UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

RELATÓRIO DE FIRMWARE

PROJETO: Blackfin IP Phone	**DATA:** 03/12/2012

ÍNDICE

1. INTRODUÇÃO

O projeto Blackfin IP Phone consiste em um telefone IP com a funcionalidade básica de realizar ligações VoIP. Essa tecnologia compreende um método de comunicação através da fala entre dois locais através de um meio digital de comunicação. A telefonia IP vem sendo abordada há alguns anos o que é motivado pela sua crescente aplicabilidade, bem como a possibilidade de agregação de outras funcionalidades o que a torna um campo de estudo bastante atrativo. A arquitetura de hardware proposta se baseia no processador ADSP-BF518 da família de processadores Blackfin, fabricada pela Analog Devices.

Este documento tem como objetivo principal indicar todos os métodos, ferramentas e detalhes de implementação do firmware desenvolvido para contemplar todas as funcionalidades do protótipo de hardware do projeto Blackfin IP Phone, Versão 1. Boa parte desta documentação é baseada em uma documentação livre disponibilizada pelo Analog Devices, através de www.docs.blackfin.uclinux.org.

A **Seção 2 – Protótipo de Hardware** contém detalhes relacionados ao protótipo de hardware a partir do qual foram estabelecidos requisitos funcionais e foi desenvolvido um firmware específico para contemplar todas suas funcionalidades.

A **Seção 3 – Ambiente de Desenvolvimento** apresenta comentários relacionados ao ambiente de desenvolvimento, sua configuração e instalação. Nesta seção são descritos os utilitários contidos na toolchain para desenvolvimento de firmware voltado para processadores da família Blackfin. Além disso, são indicadas as configurações da estação de trabalho que foi considerada como padrão para todos os desenvolvedores do firmware do projeto.

A **Seção 4 – Bootloader** apresenta todas as etapas relacionadas ao desenvolvimento e adequação do bootloader utilizado no projeto. Esta seção contém uma descrição de todos os processos de gravação do bootloader na placa, bem como a sequência de implementação e detalhes relacionados a sua execução.

A **Seção 5 – uClinux** apresenta detalhes sobre a configuração do sistema operacional, bem como aplicações de teste desenvolvidas para validar as interfaces de hardware. Além disso, são apresentados detalhes de configuração e compilação do sistema operacional.

A **Seção 6 – Aplicação Principal** contém informações sobre a aplicação do telefone VoIP, apresentando a arquitetura de firmware e destacando a função das unidades em que foi feito reuso, baseado em projetos disponíveis livremente para a comunidade de desenvolvedores. Nesta seção é descrita a máquina de estados principal, apresentadas as fontes de eventos e como eles são

administrados.

Por fim, a **Seção 7 – Gestão de Artefatos** trata sobre as ferramentas utilizadas para gerir todos os documentos gerados ao longo do projeto. São apresentados os modos de acesso ao repositório e a ferramenta de controle de versão, o que permitiu a divulgação do projeto na comunidade de desenvolvedores, bem como melhorou a dinâmica de implementação das atividades em paralelo pelos desenvolvedores do projeto.

2. PROTÓTIPO DE HARDWARE

O objeto de desenvolvimento do firmware do projeto Blackfin IP Phone é um hardware baseado no processador ADSP-BF518, elemento da família de processadores Blackfin. Dentre as funcionalidades da arquitetura de hardware encontra-se a interface com o usuário constituída por mecanismos intuitivos e de fácil utilização, o que é mandatório em equipamentos eletrônicos com acessibilidade satisfatória. São ainda disponíveis interface ethernet, conexão com um cartão de memória microSD, conexão com um display de caracteres e teclado matricial. É utilizada memória SDRAM conectada ao barramento externo do processador e memória flash acessível através de interface SPI. Através da conexão USB são disponibilizadas as interfaces JTAG ou serial, o que é contemplado através de um conversor de USB para serial e/ou interface FIFO. A interface serial do processador também é disponibilizada de modo direto através de um conversor de nível lógico. A **Figura 2.1** representa a arquitetura de hardware do projeto Blackfin IP Phone, Versão 1.

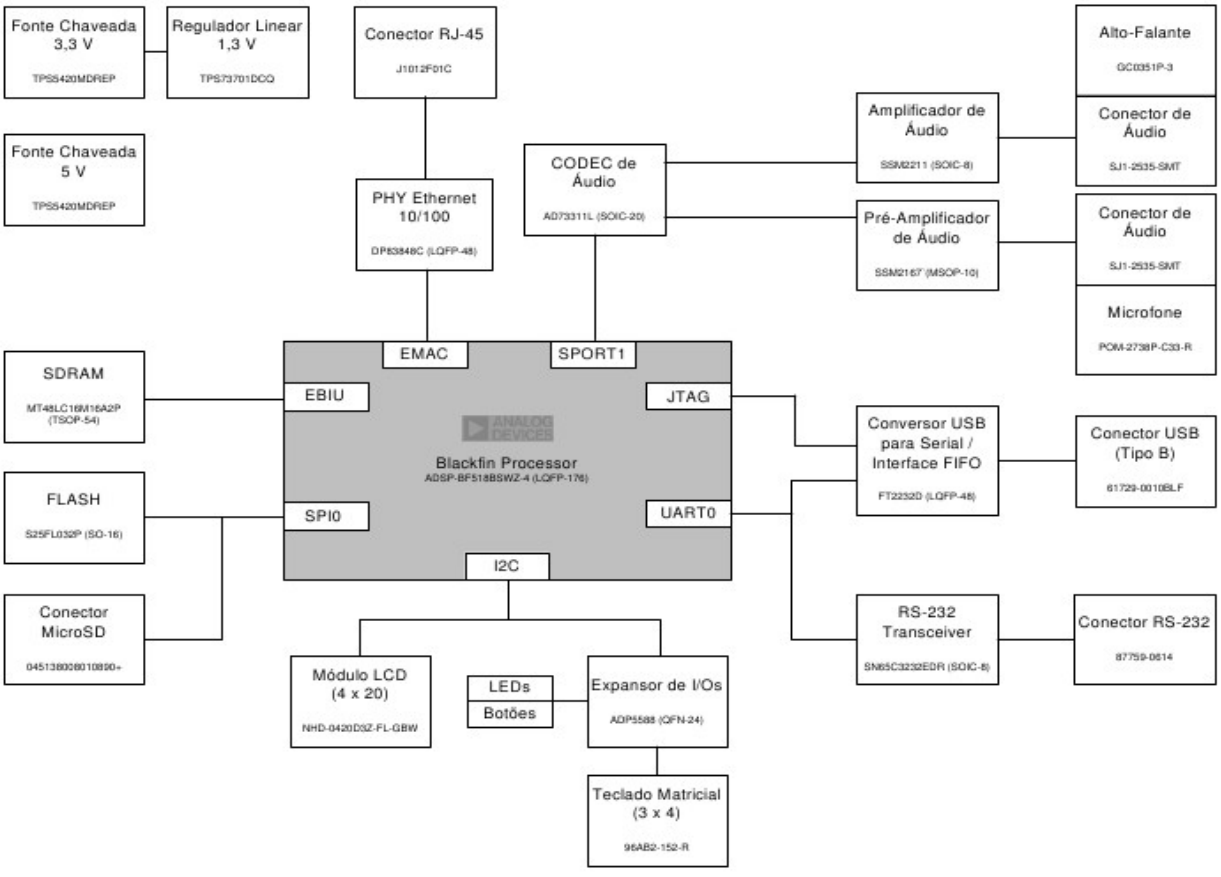

Figura 2.1 - Arquitetura de Hardware

Para maiores informações sobre o desenvolvimento de hardware do projeto Blackfin IP Phone, Versão 1, consultar a página do projeto, disponível em http://code.google.com/p/blackfin-ip-phone-hw. Na página do projeto, é possível encontrar a lista de materiais, bem como todos os arquivos necessários para a fabricação da placa. A **Figura 2.2** contém uma imagem do protótipo do projeto Blackfin IP Phone.

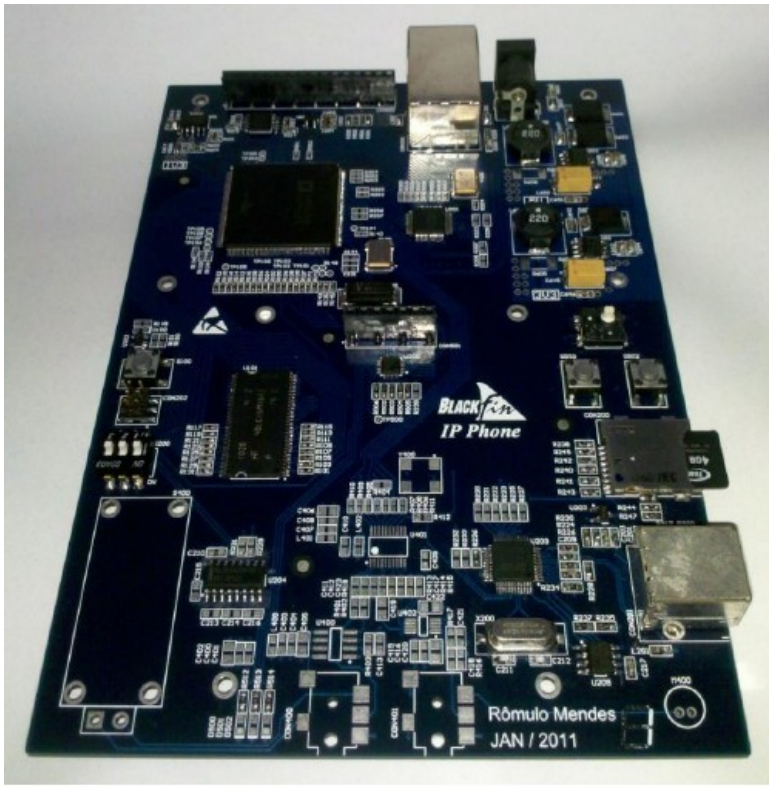

Figura 2.2 - Protótipo de Hardware (1a Versão)

3. AMBIENTE DE DESENVOLVIMENTO

Nesta seção, são apresentadas as características do ambiente de desenvolvimento, bem como os procedimentos para instalação das ferramentas e recursos necessários.

3.1. ESTAÇÃO DE TRABALHO

O desenvolvimento de firmware foi realizado em sua totalidade em ambiente Linux, distribuição Ubuntu, versão do Kernel 2.6.35. Em relação a configuração mínima de memória, deve ser suficiente para armazenamento do toolchain e dos códigos-fonte do bootloader e do uClinux.

3.2. TOOLCHAIN

O toolchain do Blackfin consiste de um conjunto de utilitários necessários para a compilação, depuração e geração de programas executáveis a partir do código fonte dos artefatos de firmware como bootloader e uClinux, por exemplo. A partir desses utilitários podem ser gerados diversos tipos de saídas, a partir das quais são gravados os programas executáveis. Segue abaixo o procedimento para download e instalação do toolchain para processadores da família Blackfin. O download e instalação podem ser feitos em linha de comando.

3.2.1. INSTALAÇÃO DO TOOLCHAIN EM LINHA DE COMANDO

Executar os seguintes comandos no bash:

```
$ wget http://download.analog.com/27516/distros/debian/apt.key
$ sudo apt-key add apt.key
$ sudo su -
# mkdir -p /etc/apt/sources.list.d
# cd /etc/apt/sources.list.d
# echo deb http://download.analog.com/27516/distros/debian stable main >
```

```
blackfin.sources.list
# exit
$ sudo apt-get update
$ sudo apt-get install blackfin-toolchain-uclinux blackfin-toolchain-linux-uclibc
blackfin-toolchain-elf
```

Neste procedimento, são realizados o download e a instalação do toolchain. Logo após deverá ser feita a configuração da variável de ambiente `PATH`.

3.2.2. INSTALAÇÃO DO TOOLCHAIN A PARTIR DO CÓDIGO-FONTE

Após o download das imagens pŕe-compiladas do toolchain e da biblioteca uClibc, devem ser extraídos para a pasta `/opt`.

No caso da versão 2011R1-RC4_45, após descompactar o toolchain (blackfin-toolchain-2011R1-RC4_45.i386.tar.bz2) e a biblioteca uClibc (blackfin-toolchain-uclibc-full-2011R1-RC4_45.i386.tar.bz2), disponibilizadas pelo projeto *GNU Toolchain for the Blackfin Processor* no Blackfin Koop, todos os arquivos devem ser organizados em uma única árvore de diretórios e levados até a pasta `/opt`. A seguir, deve ser realizada a configuração da variável de ambiente `PATH`.

3.2.3. CONFIGURAÇÃO DA VARIÁVEL DE AMBIENTE PATH

- Configuração da variável de ambiente PATH como usuário não root.

Abrir o arquivo .bashrc:

```
$ gedit ~/.bashrc
```

Inserir as linhas a seguir no final do arquivo .bashrc:

```
# Configuração da variável de ambiente PATH para a toolchain do Blackfin
export PATH=$PATH:/opt/uClinux/bfin-uclinux/bin:/opt/uClinux/bfin-linux-
uclibc/bin:/opt/uClinux/bfin-elf/bin
```

Salvar o arquivo antes de fechá-lo. Reiniciar o bash após o fechamento do arquivo.

- Configuração da variável de ambiente PATH como usuário root.

Abrir o arquivo .bashrc no contexto do usuário root:

```
$ sudo su -
# gedit ~/.bashrc
```

Inserir as linhas a seguir no final do arquivo .bashrc:

```
# Configuração da variável de ambiente PATH para a toolchain do Blackfin
export PATH=$PATH:/opt/uClinux/bfin-uclinux/bin:/opt/uClinux/bfin-linux-
uclibc/bin:/opt/uClinux/bfin-elf/bin
```

Salvar o arquivo antes de fechá-lo. Reiniciar o bash após o fechamento do arquivo.

4. BOOTLOADER

O bootloader utilizado no projeto Blackfin IP Phone é o U-Boot, versão 2010r1, disponível em www.blackfin.uclinux.org através do projeto *Das U-Boot for the Blackfin Processor*. Dentre os diversos processadores suportados por esse bootloader encontra-se o Blackfin.

4.1. GERAÇÃO DA PRIMEIRA IMAGEM DO BOOTLOADER

O procedimento que se segue é adequado para situações em que não há bootloader integrado ao hardware ou quando o mesmo encontra-se corrompido e não é executado de modo apropriado. Desse modo, em um cenário em que não há bootloader integrado ao protótipo de hardware, uma vez que o sistema foi desenvolvido a partir de um hardware sem cargas prévias de bootloader, é necessária a geração de um arquivo binário com a imagem do U-Boot contendo uma configuração básica da placa alvo. A imagem do U-Boot a ser gerada deve considerar o boot pela interface serial, uma vez que este será o meio de download para a placa alvo. Além disso, na primeira carga do U-Boot, a configuração da interface de rede é opcional. A edição incremental da imagem do U-Boot tem por objetivo a validação gradual das interfaces de hardware. Portanto, na primeira carga do U-Boot deverão ser feitas as configurações das memórias volátil e não-volátil, bem como a seleção do modo de boot através da interface serial.

Deve ser observado que a seleção do modo de boot é realizada em hardware através dos pinos BMODE[2..0] do processador. A configuração do modo de boot é feita durante o procedimento de reset, quando os sinais BMODE[2..0] são coletados pelo processador.

Para geração da imagem inicial do U-Boot foram criados os artefatos citados abaixo e armazeanados na árvore de diretórios do U-Boot. Os caminhos relativos dos artefatos, suas descrições e conteúdo para geração da primeira imagem do bootloader são relacionados abaixo.

- ../u-boot-2010.06-2010R1-RC2/include/configs/bf518-ipphone.h

Este arquivo foi gerado a partir de ../u-boot-2010.06-2010R1-RC2/include/configs/bf518f-ezbrd.h e contém a configuração da placa Blackfin IP Phone. Este arquivo deve conter um perfil básico de configuração a partir do qual os dispositivos da placa serão reconhecidos. Assim, deve estabelecer as

configurações relacionadas ao modo de boot do processador, clock, memórias, rede, dentre outros.

Para geração da primeira imagem do bootloader, que permite o boot pela interface serial, devem ser feitas as configurações que se seguem. A seção de configuração do processador deve estar de acordo com o que se segue.

```
#define CONFIG_BFIN_CPU             bf518-0.1
#define CONFIG_BFIN_BOOT_MODE       BFIN_BOOT_SPI_MASTER
```

A seção de configuração do clock deve estar de acordo com o que se segue.

```
#define CONFIG_CLKIN_HZ             25000000
#define CONFIG_CLKIN_HALF           0
#define CONFIG_PLL_BYPASS           0
#define CONFIG_VCO_MULT             16
#define CONFIG_CCLK_DIV             1
#define CONFIG_SCLK_DIV             4
```

A primeira imagem do bootloader deve conter também as configurações da memória volátil, a partir da qual o U-boot deverá ser executado. Para tanto, deve ser utilizado uma planilha para cálculo das configurações da SDRAM a partir de seus parâmetros de temporização. A planilha BfSdcCalculation_Release.xlsx é disponibilizada pela Analog Devices e, a partir dela, a seção de configuração da memória SDRAM foi gerada.

```
#define CONFIG_MEM_ADD_WDTH         9
#define CONFIG_MEM_SIZE             32
#define CONFIG_EBIU_SDRRC_VAL       0x0305
#define CONFIG_EBIU_SDGCTL_VAL      0x8091118D
#define CONFIG_EBIU_SDBCTL_VAL      0x0013
```

A seção de configurações diversas deve estar de acordo com o que se segue. Esta seção está localizada no final do arquivo de configuração.

```
#define CONFIG_UART_CONSOLE         0
#define CONFIG_BAUDRATE             115200
```

Quando necessária a visualização de mensagens de debug durante a execução do U-Boot, deve ser feita a seguinte definição:

```
#define DEBUG
```

- ../u-boot-2010.06-2010R1-RC2/board/bf518-ipphone/bf518-ipphone.c

Para criação deste arquivo foi utilizado como referência um arquivo análogo para a placa de desenvolvimento com o processador ADSP-BF518. O arquivo de referência é o ./u-boot-2010.06-2010R1-RC2/board/bf518f-ezbrd/bf518f-ezbrd.c. Este arquivo contém funções de inicialização da placa. Inicialmente deve conter apenas a função de apresentação da placa, de acordo com o representado abaixo. Essa função é executada logo após o boot, de modo que, utilizando uma interface serial como interface padrão de entrada e saída, deve ser possível visualizar a mensagem de apresentação da placa.

```
int checkboard(void)
{
      printf("Board: Blackfin IP-Phone board\n");
      printf("    Please, visit http://code.google.com/p/blackfin-ip-phone-hw\n");
      return 0;
}
```

- ../u-boot-2010.06-2010R1-RC2/board/bf518-ipphone/Makefile

Para criação deste arquivo foi utilizado como referência um arquivo análogo para a placa de desenvolvimento com o processador ADSP-BF518. O arquivo de referência é o ./u-boot-2010.06-2010R1-RC2/board/bf518f-ezbrd/Makefile. Não foram feitas alterações significativas em relação ao arquivo de referência.

- ../u-boot-2010.06-2010R1-RC2/board/bf518-ipphone/config.mk

Para criação deste arquivo foi utilizado como referência um arquivo análogo para a placa de desenvolvimento com o processador ADSP-BF518. O arquivo de referência é o ../u-boot-2010.06-2010R1-RC2/board/bf518f-ezbrd/config.mk. Não foram feitas alterações significativas em relação ao

arquivo de referência.

Além da criação desses arquivos, para que seja possível a compilação do U-Boot e geração de sua imagem é necessário que o makefile principal seja editado. O mesmo encontra-se em ../u-boot-2010.06-2010R1-RC2/Makefile. Neste arquivo, a seção onde as placas alvo são relacionadas deve ser editada, de modo a se assemelhar com o indicado abaixo.

```
# Analog Devices boards
BFIN_BOARDS = bf518f-ezbrd bf526-ezbrd bf527-ezkit bf533-ezkit bf533-stamp \
      bf537-pnav bf537-stamp bf538f-ezkit bf548-ezkit bf561-ezkit bf527-ad7160-eval

# Blackfin IP Phone Board
BFIN_BOARDS += bf518-ipphone

# Bluetechnix tinyboards
BFIN_BOARDS += cm-bf527 cm-bf533 cm-bf537e cm-bf537u cm-bf548 cm-bf561 \
      tcm-bf518 tcm-bf537

# Misc third party boards
BFIN_BOARDS += bf537-minotaur bf537-srv1 bf561-acvilon blackstamp ip04

# I-SYST Micromodule
BFIN_BOARDS += ibf-dsp561
```

Para geração da imagem do U-Boot o comando abaixo deve ser executado no bash da estação de trabalho.

```
$ make bf518-ipphone
```

Dentre os diversos arquivos gerados como resultado da compilação do código-fonte do bootloader, encontra-se o arquivo u-boot.ldr, localizado no diretório raiz do U-Boot. Este é o arquivo que deverá ser carregado para a placa alvo durante o processo de boot via interface serial. Para tanto, deve ser executado o comando abaixo no bash da estação de trabalho. Este comando deve ser executado a partir do diretório raiz do U-Boot.

```
$ bfin-uclinux-ldr -l uboot_path/u-boot.ldr /dev/ttyUSB0 && kermit -l /dev/ttyUSB0
```

```
-b 115200 -C connect
```

Observe que, neste caso, a interface serial da placa alvo é acessada através de um conversor USB/serial, de modo que é identificado pela estação de trabalho como uma interface ttyUSB0. Observe ainda que foi utilizado como utilitário de comunicação serial o software kermit. Após a execução deste comando o procedimento de boot deve ser iniciado. Seu sucesso é indicado através da visualização da mensagem de apresentação da placa e atualização para o estado de espera do bootloader, o que é apresentado abaixo.

```
U-Boot 2010.06 (ADI-2010R1-RC2) (Aug 09 2011 - 14:00:47)

CPU:   ADSP bf518-0.1 (Detected Rev: 0.1) (spi flash boot)
Board: Blackfin IP-Phone board
       Please, visit http://code.google.com/p/blackfin-ip-phone-hw
Clock: VCO: 400 MHz, Core: 400 MHz, System: 100 MHz
RAM:   32 MiB
MMC:
SF: Detected S25FL032P with page size 256, total 4 MiB
In:    serial
Out:   serial
Err:   serial
KGDB:  [on serial] ready
Net:
Hit any key to stop autoboot:  0
bfin>
```

4.2. SUPORTE A MEMÓRIA FLASH E PERSISTÊNCIA DO BOOT

Uma vez que o U-Boot é executado com sucesso a partir da carga pela interface serial, deve ser gerada uma nova imagem do U-Boot considerando o modo de boot pela memória flash acessada pelo processador através da interface SPI. Essa imagem deve ser gravada na memória SDRAM durante a execução do U-Boot que foi carregado através da interface serial. O arquivo a ser carregado é a imagem do U-Boot que deverá ser gravada na memória flash e tem extensão LDR. Uma vez que essa nova imagem foi gravada na memória SDRAM, deve ser executado o comando de gravação da

memória flash, onde são utilizados como parâmetro o endereço de memória onde o arquivo foi carregado, um endereço de offset a partir de onde a imagem será gravada na memória flash e, por fim, o tamanho da imagem a ser gravada. Esta é a descrição geral do procecimento para a gravação da imagem do U-Boot na memória flash, o que é necessário e suficiente para a persistência do processo de boot.

A imagem do U-Boot consistente com o suporte a memória flash e persistência do boot deve ser gerada a partir da modificação de alguns arquivos do código-fonte do bootloader. A seguir são apresentados os arquivos que necessitam ser editados e todas as modificações necessárias para geração dessa imagem.

- ../u-boot-2010.06-2010R1-RC2/include/configs/bf518-ipphone.h

Deve ser alterado o modo de boot da imagem para boot via interface SPI no modo master. Para tanto, a definição previamente existente deve ser alterada como segue.

```
#define CONFIG_BFIN_BOOT_MODE       BFIN_BOOT_SPI_MASTER
```

Este arquivo deve conter também as configurações relacionadas a comunicação entre o processador e a memória flash através da interface SPI. Assim, a seção de configuração da interface SPI deve se assemelhar com o que se segue.

```
#define CONFIG_BFIN_SPI
#define CONFIG_SPI_FLASH
#define CONFIG_CMD_SF
#define CONFIG_SF_DEFAULT_SPEED     50000000
#define CONFIG_ENV_IS_IN_SPI_FLASH
#define CONFIG_ENV_SPI_MAX_HZ       50000000
#define CONFIG_ENV_SPI_BUS          0
#define CONFIG_ENV_SPI_CS           2 /include/configs/bf518-ipphone.h
#define CONFIG_SPI_FLASH_SPANSION
```

Por fim, o arquivo de configuração deve conter também as configurações relacionadas ao armazenamento das variáveis de ambiente. A seção de configuração do armazenamento das variáveis de ambiente deve estar de acordo com o que se segue.

```
#define CONFIG_ENV_IS_IN_SPI_FLASH
#define CONFIG_ENV_OFFSET          0x10000
#define CONFIG_ENV_SIZE            0x2000
#define CONFIG_ENV_SECT_SIZE       0x10000
#define CONFIG_ENV_IS_EMBEDDED_IN_LDR
```

- ../u-boot-2010.06-2010R1-RC2/drivers/mtd/spi/spansion.c

As configurações realizadas no cabeçalho do arquivo de configurações são todas relacionadas a características de armazenamento e comunicação com a interface SPI. As informações específicas da memória flash devem ser direcionadas para o arquivo spansion.c. Neste arquivo devem ser definidos os identificadores da memória flash, bem como uma estrutura com seus parâmetros (identificação, tamanho de página, tamanho de setor, quantidade de setores e nome). Assim, as definições a seguir devem ser inseridas.

```
#define SPSN_ID_S25FL032P          0x0215
#define SPSN_EXT_ID_S25FL032P_64KB  0x4D00
```

Além da definição dos identificadores da memória flash, deve ser criada a estrutura onde essas definições serão encaminhadas para o bootloader. Essa estrutura deve ser inserida no vetor de estruturas que contém os parâmetros de todas as memórias fabricadas pela Spansion e suportadas pelo U-Boot. Assim, após adição do código, o vetor de estruturas deve se assemelhar com o que se segue.

```
static const struct spansion_spi_flash_params spansion_spi_flash_table[] = {

...

/* Suporte a memória flash SPI Spansion S25FL032P */
     {
           .idcode1 = SPSN_ID_S25FL032P,
           .idcode2 = SPSN_EXT_ID_S25FL032P_64KB,
           .page_size = 256,
           .pages_per_sector = 256,
           .nr_sectors = 64,
           .name = "S25FL032P",
```

```
      },

...

}
```

A nova imagem do bootloader deve ser gerada considerando as alterações descritas nesta seção. Assim, após execução do comando `make bf518-ipphone`, o arquivo u-boot.ldr deverá ser gravado na memória SDRAM. Esse procedimento é realizado durante a execução do U-Boot a partir da imagem que foi gerada considerando o boot através da interface serial. Assim, durante a execução do U-Boot, deve ser utilizado o comando `loadb` para gravação de um arquivo binário na memória SDRAM, a partir do endereço `0x01000000`, utilizando a interface serial.

```
bfin> loadb 0x01000000
```

Após a chamada deste comando, deve-se partir para o ambiente do kermit, onde o comando a seguir deve ser executado. O retorno ao ambiente do kermit através da desconexão com a placa alvo é realizado pressionando-se as teclas Ctrl+\ e C, nesta sequência.

```
Linux Kermit> send ../u-boot-2010.06-2010R1-RC2/u-boot.ldr
```

Após o término da gravação da imagem do U-Boot na memória SDRAM, deve-se retornar a conexão com a placa alvo através do comando que se segue.

```
Linux Kermit> connect
```

Neste instante, o booloader que está sendo executado é o mesmo que foi utilizado inicialmente. A partir de então, deve ser executado um comando para gravação do arquivo binário que foi copiado para a memória SDRAM na memória flash. Isso é feito através da seguinte sequência de comandos.

```
bfin> sf probe 2
bfin> sf erase 0x0 0x400000
bfin> sf write 0x01000000 0x0 0x27A18
```

Observe que antes da gravação da memória flash é necessário seu apagamento e que o último parâmetro do comando `sf write` depende do tamanho, em bytes, da imagem a ser carregada.

4.3. SUPORTE A INTERFACE DE REDE

O suporte da interface de rede é realizado habilitando-se o dispositivo MAC ethernet do processador. Para tanto, devem ser feitas alterações em arquivos do código-fonte do U-Boot. Segue abaixo a lista de alterações necessárias para que a interface de rede seja suportada, viabilizando o boot via servidor TFTP. Após a implementação dessas modificações uma nova imagem do U-Boot deve ser gerada e gravada na memória flash.

Uma vez configurado adequadamente, a interface de rede do bootloader proporcionará acesso a rede local. Desse modo, a imagem do sistema operacional poderá ser carregada de forma rápida e segura para, a partir de então, ser realizado o procedimento de inicialização do sistema.

- ../u-boot-2010.06-2010R1-RC2/include/configs/bf518-ipphone.h

Deve ser inserida a seção de configuração da interface de rede. Esta seção deve ser implementada como segue.

```
#if !defined(__ADSPBF512__) && !defined(__ADSPBF514__)
#define ADI_CMDS_NETWORK      1
#define CONFIG_BFIN_MAC
#define CONFIG_NETCONSOLE     1
#define CONFIG_NET_MULTI      1
#endif
#define CONFIG_HOSTNAME       bf518-ipphone
#define CONFIG_PHY_ADDR       1
#define CONFIG_ETHADDR  02:80:ad:20:31:e8 /* endereço MAC */
```

O endereço do PHY ethernet é definido em hardware. Nesta situação foi utilizado sua configuração padrão, o que está de acordo com os sinais [PHYAD4..PHYAD0] = [00001]. Estes sinais estão multiplexados com o barramento RX do PHY ethernet.

- ../u-boot-2010.06-2010R1-RC2/board/bf518-ipphone/bf518-ipphone.c

Após a função de apresentação da placa, deve ser inserida a função de inicialização da interface ethernet do Blackfin de acordo com o que se segue.

```
#if defined(CONFIG_BFIN_MAC)
int board_eth_init(bd_t *bis)
{
      return bfin_EMAC_initialize(bis);
}
#endif
```

Após realizar essas alterações, uma nova imagem do U-Boot deve ser gerada. Essa imagem deve ser gravada na memória flash através do processo descrito na **Seção 4.2 – Suporte a Memória Flash e Persistência do Boot**.

5. uClinux

O uClinux é uma distribuição Linux adaptada para sistemas embarcados. Esse sitema operacional é adequdo em aplicações contendo processadores sem unidade de gerenciamento de memória. Assim, suas bibliotecas são menores e as chamadas ao sistema são semelhantes as da distribuição Linux utilizada nas estações de trabalho, provendo um sitema estável, confiável e com escalabilidade adequada para sistemas embarcados.

Esta seção contém todos os detalhes relacionados ao uso do uClinux, cuja distribuição utilizada é a 2011R1-RC3. Nesta seção, o diretório de trabalho do uClinux será denotado por uclinuxdir/.

5.1. MENU DE CONFIGURAÇÃO

O menu de configuração consiste em uma interface que permite ao desenvolvedor selecionar as bibliotecas e módulos que deverão ser considerados durante a compilação do uClinux. Nesta interface de configuração, podem ser escolhidas diversas opções que estão relacionadas as configurações do sistema. O menu de configuração é iniciado após a execução do comando `make menuconfig` no diretório de trabalho do uClinux.

5.2. CRIAÇÃO DO PERFIL DO SISTEMA

Na seção `Vendor Product Selection` do menu de configuração é feita a seleção do fabricante e da plataforma. No início do projeto é possível utilizar os artefatos relacionados a uma outra placa de referência baseada em um processador semelhante. Ainda assim, em algum momento, é necessário que seja criado um perfil particular para o sistema em desenvolvimento. Para tanto, deve ser realizado o seguinte procedimento:

1. Criar as seguintes pastas:
   - uclinuxdir/vendors/DETI-UFC
   - uclinuxdir/vendors/DETI-UFC/BF518-IPPHONE

- uclinuxdir/vendors/DETI-UFC/common

2. Copiar o conteúdo de uclinuxdir/vendors/AnalogDevices/BF518F-EZBRD para uclinuxdir/vendors/DETI-UFC/BF518-IPPHONE.
3. Copiar o conteúdo de uclinuxdir/vendors/AnalogDevices/common para uclinuxdir/vendors/DETI-UFC/common.
4. Criar o arquivo bf518-ipphone.c em uclinuxdir/linux-2.6.x/arch/blackfin/mach-bf518/boards/, baseado no arquivo ezbrd.c localizado neste mesmo diretório.
5. Alterar os arquivos Kconfig e Makefile em uclinuxdir/linux-2.6.x/arch/blackfin/mach-bf518/boards/.

Após esse procedimento o perfil da placa será visível a partir do menu de configuração. Antes da compilação do uClinux deve ser feita a seleção do fabricante e da plataforma no menu de configuração. Além disso, é necessário que o sistema seja configurado na seção `Customize Kernel Settings`, onde é feita a customização do kernel. Na seção `Customize Kernel Settings > Blackfin Processor Options` deve ser escolhido o valor correto para `Silicon Rev` e no campo `System Type` deve ser selecionado o sistema `Blackfin IP Phone`.

Após realizar todas essas alterações uma imagem do uClinux pode finalmente ser obtida a partir da execução do comando `make`.

5.3. CONFIGURAÇÃO DO SINAL DE RESET

O reset da placa pode ser realizado através do pressionamento do botão de reset designado por S100. Este sinal de reset é condicionado em hardware e é direcionado apenas para a entrada de reset do processador. Deste modo, o reset de todos os outros dispositivos deve ser realizado logo após o procedimento de boot. O pino PG6 do processador é configurado como um GPIO de saída, proporcionando o sinal de reset para os outros dispositivos da placa como PHY ethernet e codec de áudio, dentre outros.

A configuração do pino PG6, que atuará logo após o procedimento de boot é realizada utilizando as rotinas de manipulação de GPIOs do uClinux. Para tanto, o seguinte trecho de código deve ser adicionado ao final do script de inicialização rc, localizado em uclinuxdir/vendors/DETI-UFC/common/.

```
#
# RESET INITIALIZATION
# Resets all devices that can be reseted by Blackfin Processor through PG6
# (reset output), available on pin 34 from ADSP-BF518 (LQFP-176). This is the
# first software reset. After that, this I/O is released by an unexport
# operation for future software reset routines. See gpio.h on path
# "/linux-2.6.x/arch/blackfin/mach-bf518/include/mach" for GPIO_PG6 definition.
#
echo 22 > /sys/class/gpio/export

cont = 0
if [ $cont=100 ] ; then
      cont=$(($cont+1));
fi

echo high > /sys/class/gpio/gpio22/direction
echo 22 > /sys/class/gpio/unexport
```

5.4. APLICAÇÕES DE TESTE

Todos os dispositivos da placa foram testados separadamente. Para tanto, foi necessária a criação de aplicações de teste, com o intuito de desenvolver uma camada secundária entre a aplicação e os drivers desses dispositivos. Para que sejam executadas, as aplicações de teste também devem ser habilitadas nos menus de configuração, bem como os drivers dos dispositivos correspondentes. Os códigos-fonte de todas as aplicações de teste encontram-se em uclinuxdir/user/blackfin_ip_phone-test.

Para que as aplicações de teste sejam facilmente incluídas ou retiradas no processo de compilação as mesmas foram cadastradas na aba `Blackfin IP Phone test programs`, contida na seção de customização de aplicações do menu de configuração. O cadastramento dessas aplicações requer que sejam criados ou alterados os arquivos relacionadas abaixo, onde o cadastramento da aplicação `adp5588_test` é considerado como exemplo.

- uclinuxdir/user/Kconfig.local

```
      menu "Blackfin IP Phone test programs"
```

```
config USER_BLACKFIN_IP_PHONE_ADP5588_TEST
        bool "ADP5588 keypad and gpi capabilities test program"
        help
          This is the ADP5588 keypad controller capabilities test
          program for Blackfin IP Phone board. This test is based
          on event_test.
endmenu
```

- uclinuxdir/user/Makefile.local

```
dir_y += blkfin-apps
dir_y += blkfin-test
dir_y += blkfin-drvs
dir_y += blackfin_ip_phone-test
```

- uclinuxdir/user/blackfin_ip_phone-test/Makefile

```
dir_y =
dir_n =
dir_  =
dir_all = $(dir_y) $(dir_n) $(dir_)

dir_$(CONFIG_USER_BLACKFIN_IP_PHONE_AD73311_TEST)  += ad73311_test
dir_$(CONFIG_USER_BLACKFIN_IP_PHONE_ADP5588_TEST)  += adp5588_test
dir_$(CONFIG_USER_BLACKFIN_IP_PHONE_MAIN_APP)      += blackfin_ip_phone
dir_$(CONFIG_USER_BLACKFIN_IP_PHONE_LCD_TEST)      += lcd_test

all romfs:
  for i in $(dir_y)   ; do $(MAKE) -C $$i $@ || exit $$? ; done

clean:
  for i in $(dir_all) ; do $(MAKE) -C $$i $@ || exit $$? ; done

.PHONY: all clean romfs
```

- uclinuxdir/user/blackfin_ip_phone-test/adp5588_test/Makefile

```
CFLAGS ?= -g -O2
CFLAGS += -Wall
FLTFLAGS += -s 0x2000

all: adp5588_test

romfs:
  $(ROMFSINST) /bin/adp5588_test

clean:
  rm -f adp5588_test *.elf *.gdb *.o

.PHONY: all clean romfs
```

5.4.1. EXPANSOR DE I/Os

A aplicação de teste adp5588_test é baseada na aplicação event_test, onde eventos são escritos no arquivo `/dev/input/event0` após o pressionamento dos botões associados ao expansor de I/Os. Essa aplicação foi posteriormente integrada a aplicação principal, onde uma máquina de estados é sensível aos eventos gerados pelo expansor de I/Os.

Devido a constantes modificações dos drivers da interface I2C e do dispositivo em questão, foi necessária a atualização dos arquivos indicados abaixo para que a aplicação funcionasse adequadamente.

- uclinuxdir/linux-2.6.x/drivers/input/keyboard/adp5588-keys.c
- uclinuxdir/linux-2.6.x/include/linux/i2c/adp5588.h

Para que a aplicação seja executada de modo apropriado, os drivers dos dispositivos devem ser habilitados no menu de configuração, cujos caminhos de configuração estão relacionados abaixo.

- "Event interface" em "Kernel Settings > Device Drivers > Input device support > Generic input layer"
- "ADP5588/87 I2C QWERTY Keypad and IO Expander" em "Kernel Settings > Device Drivers > Input device support > Generic input layer > Keyboards"

- "/sys/class/gpio/ sysfs interface" em "Kernel Settings > Device Drivers > GPIO Support"

5.4.2. CODEC DE ÁUDIO

A aplicação de teste `ad73311_test` é baseada na aplicação `sport_test` do uClinux. Para que seja executado de modo apropriado o driver do AD73311 e a interface SPORT devem ser habilitados no menu de configuração, cujo caminho é indicado abaixo.

- "Blackfin SPORT driver for direct raw access" em "Kernel Settings > Device Drivers > Character devices"

Além disso, devido a incompatibilidade entre o hardware projetado e os requisitos de firmware, foram necessárias as seguintes alteração no hardware:

- Para suporte ao sinal `SPORT_ENABLE`. Esse sinal do codec foi associado ao pino PG4 do processador (ver uclinuxdir/linux-2.6.x/arch/blackfin/mach-bf518/include/mach/gpio.h), acessível através do *test point* TP301. Para tanto, é necessária a retirada do resistor R206.
- O resistor R404 de interligação entre os sinais de sincronismo de *frame* de transmissão e recepção deve ser montado.

5.4.3. LCD

O dispositivo LCD utiliza a interface I2C, cujo driver deve ser habilitado no menu de configuração para que funcione adequadamente. Desse modo, foram desenvolvidas funções de leitura e escrita do *file descriptor* associado ao dispositivo, além das funções de controle do LCD, como apagamento de tela, alteração de brilho e contraste, por exemplo. De posse dessas funções, foi desenvolvida a aplicação de teste `lcd_test`, baseado na apliação `twi_lcd-test` do uClinux, para teste dos dispositivos e validação do driver.

6. APLICAÇÃO PRINCIPAL

A aplicação principal consiste em um telefone IP. Essa aplicação é baseada no projeto Liblinphone (www.linphone.org), uma biblioteca para desenvolvimento de aplicações VoIP. Segue abaixo algumas das principais características da aplicação principal.

- Interface de usuário amigável.
- Armazenamento de lista de contatos.
- Armazenamento de histórico de chamadas (perdidas, recebidas e realizadas).
- Informações de data e hora podem ser atualizadas automaticamente ou pelo usuário.
- Funciona com qualquer servidor SIP (Asterisk, por exemplo).
- Microfone e auto-falante em hardware (opcionalmente pode ser utilizado *headset*).
- Configurações de conta e rede podem ser estabelecidas diretamente no telefone.

A **Figura 6.1** esquematiza as principais partes de um sistema VoIP. A criação, manutenção e término da conexão entre os nós de comunicação são realizados pelo protocolo de sinalização. Com o intuito de reduzir os requerimentos de largura de banda da rede, o áudio é codificado antes da transmissão e decodificado na recepção através de vários codecs padrões. Os sinais de voz codificados são transmitidos através da rede governados por um ou mais protocolos de transporte.

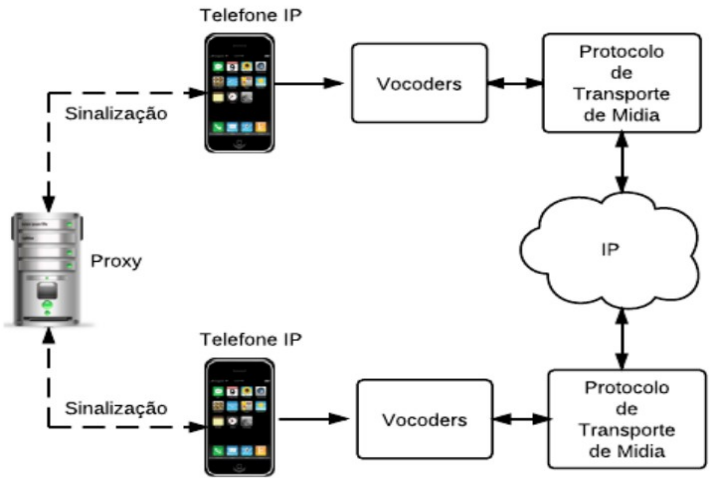

Figura 6.1 - Cenário de uso de um sistema VoIP.

Um conjunto de bibliotecas e artefatos disponíveis de modo livre para a comunidade de desenvolvedores foi considerado durante a elaboração da arquitetura de firmware da aplicação, cujas unidades básicas estão representados na **Figura 6.2**. Desse modo, baseando-se em projetos livres razoavelmente maduros e bem documentados, foi possível elaborar a aplicação principal através da integração entre todas essas unidades de desenvolvimento e através da implementação da interface com o usuário, adaptada aos dispositivos disponíveis em hardware.

Figura 6.2 - Arquitetura da aplicação.

As camadas *Interface com o Usuário* e *IPphone* foram implementadas em tempo de projeto, bem como a integração entre todas essas unidades. Abaixo, são apresentadas as características das unidades que compõem a arquitetura da aplicação.

- *Interface com o Usuário*

  No nível mais alto da arquitetura está a camada de software responsável pela interface com o usuário. A sua implementação é baseada em uma máquina de estados finitos com eventos gerados tanto pelo usuário, através do teclado e botões de funções do telefone, quanto pela camada *IPphone*, responsável pelos eventos relacionados à telefonia.

- *IPphone*

  Nesta camada, são gerados todos os eventos telefônicos utilizados pelo nível acima, como “Usuário Ocupado”, “Chamada em Curso”, entre outros. O gerenciamento das listas de chamadas, a manutenção da lista de contatos e a interface com a camada *Liblinphone* são

realizados pelas funções disponibilizadas pela camada *IPphone*.

- *Liblinphone*

  Liblinphone é uma biblioteca que implementa todas as funcionalidades de telefonia IP, como estabelecimento de sessão, gerenciamento de chamadas e proxy, controle de parâmetros de mídia e rede dentre outros. Suas funções são de alto nível e fazem uso de outros componentes de software como *oRTP*, *eXosip2* e *mediastreamer2*.

- *eXosip2*

  A aplicação proposta é um telefone VoIP compatível com o protocolo SIP. O protocolo de controle de sessão SIP objetiva estabelecer a presença e localização de usuários, assim como configuração, modificação e término de sessões. SIP é utilizado em conjunto com um protocolo de controle de mídia denominado SDP, que negocia parâmetros de inicialização de mídia, como tipo de codec e endereço da porta de destino. *eXosip2* é uma biblioteca de agente usuário SIP que oculta a complexidade de utilizar o protocolo SIP para estabelecer sessões multimídia. Essa biblioteca é utilizada por *Liblinphone*.

- *oRTP*

  Após o estabelecimento e configuração da sessão, a mídia pode fluir entre os nós da rede através de algum protocolo de transporte, tais como TCP ou UDP. As redes IP prestam serviços de melhor esforço, portanto não há garantias quanto ao atraso e sua variação, o que compromete a qualidade das transmissões multimídia de tempo-real. Dessa forma, deve ser utilizado algum protocolo de transporte de mídia na camada de aplicação para garantir a qualidade do serviço. O protocolo RTP sobre UDP é largamente utilizado para transporte de áudio e vídeo em aplicações de tempo-real. A biblioteca *oRTP*, que implementa a pilha RTP, é utilizada pela API *Liblinphone*.

- *MediaStreamer2*

Todo o recebimento e envio de fluxos de áudio, incluindo captura, codificação e decodificação, é realizado pela biblioteca *MediaStreamer2*. Outras tarefas desempenhadas por esta biblioteca incluem o cancelamento de eco, o controle automático de ganho e o envio e recebimento de pacotes RTP. Os recursos de som são acessados através de dispositivos ALSA e diversos codecs tais como Speex, G.711, GSM e iLBC são suportados.

Uma vez que a aplicação está intimamente associada ao conjunto de estados que o sistema pode assumir, o firmware foi projetado de acordo com uma máquina de estados, sensível a eventos proporcionados pelo hardware ou por eventos gerados por máquinas de estados secundárias.

6.1. MÁQUINA DE ESTADOS PRINCIPAL

A máquina de estados principal consiste de um conjunto de funções associadas aos diferentes estados que o sistema pode assumir. Cada função contém um conjunto de instruções que devem ser executados logo após a mudança de estado do sistema. Por exemplo, no estado `FSM_ST_IDLE` são executadas instruções para atualização do dispositivo LCD com informações de data e hora, além de informações de usuário, como nome e número do telefone. O código-fonte contendo as funções associadas a cada estado encontra-se em blackfin_ip_phone/fsm. Abaixo, são indicados os possíveis estados do sistema.

```
typedef enum {
  FSM_ST_IDLE,
 FSM_ST_CALL_STATUS,
 FSM_ST_INCOMING_CALL,
  FSM_ST_DIALING,
 FSM_ST_MENU,
  FSM_ST_MENU_CONTACTS,
  FSM_ST_MENU_CALL_LOGS,
  FSM_ST_MENU_SETTINGS,
 FSM_ST_CONTACTS_LIST,
  FSM_ST_CONTACTS_EDIT,
  FSM_ST_CONTACTS_EDIT_FIELDS,
  FSM_ST_CONTACT_ADD,
  FSM_ST_CONTACT_DELETE,
```

```
  FSM_ST_CALL_LOGS_MISSED,
  FSM_ST_CALL_LOGS_RECEIVED,
  FSM_ST_CALL_LOGS_OUTGOING,
  FSM_ST_CALL_LOGS_VIEW,
 FSM_ST_RECENT_LOST_CALLS,
 FSM_ST_SETTINGS_ACCOUNT,
  FSM_ST_SETTINGS_NETWORK,
  FSM_ST_SETTINGS_DATE_TIME,
 FSM_ST_NETWORK_STATIC,
 FSM_ST_EDIT_DATE,
  FSM_ST_EDIT_TIME,
 FSM_ST_NULL
} fsm_state_t;
```

Cada função da máquina de estados está associada a uma tela do display. O documento fsm_screens.pdf, que pode ser encontrado no repositório de projeto, contém um diagrama com o conteúdo de todas as telas, o que é representado na **Figura 6.3**. Além disso, é possível verificar quais os estados adjacentes e as possíveis opções de cada tela.

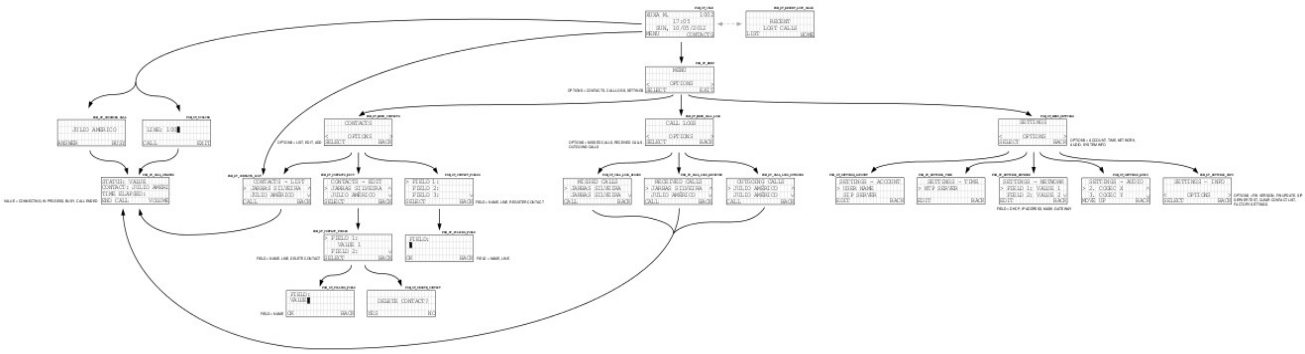

Figura 6.3 - Diagrama de telas.

6.2. EVENTOS

A máquina de estados principal é sensível a um conjunto de eventos proporcionados pelos dispositivos que constituem o hardware e pelas máquinas de estados secundárias. Por exemplo, o aperto de um botão ligado ao expansor de I/Os ocasionará a geração de um evento que será armazenado em uma fila de eventos. A fila de eventos é responsável por administrar todos os eventos listados abaixo,

mantendo a sequência de processamento pela máquina de estados de acordo com um buffer FIFO.

```
typedef enum {
  FSM_EVNT_KEYPAD_1,
  FSM_EVNT_KEYPAD_2,
  FSM_EVNT_KEYPAD_3,
  FSM_EVNT_KEYPAD_4,
  FSM_EVNT_KEYPAD_5,
  FSM_EVNT_KEYPAD_6,
  FSM_EVNT_KEYPAD_7,
  FSM_EVNT_KEYPAD_8,
  FSM_EVNT_KEYPAD_9,
  FSM_EVNT_KEYPAD_ASTERISK,
  FSM_EVNT_KEYPAD_0,
  FSM_EVNT_KEYPAD_SHARP,
 FSM_EVNT_NAVSWITCH_LEFT,
  FSM_EVNT_NAVSWITCH_RIGHT,
  FSM_EVNT_NAVSWITCH_DOWN,
  FSM_EVNT_NAVSWITCH_UP,
  FSM_EVNT_NAVSWITCH_SELECT,
 FSM_EVNT_GPBUTTON_RIGHT,
  FSM_EVNT_GPBUTTON_LEFT,
 FSM_EVNT_QUEUE_IS_EMPTY,

  FSM_EVNT_CALL_IN_INVITE,
  FSM_EVNT_USER_BUSY,
  FSM_EVNT_USER_NO_WANT_DISTURBED,
  FSM_EVNT_COULD_NOT_REACH_DESTINATION,
  FSM_EVNT_CALL_END,
  FSM_EVNT_USER_UNAVAILABLE,
  FSM_EVNT_USER_CANNOT_FOUND,
  FSM_EVNT_CALL_DECLINED,
  FSM_EVNT_REG_FAILED,
  FSM_EVNT_CALL_OUT_CONNECTED,
  FSM_EVNT_CALL_IN_CONNECTED,
 FSM_EVNT_LINPHONE_CALL_FAIL,
 FSM_EVNT_NULL
} fsm_evnt_t;
```

De tempos em tempos é checado se há algum evento diferente de `FSM_EVNT_NULL` na fila de eventos. Caso haja, esse evento servirá como parâmetro de entrada para o funcionamento da máquina de estados, que executará a função associada ao estado corrente. Dependendo do evento e do estado, ao fim da execução da função um novo estado deverá ser atribuído a máquina, que entrará em repouso, aguardando a geração de um novo evento.

6.2.1. EVENTOS GERADOS POR DISPOSITIVOS

O conjunto de eventos é formado por eventos diretamente relacionados aos dispositivos (aperto de botões, por exemplo) e eventos gerados pela camada *IPphone*. A maioria dos eventos gerados por dispositivos é proveniente dos botões ligados ao expansor de I/Os. Desse modo, foi necessário integrar a aplicação de teste `adp5588_test` a aplicação principal de modo que os eventos escritos no arquivo `/dev/input/event0` são lidos constantemente e associados a eventos da máquina de estados principal através da fila de eventos. Desse modo, cada botão foi associado a um evento e a fila de eventos do expansor de I/Os é administrada dinamicamente pela camada secundária entre o driver do dispositivo e a aplicação principal. Os códigos-fonte relacionados a essa camada secundária estão contidos em ../blackfin_ip_phone/drivers/event.

6.2.2. EVENTOS GERADOS PELA CAMADA IPPHONE

Os eventos da camada *IPphone* são gerados pela máquina de estados secundária administrada pela camada *Liblinphone*. Esses eventos estão relacionados ao uso do telefone. Por exemplo, em qualquer estado do sistema, ao receber uma ligação, o evento `FSM_EVNT_CALL_IN_INVITE` será gerado e instruções serão executadas com o objetivo de alterar o estado do sistema para `FSM_ST_INCOMING_CALL`, onde o usuário poderá decidir se a ligação será atendida ou rejeitada.

6.3. FORMATAÇÃO E PARTICIONAMENTO DE MEMÓRIAS NÃO-VOLÁTEIS

A memória S25FL064P tem capacidade de 8 MB e é organizada em 128 setores de 64 kB. Os

dois primeiros setores da memória, a partir do endereço 0x0, são divididos em 16 subsetores de 4 kB, cada. A versão do U-Boot de 2010 não suporta o apagamento de subsetores. Por esse motivo, é necessário gravar suas variáveis de ambiente no terceiro setor, que é utilizado exclusivamente para este propósito. Segue abaixo a organização dos setores da memória flash.

- 1o e 2o setores: U-Boot (parte inicial)
- 3o setor: variáveis de ambiente do U-Boot
- 4o setor: U-Boot (parte final)
- 5o ao 124o setor: uClinux
- 125o ao 128o setor: parâmetros e configurações da aplicação principal

Para contemplar esse particionamento, é necessário que os tamanhos de particionamento sejam enumerados no arquivo de declaração dos dispositivos da placa, ../blackfin-linux-dist/linux-2.6.x/arch/blackfin/mach-bf518/boards/bf518-ipphone.c, o que é representado abaixo.

```
#if defined(CONFIG_MTD_M25P80) || defined(CONFIG_MTD_M25P80_MODULE)
static struct mtd_partition bfin_spi_flash_partitions[] = {
  {
    .name       = "bootloader(nor)",
    .size       = 0x40000,
    .offset     = 0,
  }, {
    .name       = "linux kernel(nor)",
    .size       = 0x6C0000,
    .offset     = MTDPART_OFS_APPEND,
  }, {
    .name       = "jffs2(nor)",
    .size       = 0x100000,
    .offset     = MTDPART_OFS_APPEND,
  }
};

static struct flash_platform_data bfin_spi_flash_data = {
  .name = "m25p80",
  .parts = bfin_spi_flash_partitions,
  .nr_parts = ARRAY_SIZE(bfin_spi_flash_partitions),
```

```
  .type = "s25sl064a",
};

/* SPI flash chip (m25p64) */
static struct bfin5xx_spi_chip spi_flash_chip_info = {
  .enable_dma = 0,         /* use dma transfer with this chip*/
  .bits_per_word = 8,
};
#endif
```

Para criar a imagem do sistema de arquivos JFFS2 no host, o linha de comando abaixo deve ser executada utilizando a aplicação mkfs.jffs2.

```
$ ./mkfs.jffs2 -v -r config -p0x100000 -s 256 -e 64 -o config.jffs2
```

Procedimento semelhante deve ser feito para o cartão de memória, que armazenará informações de uso da aplicação, tais como agenda telefônica e histórico de chamadas (realizadas, perdidas e recebidas). O particionamento e atribuição do tipo de sistema de arquivos do cartão de memória deve ser feito no host utilizando a aplicação fdisk. Para o cartão de memória a partição a ser utilizada é a mmcblk0p1 com sistema de arquivos do tipo ext3/ext4. Essa partição deve ter tamanho suficiente para armazenamento de informações como lista de contatos e registros de chamadas.

Para montar os dispositivos de armazenamento não-voláteis as linhas de comando abaixo devem ser executados logo após a inicialização do sistema. Para tanto, essas linhas de comando devem ser adicionadas ao arquivo ../blackfin-linux-dist/vendors/DETI-UFC/common/rc.

```
$ mount -t jffs2 /dev/mtdblock2 /etc/config/
$ mount /dev/mmcblk0p1 /mnt/
```

6.4. GRAVAÇÃO DA IMAGEM DO uClinux NA MEMÓRIA FLASH

De posse da imagem do uClinux já contendo a aplicação principal, realizar o procedimento descrito abaixo a partir do bootloader.

1. Copiar a imagem do uClinux para a RAM.

```
bfin> tftpboot $(loadaddr) $(ramfile);run ramargs;run addip;
```

Exemplo:

```
bfin> tftpboot 0x1000000 uImage; run ramargs; run addip;
```

2. Identificar o dispositivo SPI.

```
bfin> sf probe 2;
```

3. Copiar o arquivo da memória RAM para a memória flash.

```
bfin> sf erase 0xOFFSET 0xLENGHT; sf write $(loadaddr) 0xOFFSET(FLASH) 0xLENGHT
```

Exemplo:

```
bfin> sf erase 0x40000 0x7C0000; sf write 0x1000000 0x40000 0x48094A;
```

Após realizar esse procedimento, o sistema será capaz de executar a imagem do uClinux contida na memória flash imediatamente após sua inicialização, não sendo necessária a carga dessa imagem via interface de rede, o que deve ser feito apenas durante a fase de desenvolvimento do projeto. Portanto, esse é um procedimento que deve ser executado somente após a conclusão da aplicação principal, uma vez que, a partir de então, o sistema será configurado para realizar o processo de boot a partir da memória. Nesta situação, a imagem contida na memória flash será descomprimida na memória RAM para, então, ser executada, o que é feito através do comando `bootm`.

Para testar se a gravação da imagem do uClinux foi realizada com sucesso, executar os seguintes comandos a partir do bootloader.

1. Copiar a imagem que está na flash para a RAM.

```
bfin> sf read $(loadaddr) 0xOFFSET(FLASH) 0xLENGHT
```

Exemplo:

```
bfin> sf read 0x1000000 0x40000 0x48094A;
```

2. Realizar o boot usando a imagem contida na RAM.

```
bfin> bootm;
```

Os procedimentos citados acima exigem a execução de cada comando pelo desenvolvedor. Desse modo, é interessante o uso de um script para gravação da imagem do uClinux, sendo necessário apenas que sejam configurados o tamaho da imagem e informações da memória flash. O script em questão será executado e armazenado pelo u-Boot. Segue abaixo os comandos para estabelecer esse script a partir das variáveis de ambiente do bootloader.

```
bfin> setenv uclinuxoffset 0x40000
bfin> setenv uclinuxsize 0x51d0c2
bfin> setenv flash_top_addr 0x7C0000
bfin> setenv uclinuxsave_part_1 'tftpboot $(loadaddr) $(ramfile); run ramargs; run
addip; sf probe 2;'
bfin> setenv uclinuxsave_part_2 'sf erase $(uclinuxoffset) $(flash_top_addr); sf
write $(loadaddr) $(uclinuxoffset) $(uclinuxsize);'
bfin> setenv uclinuxsave 'run uclinuxsave_part_1; run uclinuxsave_part_2; saveenv;'
```

O valor default de `bootcmd` é `run ramboot`, o que corresponde a realizar o boot a partir da imagem do uClinux proveniente da interface de rede. Para substituir o comando de boot atual deve-se sobrescrever a variável de ambiente `bootcmd`, como descrito a seguir.

```
bfin> setenv uclinuxboot 'sf probe 2; sf read $(loadaddr) $(uclinuxoffset) $
(uclinuxsize); bootm;'
bfin> setenv bootcmd run uclinuxboot;
bfin> saveenv
```

7. GESTÃO DE ARTEFATOS

Este projeto foi realizado por uma equipe constituída por três desenvolvedores com competências e habilidades distintas. Isso é bastante comum quando se trata de projetos que envolvem algum tipo de inovação tecnológica, onde a complementaridade das competências é algo imprescindível para seu sucesso. Desse modo, os desenvolvedores podem trabalhar de modo independente, permitindo uma diminuição do tempo de desenvolvimento, bem como a modularização da arquitetura do projeto. O uso de uma ferramenta de controle de versão em conjunto com um repositório de projeto é o que permite a realização das atividades de modo dinâmico. A configuração dessa ferramenta e os métodos de acesso ao repositório serão tratados nessa seção.

7.1. REPOSITÓRIO E FERRAMENTA DE CONTROLE DE VERSÃO

Em projetos de longa duração é fundamental que haja um controle das versões intermediárias de todos os artefatos. Dentre os benefícios desse controle de versões encontram-se a documentação do histórico do projeto, identificação dos desenvolvedores que realizaram as alterações e a facilidade de encontrar erros decorrentes de alterações defeituosas. Para atender a esses requisitos são utilizadas ferramentas de controle de versão intimamente associadas a projetos de software.

Além da funcionalidade básica de controlar as versões intermediárias de todos os artefatos do projeto, as ferramentas de controle de versão permitem a configuração de permissões de acesso a documentos restritos, lançamento de baselines de projeto, documentação das alterações realizadas em cada versão intermediária, deleção de versões intermediárias defeituosas, criação de branches, dentre outras funcionalidades.

Para utilização de uma ferramenta de controle de versão é necessário que haja uma árvore de diretórios de projeto bem definida, o que permite a fácil identificação dos artefatos e facilita a configuração da ferramenta. A definição da árvore de diretórios e o acompanhamento e suporte da ferramenta de controle de versão são realizadas pelo gestor de configuração do projeto.

Nesse projeto foi utilizada a ferramenta Subversion para controle de versão de todos os artefatos. O Subversion é um sistema de controle de versão, baseado em software livre, utilizado para gerenciar arquivos e diretórios em um cliente para sistemas de controle de versão do tipo subversion. Os arquivos e a árvore de diretórios são armazenados em um servidor, chamado repositório. O

repositório é basicamente um servidor de arquivos com a funcionalidade de armazenar versões intermediárias de projeto, o que é gerenciado pela ferramenta de controle de versão.

Foi utilizado como repositório de projeto o Google Code. O Google Code é um serviço de hospedagem de projetos open source que permite a criação de projetos de qualquer natureza de modo instantâneo com um espaço de armazenamento de até 1 GB. Além das funcionalidades básicas desejadas em um repositório, pelo fato de ser um hospedeiro de projetos open source, o Google Code possui ferramentas que auxiliam e incentivam a contribuição entre desenvolvedores na forma de discussões técnicas e pesquisas em repositórios de outros projetos. O Google Code proporciona a criação de uma página WEB para o projeto, que pode conter sua descrição detalhada, bem como documentos importantes, manuais, notas de instalação, links associados a fóruns de discussão e vídeos, por exemplo. As informações contidas nessa página de projeto são facilmente editáveis, o que pode ser feito por qualquer um dos desenvolvedores do projeto. Quando bem elaborada, é algo bastante atrativo para aqueles que desejam conhecer com detalhes o projeto ou apenas fazer uso do produto final. A **Figura 7.1** contém a página inicial do projeto, a qual pode ser acessada através do link http://code.google.com/p/blackfin-ip-phone-hw/. Também é através deste link que é possível acessar todos os artefatos do projeto. Opcionalmente, utiliza-se a ferramenta de controle de versão para fazer o download de todos os artefatos, de modo que o respositório é sensível a qualquer alteração.

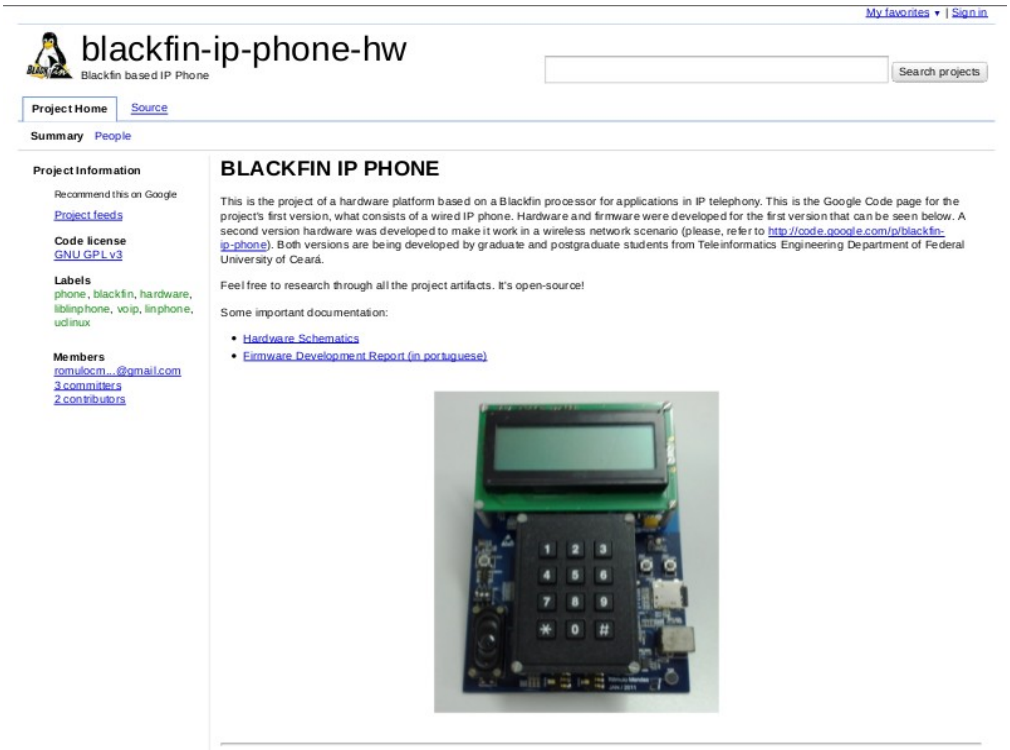

Figura 7.1 - Página do projeto.

Caso o usuário queira realizar apenas operações de leitura dos artefatos do projeto, o comando abaixo deverá ser executado.

```
$ svn checkout http://blackfin-ip-phone-hw.googlecode.com/svn/trunk/ blackfin-ip-phone-hw-read-only
```

Os desenvolvedores, que precisam realizar operações de escrita no repositório, devem possuir um nome de usuário previamente cadastrado pelo gestor de configuração do projeto para então executar o comando abaixo, a partir de onde uma senha será solicitada.

```
$ svn checkout https://blackfin-ip-phone-hw.googlecode.com/svn/trunk/ blackfin-ip-phone-hw --username USERNAME
```

O gestor de configuração é o responsável por criar os perfis de usuário e fornecer-lhes as senhas para acesso ao repositório. A geração de senhas é feita logando-se na página do projeto, na seção Checkout.

7.2 INTEGRAÇÃO ENTRE CÓDIGOS-FONTE E REPOSITÓRIO

Devido ao tamanho do código-fonte do uClinux e da sua frequência de atualização, apenas alguns artefatos foram incluídos no repositório de projeto. Os artefatos que estão sujeitos a controle de versão não existem ou são diferentes daqueles contidos no código-fonte original. Durante o desenvolvimento é necessário que o código-fonte do uClinux esteja íntegro e com a estrutura de diretórios completa. Por esse motivo, para que seja feita a integração do código-fonte original com os artefatos contidos no repositório é necessária a execução do comando abaixo.

```
$ cp -r -n /src /dest
```

Os caminhos `/src` e `/dest` representam a localização do diretório do código-fonte original e a pasta raiz do uClinux dentro do repositório, respectivamente. Desse modo, apenas os artefatos diferentes daqueles já existentes no repositório serão copiados para a pasta destino.

Esse procedimento permite que apenas uma parte do código-fonte do uClinux esteja sujeito a

controle de versão, possibilitando uma fácil atualização dos artefatos e o uso do repositório de modo econômico. Possivelmente, após essa integração, deverá ser necessária a execução do comando `make oldconfig` para atualização das opções de configuração relacionadas ao local onde o kernel será sendo compilado.