UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RENÊ AUGUSTO BENVENUTI

PLATAFORMA EMBARCADA DE TEMPO REAL INTEGRADA AO MATLAB SIMULINK

Porto Alegre
2011

RENÊ AUGUSTO BENVENUTI

PLATAFORMA EMBARCADA DE TEMPO REAL INTEGRADA AO MATLAB SIMULINK

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Engenheiro Eletricista.

ORIENTADOR: Prof. Dr. Marcelo Götz

Porto Alegre
2011

RENÊ AUGUSTO BENVENUTI

PLATAFORMA EMBARCADA DE TEMPO REAL INTEGRADA AO MATLAB SIMULINK

Este Projeto foi julgado adequado para a obtenção dos créditos da Disciplina Projeto de Diplomação do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: ____
Prof. Dr. Marcelo Götz,
Doutor pela Universität Paderborn - Paderborn, Alemanha

Banca Examinadora:

Prof. Dr. Marcelo Götz,
Doutor pela Universität Paderborn - Paderborn, Alemanha

Prof. Dr. Altamiro Amadeu Susin,
Doutor pelo Institut National Polytechnique de Grenoble - Grenoble, França

Prof. Dr. Luís Fernando Alves Pereira,
Doutor pelo Instituto Tecnológico de Aeronáutica, ITA - São José dos Campos, Brasil

Chefe do DELET: ______________________________________
Prof. Dr. Altamiro Amadeu Susin

Porto Alegre, dezembro de 2011.

DEDICATÓRIA

Aos meus pais, por todo seu apoio nesta jornada.

AGRADECIMENTOS

Agradeço à UFRGS por ser uma faculdade pública e de qualidade.

Aos professores que tornaram o desenvolvimento desta obra possível e fizeram da jornada deste curso engrandecedora.

Aos colegas por seu apoio e auxílio, amigos que levarei pelo resto da vida.

RESUMO

Ferramentas de auxílio computacional vem sendo amplamente utilizadas no projeto de controladores e na simulação do comportamento de sistemas reais. Pensando-se em criar uma forma de tornar os algoritmos desenvolvidos em ambiente de simulação funcionais, é que propõem-se o uso da *toolbox* RTW, disponível com a ferramenta Matlab/Simulink, juntamente com a aplicação desenvolvida neste trabalho utilizando as bibliotecas e API do Linux, tornar o algoritmo do controlador ou sistema sob simulação um executável em tempo real em uma plataforma embarcada. Para conseguir estes objetivos, será desenvolvido um código em linguagem C - aqui chamado de aplicação - com o qual será possível a integração e execução do código exportado pela ferramenta RTW à plataforma embarcada i.MX25 da Freescale.

Palavras-chave: Sistemas Operacionais Embarcados, Threads, Linux kernel.

ABSTRACT

Computational aid tools has been widely used for controller design and simulation of real systems. Thinking on creating a way to make the algorithms developed in CACSDS environment functional, has been proposed the use of RTW toolbox, available with the Matlab / Simulink tool, along with the application developed in this work using Linux's libraries and API, to make the algorithm executable in a real time embedded platform. To achieve these goals, it shall be developed a code in C language - called here application - with which will be possible to integrate and run the code exported by RTW tool into the Freescale i.MX25 embedded platform.

Keywords: Embedded Operational Systems, Threads, Linux kernel.

LISTA DE ILUSTRAÇÕES

Figura 1: Arquitetura do Matlab Simulink RTW 15
Figura 2: Família de plataformas i.MX 16
Figura 3: Diagrama de blocos simplificado da plataforma i.MX25 16
Figura 4: Passos necessários à aplicação proposta 18
Figura 5: Boot do sistema operacional na plataforma de desenvolvimento . . . 24
Figura 6: Visão geral da aplicação 25
Figura 7: Diagrama de blocos da aplicação 30
Figura 8: Sistema genérico para testes 32
Figura 9: Diagrama de blocos do sistema de testes 33
Figura 10: Sistema simulado em ambiente Matlab/Simulink 34
Figura 11: Aplicação rodando sem carga 35
Figura 12: Aplicação rodando com carga em baixa prioridade 35
Figura 13: Aplicação rodando com carga em alta prioridade 36

LISTA DE TABELAS

Tabela 1: Principais funcionalidades da plataforma i.MX25 17
Tabela 2: Configurações aplicadas ao sistema em teste. 32
Tabela 3: Resultados experimentais 36

LISTA DE ABREVIATURAS

API	Application Programming Interface
ATK	Advance Tool Kit
BSP	Board Suport Package
CACSDS	Computer Aided Control System Design Software
HAL	Hardware Abstraction Layer
LTIB	Linux Target Image Builder
LTS	Long Term Stable
NFS	Network File System
PDA	Personal Digital Assistant
RT	Real Time
RTW	Real Time Workshop
TFTP	Tivial File Transfer Protocol

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Visão Geral e Motivação 12
1.2 Objetivos e Metodologia 12
1.3 Organização deste Trabalho 13
2 CONTEXTO 14
2.1 Sistemas de Tempo Real 14
2.2 Matlab/Simulink - Real Time Workshop 14
2.3 Plataforma de Desenvolvimento 15
2.3.1 Família Freescale i.MX 15
2.3.2 Plataforma de Desenvolvimento i.MX25 16
3 SETUP DE DESENVOLVIMENTO 18
3.1 Pré-requisitos da Aplicação 18
3.2 Ferramentas Freescale 20
3.3 Setup do Host PC Linux 20
3.3.1 Escolha da Distribuição Linux 20
3.3.2 LTIB 21
3.3.3 Setup do Servidor TFTP 21
3.3.4 Setup NFS 22
3.4 Setup da Placa de Desenvolvimento 22
3.4.1 Bootloader 22
3.4.2 Imagem de Kernel 22
3.4.3 Iniciando o Linux na placa i.MX25 23
4 APLICAÇÃO 25
4.1 Visão Geral 25
4.2 Requisitos do Kernel 26
4.3 Criando as Threads 26
4.4 Implementando Threads Periódicas 27
4.5 Configurando a Política de Escalonamento 28
4.6 Configurando a Prioridade de Execução 29
4.7 Integrando a Aplicação ao Código Gerado pelo Matlab 29
4.8 Sistema Final 30
4.8.1 Processo Principal 30
4.8.2 Thread Matlab 31
5 TESTES 32
5.1 Abordagem e Metodologia de Testes 32
5.2 Resultados Experimentais 34

6 CONCLUSÕES 37
6.1 Trabalhos Futuros 37
REFERÊNCIAS 39
ANEXO A 41
ANEXO B 45
ANEXO C 46

1 INTRODUÇÃO

1.1 Visão Geral e Motivação

Simulação e design de modelos matemáticos para sistemas físicos vem sendo o método mais utilizado para análise comportamental de sistemas dinâmicos nos últimos 20 anos. Ferramentas de software como as assim chamadas *Computer Aided Control System Design Software* (CACSDS), são cada vez mais utilizadas e fundamentais para rápida modelagem e simulação dos sistemas de controle.

A evolução dos microprocessadores, excepcionalmente em desempenho e eficiência energética, é notável e proporciona aos engenheiros uma ampla gama de aplicações. A possibilidade de embarcar um sistema operacional em uma plataforma móvel vem sendo difundida graças a crescente demanda por aparelhos eletrônicos conhecidos por PDAs (*Personal Digital Assistant*), *tablets e smartphones*, dispositivos estes que exigem grande capacidade computacional assim como baixo consumo energético e portabilidade. A família de processadores ARM (*Advanced RISC Machine*), é atualmente a mais utilizada para estas aplicações. Desenvolvida com foco na eficiência energética, esta arquitetura hoje suporta sistemas operacionais como o Linux, Android e Windows CE, e também a maior parte dos sistemas de tempo real disponíveis (ARM, 2011).

Estas ferramentas, hoje disponíveis, são muito completas e oferecem infindáveis possibilidades de aplicações, porém, a complexidade de aprendizado destas ferramentas e a escassez de tempo para execução dos projetos por vezes limita os engenheiros. Se faz necessário a utilização de interfaces *user friendly* que possibilitem a rápida aprendizagem e execução das tarefas.

1.2 Objetivos e Metodologia

Pensando nos fatos supracitados, este trabalho tem o objetivo de integrar ambiente de desenvolvimento e simulação Matlab/Simulink com a plataforma de desenvolvimento i.MX25 da Freescale, proporcionando um ambiente de desenvolvimento e simulação de

rápido aprendizado e transparente ao usuário quanto a utilização de um sistema de tempo real. Desta forma mantem-se a vantagem da simplicidade do design e simulação em Matlab/Simulink e obtém-se um meio para tornar este algoritmo funcional em uma plataforma embarcada.

Utilizando-se o *toolbox* RTW, exporta-se o código fonte do sistema desenvolvido no Matlab/Simulink, a aplicação desenvolvida então integra este código fonte, threads e bibliotecas do Linux em um aplicativo, que executará o algoritmo em tempo real. A aplicação é então cross-compilada possibilitando a execução em ambiente embarcado do algoritmo do sistema de forma rápida para o desenvolvedor. Como plataforma alvo é utilizada a placa de desenvolvimento da Freescale i.MX25, baseada no processador ARM9, um dos processadores de alta performance mais utilizados em sistemas embarcados.

1.3 Organização deste Trabalho

Os capítulos seguintes desta obra estão estruturados da seguinte maneira:

CONTEXTO : Breve explicação dos itens essenciais ao desenvolvimento da aplicação final.

SETUP DE DESENVOLVIMENTO : Descreve as ferramentas utilizadas para compilação e o ambiente de trabalho necessário ao desenvolvimento.

APLICAÇÃO : O capítulo aplicação descreve como foi implementado o código que integra as fontes do algoritmo ao sistema de tempo real de execução.

TESTES : Descreve a metodologia utilizada para testes da aplicação assim como os resultados obtidos nestes.

CONCLUSÕES : Apresenta as conclusões tomadas sobre o projeto desenvolvido, assim como ideias para desenvolvimentos futuros.

2 CONTEXTO

2.1 Sistemas de Tempo Real

Em geral o termo “tempo real” deriva seu nome dos primórdios da simulação computacional, servindo para designar processos computacionais suficientemente rápidos a ponto de igualar as taxas de saída esperadas do sistema dinâmico sendo simulado.

Um processo é dito estar rodando em tempo real se seguir as exigências temporais impostas a ele e concluir sem nenhuma falha dentro de um tempo especificado. Um prazo é uma boa analogia ao que sistemas de tempo real tentam atingir. Basicamente, sistemas de tempo real podem ser classificados em duas categorias: *hard real time* e *soft real time* (SANKARAYOGI, 2005). O principal critério por trás desta classificação é o grau de tolerância à perda destes prazos. Em um sistema de tempo real *hard*, prazos não podem ser perdidos sob qualquer circunstância, significando a perda de algum destes prazos uma total falha do sistema, enquanto que em sistemas de tempo real *soft*, há certo grau de tolerância à perda de prazos. Para exemplificar, considerando-se um aplicativo para reprodução de vídeos, se este perder algum quadro não afetará a qualidade do vídeo, porém há um certo limite para estas perdas e para a frequência em que ocorrem. Esse tipo de sistema pode ser considerado do tipo *soft real time*. Agora se considerarmos a aplicação de sistemas de controle de aeronaves, perder dados ou não deixar de atuar podem se provar catastrófico, sendo então classificado como um sistema *hard real time*.

2.2 Matlab/Simulink - Real Time Workshop

O Matlab, cujo nome deriva de *Matrix Laboratory* (MATHWORKS, 2011), é uma poderosa ferramente de análise e simulação que provê linguagem Matlab, biblioteca de funções matemáticas, ambiente e assistentes de desenvolvimento, ambiente gráfico e interfaces para programação em diversas linguagens, como por exemplo C. O Simulink é um pacote integrado ao Matlab usado para modelar, simular e analisar sistemas contínuos, discretos e híbridos. É composto por uma interface com usuário na qual é possível

a criação de modelos por meio de “arrastar e largar” blocos com funções pré definidas.

O *Real Time Workshop* (RTW), é um pacote integrado ao Matlab/Simulink que provê geração de código C e também HDL (*Hardware Description Language*), a partir do diagrama funcional estruturado em Simulink, possibilitando rápidos desenvolvimento e prototipação do sistema. A Figura 1 ilustra o conceito de simulação e geração de código integrando Matlab/Simulink e RTW. Neste trabalho será explorada apenas a funcionalidade de exportação de código C, não sendo abordada aqui a exportação e uso do código HDL.

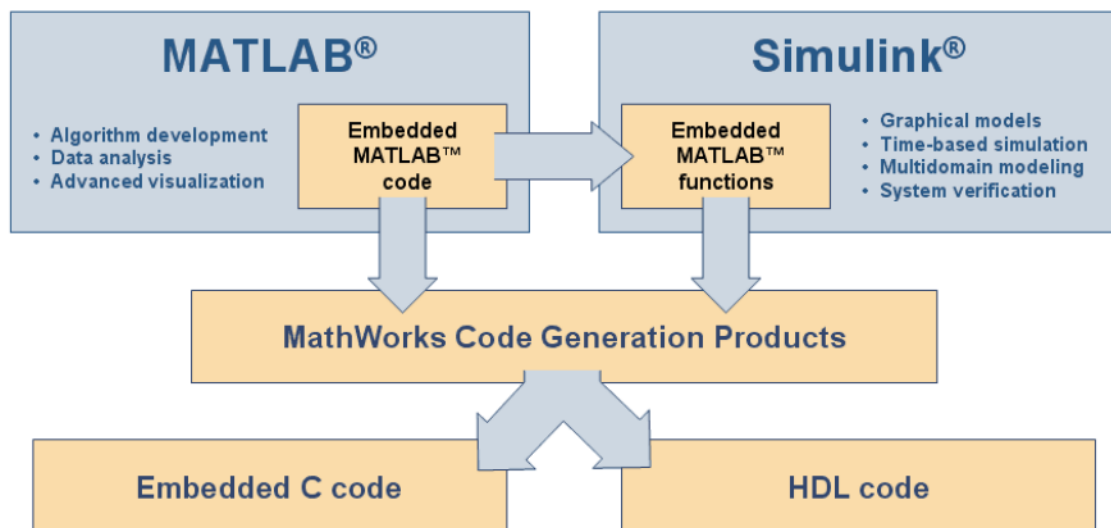

Figura 1: Arquitetura do Matlab Simulink RTW

2.3 Plataforma de Desenvolvimento

2.3.1 Família Freescale i.MX

Baseadas na arquitetura ARM, as placas de desenvolvimento i.MX da Freescale buscam atingir aplicações que vão do setor industrial e automotivo até aplicações de multimídia voltadas ao consumidor. Fortemente engajadas na ideia de performance energética, esta família de placas de desenvolvimento já atraiu uma comunidade de desenvolvedores pelo mundo (FREESCALE, 2011). A Figura 2 ilustra o portfólio de placas de desenvolvimento disponíveis da família i.MX.

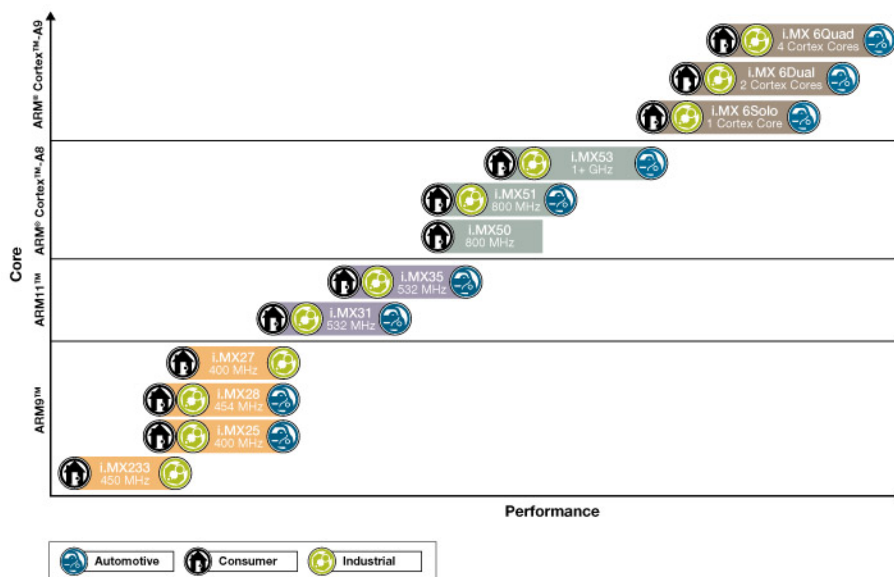

Figura 2: Família de plataformas i.MX

2.3.2 Plataforma de Desenvolvimento i.MX25

Voltadas para o setor automotivo, mas com forte apelo multimídia, esta placa oferece boa performance com custo relativamente baixo. A Figura 3 ilustra o diagrama de blocos simplificado da placa e a Tabela 1 descreve as suas principais funcionalidades.

Foi disponibilizada para o desenvolvimento deste trabalho pelo prof. Dr. Marcelo Götz uma placa Freescale i.MX25.

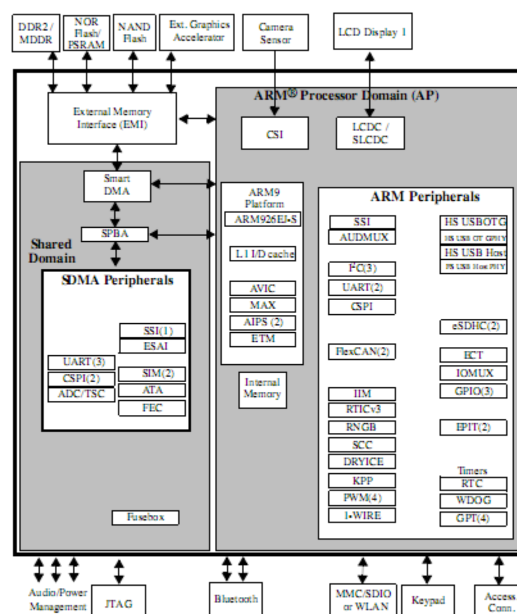

Figura 3: Diagrama de blocos simplificado da plataforma i.MX25

Funcionalidade	**Plataforma i.MX25**
Core	ARM926EJ-S
Freq. Máxima CPU	400 MHz
Memória	64 MB
Controlador LCD	Contém
TouchScreen	Contém
CAN	Contém
Ethernet	Contém
12 bit ADC	Contém
SD/SDIO/MMC	Contém
I2C	Contém
SSI/I2S	Contém
UART	Contém

Tabela 1: Principais funcionalidades da plataforma i.MX25

3 SETUP DE DESENVOLVIMENTO

3.1 Pré-requisitos da Aplicação

Buscando-se criar a integração entre Matlab/Simulink com a plataforma embarcada, projetou-se uma sequência de objetivos parciais que são necessários à construção do projeto como um todo. Estes objetivos parciais são descritos nos itens a seguir:

Objetivo Parcial A : Simular e Verificar o Sistema em ambiente Matlab/Simulink.

Objetivo Parcial B : Exportar código C com o auxílio do *toolbox* RTW.

Objetivo Parcial C : Anexar o algoritmo exportado à um sistema periódico de tempo real.

Objetivo Parcial D : Cross-compilar o código para a arquitetura alvo (plataforma i.MX25 da Freescale).

Objetivo Parcial E : Executar o algoritmo na plataforma embarcada.

O diagrama de blocos da Figura 4 ilustra a abordagem.

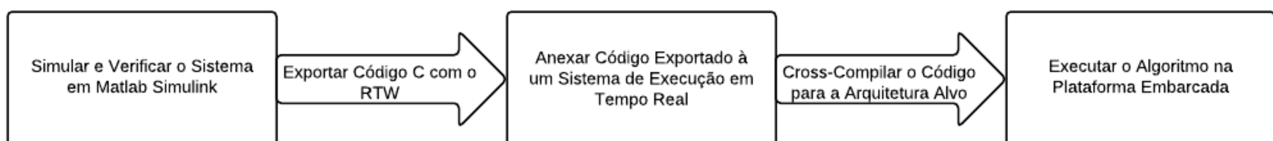

Figura 4: Passos necessários à aplicação proposta

Assim sendo, como pré-requisitos para que se tenha um ambiente de desenvolvimento funcional, são necessárias análises e também preparações para cada objetivo parcial, ou seja, para o objetivo parcial:

A : É necessário compreender-se o funcionamento da ferramenta Matlab/Simulink e dos sistemas que estão sob simulação.

B : É preciso entender-se a operação e configuração do *toolbox* RTW, assim como analisar-se o código C que é por ele exportado.

C : É necessário compreender-se o conceito de tempo real e as alternativas de implementação de tal sistema na plataforma alvo do projeto.

D : Devemos possuir um ambiente de desenvolvimento (*host* PC), com todos os pacotes e bibliotecas necessárias à cross-compilação. Configurar-se variáveis de ambiente e ferramentas para que funcionem corretamente.

E : Precisamos garantir um sistema operacional rodando na plataforma alvo com versão e funcionalidades de kernel que deem suporte a aplicação desenvolvida.

Não são raras as vezes em que o tempo gasto na preparação do ambiente de desenvolvimento é superior ao tempo de desenvolvimento da própria aplicação. Manuais incompletos e/ou imprecisos geram inúmeras dúvidas na hora de se estabelecer um primeiro contato com a placa de desenvolvimento.

No caso específico da plataforma i.MX25 da Freescale observou-se que a documentação é muito falha, contendo descrições não funcionais e tampouco genéricas. Um esforço de pesquisa teve de ser realizado buscando-se referências funcionais para o *setup* da aplicação.

Tendo em vista estes motivos supracitados, as próximas seções deste capítulo, juntamente com os anexos A, B e C, tem por objetivo descrever as ferramentas utilizadas e também passos funcionais para o *setup* do *host* PC Linux, usado para compilação das imagens de kernel, boot e também para geração de aplicativos para a plataforma alvo, como também para o *setup* da placa de desenvolvimento. Informações complementares podem ser encontradas na wiki da comunidade de desenvolvedores da família i.MX (WIKI, 2011).

3.2 Ferramentas Freescale

As ferramentas oferecidas pela Freescale para o desenvolvimento de aplicações na placa i.MX são:

BSP : o BSP (*Board Support Package*), é um conjunto de softwares utilizados para iniciar os dispositivos de hardware na placa e implementar as rotinas específicas da placa. O BSP não deixa de ser um HAL (*Hardware Abstraction Layer*), visto que ele é uma camada de abstração entre o kernel e o hardware específico da placa (NEELAKANDAN; LAD; RAGHAVAN, 2006).

LTIB : o LTIB (Linux Target Image Builder), é uma ferramenta que possibilita modificar, desenvolver, compilar e instalar pacotes para diversas placas da Freescale. Esta ferramenta é utilizada para fazer alterações na imagem de kernel e boot, assim como gerar o sistema de arquivos para o sistema operacional da placa. Ela também nos permite a instalação de pacotes como o VIm para edição de arquivos de texto ou ainda o GTK+ para criação de aplicações com interfaces gráficas, entre outros. O procedimento para instalação da LTIB é mostrado mais adiante neste texto (LTIB, 2011).

ATK : o ATK (Advanced Tool Kit), é uma ferramenta GUI desenvolvida para fazer o download do *bootloader* para a placa. Também é possível utiliza-lo para mudar-se o formato do sistema de arquivos gerado pela LTIB.

3.3 Setup do Host PC Linux

Abaixo seguem os passos para criar-se um *setup* funcional de um *host* PC Linux para o desenvolvedor que queira utilizar as ferramentas da Freescale.

3.3.1 Escolha da Distribuição Linux

O Linux propriamente dito é na verdade apenas o núcleo do sistema operacional. Sobre o núcleo Linux rodam um conjunto variável de softwares dependendo de seus propósitos. Este conjunto de softwares é conhecido por distribuição, ou simplesmente distro

(WIKIPEDIA, 2011). Dentre as diversas distribuições disponíveis, o Ubuntu se sobressai como sendo a mais amplamente utilizada no mundo. Baseado no Debian, o Ubuntu, conta com uma ampla comunidade ao redor do mundo e suporte online. A comunidade disponibiliza atualizações semestrais de versão além de versões LTS (*Long Term Stable*), que são versões da distribuição testadas por pelo menos 3 anos e que apresentam poucos *crushes* ou *bugs*, as quais são preferidas por empresas de pesquisa e desenvolvimento por oferecer maior confiabilidade ao sistema operacional.

Como sistema operacional para o *host* PC Linux, foi escolhido o Ubuntu 10.04 32 bits LTS , pois além de possuir as vantagens supracitadas, é o mais amplamente utilizado nos exemplos das comunidades de desenvolvimento. Pode-se obter o Ubuntu 10.04 LTS gratuitamente pelo site da comunidade Ubuntu (UBUNTU, 2011).

3.3.2 LTIB

O procedimento para instalação da LTIB pode ser bastante frustrante para o usuário menos habituado com um sistema operacional baseado em GNU/Linux. Além de os procedimentos descritos nos manuais disponibilizados pela Freescale não oferecerem um método genérico de instalação que funcione nas diversas distribuições, os *scripts* da própria LTIB devem ser alterados para funcionarem na distribuição escolhida.

Alguns passos e alterações devem ser efetuados para o correto funcionamento da LTIB no host. Pensando em facilitar a vida do desenvolvedor foi criado o roteiro de ações e *shell scripts* descritos no ANEXO A.

3.3.3 Setup do Servidor TFTP

O TFTP (*Trivial File Transfer Protocol*) é um protocolo de transferência de arquivos que utiliza a interface ethernet para realizar transferência de arquivos diversos. No contexto de desenvolvimento de aplicações para sistemas operacionais embarcados, o TFTP é utilizado para transferir as imagens de kernel e sistema de arquivos do computador host para a placa de desenvolvimento. O ANEXO B explica como deve ser feita a configuração do TFTP no *host PC*.

3.3.4 Setup NFS

O NFS (*Network File System*), na verdade não é o sistema de arquivos em si, e sim um método para montar um sistema de arquivos sobre uma rede (NEELAKANDAN; LAD; RAGHAVAN, 2006). Usando NFS podemos exportar um sistema de arquivos *desktop* (ext2 ou ext3) para a placa. O uso do NFS é interessante pois quando o desenvolvedor está *debugando* ou modificando o projeto não é prático nem recomendado (visto que a flash tem uma capacidade de escritas limitada), ficarmos gravando e desgravando a memória flash da placa de desenvolvimento. Podemos então gerar o sistema de arquivos usando a LTIB e mantê-lo no host Linux, acessando-o pela placa i.MX usando NFS.

Para a geração do sistema de arquivos e posterior acesso deste pela placa i.MX são necessários alguns procedimentos, estes descritos no ANEXO C.

3.4 Setup da Placa de Desenvolvimento

3.4.1 Bootloader

O *bootloader* ou *bootstrap loader*, é basicamente um pequeno programa que tem por função carregar dados e inicializar os processos do sistema operacional.

Utilizando-se a ferramenta ATK é possível baixar a imagem de *bootloader* gerada pela LTIB para a placa. O procedimento para isso é documentado nos manuais de operação da placa (I.MX ADVANCED TOOLKIT - ATK, 2009), e não será abordado nesta obra. Para este trabalho foi utilizada a imagem de boot do `RedBoot`, disponibilizada juntamente com a mídia provida pela Freescale.

3.4.2 Imagem de Kernel

Esta imagem é gerada utilizando-se as fontes do kernel Linux e a ferramenta LTIB. Quando compiladas, as fontes geram duas imagens de kernel diferentes: uma imagem com o nome de `uImage` utilizada pelo *bootloader* `u-boot` e uma imagem com o nome `zImage`, que é utilizada pelo `RedBoot`. Esta imagem é passada para a placa utilizando-se o TFTP, e então gravada na memória flash para que seja executada quando o sistema é iniciado.

3.4.3 Iniciando o Linux na placa i.MX25

Após gravadas a imagem de *bootloader* e a imagem de kernel, é necessário configurar os argumentos do *bootloader* (*bootargs*), para que o sistema seja inicializado da forma que se deseja. Neste trabalho foi optado por utilizar-se o sistema de arquivos por NFS, assim é necessário que seja estabelecida e configurada uma conexão ethernet entre a placa e o *host* PC. Após configurado corretamente a conexão, o seguinte comando inicializa o sistema:

exec -b 0x100000 -l 0x200000 -c "noinitrd console=tty0 console=ttymxc0 root=/dev/nfsroot rootfstype=nfsroot nfsroot=192.168.0.1:/home/engenheiro/freescale/ltib/rootfs ip=192.168.0.2"

Neste caso o sistema de arquivos será carregado por NFS e está localizado na pasta `/home/engenheiro/freescale/ltib/rootfs`, a conexão estabelecida é ponto-a-ponto onde o *host* PC possui o IP 192.168.0.1 e a plataforma está configurada com o IP 192.168.0.2.

Uma vez efetuados os comandos acima e também realizados todos os passos de configurações anteriores, a placa deve iniciar o sistema operacional e será possível à aplicação rodar sobre ele. A Figura 5 demostra a inicialização do sistema operacional na plataforma.

Figura 5: Boot do sistema operacional na plataforma de desenvolvimento

4 APLICAÇÃO

4.1 Visão Geral

A aplicação final consiste em utilizar partes do código gerado pelo *toolbox* RTW e integra-lo a um sistema periódico de execução em tempo real, de modo a tornar o modelo desenvolvido em Matlab/Simulink funcional. O Diagrama de blocos da Figura 6 ilustra o ciclo de trabalho objetivado pela aplicação.

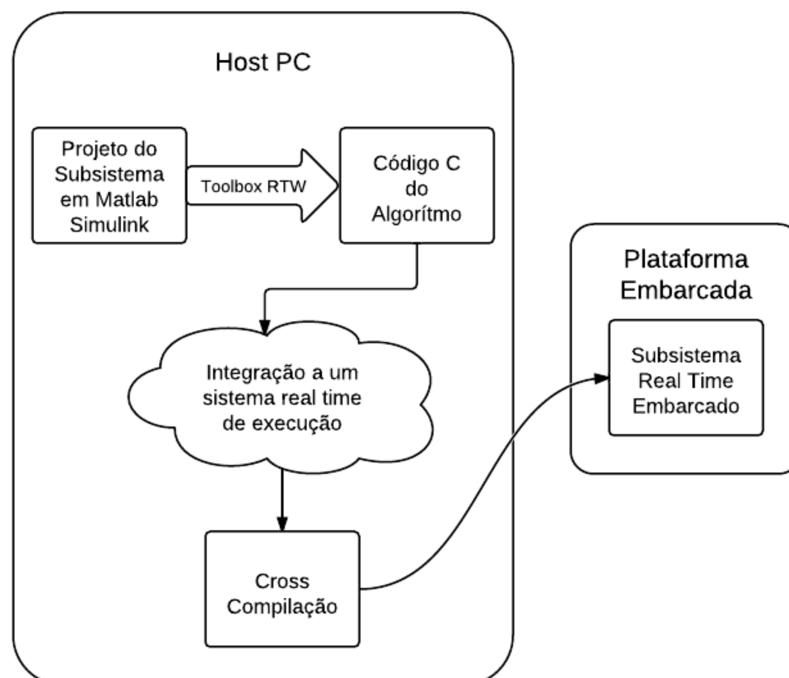

Figura 6: Visão geral da aplicação

Utilizando-se o Matlab/Simulink desenvolve-se o sistema de interesse, o qual tem o algoritmo exportado utilizando-se a *toolbox* RTW. A aplicação desenvolvida integra o algoritmo exportado a um sistema periódico de tempo real de execução, que é então cross-compilado para a arquitetura de interesse obtendo-se um binário executável em ambiente embarcado do subsistema desenvolvido e simulado em Matlab/Simulink.

Para a tarefa de tornar o sistema embarcado periódico, foi optado pela utilização da API do Linux, juntamente com a biblioteca *POSIX threads* (`libpthread`) e a biblioteca *POSIX realtime extension* (`librt`). Estas bibliotecas possuem uma API que possibilita a

criação e alteração dos atributos das threads, assim como alteração da política de escalonamento das mesmas. Mais detalhes sobre o desenvolvimento são explicitadas nas seções seguintes.

4.2 Requisitos do Kernel

Para maximizar a performance do sistema multithreads de tempo real, foram necessárias algumas alterações na configuração padrão do kernel utilizado na plataforma embarcada, de forma a torná-lo mais preemtivo e eficiente (RUSLING, 1999) (KERNEL, 2011).

High Resolution Timers : O uso de *High Resolution Timers* permite que se tenha a máxima resolução nas funções disponibilizadas pela API *POSIX timers*, cujo padrão é de 10 ms. Disponível a partir do kernel 2.6.17, esta alteração permitirá que as threads da aplicação tenham períodos de execução menores do que 10 ms.

Preemptive Patch : O patch de preempção permite diminuir-se a latência do escalonador de tarefas, ou seja, o atraso que ocorre entre a solicitação de atendimento de uma tarefa e o atendimento desta tarefa propriamente dito. Foi desenvolvido primeiramente no kernel 2.6.0, sendo o resultado desta alteração a redução no tempo total para atendimento da thread.

Timerfd System Call : Primeiramente implementado na GNU libc 2.8 e disponibilizado a partir do kernel 2.6.25, o *Timerfd System Call* possibilita que sejam utilizadas as funções do *header* `timerfd.h` (LINUX, 2007). Esta API será utilizada na aplicação para tornar as threads periódicas e possibilitar que a execução do algoritmo seja em tempo real.

4.3 Criando as Threads

Uma thread nada mais é do que uma tarefa do sistema operacional. Utilizando-se a API disponível no *header* `pthread.h` é possível criar-se dentro do processo da aplicação quantas threads forem necessárias. Para tal é utilizada a função `pthread_create()`

e passado como atributos o identificador da thread e a função que esta thread deverá executar.

Ao encerrar o processo utilizamos a função `pthread_exit()` para encerrar todas as threads criadas.

4.4 Implementando Threads Periódicas

Grande parte do trabalho da aplicação consiste em executar o algoritmo exportado pelo RTW periodicamente, garantindo-se que este período seja igual ou tão próximo quanto possível do período estabelecido na especificação e verificação do sistema em Matlab/Simulink. Uma solução rude seria a execução da rotina principal seguida por um comando de `sleep()`, porém esta solução é muito pouco confiável pois a periodicidade irá variar dependendo do tempo que o sistema levar para a execução do algoritmo.

O Linux oferece algumas interfaces com o relógio, dentre elas a que sobressai como mais precisa e recente é a interface timerfd, disponível para GNU libc 2.8 e versões de kernel 2.6.25 ou posteriores. A seguir são explicitadas as funções utilizadas do *header* `timerfd.h` para criar-se um evento periódico configurável que possa servir de referência para a thread.

Criou-se um *timer* chamando-se a função `timerfd_create` e atribuindo-se como parâmetro para a função o *POSIX clock id* `CLOCK_MONOTONIC`. Esta função retorna um *file descriptor* para o *timer*, que funciona basicamente como um arquivo no qual podemos ler eventos de relógio.

Após ser criado o *file descriptor*, utiliza-se a função `timerfd_settime()` com os seguintes atributos para que possamos passar ao *timer* as configurações necessárias para torná-lo periódico:

- *file descriptor* : É o *timer* criado anteriormente pela função `timerfd_create()` e que será a fonte de eventos;
- 0 : Parâmetro indicando que desejamos um temporizador que utilize tempo relativo e não absoluto;

- *struct itimerspec* : Estrutura contendo o valor em segundos e também em nanosegundos do período que desejamos dar ao *timer*.

Com estas configurações, o *file descriptor* criado agora fornece um evento periódico seguindo os parâmetros configurados pela função `timerfd_settime()`.

Uma vez criado e configurado o *timer*, utiliza-se a função `read()`, passando como atributo o *file descriptor*, para que o sistema aguarde o evento de *timer*.

4.5 Configurando a Política de Escalonamento

O escalonador (*scheduler*), do kernel Linux, é o elemento do responsável por dividir o tempo de processamento entre as diversas tarefas do sistema. Utilizando-se a API disponível no *header* `sched.h`, temos acesso a funções que permitem dizer ao escalonador como queremos que certa thread seja escalonada.

Inicialmente cria-se o atributo que será passado para a thread com a função `pthread_attr_init()`, depois cria-se a estrutura `sched param`, que armazenará a política de escalonamento desejada. Existem 3 possíveis políticas de escalonamento disponíveis no sistema operacional Linux, estas são:

- SCHED_OTHER : Política de escalonamento padrão. Atribuída a toda tarefa que não tenha sua política de escalonamento explicitada.

- SCHED_FIFO : Utiliza a política de escalonamento *First In First Out*.

- SCHED_RR : Utiliza a política de escalonamento *Round Robin*.

Utilizando-se a função `pthread_attr_setschedpolicy()`, configuramos a política de escalonamento desejada ao atributo da thread e por fim, utilizando-se a função `pthread_attr_setinheritsched()`, indicamos que a thread criada terá sua política de escalonamento explicitada. Assim quando for criada a thread com este atributo, ela será escalonada como solicitado.

4.6 Configurando a Prioridade de Execução

Utilizando-se a API disponibilizada no *header* `pthread.h` pode-se atribuir às threads criadas uma prioridade de execução. A prioridade funciona exatamente como o nome sugere, uma thread que tenha uma prioridade elevada será executada antes de outra thread que tenha prioridade menor. Se duas threads possuem a mesma prioridade, o escalonador do kernel disponibiliza tempos iguais de execução para as duas tarefas.

Para atribuir-se a maior prioridade possível à thread (o que é desejável para a aplicação), primeiramente obtemos o valor máximo de prioridade possível com a função `sched_get_priority max()`, e depois igualamos este valor máximo ao atributo da thread com a função `pthread_attr_setschedparam()`. Assim quando criarmos a thread com este atributo, ela terá o valor de prioridade solicitado.

4.7 Integrando a Aplicação ao Código Gerado pelo Matlab

Uma vez criada a thread e desenvolvidos meios para tornar sua execução periódica em tempo real, é necessário integrar o algoritmo que é exportado pelo RTW à aplicação. Este código que é exportado é composto por uma série de arquivos em linguagem C que efetivamente implementam o algoritmo do sistema projetado em ambiente Simulink, porém este código precisa ser anexado a um sistema que execute suas funções com a periodicidade apropriada e compilado, juntamente com a aplicação, para a plataforma de interesse.

Para execução de tal tarefa foi criado um *shell script* com função de copiar os arquivos fonte gerados pelo RTW para a pasta principal do projeto, unir este código ao código desenvolvido, executar o cross-compilador e, por fim, copiar o executável do algoritmo para o sistema de arquivos da placa. Desta forma, após desenvolver em ambiente Simulink o projeto de interesse e exportar seu código utilizando-se o RTW, é necessário apenas rodar o *script* desenvolvido para que se tenha o algoritmo do sistema coss-compilado e pronto para rodar na plataforma embarcada. Assim, conseguiu-se a automação da integração do código exportado pelo RTW ao sistema embarcado.

4.8 Sistema Final

O diagrama de blocos mostrado na Figura 7 ilustra como está organizado o código da aplicação. Em seguida é explicado o funcionamento dos blocos que compõem a aplicação.

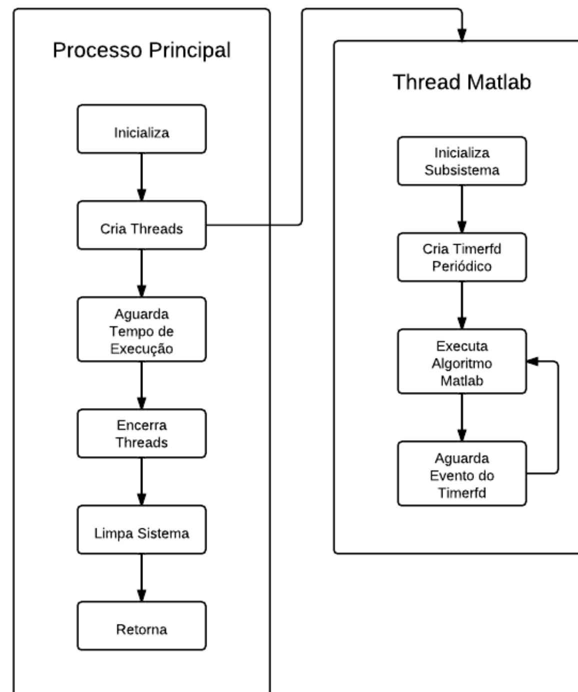

Figura 7: Diagrama de blocos da aplicação

4.8.1 Processo Principal

Inicializa : Este bloco é composto pelas funções encarregadas de inicializar as variáveis do sistema e configurar a estrutura `pthread_attr_t` e a estrutura `sched param`.

Cria Threads : Este bloco é responsável por efetivamente atribuir os parâmetros para a thread Matlab e iniciar sua execução.

Aguarda Tempo de Execução : Este bloco aguarda pelo tempo definido de execução da tarefa. Caso se deseje a execução contínua deve-se ignorar este bloco.

Encerra Threads : Encerra a execução das threads que estão rodando na aplicação. Isso é extremamente importante, pois evita que a thread principal execute o algoritmo por mais tempo do que o desejado.

Limpa Sistema : Este bloco tem a função de eliminar os atributos criados para as threads, para que não permaneçam no sistema após o encerramento da aplicação.

Retorna : Encerra o processo e as demais variáveis criadas pela aplicação.

4.8.2 Thread Matlab

Esta é a principal tarefa da aplicação, nela roda o algoritmo exportado pelo RTW.

Inicializa Subsistema : Cria variáveis necessárias e chama as funções responsáveis por inicializar o algoritmo do subsistema exportado pelo Matlab/Simulink.

Cria Timerfd Periódico : Responsável pela criação e configuração do *file descriptor* do *timer*, permitindo a execução periódica de eventos de relógio.

Executa Algorítmo Matlab : Este bloco é responsável por carregar a entrada do sistema, chamar a função que executa o algorítmo exportado pelo RTW e coletar as saídas.

Aguarda Evento do Timerfd : Este bloco lê o *file descriptor* do relógio e aguarda que outro evento apareça, evitando que o algoritmo do subsistema seja executado demasiadas vezes.

5 TESTES

5.1 Abordagem e Metodologia de Testes

Para testes da aplicação desenvolvida foi criado um subsistema genérico em ambiente Matlab/Simulink, este ilustrado na Figura 8, com o objetivo de simular um sistema real. As configurações dadas ao sistema são descritas na Tabela 2.

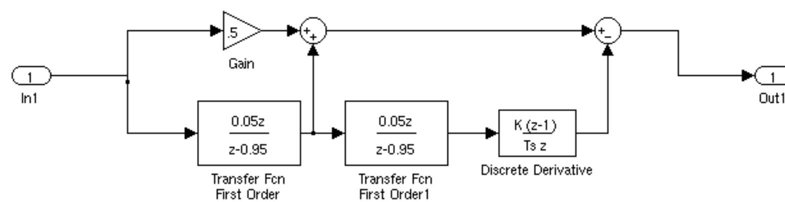

Figura 8: Sistema genérico para testes

Campo	Descrição	Valor
Step Time	Período de amostragem do sistema.	5 ms
Total Time	Tempo total de execução.	10 s

Tabela 2: Configurações aplicadas ao sistema em teste.

O código deste subsistema foi gerado utilizando-se a ferramenta RTW e incorporado à aplicação desenvolvida utilizando-se o script *maker*, como descrito no capítulo anterior.

Foi então introduzido na aplicação um mecanismo para coleta de *timestamps*, ou seja, retratos do relógio do sistema, para que fosse possível fazer a medida do intervalo entre as execuções do algoritmo exportado e então confrontá-lo com o período esperado. A saída do sistema foi armazenada em um arquivo de texto para que os dados pudessem ser armazenados e confrontados com os da simulação feita em Simulink. O diagrama de blocos da Figura 9 ilustra o sistema de testes.

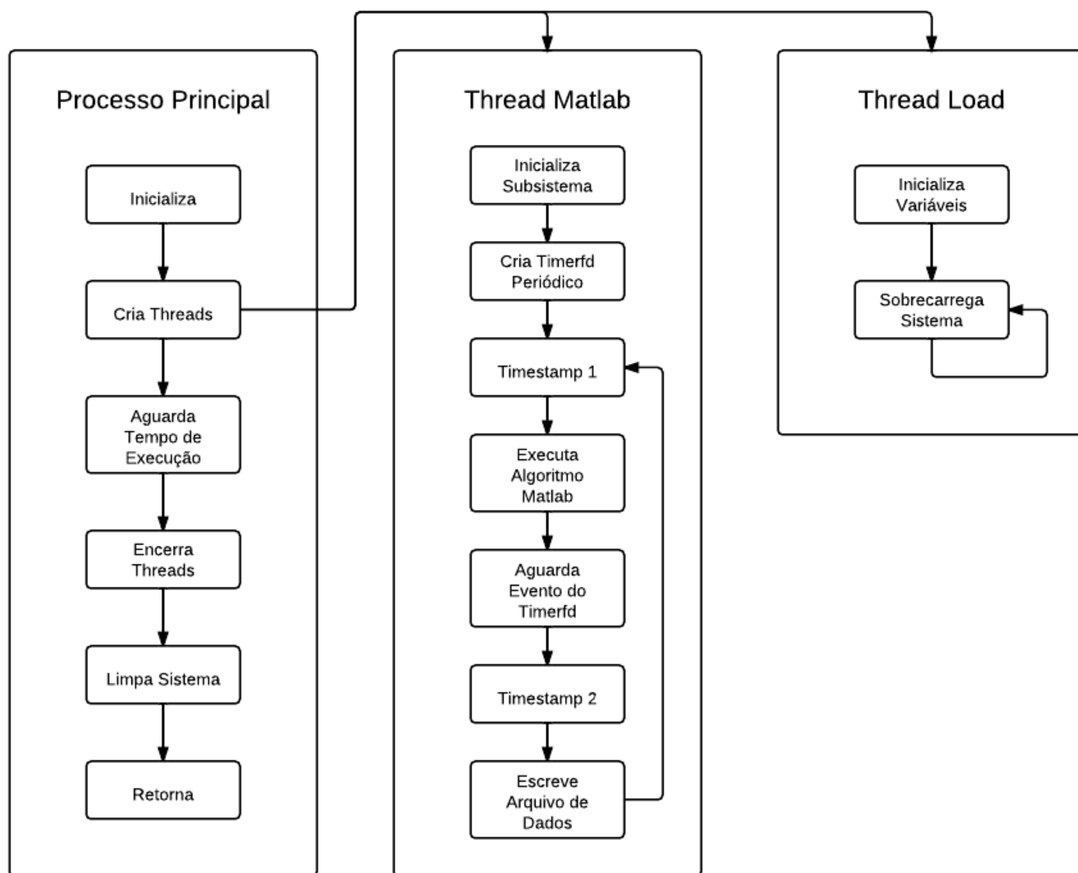

Figura 9: Diagrama de blocos do sistema de testes

Com o objetivo de verificar a manutenção das garantias temporais das threads da aplicação alvo frente a carga do sistema por outras aplicações genéricas, foram cogitadas e testadas três hipóteses de condições de operação, estas descritas a seguir:

Sistema sem carga : Hipótese que representa o sistema operacional ocioso, ou seja, apenas a thread encarregada de executar o algoritmo do Matlab (thread principal), está executando em máxima prioridade;

Sistema com carga em baixa prioridade : Representa a hipótese do sistema ocupado com operações em paralelo, não relacionadas a aplicação alvo, juntamente com a execução da thread principal. Porém, nesta hipótese, a thread principal esta rodando com prioridade mais elevada do que a prioridade dada para as demais tarefas;

Sistema com carga em alta prioridade : Representa o sistema operacional carregado com tarefas paralelas de igual prioridade a dada para a thread principal.

Para representar a sobrecarga do sistema, foi criada uma segunda thread, chamada aqui de *threadLoad*, que executa em laço uma sequência de cálculos utilizando variáveis de ponto flutuante.

5.2 Resultados Experimentais

A seguir são ilustrados os resultados obtidos seguindo o processo supracitado. A Figura 10 representa a saída do sistema simulado em ambiente Matlab/Simulink, as Figuras 11, 12 e 13, representam as saídas do sistema embarcado rodando o algoritmo exportado para as hipóteses de sistema sem carga (ocioso), com carga em baixa prioridade e com carga em alta prioridade, respectivamente.

A Tabela 3 resume os principais resultados deste teste, onde foi definido como erro máximo a diferença entre o máximo período medido e o valor esperado (5 ms).

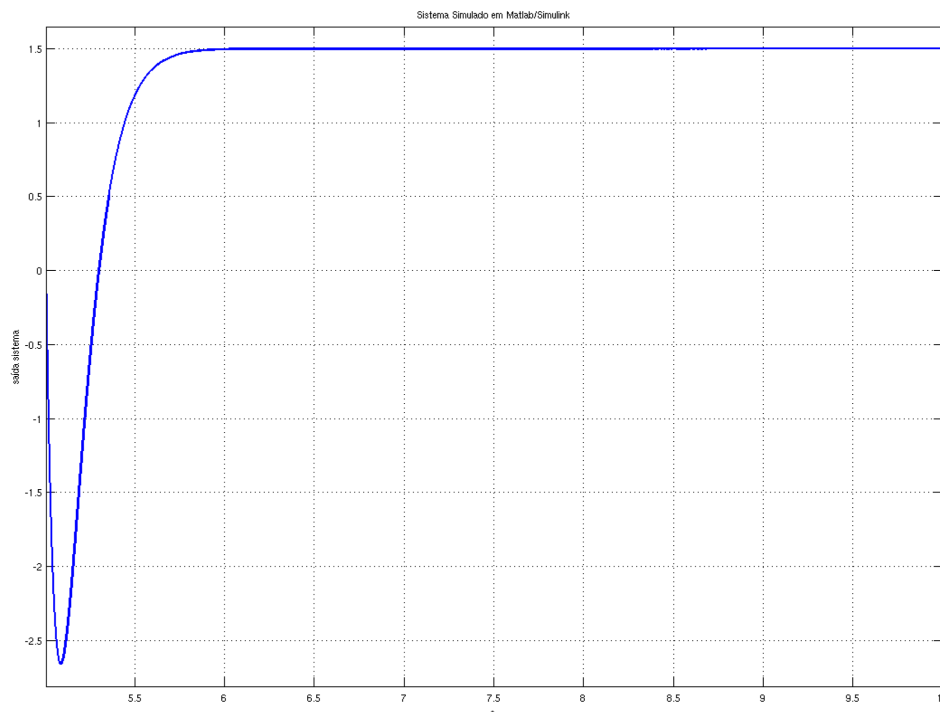

Figura 10: Sistema simulado em ambiente Matlab/Simulink

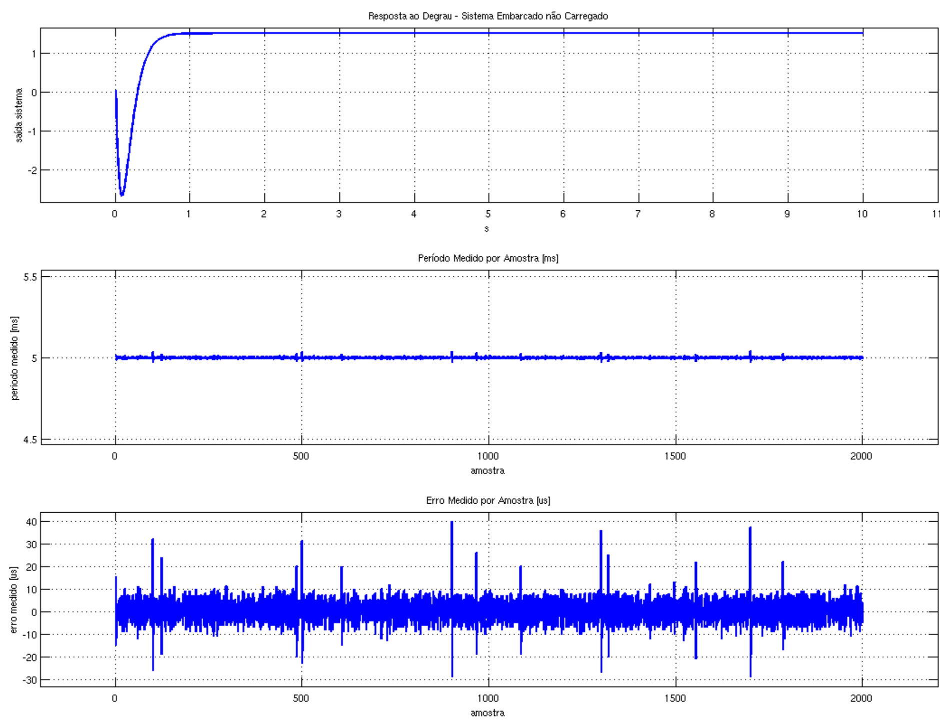

Figura 11: Aplicação rodando sem carga

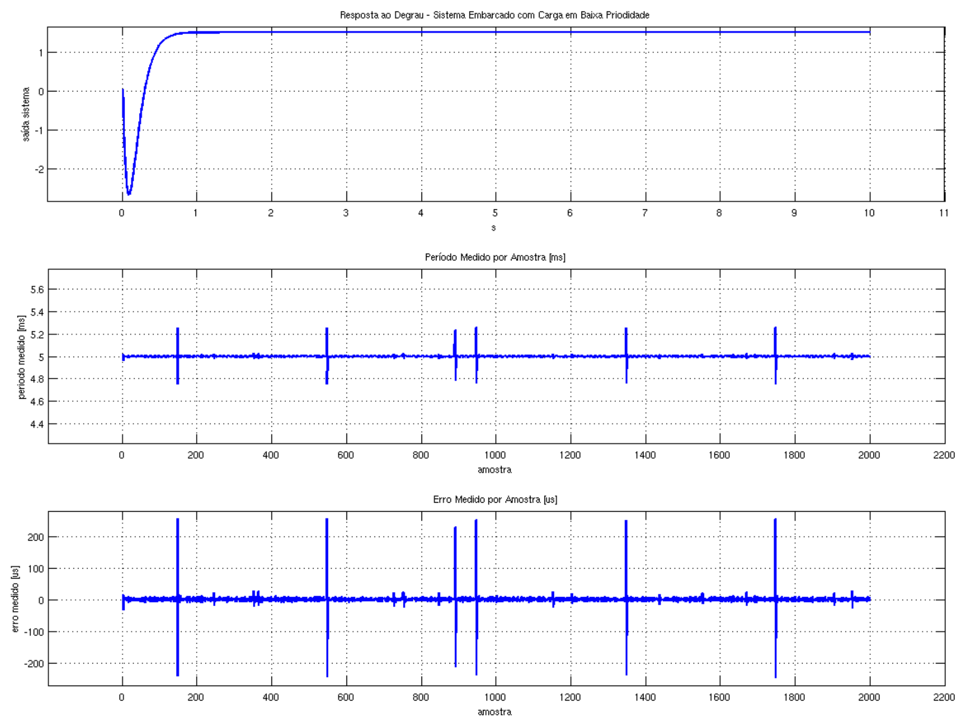

Figura 12: Aplicação rodando com carga em baixa prioridade

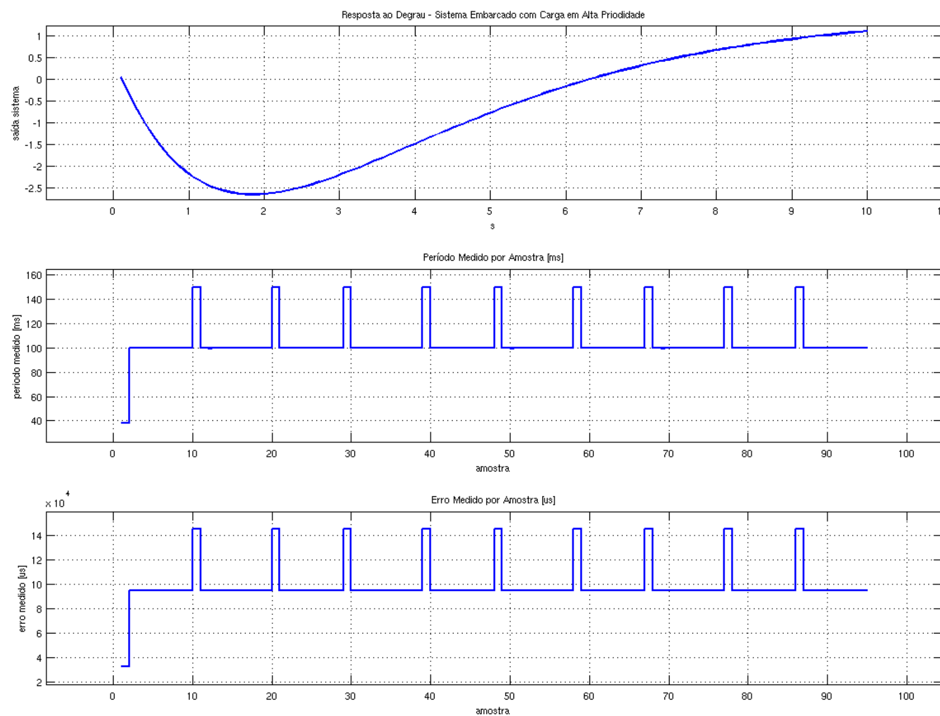

Figura 13: Aplicação rodando com carga em alta prioridade

Tabela 3: Resultados experimentais

Hipótese	Thread Load	Período Médio [ms]	Erro Máximo [us]
Sistema sem Carga	Desativada	5,0000	40
Sistema com Carga em Baixa Prioridade	Ativa	5,0000	255
Sistema com Carga em Alta Prioridade	Ativa	104,0824	145.096

Analisando-se os dados experimentais, observou-se nitidamente o incremento do erro a medida que o sistema operacional é sobrecarregado com tarefas concorrentes a execução da thread principal, o que era o resultado esperado.

Observou-se também que, para o sistema em testes, o período estabelecido é seguido com erros relativamente pequenos para as hipóteses de sistema sem carga e com carga em baixa prioridade. Revelando que o sistema implementado por threads com o uso do `Timerfd` realmente fornece tempo real de execução tipo *soft*.

6 CONCLUSÕES

Os testes mostram que a aplicação desenvolvida consegue integrar o algoritmo exportado pelo RTW ao sistema de threads periódicas, tornando o algoritmo um executável em tempo real na plataforma alvo utilizada e possivelmente em qualquer plataforma embarcada que utilize o kernel Linux. Porém, mais testes devem ser realizados futuramente para efetivamente garantir-se a periodicidade da aplicação para quaisquer sistemas.

A abordagem adotada para desenvolvimento da aplicação por meio de threads utilizando o `timerfd system call` é funcional e possibilita que diversos subsistemas rodem simultaneamente em tempo real, sendo necessário apenas criar-se novas threads para cada algoritmo. Porém, caso o sistema estiver atarefado (ou seja, carregado), não é mais garantido que a execução siga periódica como esperado. Para garantir este tipo de execução seria necessário a adoção de um sistema operativo que disponibiliza-se preempção total de kernel, ou seja, tempo real tipo *hard*.

A utilização do *patch RTAI* (RTAI, 2011), como uma forma de tornar o sistema *hard real time* foi estudada e cogitada para o projeto, porém, após tentativas frustradas, entrou-se em contato com os desenvolvedores deste pacote para a arquitetura ARM, os quais relataram que não existe ainda um *patch* compatível com a plataforma de desenvolvimento utilizada neste trabalho. Como fazer a portabilidade de um pacote RTAI que suportasse a plataforma alvo utilizada nesta obra estaria fora do escopo e dos objetivos principais da proposta, a hipótese de utilização do RTAI foi descartada.

6.1 Trabalhos Futuros

Como trabalhos futuros é sugerido o estudo das novas implementações do kernel Linux 3.0 e do pacote de preempção RT. No desenvolvimento deste trabalho observou-se que muitos esforços vem sendo tomados pela comunidade de desenvolvedores para tornar o Linux totalmente preemptivo. A API `timerfd`, utilizada neste trabalho, é um exemplo de esforço para melhorar a interface do usuário com o relógio do sistema e é

relativamente recente (foi desenvolvida em 2007). Caso o kernel Linux padrão viesse a possibilitar temporização *hard real time*, permitiria também à aplicação rodar em tempo real tipo *hard* sem ser necessário o uso de sistemas operativos pagos ou dependências de sistemas desenvolvidos por outras frentes que não a principal do kernel (como é o caso do RTAI).

Poder-se-ia testar a aplicação desenvolvida como controlador de alguma planta real. Isso poderia ser feito com o desenvolvimento de *drivers* que suportassem a entrada e saída de sinais para a aplicação. Tornando assim o projeto de controladores e sua execução real muito mais prático e fácil.

Simulações *hardware in the loop* poderiam ser realizadas utilizando-se a aplicação desenvolvida juntamente com algum tipo de sistema para entrada e saída de dados.

O uso do LCD e da *touchscreen*, disponíveis na placa i.MX25 usada neste trabalho, poderia ser explorado para configuração em tempo real do algoritmo que esta rodando na aplicação desenvolvida. Isso possibilitaria um amplo leque de aplicações de interface homem máquina e ainda poderia utilizar o sistema desenvolvido de threads periódicas para execução de outras tarefas como varredura do *display*.

REFERÊNCIAS

ARM. **The Archtecture for the Digital World**. Disponível em: http://www.arm.com/community/software-enablement/. Acesso em: novembro 2011.

FREESCALE. **i.MX25 Multimedia Applications Processors**. Disponível em: http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=IMX25_FAMILY. Acesso em: Agosto 2011.

I.MX Advanced Toolkit - ATK. [S.l.]: Freescale, 2009.

KERNEL. **The Linux Kernel Archives**. Disponível em: http://www.kernel.org/doc/. Acesso em: Novembro 2011.

LINUX. **The Linux Jornal**. Disponível em: http://lwn.net/Articles/245533/. Acesso em: Novembro 2011.

LTIB. **Linux Target Image Builder**. Disponível em: http://ltib.org/. Acesso em: Setembro 2011.

MATHWORKS. **Products and Services**. Disponível em: http://www.mathworks.com/products/?s_cid=global_nav. Acesso em: julho 2011.

NEELAKANDAN, S.; LAD, A.; RAGHAVAN, P. **Embedded Linux System Design and Development**. [S.l.]: Auerbach Publications, 2006.

RTAI. **RealTime Application Interface**. Disponível em: https://www.rtai.org/. Acesso em: Novembro 2011.

RUSLING, D. A. **The Linux Kernel**. [S.l.: s.n.], 1999. Disponível em: http://www.tldp.org/LDP/tlk/tlk.html. Acesso em: Novembro 2011.

SANKARAYOGI, R. **Software Tools for Real-Time Simulation and Control**. 2005. Dissertação (Mestrado) — West Virginia University. Disponível em: `https://eidr.wvu.edu/etd/documentdata.eTD?documentid=4397`. Acesso em: julho 2011.

UBUNTU. **Distribution Community**. Disponível em: `http://www.ubuntu.com/download/ubuntu/download`. Acesso em: Novembro 2011.

WIKI i.MX. **A Wiki for i.MX Developers**. Disponível em: `http://www.imxdev.org/wiki/index.php?title=Main_Page`. Acesso em: Agosto 2011.

WIKIPEDIA. **Distribuição Linux**. Disponível em: `http://pt.wikipedia.org/wiki/Distribui\%C3\%A7\%C3\%A3o_Linux`. Acesso em: Outubro 2011.

ANEXO A

O roteiro a seguir tem o objetivo de demonstrar passos funcionais para instalação da LTIB em um *host* PC Linux rodando a versão LTS 10.04 da distribuição Ubuntu.

A LTIB por si só não é autossuficiente, ou seja, ela necessita que estejam instalados no *host* um conjunto bibliotecas dando-lhe suporte. Para facilitar o processo de instalação foi criado um *shell script* que realiza a tarefa de baixar e instalar as bibliotecas necessárias. O código deste *shell script* pode ser visto na Figura 14 abaixo:

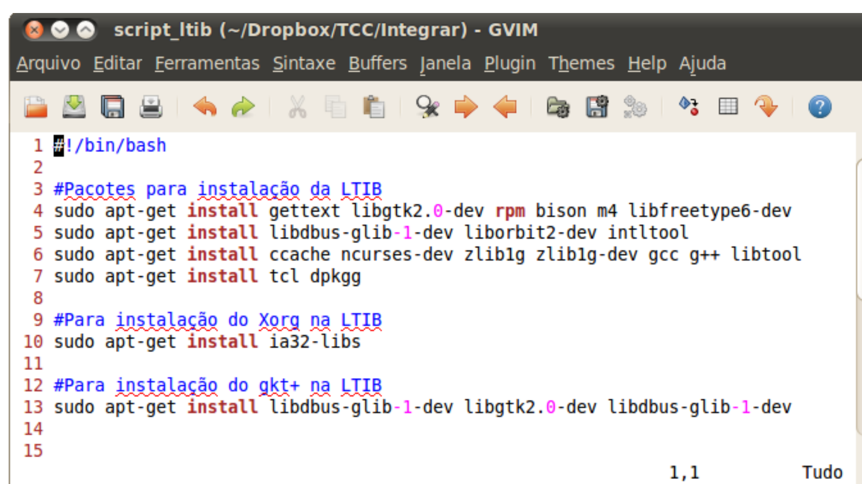

Figura 14: Script LTIB

Para execução do *script* é necessário apenas que o usuário vá até a pasta onde este está localizado e insira o comando ./script_ltib como administrador (*root*). Caso o *script* seja executado sem autorização de administrador, a senha será solicitada antes dos pacotes serem instalados.

Após o download e instalação das bibliotecas necessárias, é possível proceder com a instalação da LTIB. Para tal é preciso que se faça o download dos arquivos-fonte na página da Freescale, os quais são mais atualizados e tem maiores implementações do que os disponibilizados no CD do kit. O último pacote fonte disponível para a placa i.MX25 é o `L2.6.31_09.12.00_SDK_source`. Realizado o download, é necessário descompactar o arquivo em alguma pasta e então executar o script install, localizado dentro da pasta com os arquivos-fonte. Para execução do script execute no terminal:

```
cd <Fontes LTIB>./install
```

Realizados estes passos a instalação deve ser iniciada e será solicitado que o usuário entre com o caminho escolhido para a pasta da LTIB. Ao final da instalação deverá constar a pasta "ltib" no caminho solicitado e também a pasta /opt/freescale/. Caso não seja criada a pasta freescale em /opt, deverá ser criado um link simbólico para a pasta onde foi instalada a LTIB, isso pode ser feito com o comando:

```
sudo ln -s <caminho indicado para instalação da LTIB> /opt/freescale
```

Deve-se também alterar as permissões de acesso a pasta /opt, isso é feito executando-se o comando:

```
sudo chmod 777 /opt
```

Após realizados estes passos é necessário informar o caminho para o *script* rpm, necessário para a compilação dos pacotes pela LTIB. Isso é feito editando-se o arquivo /etc/sudoers. Para isso abra o arquivo digitando no terminal o comando:

```
sudo /usr/sbin/visudo
```

Ou então usando:

```
gksudo gvim /etc/sudoers
```

caso queira utilizar o editor gvim. E então adicione a linha: <usuário> ALL = NOPASSWD: /usr/bin/rpm, /opt/freescale/ltib/usr/bin/rpm. A Figura 15 ilustra o arquivo editado.

```
# /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for details on how to write a sudoers file.
#

Defaults        env_reset

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL) ALL
benvenuti ALL = NOPASSWD: /usr/bin/rpm, /opt/freescale/ltib/usr/bin/rpm

# Allow members of group sudo to execute any command after they have
# provided their password
# (Note that later entries override this, so you might need to move
# it further down)
```

Figura 15: Arquivo sudoers editado

Após a edição do arquivo sudoers, é necessário alterar uma configuração da LTIB para forçar a instalação de pacotes Debian. Isso é feito executando-se os seguintes passos:

Abra o arquivo Ltibutils.pm com o comando:

```
gvim /opt/freescale/ltib/bin/Ltibutils.pm
```

Edite a linha 563 inserindo a linha:

'glibc-devel' => sub -f '/usr/lib/libm.so' || -f '/usr/lib64/libz.so' || -f '/usr/lib/i386-linux-gnu/libm.so',

Edite as linhas 583 e 584 inserindo as linhas:

zlib => sub my @f = (glob('/usr/lib/libz.so*'), glob('/lib/libz.so*'), glob('/lib/i386-linux-gnu/libz.so*')); @f > 1 ? 1 : 0 , 'zlib-devel' => sub -f '/usr/include/zlib.h' ,

Realizados estes passos, vá até o diretório onde foi instalada a LTIB e execute o comando:

```
./ltib -c
```

Uma interface deverá ser iniciada no terminal permitindo a escolha da placa alvo, para o caso da i.MX25 deve ser escolhido o imx25_3stack.

Após ser escolhida a placa alvo, é possível realizar diversas modificações no modo como a LTIB monta a imagem de kernel, boot e nos pacotes a serem instalados. A Figura 16 ilustra a interface da LTIB.

Após selecionar os pacotes e as alterações desejadas, deve-se dar o comando exit. A Interface pedirá se é desejado que as alterações sejam salvas e então iniciará o processo de compilação das imagens de kernel e boot, e também irá gerar o sistema de arquivos (rootfs).

Caso haja mais alguma falha na compilação é possível obter mais suporte na página da wiki da comunidade dos desenvolvedores da iMX.

```
Freescale iMX25 3-Stack Board
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while <N> will
exclude a feature.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] feature is selected  [ ] feature is excluded

(imx25_3stack) Platform
--- LTIB settings
    System features  --->
--- Choose the target C library type
    Target C library type (glibc)  --->
    C library package (from toolchain only)  --->
    Toolchain component options  --->
--- Toolchain selection.
    Toolchain (ARMv5te gcc-4.1.2,Multi-lib,gnueabi/glibc-2.5-nptl-3)  --->
(-O2 -fsigned-char -msoft-float) Enter any CFLAGS for gcc/g++
--- Choose your bootloader
    Bootloader (u-boot)  --->
--- Choose your board
--- Choose your Kernel
    Kernel (Linux 2.6.31-imx)  --->
[ ] Always rebuild the kernel
[ ] Produce cscope index
[*] Include kernel headers
[ ] Configure the kernel
--- Leave the sources after building
--- Package selection
    Package list  --->
--- Target System Configuration
    Options  --->
--- Target Image Generation
    Options  --->
---
Load an Alternate Configuration File
Save Configuration to an Alternate File

<Select>    < Exit >    < Help >
```

Figura 16: Interface da LTIB

ANEXO B

Para facilitar o processo de instalação do servidor TFTP e alteração dos arquivos de configuração necessários foi desenvolvido o script ilustrado na Figura 17. Para executá-lo deve-se ir, pelo terminal, até a pasta no qual ele esta localizado e digitar o comando:

```
sudo ./script_tftp
```

```
1 #!/bin/bash
2
3 sudo apt-get install atftpd
4 sudo apt-get install xinetd
5 if ! ls /tftpboot ; then sudo mkdir -f /tftpboot; fi;
6 sudo chmod a+x /tftpboot
7 cd /etc/xinetd.d/
8 sudo rm tftp
9 sudo touch tftp
10 cat >> tftp <<EOF
11 service tftp
12 {
13 socket_type = dgram
14 protocol = udp
15 wait = yes
16 user = root
17 server = /usr/sbin/in.tftpd
18 server_args = /tftpboot
19 disable = no
20 per_source = 100 2
21 flags = IPv4
22 }
23 EOF
24 sudo /etc/init.d/xinetd restart
25 clear
26 echo "Instalado tftp. Pasta /tftpboot"
```

Figura 17: Script TFTP

Isso irá instalar os pacotes necessários e editar os arquivos de configuração. Deverá também ser criada a pasta /tftpboot. Nela deverão ser colocados os arquivos das imagens de kernel e sistema de arquivos para que possam ser transferidos para a placa iMX.

ANEXO C

Para a geração do sistema de arquivos e posterior acesso deste pela placa i.MX são necessários alguns procedimentos, descritos a seguir.

Primeiramente deve-se gerar alterar a configuração da LTIB para gerar o sistema de arquivos NFS. Para isso é necessário seguir até o diretório onde foi instalado a LTIB e proceder o comando de configuração:

```
./ltib -c
```

O utilitário de configuração da LTIB será aberto e então deve-se seguir Target Image Generation → Target Image → NFS Only. A Figura 18 ilustra a configuração da LTIB para gerar o sistema de arquivos por NFS.

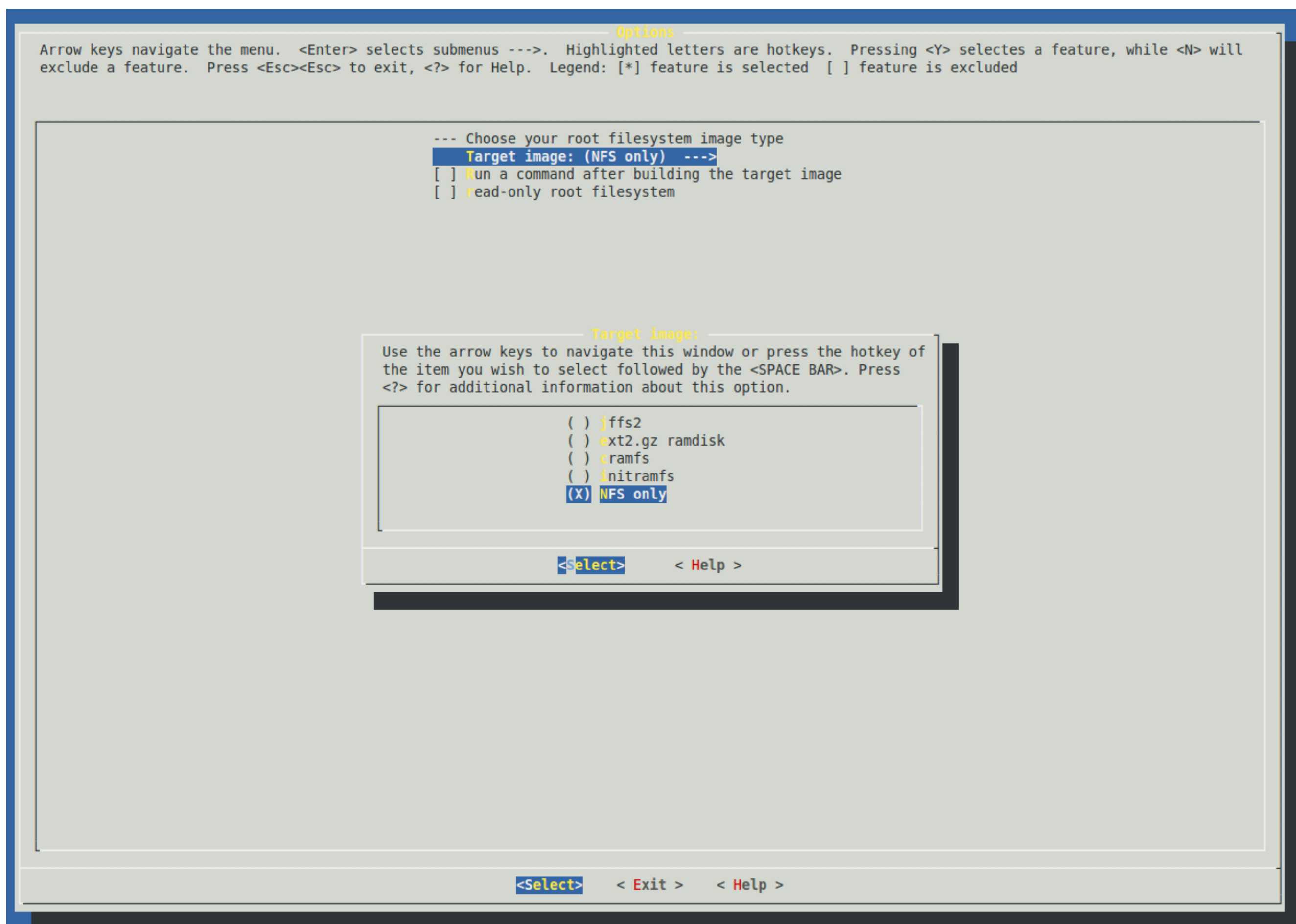

Figura 18: Configuração da LTIB para NFS

Após editar esta configuração e a LTIB proceder com a geração do sistema de arquivos

é necessário a instalação do cliente NFS. Para facilitar o processo foi desenvolvido um *shell script* que instala e exibe as configurações que devem ser editadas. A Figura 19 ilustra o *script* NFS, para executá-lo é necessário seguir até a pasta onde ele está localizado e digitar o comando:

```
./script_nfs
```

```
1 #!/bin/sh
2
3 #instala server nfs
4 sudo apt-get install nfs-kernel-server
5
6 echo "----------------------------------------------------------------------"
7 echo "Você deve incluir o caminho para pasta rootfs no arquivo /etc/exports"
8 echo "Inclua a linha como indicado abaixo"
9 echo "<caminho para ltib>/rootfs *(rw,no_root_squash,no_subtree_check,async)"
10 echo "----------------------------------------------------------------------"
11
12
~
```

Figura 19: Script NFS

Feito isso é necessário editar o arquivo /etc/exports e inserir a linha:

<caminho para ltib>/rootfs *(rw,no_ root_ squash,no_ subtree_ check,async)

A Figura 20 ilustra o arquivo exports editado.

```
1 # /etc/exports: the access control list for filesystems which may be exported
2 #»··»···to NFS clients.  See exports(5).
3 #
4 # Example for NFSv2 and NFSv3:
5 # /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
6 #
7 # Example for NFSv4:
8 # /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
9 # /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
10 # /home 192.168.0.2/255.255.255.0(rw,sync,no_subtree_check)
11 # /usr/local 192.168.0.2/255.255.255.0(rw,sync,no_subtree_check)
12
13 #/tftpboot/rootfs/ *(rw,no_root_squash,no_subtree_check,async)
14 #/tmp/fs *(rw,no_root_squash)
15
16 /home/benvenuti/Freescale/ltib/rootfs *(rw,no_root_squash,no_subtree_check,async)
17
```

Figura 20: Arquivo exports editado

Para que o sistema NFS seja montado pela placa quando iniciamos a carga da imagem de kernel, é necessário certificar-se de que a conexão entre a plataforma e o *host* PC mantenha-se. Ocorre que a placa em sua inicialização *reseta* a porta ethernet, o que ocasiona a perda do link, e consequentemente do IP, do *host*. Para contornar-se este problema, deve-se configurar o IP do *host* continuamente até que a placa tenha finalizado sua inicialização.