	○	5				
R30	49R9	R0805	5,89	1,99	○	5				
R31	49R9	R0805	5,78	2,29	○	5				
R20	6.49k	R0805	5,35	1,26	○	5				
U$11	ETHERNET	CONECTOR	0,88	0,61	●	5				
U$9	KS8721	KS8721	2,34	1,94	●	5				
JP3	PIN FEMALE	2X06	2,77	0,83	●	6				
JP5	PIN FEMALE	2X06	6,46	2,62	●	6				
R25	15K	R0805	4,42	5,84	○	6				
R26	22K	R0805	3,99	5,67	○	6				
~~C87~~	~~CANCELADO~~	~~CANCELADO~~								
~~SJ1~~	~~CANCELADO~~	~~CANCELADO~~								
~~SJ1~~	~~CANCELADO~~	~~CANCELADO~~								

APÊNDICE D – Programa AT91Bootstrap

```
/* ----------------------------------------------------------------------------
 *         ATMEL Microcontroller Software Support  -  ROUSSET  -
 * ----------------------------------------------------------------------------
 * Copyright (c) 2006, Atmel Corporation

 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the disclaimer below.
 *
 * Atmel's name may not be used to endorse or promote products derived from
 * this software without specific prior written permission.
 *
 * DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 * DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
 * OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ----------------------------------------------------------------------------
 * File Name           : main.c
 * Object              :
 * Creation            : ODi Apr 19th 2006
 *----------------------------------------------------------------------------
 */

#include "include/part.h"
#include "include/main.h"
#include "include/debug.h"
#include "include/dataflash.h"
#include "include/nandflash.h"
#include "include/norflash.h"
#include "include/gpio.h"

void ARE(void)
{
	const struct pio_desc ark_pinos[] = {
		{"BOT0", AT91C_PIN_PB(26), 0, PIO_PULLUP, PIO_INPUT},
		{"BOT1", AT91C_PIN_PB(27), 0, PIO_PULLUP, PIO_INPUT},
		{"BOT2", AT91C_PIN_PB(28), 0, PIO_PULLUP, PIO_INPUT},
		{"LED0", AT91C_PIN_PB(31), 0, PIO_DEFAULT, PIO_OUTPUT},
		{"LED1", AT91C_PIN_PB(30), 0, PIO_DEFAULT, PIO_OUTPUT},
		{"LED2", AT91C_PIN_PB(29), 0, PIO_DEFAULT, PIO_OUTPUT},
		{(char *) 0, 0, 0, PIO_DEFAULT, PIO_PERIPH_A},
	};

	/* Configure the PIO controller */
	writel((1 << AT91C_ID_PIOB), PMC_PCER + AT91C_BASE_PMC);
	pio_setup(ark_pinos);

	while(1) {
		pio_set_value(AT91C_PIN_PB(31),!pio_get_value(AT91C_PIN_PB(26)));
		pio_set_value(AT91C_PIN_PB(30),!pio_get_value(AT91C_PIN_PB(28)));
		pio_set_value(AT91C_PIN_PB(29),!pio_get_value(AT91C_PIN_PB(27)));
	}
```

```
}

/*------------------------------------------------------------------------------*/
/* Function Name       : main                                                   */
/* Object              : Main function                                          */
/* Input Parameters    : none                                                   */
/* Output Parameters   : True                                                   */
/*------------------------------------------------------------------------------*/
int main(void)
{
/* ------------------ 1st step: Hardware Initialization ------------------ */
        /* Performs the hardware initialization */
#ifdef CFG_HW_INIT
        hw_init();
#endif

/* ---------------------- 2nd step: Load from media ---------------------- */
        /* Load from Dataflash in RAM */
#ifdef CFG_DATAFLASH
        load_df(AT91C_SPI_PCS_DATAFLASH, IMG_ADDRESS, IMG_SIZE, JUMP_ADDR);
#endif

        /* Load from Nandflash in RAM */
#ifdef CFG_NANDFLASH
        load_nandflash(IMG_ADDRESS, IMG_SIZE, JUMP_ADDR);
#endif

        /* Load from Norflash in RAM */
#ifdef CFG_NORFLASH
        load_norflash(IMG_ADDRESS, IMG_SIZE, JUMP_ADDR);
#endif

/* --------------------- 3rd step:  Process the Image --------------------- */
        /* Uncompress the image */
#ifdef GUNZIP
        decompress_image((void *)IMG_ADDRESS, (void *)JUMP_ADDR, IMG_SIZE);     /*
NOT IMPLEMENTED YET */
#endif /* GUNZIP */

/* --------------------- 4th step: Start the application -------------------- */
        /* Set linux arguments */
#ifdef LINUX_ARG
        linux_arg(LINUX_ARG);      /* NOT IMPLEMENTED YET */
#endif /* LINUX_ARG */

        ARE();

        /* Jump to the Image Address */
        return JUMP_ADDR;
}
```

APÊNDICE E – Resultado do *boot*: AT91Bootstrap, U-Boot e *Kernel*

```
ARK: Start AT91Bootstrap...
ARK: Nand Init...


U-Boot 2010.06 (Sep 24 2010 - 22:53:34)

DRAM:  32 MiB
NAND:    ID0=ec,ID1=d5,ID2=55,ID3=25,Write:2048,OOB:64,Erase:262144,busw:0,size:2048
MiB
In:    serial
Out:   serial
Err:   serial
Net:   macb0
macb0: Starting autonegotiation...
macb0: Autonegotiation complete
macb0: link up, 100Mbps full-duplex (lpa: 0x4de1)
Hit any key to stop autoboot:  0
Using macb0 device
TFTP from server 192.168.0.151; our IP address is 192.168.0.200
Filename 'uImage'.
Load address: 0x20000000
Loading: ##################################################################
         ##################################################################
done
Bytes transferred = 1842928 (1c1ef0 hex)
## Booting kernel from Legacy Image at 20000000 ...
   Image Name:   Linux-2.6.33.2Albert-19/9/2010
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1842864 Bytes = 1.8 MiB
   Load Address: 20008000
   Entry Point:  20008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[      0.000000] Linux version 2.6.33.2Albert-19/9/2010 (tf@linuxtf) (gcc version
4.3.4 (ARK-19.09.2010) ) #4 Mon Sep 27 21:18:23 BRT 2010
[    0.000000] CPU: ARM926EJ-S [41069265] revision 5 (ARMv5TEJ), cr=00053177
[    0.000000] CPU: VIVT data cache, VIVT instruction cache
[    0.000000] Machine: Atmel AT91SAM9260-EK
[    0.000000] Memory policy: ECC disabled, Data cache writeback
[    0.000000] Clocks: CPU 198 MHz, master 99 MHz, main 18.432 MHz
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
8128
[    0.000000] Kernel command line: console=ttyS0,115200 root=/dev/nfs rw
 nfsroot=192.168.0.151:/home/tf/Desktop/nfsroot ip=dhcp
[    0.000000] PID hash table entries: 128 (order: -3, 512 bytes)
[    0.000000] Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
[    0.000000] Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Memory: 32MB = 32MB total
[    0.000000] Memory: 28764KB available (3200K code, 244K data, 128K init, 0K
 highmem)
[    0.000000] Hierarchical RCU implementation.
[    0.000000] NR_IRQS:192
[    0.000000] AT91: 96 gpio irqs in 3 banks
[    0.000000] Console: colour dummy device 80x30
[    0.000000] console [ttyS0] enabled
[    0.100000] Calibrating delay loop... 99.12 BogoMIPS (lpj=495616)
[    0.310000] Mount-cache hash table entries: 512
```

```
[    0.310000] CPU: Testing write buffer coherency: ok
[    0.330000] NET: Registered protocol family 16
[    0.340000] AT91: Power Management
[    0.350000] AT91: Starting after user reset
[    0.400000] bio: create slab <bio-0> at 0
[    0.410000] SCSI subsystem initialized
[    0.410000] usbcore: registered new interface driver usbfs
[    0.420000] usbcore: registered new interface driver hub
[    0.420000] usbcore: registered new device driver usb
[    0.430000] Advanced Linux Sound Architecture Driver Version 1.0.21.
[    0.440000] cfg80211: Calling CRDA to update world regulatory domain
[    0.450000] Switching to clocksource tcb_clksrc
[    0.460000] NET: Registered protocol family 2
[    0.460000] IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.470000] TCP established hash table entries: 1024 (order: 1, 8192 bytes)
[    0.480000] TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
[    0.480000] TCP: Hash tables configured (established 1024 bind 1024)
[    0.490000] TCP reno registered
[    0.490000] UDP hash table entries: 256 (order: 0, 4096 bytes)
[    0.500000] UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
[    0.500000] NET: Registered protocol family 1
[    0.510000] RPC: Registered udp transport module.
[    0.510000] RPC: Registered tcp transport module.
[    0.520000] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.530000] JFFS2 version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
[    0.540000] msgmni has been set to 56
[    0.550000] alg: No test for stdrng (krng)
[    0.550000] io scheduler noop registered (default)
[    0.610000] atmel_usart.0: ttyS0 at MMIO 0xfefff200 (irq = 1) is a ATMEL_SERIAL
[    0.620000] atmel_usart.1: ttyS1 at MMIO 0xfffb0000 (irq = 6) is a ATMEL_SERIAL
[    0.640000] brd: module loaded
[    0.660000] loop: module loaded
[    0.680000] NAND device: Manufacturer ID: 0xec, Chip ID: 0xd5 (Samsung NAND 2GiB
3,3V 8-bit)
[    0.690000] AT91 NAND: 8-bit, Software ECC
[    0.690000] Scanning device for bad blocks
[    1.060000] Creating 2 MTD partitions on "atmel_nand":
[    1.070000] 0x000000000000-0x000000080000 : "Partition 1"
[    1.080000] 0x000000080000-0x000080000000 : "Partition 2"
[    1.100000] MACB mii_bus: probed
[    1.100000] eth0: Atmel MACB at 0xfffc4000 irq 21 (08:00:27:0e:03:46)
[    1.110000] eth0: attached PHY driver [Generic PHY]
(mii_bus:phy_addr=ffffffff:01, irq=-1)
[    1.120000] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[    1.120000] at91_ohci at91_ohci: AT91 OHCI
[    1.130000] at91_ohci at91_ohci: new USB bus registered, assigned bus number 1
[    1.130000] at91_ohci at91_ohci: irq 20, io mem 0x00500000
[    1.200000] usb usb1: New USB device found, idVendor=1d6b, idProduct=0001
[    1.210000] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    1.210000] usb usb1: Product: AT91 OHCI
[    1.220000] usb usb1: Manufacturer: Linux 2.6.33.2Albert-19/9/2010 ohci_hcd
[    1.220000] usb usb1: SerialNumber: at91
[    1.230000] hub 1-0:1.0: USB hub found
[    1.230000] hub 1-0:1.0: 2 ports detected
[    1.240000] Initializing USB Mass Storage driver...
[    1.240000] usbcore: registered new interface driver usb-storage
[    1.250000] USB Mass Storage support registered.
[    1.250000] usbcore: registered new interface driver usbserial
[    1.260000] USB Serial support registered for generic
[    1.260000] usbcore: registered new interface driver usbserial_generic
[    1.270000] usbserial: USB Serial Driver core
[    1.270000] udc: at91_udc version 3 May 2006
[    1.280000] g_serial gadget: Gadget Serial v2.4
[    1.280000] g_serial gadget: g_serial ready
[    1.290000] mice: PS/2 mouse device common for all mice
[    1.300000] input: gpio-keys as /devices/platform/gpio-keys/input/input0
[    1.300000] Registered led device: ds5
```

```
[    1.310000] Registered led device: ds1
[    1.310000] usbcore: registered new interface driver snd-usb-audio
[    1.320000] ALSA device list:
[    1.320000]   No soundcards found.
[    1.330000] IPv4 over IPv4 tunneling driver
[    1.330000] TCP cubic registered
[    1.340000] NET: Registered protocol family 17
[    1.340000] lib80211: common routines for IEEE802.11 drivers
[    2.870000] Sending DHCP requests .,
[    3.100000] eth0: link up (100/Full)
[    3.390000]  OK
[    3.390000] IP-Config: Got DHCP answer from 0.0.0.0, my address is 192.168.0.102
[    3.390000] IP-Config: Complete:
[    3.400000]      device=eth0, addr=192.168.0.102, mask=255.0.0.0,
 gw=192.168.0.1,
[    3.400000]      host=192.168.0.102, domain=, nis-domain=(none),
[    3.410000]      bootserver=0.0.0.0, rootserver=192.168.0.151, rootpath=
[    3.420000] Looking up port of RPC 100003/2 on 192.168.0.151
[    3.430000] Looking up port of RPC 100005/1 on 192.168.0.151
[    3.440000] VFS: Mounted root (nfs filesystem) on device 0:12.
[    3.450000] Freeing init memory: 128K
```