Centro Universitário do Leste de Minas Gerais

Unileste-MG

Programa de Iniciação Científica - PIC

Estudo e Desenvolvimento de Sistemas Embarcados Usando Linux

Bolsista: Alan Carvalho de Assis

Orientador: Ms. Afonso Cunha Silva Júnior

Cel. Fabriciano / 2003

Centro Universitário do Leste de Minas Gerais

Unileste-MG

Programa de Iniciação Científica - PIC

Curso: Computação - Sistemas de Informação

Área do Conhecimento: Engenharias

Linha de Pesquisa: Circuitos Digitais

Estudo e Desenvolvimento de Sistemas Embarcados Usando Linux

Bolsista: Alan Carvalho de Assis

Orientador: Ms. Afonso Cunha Silva Júnior

Cel. Fabriciano / 2003

Pesquisa aprovada em

_____de _____de _____

pelo Conselho de Pesquisa - PIC / UnilesteMG.

Prof. Otton Fava

Presidente do Conselho de Pesquisa PIC / UnilesteMG

Profa. Dra. Myriam Evelyse Mariani

Coordenadora do PIC / UnilesteMG

Agradecimentos

Ao Professor Ms. Afonso Cunha Silva Júnior pela sua ajuda no desenvolvimento deste projeto e por suas idéias "malucas" sobre as possibilidades e potencialidades deste trabalho.

Aos professores do curso de Computação - Sistemas de Informação pela amizade e pelos conhecimentos passados ao longo de todos estes períodos.

Ao amigo e colaborador Marcelo Barros de Almeida que sempre me ajudou com idéias e incentivando o desenvolvimento deste projeto.

Aos colegas do LTR pela boa convivência e pela colaboração no período em que convimos junto no mesmo laboratório.

Ao Programa de Iniciação Científica - PIC, pelo apoio e incentivo ao engrandecimento científico.

Ao UnilesteMG e à Fundação Geraldo Peringeiro de Abreu - FGPA, pelo apoio financeiro dado para o desenvolvimento desde estudo.

Dedicatória

Dedico este trabalho aos meus familiares, por toda a dedicação e empenho para minha formação, e principalmente à minha mãe que com seu carinho me ajudou a superar os momentos mais difíceis.

Resumo

Nesta pesquisa foram estudados e implementados sistemas embarcados usando Linux como sistema operacional. Utilizou-se o uClinux, uma versão do sistema operacional Linux para microcontroladores e DSP's que não possuem unidade de gerenciamento de memória (*MMU*).

Como dispositivo alvo, usou-se uma placa de testes Motorola Coldfire Evaluation Kit, da AVNet, que utiliza o processador RISC Motorola ColdFire MCF5282. Este processador enquadra-se na categoria SOC (*System On Chip*), onde tem-se num único chip a infra-estrutura necessária para a execução de um sistema operacional embarcado.

Palavras Chaves: Linux, Sistemas Embarcados, Sistemas Embutidos, uClinux, MCF5282.

Abstract

In this research were studied and implemented embedded systems using Linux as operating system. It was used uClinux, a version of the Linux operating system for microcontroller ans DSP's which do not have memory management unit (MMU).

As target device was been used an evaluation test board, known as Motorola Coldfire Evaluation Kit, developed by AvNet. This board use the MCF5282 RISC processor from Motorola. This processor is related to the SOC (*System On Chip Category*), in which there is an unique chip, the necessary to execute an embedded system.

Key Words: Linux, Embedded System, uClinux, Coldfire, MCF5282.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Sistemas embarcados são sistemas computacionais dedicados a uma aplicação específica, o que os distinguem de sistemas computacionais de uso geral. Até pouco tempo atrás os sistemas embarcados eram também caracterizados por possuirem recursos computacionais reduzidos, mas atualmente isto não é mais regra, visto que existem muitos sistemas embarcados com mais poder computacional que um computador pessoal (PC).

Geralmente sistemas embarcados podem ser implementados através de microcontroladores (uC), processadores de sinais digitais (DSP) ou microprocessadores (uP) possuindo um *software* ou um sistema operacional agregado. Os sistemas embarcados modernos geralmente são desenvolvidos utilizando-se um sistema operacional (SO) (freqüentemente um SO embarcado)[2].

O Linux começou a ser usado como sistema operacional para sistemas embarcados no final da década de 90, com o porte do kernel do Linux para os processadores sem unidade de gerenciamento de memória (MMU), este porte foi realizado através do projeto conhecido como `uClinux`.

Usando `uClinux` o desenvolvedor poderá ter todas as potencialidades comuns num sistema Linux normal, como por exemplo, excelente suporte ao protocolo de comunicação TCP/IP, tudo isso em menos de 1MB de memória.

Uma vez que o `uClinux` está funcionando no dispositivo alvo, as demais aplicações que existem para Linux poderão ser facilmente portadas para fun-

cionar nele. Em geral quase todas as aplicações necessárias para a maiorias das necessidades já se encontram portadas para o `uClinux`.

Possuir milhares de desenvolvedores trabalhando juntos para resolver os problemas existentes e propor novas melhorias, esta é, sem dúvida, uma das maiores vantagens de utilizar-se um sistema operacional embarcado que é licenciado como `software livre`.

1.1 Objetivos

O objetivo deste projeto é estudar e desenvolver aplicações embarcadas utilizando o sistema operacional Linux como opção viável, econômica e eficaz.

Esta pesquisa científica objetiva instalar e testar o `uClinux` em uma placa embarcada e, após finalizar esta tarefa, esta placa embarcada será utilizada em controles industriais e automação de equipamentos eletrônicos.

A placa utilizada possui interface `Ethernet`, assim as aplicações desenvolvidas poderão ser controladas remotamente através de uma rede local (*LAN - Local Area Network*) e/ou através da Internet.

1.2 Justificativa

Até 1998 o uso do Linux como sistema operacional embarcado era insignificante, inferior a 1%, mas nos anos seguintes cresceu a ponto de representar 37% deste mercado em 2003 segundo pesquisa do `Evans Data Corp`.

O Linux suporta as APIs (*Application Program Interface*) do padrão **POSIX** (*Portable Operating System Interface for Unix*). Este é o padrão (IEEE 1003.1) que define a linguagem das interfaces usadas entre os programas e o sistema operacional Unix. Este padrão assegura compatibilidade

quando um programa é movido de um sistema Unix para outro[3].

O `uClinux` por sua vez adere ao padrão **POSIX** o que torna os programas desenvolvidos para Linux e outros Unix compatível com o `uClinux`, assim, na maioria das vezes, bastam pequenas alterações nos mesmos para passem a funcionar no `uClinux`. Em alguns casos não é necessários nem mesmo realizar alterações, sendo necessário apenas recompilar o aplicativo para a nova plataforma.

Além destas vantagens o `uClinux` possui muitas outras que o tornam adequado para o desenvolvido de sistemas embarcados. Pode-se citar, de maneira sucinta, as principais vantagens do `uClinux` para o desenvolvimento de sistemas embarcados:

- Livre, não há pagamento de licenças e nem de *royaties* por unidades vendidas;
- Código fonte aberto, o acesso ao código fonte é garantido pela licença **GPL** (*General Propose License*), portanto o desenvolvedor poderá modificar o sistema segundo sua necessidade;
- Ambiente de desenvolvimento gratuito, incluindo compiladores, depuradores, e outras ferramentas de desenvolvimento;
- Inúmeros aplicativos disponíveis de forma livre;
- Verificação de falhas (*bugs*) rápido e eficiente;
- Extensão para suporte a tempo real disponível;
- Suporte a vários processadores, o que o torna independente de um único fabricante de processadores;
- Suporte a vários periféricos e *hardware* de diferentes fornecedores.

1.3 Definições de Termos/Glossário

Os seguintes termos são utilizados ao longo deste trabalho, sendo que alguns termos foram mantidos na língua inglesa por são serem termos já consagrados na literatura científica nacional.

ANSI - Instituto de normas americano (*American Natinal Standards Institute*)

API - Interface do programa de aplicação

CPU - Unidade central de processamento

ELF - Formato binário de ligação e aplicação

ext2 - Sistema de arquivo extendido segundo

Flash - Memória não volátil capaz de reter informações mesmo após cessar alimentação

FTP - Protocolo de transferência de arquivo

GB - Gyga bytes, equivale a $2^{30} bytes$

GNU - Projeto iniciado em 1984 por Richard Stallman, cujo objetivo era criar um sistema operacional livre.

GPL - Licença pública geral

GUI - Interface gráfica do usuário

HTTP - Protocolo de transferência de hipertexto

HW - Hardware, parte física de um computador ou de um equipamento eletrônico

I/O - Entrada e Saida

IEEE - Instituto de elétrica e eletrônica

IP - Protocolo de internet

IPC - Comunicação interprocesso

LGPL - Licença de proposito geral restrita

Linux - Sistema Operacional de código fonte aberto, atualmente é muito utilizado em servidores de rede e de Internet e também no meio científico.

MHz - Mega Hertz, um milhão de ciclo por segundo

MMU - Unidade de gerenciamento de memória

NFS - Sistema de arquivo de rede

OS - Sistema operacional

PC - Computador pessoal

PHY - Circuito de interface física

POSIX - Padrão de interface para sistemas UNIX

RAM - Memória volátil

ROM - Memória não volátil

SDRAM - Memória volátil de sincronismo dinâmico

SW - Software

TCP - Protocolo de controle de transmissão

TFTP - Protocolo de transferência de arquivo simplificado

UART - Transmissor e receptor serial assíncrono

uClinux - Linux para microcontroladores

UNIX - Sistema operacional desenvolvido nos anos 70 por Dennis Ritchie e Ken Thompson

2 Revisão de Literatura

2.1 Placa de Avaliação da AvNet

A placa de avaliação `Motorola Coldfire Evaluation Kit` foi desenvolvida pela fabricante de equipamentos eletrônicos chamada AvNet. Esta placa utiliza o microcontrolador Motorola Coldfire `MCF5282`.

A foto desta placa pode ser vista na Figura 1, ela possui todos os principais recursos de *hardware* que a tornam adequada para o desenvolvimento de sistemas embarcados.

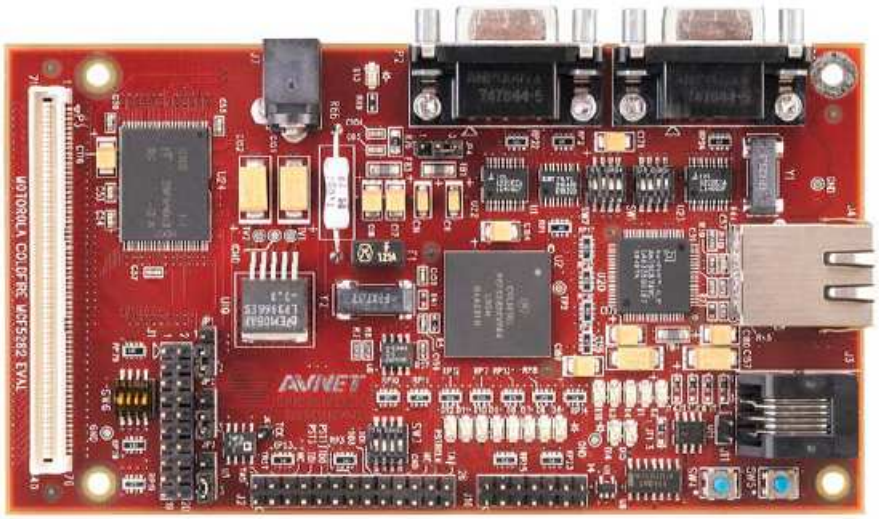

Figura 1: Foto da placa AvNet Motorola Coldfire Evaluation Kit

Esta placa possui os seguintes recursos de *hardware*:

- Processador Motorola Coldfire `MCF5282`;
- Interface `Ethernet` 10/100 Mbps;
- 2 portas seriais `RS-232`;
- Interface `CAN` (*Controller Area Network*);
- 16 Mbytes de memória `SDRAM`;
- 8 Mbytes de memória `Flash`;
- Conector `BDM/JTAG`;
- Conversor Analógico/Digital de 8 canais/10 bits (`QADC`);
- Queued `SPI` (*Serial Peripheral Interface*);
- Porta `I2C` (*Inter Integrated Circuit*);
- Conector de expansão (`AvBus`).

A placa `Motorola Coldfire Evaluation Kit` foi escolhida por possuir interface `Ethernet`, interface `CAN` e razoável quantidade de memória (`RAM` e `Flash`), o que permite desenvolver aplicações para Internet e controle de aplicações industriais que utilizam o barramento `CAN`.

Outro fator decisivo foi o fato do `uClinux` já possuir porte para o microcontrolador `MCF5282`, o que tornaria o porte para esta placa uma tarefa menos árdua.

Infelizmente a placa `Motorola Coldfire Evaluation Kit` não vem de fábrica com um *bootloader*, ao contrário da placa *M5282EVB* da Motorola

que possui o `dBUG`. Porém esta placa da Motorola tem um custo muito maior que a da AvNet, o que justificou a aquisição desta.

A ausência de um *bootloader* motivou o pesquisador deste projeto a portar o `U-Boot` para a placa `Motorola Coldfire Evaluation Kit`, assim outros desenvolvedores que venham a utilizar a mesma placa poderão beneficiar-se deste *bootloader*.

Um *bootloader* geralmente é responsável por configurar e inicializar os recursos de uma placa embarcada, servindo como base para a inicialização de um sistema operacional na memória da mesma.

A placa `Motorola Coldfire Evaluation Kit` possui 16 MB de memória SDRAM externa, 64KB de SRAM interna (no processador), 8MB de memória Flash externa e 512KB de memória flash interna. O mapa de memória da placa pode ser visto na Tabela 1.

Tabela 1: Tabela do Mapa de Memória

INÍCIO	**FIM**	**DESCRIÇÃO**
FF80 0000	FFFF FFFF	Flash Externa
F000 0000	F007 FFFF	Flash Interna
4000 0000		IPSBAR
2000 0000	2000 FFFF	SRAM Interna
0000 0000	00FF FFFF	SDRAM Externa

O `U-Boot` foi portado para funcionar na memória flash externa, e é ele o responsável por inicializar o `uClinux`. A Seção 2.1.1 possui mais informações sobre o `U-Boot`.

2.1.1 O bootloader U-Boot

O *bootloader* escolhido para ser utilizado em nossa BSP foi o `U-Boot`. Ele foi escolhido por suportar o processador Motorola Coldfire MCF5282.

Como o nome sugere o `U-Boot` é um *Universal Bootloader* disponível com o código fonte completo sob licença `GPL`. Vários processadores são suportados incluindo: PowerPC (MPC5xx, MPC8xx, MPC82xx, MPC7xx, MPC74xx, 4xx), ARM (ARM7, ARM9, StrongARM, XScale), MIPS (4Kc,5Kc), x86, Coldfire e outros.

O projeto `U-Boot` foi iniciado por Wolfgang Denk em 2000 e a página oficial do projeto é: `http://sourceforge.net/projects/u-boot`, onde se encontra disponível atualmente a versão 1.0.0.

O `U-Boot` é diretamente relacionado ao Linux, sendo projetado para inicializar (*bootar*) as imagens do kernel do Linux.

As principais características do `U-Boot` são:

- Diagnóstico de Inicialização (*POST - Power On Self Test*)
- Suporte a vários dispositivos de boot:

  Ethernet

  USB

  Dispositivos IDE

  Dispositivos SCSI

  Dispositivos ATAPI

  Dispositivos de discos flexíveis

- Variáveis de ambiente salvas em memória não volátil
- Console serial e console VGA

- Suporte a inicialização por imagens de kernel compactadas (gzip).

O `U-Boot` suporta `TFTP`, assim a imagem do *kernel* do Linux ou de qualquer outro sistema operacional pode ser descarregado rapidamente através da rede usado uma interface `Ethernet`.

2.2 Sistemas Embarcados

Sistema embarcado podem ser definidos como: *"Um sistema cuja função principal não é computacional, mas que é controlado por um computador integrado neste sistema"*[4].

Estes sistemas geralmente são designados para possuirem certas características que não são importantes em um sistema computacional de propósito geral. As principais características são: baixo consumo de energia, espaço físico limitado e baixo custo, em geral são desenvolvidos utilizando microcontroladores e/ou processadores de baixo consumo e de custos acessíveis.

Os dispositivos embarcados, em sua maioria, possuem interface com o usuário muito simples e com poucos recursos. Normalmente possuindo apenas alguns poucos botões, telas sensíveis ao toque e em alguns casos não possuem interface externa com o usuário. Neste último caso a configuração dos mesmos é realizada através de um *console serial* ou através da Internet usando um pequeno servidor *web* rodando internamente no dispositivo embarcado.

Existem várias vantagens de utilizar-se um sistema embarcado controlado via Internet[5], só para citar as principais:

- Controle remoto de qualquer local onde haja conexão com Internet;
- O fabricante pode monitorar o dispositivo embarcado e dar suporte de qualquer local e a qualquer hora;

- Reduz custos, uma vez que elimina gasto com interfaces físicas com o usuário;

Os primeiros sistemas embarcados desenvolvidos por volta de 1970 utilizam sistemas operacionais "caseiros", criados pelos próprios desenvolvedores (*"roll-your-own"*). Estes sistemas eram muito rudimentares, e o sistema operacional se confundia com os próprios aplicativos.

O sistema embarcado daquela época era equivalente a um módulo com um microcontrolador. Em geral estes sistemas eram desenvolvidos diretamente em linguagem de máquina (*assembly*). Por isso o código desenvolvido para um processador, quase sempre, deveria ser reescrito quando fosse utilizado em outro processador.

Com o tempo, mais e mais recursos foram exigidos destes sistemas[6]. Requisitos como interconectividade, acessibilidade, integração e tolerância a falhas passaram a ser exigidos por tais sistemas.

Para atender a estas novas exigências os sistemas embarcados tiveram que evoluir e tornaram-se mais complexos. As tarefas que antes exigiam apenas uma rotina executando dentro de uma laço de repetição (*loop*), passaram a ser realizadas como processos em execução em um sistema operacional embarcado[1].

Assim, um sistema embarcado atual em muito aproxima-se de um computador de uso geral, possuindo um razoável poder computacional, uma considerável quantidade de memória e um sistema operacional completo e complexo.

Com o avanço da micro-eletrônica e a exigência por sistemas mais complexos, aquele padrão de desenvolvimento "caseiro" foi aos poucos substituído pelo desenvolvimento usando um sistema operacional desenvolvido por terceiro, geralmente de forma comercial.

O Linux veio atender perfeitamente à estas exigências, possuindo o código fonte disponível e não necessitando de pagamento de licenças para sua utilização em sistemas embarcados.

2.2.1 O mercado atual dos Sistemas Embarcados

A indústria de computadores pessoais vende por ano em torno de 100 milhões de computadores, em contrapartida são vendidos 6 vezes mais processadores para serem usado em sistemas embarcados[7].

O mercado de sistemas operacionais para dispositivos embarcados é formado por muitas empresas, entre elas as duas maiores são, respectivamente: Wind River e Microsoft. Segundo o IDC (`www.idc.com`) no ano 2000 as empresas de sistemas operacionais para sistema embarcados faturaram, juntas, 666 milhões de dólares, sendo esperado um crescimento para 2,8 bilhões em 2005.

Várias empresas no entanto estão desenvolvendo soluções para sistemas embarcados baseadas em Linux, entre elas: MontaVista (`www.mvista.com`), SnapGear (`www.snapgear`), MetroWerks(`www.metrowerks.com`), LynuxWorks (`www.lynuxworks.com`), TimeSys (`www.timesys.com`), Cyclades (`www.cyclades.com`) entre outras.

O Linux está crescendo de forma rápida e significativa no mercado de sistemas embarcados, prova disso é o consórcio CELF(*CE Linux Forum* - `www.celinuxforum.org`) formado por empresas de renome do setor de tecnologia: Sony, Philips, Sharp, Toshiba, Nokia, Motorola, IBM, HP e outras, com o objetivo de evoluir e padronizar o Linux para os dispositivos embarcados.

2.3 O Sistema Operacional Linux

O Linux é um sistema operacional (SO) relativamente novo, ele é considerado como um clone do UNIX.

Um sistema operacional é um programa especial que atua intermediando entre o usuário e os dispositivos físicos do computador (*hardware*). Por exemplo, quando o usuário copia um arquivo para o disquete, usando um programa apropriado para esta função, na verdade o programa que o usuário esta usando solicita algumas funções ao núcleo (*kernel*) do sistema operacional para acessar o leitor de disquete e realizar a cópia.

O Linux começou a ser desenvolvido em 1991 pelo estudante de computação finlandês Linus Torvalds, inspirado no sistema operacional Minix. Ele havia criado, segundo suas próprias palavras, "um Minix melhor que o Minix".

No dia 5 de outubro de 1991 ele liberou ao público a primeira versão oficial do Linux, versão 0.02. Desde então, milhares de programadores de várias parte do mundo têm ajudado a tornar o Linux um sistema operacional robusto, seguro, flexível e dinâmico.

A Tabela 2 demonstra a evolução do *kernel* do Linux desde sua criação até os dias atuais.

Ao contrário do Minix, o Linux foi licenciado sob a licença GPL, esta licença incentiva a colaboração entre os desenvolvedores, uma vez que todo aperfeiçoamento no código fonte deverá ser disponibilizado para uso público.

Por outro lado algumas empresas não desejam disponibilizar seu código fonte proprietário, por temerem que o mesmo possa ser usado pela concorrência. Estas empresas podem utilizar em seus programas as bibliotecas com licenças **LGPL** ligadas dinamicamente ou utilizar módulos para serem carregados no *kernel*[8].

Tabela 2: História do Kernel do Linux[1]

ANO	VERSÃO	EVENTO
1991	0.01	Linus Torvalds cria o kernel do Linux.
1992	0.99	Linux passa a ser licenciado como GPL.
1994	1.0.0	Primeira versão para liberação pública.
1995	1.2	Primeira versão com suporte a multi-plataforma.
1996	2.0.0	Múltiplas arquiteturas, novos protocolos de redes.
1999	2.2	Suporte a módulos dinâmicos no kernel.
2001	2.4	Suporte a Multiprocessadores.
2003	2.6	Suporte a alguns sistemas embarcados.
	2.7	Kernel atualmente em desenvolvimento.

Atualmente o Linux continua em pleno desenvolvimento, auxiliado por milhares de programadores ao redor do mundo. O próprio Linus Torvalds continua colaborando ativamente para o desenvolvimento do Linux, trabalhando nos laboratórios da OSDN (*The Open Source Development Network*).

2.3.1 O Kernel do Linux

O *kernel* do Linux utiliza uma arquitetura conhecida como monolítica. Esta arquitetura é caracterizada pela execução dos serviços do *kernel* em modo privilegiado, conhecido como modo Kernel.

A arquitetura monolítica tem a vantagem de ser simples e mais rápida, porém, como o *kernel* é compilado todo em um único arquivo, a expansão do sistema torna-se mais difícil[9].

A arquitetura microkernel resolve este problema implementando apenas as funcionalidades básicas em modo privilegiado. Os demais serviços do *kernel* são executados em modo não privilegiado, conhecido como modo usuário.

Este tipo de arquitetura é mais lenta que a monolítica, uma vez que o *kernel* precisa realizar trocas de mensagens entre os serviços que estão sendo executados em uma camada externa, ou seja, em modo usuário.

Além disso a arquitetura microkernel utiliza uma estrutura que é mais difícil de ser desenvolvida que a monolítica[10].

Por estas razões Linus Torvalds preferiu utilizar a arquitetura monolítica quando começou seu projeto, em 1991. Para obter as vantagens de um microkernel, como a possibilidade de expansão da funcionalidades, o *kernel* do Linux passou a suportar módulos dinâmicos.

O *kernel* do Linux foi desenvolvido para ser portável entre várias plataformas de *hardware*, possuindo suporte para processadores Alpha, Arm, Cris, M68L, MIPS, PowerPC, SPARC, x86 entre vários outros.

A portabilidade do Linux foi importante para que o `uClinux`, que é baseado no `kernel` do Linux, suportasse várias plataforma de hardware, como será visto na Seção 2.4.

2.4 uClinux - O Linux para microcontroladores

O `uClinux` é uma variante do *kernel* do Linux destinado a microcontroladores e processadores sem unidade de gereciamento de memória (*MMU - Memory Management Unit*).

Processadores sem MMU são comumente usados em aplicações embarcadas, pois são mais baratos que os processadores com MMU. Processadores com MMU possuem uma implementação em hardware interna de um gerenciador de memória.

O gerenciador de memória é responsável por converter o endereço virtual, usado pelas aplicações, em endereçamento real da memória física. Isto torna

os sistemas mais seguros, pois, cada programa possui seu próprio espaço de memória virtual, não podendo invadir a área de outros programas[11].

O projeto `uClinux` começou a ser desenvolvido em 1997 baseado no *kernel* do Linux versão 2.0.33, e foi liberado para o público em fevereiro de 1998[12]. Originalmente o projeto foi desenvolvido por Jeff Dionne e Kenneth Albanowski, contando com a participação de alguns colaborados.

O `uClinux` rapidamente ganhou suporte para outros modelos de microcontroladores e processadores, entre eles Coldfire, ETRAX, ARM7, ARM9, i960, H8, entre outros. Suporta também alguns processadores desenvolvidos em FPGA, como o MicroBlaze da Xilinx, o NIOS da Altera e o OpenRISC que é um projeto de código aberto desenvolvido por engenheiros e programadores (`www.opencores.org`).

Por ser um sistema operacional voltado para aplicações embarcadas, requisitos como tamanho final (*footprint* em inglês) e quantidade de memória RAM necessária para a execução do **SO** devem ser levados em consideração. Para permitir que as aplicações no `uClinux` tivessem um tamanho reduzido foi desenvolvida um biblioteca de linguagem `C` específica para este fim.

Inicialmente foi desenvolvida a `uC-libc` baseada na biblioteca de linguagem C `Linux-8086`, que é usada no projeto ELKS. Porém esta biblioteca possuia algumas incompatibilidades com a biblioteca C padrão **POSIX** e possuia alguns códigos sob licença **GPL**, o que tornaria a utilização do uClinux em empresas de tecnologia secreta impraticável, uma vez que todo *software* desenvolvido com ela deveria ser liberado ao público.

Para solucionar estes problemas Erik Andersen iniciou a criação de uma biblioteca sob licença **LGPL**, o que tornaria a utilização do `uClinux` em empresas de tecnologia mais simples, uma vez que os programas desenvolvidos com esta biblioteca não precisariam liberar seus códigos fontes.

Esta nova biblioteca recebeu o nome de `uClibc` (o mesmo nome da anterior sem o "-"), sendo atualmente a mais usada nas aplicações para o `uClinux`.

A utilização da biblioteca `uClibc` torna a tarefa de portar os aplicativos do Linux para o `uClinux` mais fácil, uma vez que esta biblioteca é compatível com a `glibc` usada nos aplicativos para Linux[13].

Atualmente o *kernel* do `uClinux` encontra-se em sincronismo com o *kernel* do Linux. Os desenvolvedores do `uClinux` têm trabalhado em conjunto com os desenvolvedores do Linux no sentido de integrar as funcionalidades do `uClinux` no *kernel* do Linux.

O primeiro passo já foi dados neste sentido, o suporte a alguns processadores sem `MMU` começou a ser adicionado a partir da versão 2.5.46 do kernel do Linux[14].

O suporte nativo a processadores sem `MMU` é de grande importância para que o Linux possa ser utilizado largamente em aplicações embarcadas. A expectativa para o futuro é que o `uClinux` se integre diretamente ao *kernel* do Linux, como já ocorre com vários outros projetos.

3 Metodologia

A placa alvo utilizada nesta pesquisa é conhecida como `Motorola Coldfire Evaluation Kit`, possuindo uma interface de depuração conhecida como BDM (*Background Debug Mode*). Através desta interface, pode-se enviar qualquer programa diretamente à memória RAM da placa e iniciar seu processamento em seguida, o que acelera em muito o tempo de testes e de desenvolvimento.

No Linux é necessário compilar o GDB (*GNU Debugger*) com um `patch` aplicado para que o mesmo tenha suporte ao BDM, como verá visto na Seção 3.6.

3.1 Tipo de Pesquisa

A estudo apresentado neste relatório é de cunho prático exploratório.

3.2 Amostra

A pesquisa desenvolvida retratou os seguintes elementos: estudo dos sistemas embarcados, estudo das ferramentas disponíveis para desenvolvimento de sistemas embarcados, métodos de programação da memória flash e implementação do uClinux na placa embarcada da AvNet.

3.3 Instrumentos

Na realização desta pesquisa foram utilizados uma placa embarcada desenvolvida pela AvNet utilizando um microcontrolador *M*otorola Coldfire MCF5282, um computador de arquitetura $x86$ de 1.3 GHz com 128MB de memória RAM.

O computador foi utilizado para gerar o sistema operacional a ser utilizado na placa e para gravar a imagem deste sistema na mesma. Para realizar a gravação do sistema na memória da placa foi utilizado um interface conhecida como BDM conforme descreve a Seção 3.6.

Através da interface BDM o *bootloader* U-Boot pode ser gravado na placa, este método é apresentado na Seção 3.7. Com o *bootloader* instalado na placa pode-se, então, usá-lo para enviar a imagem do uClinux para a placa, este método é apresentado na Seção 3.8.

Usando a metodologia proposta neste estudo outros pesquisadores poderão desenvolver sistemas embarcados com o uso do Linux.

3.4 Procedimentos

A coleta de dados para esta pesquisa foi, em sua maioria, obtida através de sites da Internet, utilizando a ferramenta de busca *G*oogle.

Por tratar-se de uma assunto recente não foi encontrada literatura disponível em bibliotecas, a exceção é o livro *Embedded Systems Firmware Demystified* que foi adquirida pelo UnilesteMG para ser utilizado nesta pesquisa.

3.5 Tratamento dos Dados

A coleta dos dados e informações nesta pesquisa foram obtidas em lista de discussões sobre sistemas embarcados, em sua totalidade em idioma inglês, uma vez que não existe, até então, nenhuma lista de discussão em português.

Outras informações foram obtidas diretamente através de *emails* enviados a vários pesquisadores e desenvolvedores que trabalham com sistemas embarcados usando Linux.

3.6 Instalando o BDM-GDB

Inicialmente deve-se baixar o GDB e os `patches` para suportar a interface BDM.

Baixe do site `http://sourceforge.net/projects/bdm/` os arquivos:

```
bdm-gdb-20030717.tar.gz
gdb-5.3-m68k.patch
```

Baixe do site GNU, `www.gnu.org`, o arquivo:

```
gdb-5.3.tar.gz
```

Descompacte o arquivo do BDM:

```
tar -zxvf bdm-gdb-20030717.tar.gz
```

Para o processador MCF5282 é necessário realizar uma modificação no código fonte, pois o registrador RAMBAR (usado para controle da memória RAM interna) teve seu número alterado em relação aos processadores anteriores.

Modifica-se o arquivo `bdm-gdb-20030717/m68k/driver/bdm.c` na linha 1230:

```
0xc04,    /* BDM_REG_RAMBAR   */
```

Alterando-a para:

```
0xc05,    /* BDM_REG_RAMBAR   */
```

Finalmente pode-se compilar o BDM:

```
cd bdm-gdb-20030717/m68k/
./local_scripts/build-it
cd driver/linux
make install
```

Também é necessário instalar a biblioteca do BDM para ser usada pelo GDB:

```
cd bdm-gdb-20030717/m68k/lib
make
make install
```

Após instalar o BDM, segue-se com a instalação do GDB, descompactando o arquivo:

```
tar -zxvf gdb-5.3.tar.gz
```

Entra-se no diretório `gdb-5.3` e aplica-se o `patch`:

```
cd gdb-5.3
patch -p1 < ../gdb-5.3-m68k.patch
```

Após aplicar o `patch` procede-se com a compilação e instalação:

```
./configure --target=m68k-bdm-elf
make
make install
```

Deve-se verificar se o arquivo `/dev/bdmcf0` existe, caso não exista deve-se criá-lo com o comando:

```
mknod /dev/bdmcf0 c 34 4
```

Para usar o BDM deve-se adicionar o *device driver* do BDM no sistema, através do comando:

```
insmod bdm
```

3.6.1 Testando e usando o GDB

Para iniciar o GDB deve-se executar:

```
m68k-bdm-elf-gdb
```

Para conectar ao BDM deve-se executar o comando:

```
(gdb) target bdm /dev/bdmcf0
```

Uma vez conectado, pode-se executar os comandos para depurar os programas e/ou obter informações sobre o processador. Por exemplo, o comando abaixo lista todos registadores do processador bem como os valores assumidos por eles:

```
(gdb) select-frame 0
(gdb) info reg
```

Pode-se também reiniciar a placa alvo através do comando:

```
(gdb) bdm_reset
```

Outro comando que foi importante para os testes realizados neste projeto e que está presente em qualquer GDB, mesmo sem aplicar os `patchs` é:

```
(gdb) disassemble 0x10000
```

Este comando *desassembla* o código presente em determinada posição de memória, no exemplo 0x10000. O código em *assembly* exido deverá ser igual ao obtido do arquivo binário, através do comando:

```
m68k-elf-objdump -D arquivo.elf
```

Outros comandos podem ser obtidos através do `help` do próprio aplicativo, fugindo do escopo deste estudo.

3.6.2 Executando um programa através do BDM

Nesta seção será mostrado como enviar um programa para a memória da placa e executa-lo.

Para compilar este programa é necessário ter a versão das ferramentas `GNU` para o processadores `M68K` instaladas no sistema.

O Arquivo main.c

Este é o arquivo fonte principal, nele encontram-se as instruções responsáveis por acender e apagar alternadamente os LEDs da placa de testes.

```
int main()
{
    int i;
    unsigned char *ptcpar = (unsigned char *)0x4010005B;
    unsigned char *ddrtc  = (unsigned char *)0x40100024;
    unsigned char *porttc = (unsigned char *)0x40100010;
    *ptcpar = 0x00;
    *ddrtc  = 0x0F;
```

```
    while(1)
    {
        *porttc = 0x01;
        for (i=0;i<20000;i++);

        *porttc = 0x04;
        for (i=0;i<20000;i++);
    }
    return 0;
}
```

O arquivo start.s

Este é o arquivo responsável por inicializar o *hardware* de forma que os programas em linguagem C possam ser utilizados corretamente.

```
#define MEM_BASE        0x00000000
#define VBR_BASE        MEM_BASE
#define MEM_SIZE        0x01000000
.global _main
.global _start
.global _rambase
.global _ramvec
.global _ramstart
.global _ramend
.data
_rambase:
.long   0
_ramvec:
```

```
.long   0
_ramstart:
.long   0
_ramend:
.long   0
.text
_start:
        nop
        move.w  #0x2700, %sr
        move.l  #VBR_BASE, %a7
        movec   %a7, %VBR
        move.l  %a7, _ramvec
        move.l  %a7, _rambase
        move.l  #MEM_SIZE, %a0
        move.l  %a0, %d0
        move.l  %d0, %sp
        move.l  %d0, _ramend
        jsr     main
_exit:
        jmp     _exit
```

O arquivo mem.ld

Este é o arquivo onde deve-se definir o endereçamento de memória para onde o programa será enviado.

```
MEMORY {
        ram     : ORIGIN = 0x16000, LENGTH = 0x000fff
}
```

```
SECTIONS {
        .text : {
                _stext = . ;
                *(.text)
                _etext = ALIGN(0x4) ;
        } > ram
        .data BLOCK(0x4) : {
                _sdata = . ;
                __data_start = . ;
                *(.rodata)
                *(.data)
                _edata = ALIGN(0x4) ;
        } > ram
        .bss BLOCK(0x4) : {
                _sbss = . ;
                *(.bss)
                *(COMMON)
                _ebss = ALIGN(0x4) ;
                _end = ALIGN(0x4) ;
        } > ram
}
```

Makefile

O Makefile é o arquivo que contém as instruções para que o programa `make` compile, usando as ferramentas `GNU`, os códigos fontes do projeto.

```
CC_EXEC_PREFIX = /usr/local/bin/m68k-elf-
CC = $(GCC_EXEC_PREFIX)gcc
```

```
AS = $(GCC_EXEC_PREFIX)as
LD = $(GCC_EXEC_PREFIX)ld
AR = $(GCC_EXEC_PREFIX)ar
OBJCOPY = $(GCC_EXEC_PREFIX)objcopy
PROGNAME = tstart
TARGET = $(PROGNAME).s19
OBJS = start.o main.o
CFLAGS += -g -m5307
ASFLAGS += -Wa,-as -m5307
$(PROGNAME).s19: $(PROGNAME).bin
$(OBJCOPY) --input-target=binary --output-target=srec
$(PROGNAME).bin $(PROGNAME).s19
$(PROGNAME).bin: $(PROGNAME).elf
$(OBJCOPY) -O binary $(PROGNAME).elf $@
$(PROGNAME).elf: $(OBJS)
$(LD) -T mem.ld -M -o $(PROGNAME).elf $(OBJS) >$(PROGNAME).map
.S.o:
        $(CC) $(ASFLAGS) -c $<
.c.o:
        $(CC) $(CFLAGS) -nostartfiles -nostdlib -c $<
clean:
        rm *.o *.bin *.s19 *.elf *.map -f
start.o: start.S
all:    $(TARGET)
```

Compilando e enviando o programa para a placa

Para compilar o programa, basta executar o comando `make`. Se a operação transcorrer corretamente será gerado um arquivo com o nome `tstart.elf`.

Deve-se então iniciar o GDB passando como parâmetro o nome do arquivo binário:

```
m68k-bdm-elf-gdb tstart.elf
```

Em seguida, deve-se executar os comandos, na seguinte sequência:

```
(gdb) target bdm /dev/bdmcf0
(gdb) load
(gdb) set $pc=0x16000
(gdb) c
Continuing.
```

Imediatamente o LEDs da placa deverão acender de forma alternada.

O GDB foi de extrema importância neste projeto, uma vez que o mesmo foi utilizado para enviar o *bootloader* `U-Boot` para e memória da placa.

3.7 Instalando o bootloader

Após baixar o `U-Boot` do site é necessário compilá-lo para a versão do processador da placa alvo.

Deve-se iniciar o executável do `U-Boot` através do programador BDM, da mesma forma como se inicia um aplicativo qualquer, como exemplificado na Seção 3.6.2.

O `U-Boot` pode-se ser controlado através de um *console serial*, para isso deve-se configurar um programa de comunicação serial, como o **minicom**,

com as seguintes configurações: "19200 8N1" e com controle de fluxo por *hardware* e *software* desativado.

Quando o `U-Boot` for executado, será exibido pela `console serial` as informações da placa e um *prompt* estará disponível:

```
U-Boot 0.4.0 (Jan 26 2004 - 14:10:08)
CPU:   MOTOROLA Coldfire MCF5282
Board: AvNet Motorola Coldfire Evalution Kit Board
DRAM:  16 MB
FLASH:  8 MB
*** Warning - bad CRC, using default environment
AVNETMCEK>
```

Usando o *prompt* de comandos do `U-Boot` configura-se o mesmo com as informações de rede adequadas.

```
AVNETMCEK> setenv ethaddr 00:cf:52:72:c3:01
AVNETMCEK> setenv ipaddr 10.2.5.91
AVNETMCEK> setenv serverip 10.2.5.90
```

Onde o *ethaddr* é o número `MAC` da placa alvo, o *ipaddr* é o endereço `IP` da placa alvo e o *serverip* é o endereço `IP` do servidor `TFTP` onde há o arquivo binário do `U-Boot` a ser gravado na Flash (loader.bin).

Para descarregar o arquivo do servidor `TFTP` para a placa alvo, deve-se executar o seguinte comando:

```
AVNETMCEK> tftp 20000 /tftpboot/loader.bin
```

Este comando irá baixar o arquivo "loader.bin" para a memória RAM da placa a partir da posição de memória 0x20000.

O próximo passo é salvar o arquivo na memória Flash, para isso, deve-se desproteger a área de memória a ser usada e apagá-la:

```
AVNETMCEK> protect off ff800000 ff80ffff
AVNETMCEK> erase ff800000 ff80ffff
```

O comando usado para gravar o arquivo na Flash (`cp.b`) recebe como argumentos, respectivamente, o endereço da memória RAM onde encontra-se os dados a serem gravados (`0x20000`), o endereço da memória Flash para onde o arquivo será gravado (`0xff800000`) e a quantidade de bytes a serem gravados (`0x10000 - 64KB`):

```
AVNETMCEK> cp.b 20000 ff800000 10000
```

Após finalizar a cópia do `U-Boot`, deve-se salvar as configurações da placa na memória `Flash`, para isso, executa-se o seguinte comando:

```
AVNETMCEK> saveenv
```

Quando finalizar esta operação pode-se reiniciar a placa e imediatamente a mensagem inicial do `U-Boot` deverá ser apresentada através da *console serial*.

Com o `U-Boot` instalado na memória da placa, pode-se enviar o `uClinux` para a mesma.

3.8 Instalando o uClinux na BSP

Todo o desenvolvimento em `uClinux` utiliza as ferramentas disponibilizadas pela GNU [citar], tais como o compilador GCC (*GNU Compiler Collection*), linker LD e vários outros aplicativos.

Para a plataforma MCF5282 (família `M68K`), um pacote completo de desenvolvimento (*toolchain*) pode ser obtido em `www.uclinux.org/pub/uClinux/m68k-elf-tools/m68k-elf-tools-20030314.sh`.

Já o arquivo fonte do uClinux pode ser encontrado em `www.uclinux.org/pub/uClinux/dist/uClinux-dist-20030909.tar.gz`

3.8.1 Configuração e compilação do uClinux

A instalação do toolchain deve ser feita com privilégios de super-usuário. Após baixar o arquivo `m68k-elf-tools-20030314.sh`, executa-se o comando:

```
#sh m68k-elf-tools-20030314.sh
```

Para compilar o uClinux é necessário primeiramente descompactar o arquivo `uClinux-dist-20030909.tar.gz`, para isso executa-se o procedimento:

```
#tar -xzvf uClinux-dist-20030909.tar.gz
```

exit

Em seguida entra-se no diretório criado e inicializa a interface de configuração do uClinux:

```
#cd uClinux-dist
make menuconfig
```

O primeiro passo é selecionar a plataforma através da opção **Target Platform Selection**:

```
(Motorola/M5282C3)
(linux-2.4.x)
(uClibc)
[*]Customize Kernel Settings
[*]Customize Vendor/User Settings
```

Em seguida será exibida a tela de customização do kernel. Nesta tela as opções em **Processor type and features** devem ser alteradas:

```
(MCF5282) CPU
(60MHz) CPU CLOCK Frequency
--- Platform
[*] Motorola M5282C3 board support
(AUTO) RAM size
(AUTO) RAM bit width
(RAM) Kernel executes from
```

Estas são as configurações básicas para o sistema funcionar na placa AvNet `Motorola Coldfire Evaluation Kit`, os demais recursos devem ser ativados segundo a necessidade da aplicação para a qual o uClinux se baseia.

Vários protocolos de são suportados pelo `kernel` e precisam ser ativados para que aplicações que o utilizam possam funcionar. Por exemplo, para o interroperabilidade com servidores Windows, é necessário ativar o protocolo `smbfs` em **File Systems** -> **Network File Systems**.

Por fim, será exibida a tela de customização de Vendor/User. Nesta tela o usuário irá selecionar os aplicativos do uClinux que deseja utilizar.

Finalmente, a compilação do kernel pode iniciada:

```
make dep
make
```

Quando a compilação finalizar, a imagem completa do uClinux (arquivo `image.bin`) poderá ser encontrada dentro do diretório `images`. É este arquivo que deve ser enviado para a memória RAM da placa para ser executado.

3.8.2 Enviando a imagem do uClinux para a placa

O processo para enviar a imagem do `uClinux` para a placa é semelhante ao enviar um aplicativo comum. Para fazê-lo é preciso ter o arquivo `image.bin`

copiado para o diretório base do servidor de TFTP (/tftpboot).

Após inicializar a placa deve-se executar os comando para enviar o imagem do uClinux a memória RAM da mesma e executá-la:

```
tftp 400000 image.bin
go 400000
```

Imediatamente o *kernel* do uClinux deverá exibir as mensagem de inicialização, como mostrado abaixo:

```
Linux version 2.4.22-uc0 (alan@ltr03) (gcc version 2.95.3 20010315 (release4


uClinux/COLDFIRE(m5282)
COLDFIRE port done by Greg Ungerer, gerg@snapgear.com
Flat model support (C) 1998,1999 Kenneth Albanowski, D. Jeff Dionne
On node 0 totalpages: 2048
zone(0): 0 pages.
zone(1): 2048 pages.
zone(2): 0 pages.
Kernel command line:
Calibrating delay loop... 5.84 BogoMIPS
Memory available: 2136k/8192k RAM, 0k/0k ROM (666k kernel code, 206k data)
kmem_create: Forcing size word alignment - vm_area_struct
kmem_create: Forcing size word alignment - mm_struct
kmem_create: Forcing size word alignment - filp
Dentry cache hash table entries: 1024 (order: 1, 8192 bytes)
Inode cache hash table entries: 512 (order: 0, 4096 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
kmem_create: Forcing size word alignment - bdev_cache
```

```
kmem_create: Forcing size word alignment - cdev_cache
kmem_create: Forcing size word alignment - kiobuf
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 2048 (order: 1, 8192 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
kmem_create: Forcing size word alignment - sock
Initializing RT netlink socket
Starting kswapd
kmem_create: Forcing size word alignment - file_lock_cache
ColdFire internal UART serial driver version 1.00
ttyS0 at 0x40000200 (irq = 77) is a builtin ColdFire UART
ttyS1 at 0x40000240 (irq = 78) is a builtin ColdFire UART
kmem_create: Forcing size word alignment - blkdev_requests
fec.c: Probe number 0 with 0x0000
eth0: FEC ENET Version 0.2, a8:4a:c4:b7:a4:54
FEC: No PHY device found.
SLIP: version 0.8.4-NET3.019-NEWTTY (dynamic channels, max=256).
CSLIP: code copyright 1989 Regents of the University of California.
Blkmem copyright 1998,1999 D. Jeff Dionne
Blkmem copyright 1998 Kenneth Albanowski
Blkmem 1 disk images:
0: 4DA4C8-5CF8C7 [VIRTUAL 4DA4C8-5CF8C7] (RO)
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
PPP generic driver version 2.4.2
PPP MPPE compression module registered
```

```
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
kmem_create: Forcing size word alignment - ip_dst_cache
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 512 bind 512)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
VFS: Mounted root (romfs filesystem) readonly.
Freeing unused kernel memory: 24k freed (0x4c2000 - 0x4c7000)
Shell invoked to run file: /etc/rc
Command: hostname uClinux
Command: /bin/expand /etc/ramfs.img /dev/ram1
Command: mount -t proc proc /proc
Command: mount -t ext2 /dev/ram1 /var
Command: mkdir /var/tmp
Command: mkdir /var/log
Command: mkdir /var/run
Command: mkdir /var/lock
Command: mkdir /var/empty
Command: ifconfig lo 127.0.0.1
Command: route add -net 127.0.0.0 netmask 255.0.0.0 lo
Command: dhcpcd -p -a eth0 &
[11]
Command: cat /etc/motd
Welcome to
          ____ _  _
         /  __| ||_|
    _   _| |  | | _ ____  _   _  _  _
```

```
   | | | | |  | || |  _ \| | | |\ \/ /
   | |_| | |__| || | | | | |_| |/    \
   |  ___\____|_||_|_| |_|\____|\_/\_/
   | |
   |_|

For further information check:
http://www.uclinux.org/
Execution Finished, Exiting
Sash command shell (version 1.1.1)
/>
```

Se o `uClinux` inicializar corretamente, como no exemplo acima, pode-se então gravá-lo na memória Flash, para isto é necessário criar um pacote que o `U-Boot` reconhecer. Deve-se entrar no diretório `/tftpboot` e executar os seguintes comandos:

```
gzip image.bin
mkimage -A m68k -O linux -T kernel -C gzip -a 0x400000 -e 0x400000
-d image.bin.gz image.pkg
```

Este comando irá criar o pacote chamado `image.pkg`, que deverá ser enviando a placa e gravado na memória Flash, usando os seguintes comandos:

```
tftp 400000 image.pkg
erase ff840000 ffffffff
cp.b 400000 ff840000 eee65
```

Onde "eee65" é o tamanho em bytes, no formato hexadecimal, do arquivo `image.pkg`.

Para inicializar o uClinux a partir da memória Flash deve-se executar o seguinte comando:

```
setenv bootcmd bootm ff840000
setenv bootdelay 3
saveenv
```

O "bootdelay 3" é o tempo, em segundos, que o U-Boot irá aguardar antes de inicializar o *kernel* do uClinux.

Deve-se tomar cuidado ao se utilizar o "bootdelay 0", pois, caso o kernel não inicialize, não terá como recuperar o sistema e o U-Boot deverá ser reinstalado.

Como a imagem do uClinux gravada na memória flash pode-se "resetar" ou desligar e em seguida ligar novamente a placa embarcada, após alguns segundos o uClinux iniciará automaticamente.

Uma vez que o uClinux encontra-se instalado na placa, todas as características inerentes a um sistema Linux podem ser exploradas. Para verificar o funcionamento do servidor web chamado boa que é executado por padrão no uClinux, conecte o cabo de rede na placa e verifique através de um navegador web, acessando o IP da mesma.

Inúmeras aplicações podem ser devenvolvidas utilizando os recursos existente na placa embarcada. Na Secão **??** encontra-se o código de uma aplicação simples que foi desenvolvida para controlar os leds da placa através da rede.

4 Resultados e Discussões

Os objetivos maiores desta pesquisa foram atingidos, uma vez que pôde-se utilizar o conhecimento adquirido na literatura técnica e aplicá-lo no desenvolvimento de sistemas embarcados reais. O `uClinux` foi configurado, compilado e executado com sucesso na placa embarcada utilizada nesta pesquisa.

Também foi portado um *bootloader*, conhecido como `U-Boot`, para a placa embarcada. Este *bootloader* é um programa que é gravado na Flash da placa e é o responsável pela inicialização da imagem do *kernel* do `uClinux`.

Várias experiências com sistemas de arquivos e suporte a recursos de hardware foram testados, verificando assim a capacidade e flexibilidade do `uClinux` como um sistema embarcado. Em geral a imagem obtida com o `kernel` 2.4 é em torno de 2MB, porém como o `U-Boot` aceita inicializar imagens compactadas, pôde-se utilizar este artifício e obter uma imagem final gravada na memória Flash em torno de 1MB apenas.

Como aplicação prática, para a demonstração do funcionamento do `uClinux` na placa, foi desenvolvido um *device driver* para controlar os LED's da placa. Assim foi possível acender ou apagar os LED's pela `console serial`. Esta idéia evoluiu e criou-se um `CGI` que permite controlar os LED's através da Internet.

5 Conclusão e Recomendação

Este estudo apresentou o desenvolvimento de sistemas embarcados utilizando o `uClinux` como sistemas operacional. O `uClinux` vem ganhando notoriedade como um sistema operacional para sistemas embarcados, sendo utilizado por várias empresas de tecnologia, uma vez que não requer o pagamento de royalties.

Neste relatório de pesquisa foi explicado os detalhes para implementar o `uClinux` na placa de avaliação `Motorola Coldfire Evaluation Kit` da AvNet. Todos os passos envolvidos neste processo foram descritos, incluindo configuração, compilação e envio da imagem do *kernel* para a placa.

Espera-se que este estudo sirva como base para futuros projetos envolvendo o desenvolvimento de sistemas embarcados utilizando Linux, projetos práticos para aplicações industriais podem ser desenvolvidos com base no conhecimento explicado neste estudo.

Recomenda-se para aqueles que desejam implementar sistemas embarcados para controle em tempo real, utilizar o `RTAI` ou `RTLinux` aplicado sobre o `uClinux`.

Anexos

Anexo A. Controlando os LEDs da placa pelo browser

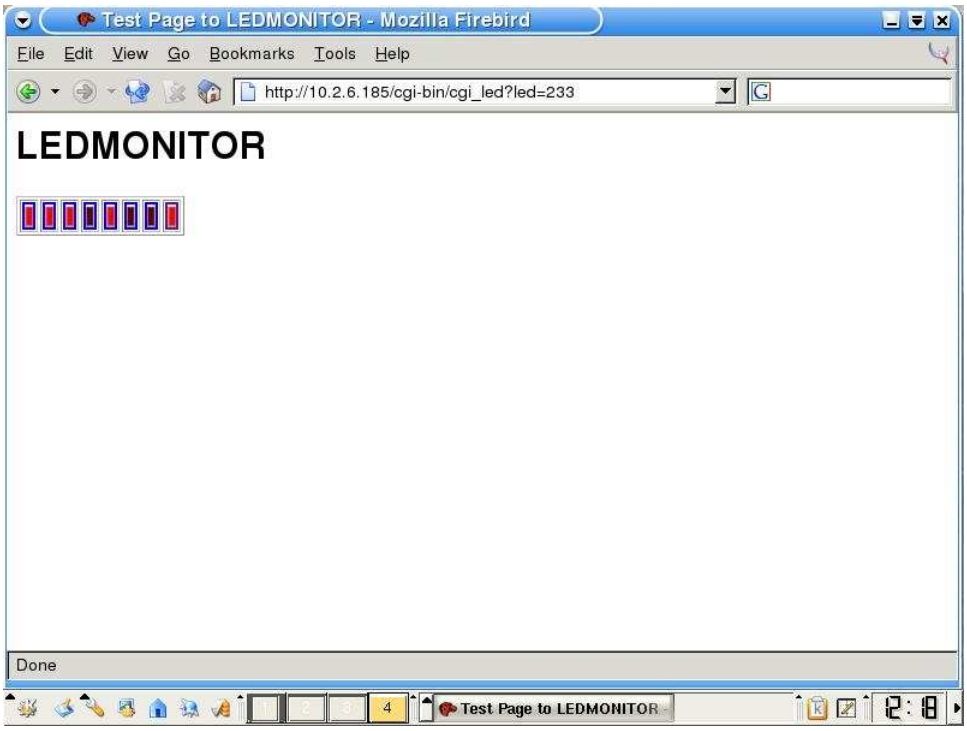

Figura 2: Página CGI para controle dos LEds da placa

Anexo B. Foto da placa com os LEDs acesos

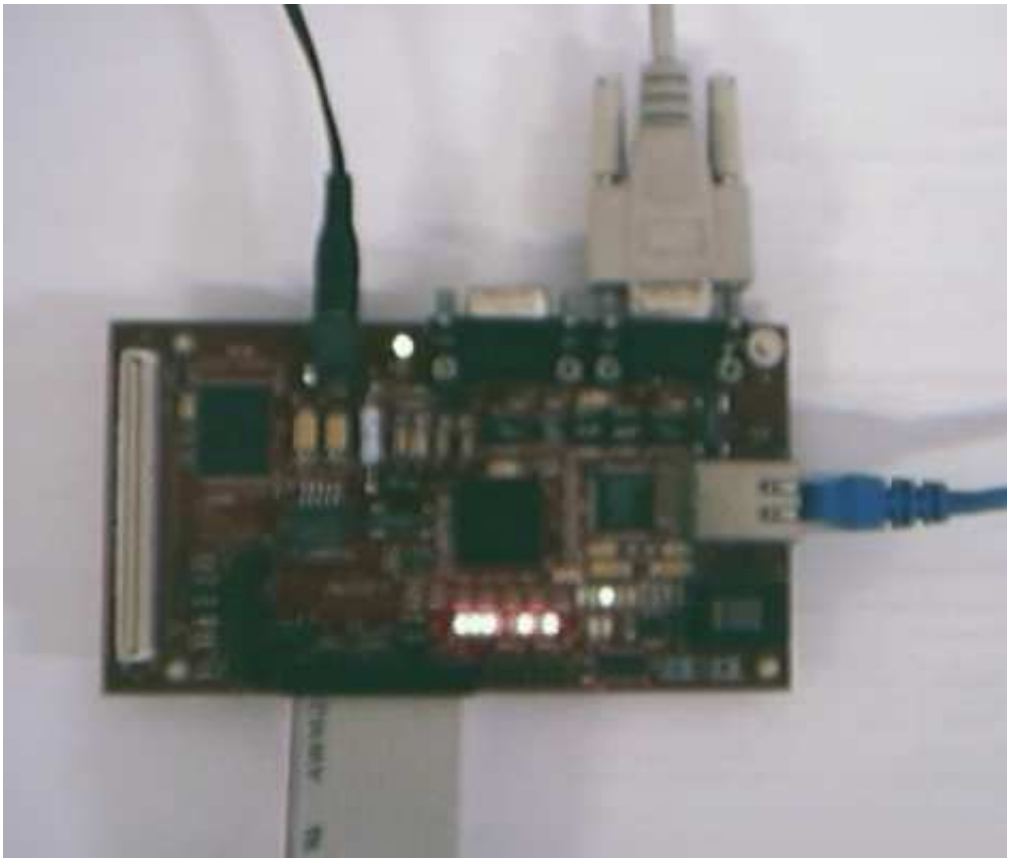

Figura 3: LEDs da placa controlados por um CGI

Anexo C. Device Driver do LedMonitor

Arquivo ledmon.c

```
/*
 * (C) Copyleft 2004
 * Alan Carvalho de Assis, alan@unilestemg.br
 *
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA 02111-1307 USA
 */

#define MODULE
#define __KERNEL__
```

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>   /* printk */
#include <linux/fs.h>       /* everything... */
#include <linux/errno.h>    /* error codes */
#include <linux/types.h>    /* size_t */
#include <linux/proc_fs.h>  /* proc file system */
#include <linux/fcntl.h>    /* O_ACCMODE */
#include <asm/system.h>   /* cli, flags */
#include <asm/uaccess.h>    /* copy from/to user */

MODULE_LICENSE("GPL");

/*Registers to get leds access*/

volatile unsigned char * gptbdr = (unsigned char *)0x401b001d;
volatile unsigned char * gptbddr= (unsigned char *)0x401b001e;
volatile unsigned char * ptcpar = (unsigned char *)0x4010005A;
volatile unsigned char * ptdpar = (unsigned char *)0x4010005B;
volatile unsigned char * ddrtc  = (unsigned char *)0x40100023;
volatile unsigned char * ddrtd  = (unsigned char *)0x40100024;
volatile unsigned char * porttc = (unsigned char *)0x4010000f;
volatile unsigned char * porttd = (unsigned char *)0x40100010;

/* Function prototypes required by led.c */

int led_open (struct inode *inode, struct file *filp);
```

```
int led_release (struct inode *inode, struct file *filp);
ssize_t led_read(struct file *filp, char *buf, size_t count, loff_t *f_pos);
ssize_t led_write (struct file *filp, const char *buf, size_t count, loff_t *f_p
void acende_leds(int leds);
void cleanup_module (void);

/* Structure to declare our file access functions. */

struct file_operations led_fops =
{
open:  led_open,
read: led_read,
write: led_write,
release: led_release
};

/* Global driver variables. */
int  led_major = 60; /* major number */
char led_buffer; /* data buffer */

int init_module(void)
{
    int result;


    /*Iniciatialize LEDs off*/
    *ptcpar = 0x00;
```

```
    *ptdpar = 0x00;
    *ddrtc  = 0x0f;
    *ddrtd  = 0x0f;
    *gptbddr= 0x0f;
    *gptbdr = 0x0f;

    /* Register the device. */
    result = register_chrdev(led_major, "led", &led_fops);
    if (result < 0)
    {
        printk("<1>LED: couldn't obtain major number %d\n",led_major);
        return result;
    }

    printk("AvNet Led driver is installed\n");
    printk("Copyleft Alan Carvalho de Assis - alan@unilestemg.br\n");
    led_buffer = 0;
    return 0;
}

/* Unload the module. */
void cleanup_module(void)
{
    unregister_chrdev (led_major, "led");
    printk("AvNet Led driver is removed\n");
}
```

```
/* Open the device file. */
int led_open(struct inode *inode, struct file *filp)
{
    MOD_INC_USE_COUNT;
    printk("Executing led_open\n");
    return 0;
}

/* Close the device file. */
int led_release(struct inode *inode, struct file *filp)
{
    MOD_DEC_USE_COUNT;
    printk("Executing led_release\n");
    return 0;
}

ssize_t led_read(struct file *filp, char *buf, size_t count, loff_t *f_pos)
{

    led_buffer = 0xFF;

    if(*porttc & 0x01)
        led_buffer ^= 0x01;
    if(*porttc & 0x04)
        led_buffer ^= 0x02;
    if(*porttd & 0x01)
        led_buffer ^= 0x04;
```

```
    if(*porttd & 0x04)
        led_buffer ^= 0x08;
    led_buffer ^= (*gptbdr<<4);


    /* Move data to the user space */
    if(copy_to_user (buf, &led_buffer, 1))
        return -EFAULT;

    *f_pos++;
    return 1;
}


/* Write to the file. */
ssize_t led_write(struct file *filp, const char *buf, size_t count, loff_t *fpos
{
    printk("Executing led_write\n");

    /* Move data to the user space */
    if(copy_from_user (&led_buffer, buf, 1))
        return -EFAULT;

    printk("Input = %c\n",led_buffer);

    show_leds(led_buffer);
```

```
    *fpos++;

    return 1;
 }

void show_leds(int leds)
{
    *porttc  = 0x05;
    *porttd  = 0x05;
    *gptbdr  = 0x0f;

    if(leds&0x01)
        *porttc ^= 0x01;
    if(leds&0x02)
        *porttc ^= 0x04;
    if(leds&0x04)
        *porttd ^= 0x01;
    if(leds&0x08)
        *porttd ^= 0x04;
    leds = leds>>4;
        *gptbdr ^= leds;
}
```

Anexo B. Arquivo fonte do CGI

Arquivo template.c baseado no cgi_generic

```
#include <stdio.h>

#include "cgivars.h"
#include "htmllib.h"



#define DEBUG 0

int led;

int template_page(char **getvars, int form_method) {
int i;
FILE *f;

addTitleElement("LEDMONITOR");

if(form_method == GET) {
for (i=0; getvars[i]; i+= 2) {
#if DEBUG
printf("<li>DEBUG: [%s] = [%s]\n", getvars[i], getvars[i+1]);
#endif
if(strcmp(getvars[i],"led")==0)
    led=atoi(getvars[i+1]);
}
```

```
}


if((f = fopen("/dev/led","wb"))==NULL)
{
    printf("File /dev/led not found or Device Driver is not loaded!\n");
    return -1;
}

fprintf(f,"%c",led);
fclose(f);

printf("<TABLE border=1>\n");
printf("<TR>\n");

for(i=128;i>=1;i=i>>1)
{
    if(led & i)
    {
printf("<TD>\n");
printf("<A href=\"/cgi-bin/cgi_led?led=%d\"><img \
 src=\"/img/on.gif\"></A>\n",led & (~i));
printf("</TD>\n");
    }
    else
    {
printf("<TD>\n");
```

```
printf("<A href=\"/cgi-bin/cgi_led?led=%d\"><img \
 src=\"/img/off.gif\"></A>\n",led | i);
printf("</TD>\n");
    }

}

printf("</TR>\n");
printf("</TABLE>\n");

return 0;
}
```

Referências Bibliográficas

1 NIKKANEN, K. *uClinux as an Embedded Solution*. [S.l.], 2003.

2 CIRCUITS, I. I. S. on; 2002, S. I. (Ed.). *EMSIM: An Energy Simulation Framework for an Embedded Operating System*. [S.l.: s.n.], May 2002.

3 DICK, N. *Embedded ColdFire - Taking Linux On-Board*. [S.l.], 2000.

4 WILMHURST, T. *An Introduction to the Design of Small-Scale Embedded Systems*. [S.l.]: Palgrave/Macmillan Press, 2001.

5 SILVA, D.; COELHO, C.; FERNANDES, A. *Conexão de Sistemas Embutidos a Internet*. [S.l.], 1999.

6 JUNIOR, J. C. *CFPPC: Um Sistema Didático Baseado em Processadores PowerPC*. [S.l.], 2001.

7 GILLEN, A.; KUSNETZKY, D. *Embedded Operating Environments: How Does Linux Compare?* [S.l.], 2001.

8 BEAL, D.; BARR, M. Embedded linux and the law. *Embedded Systems Programming*, 2002.

9 TANENBAUM, A. S.; S.WOODHULL, A. *Operating Systems Design and Implementation*. [S.l.]: Prentice-Hall, Inc, 1998.

10 TORVALDS, L. Microkernels. 2002. Disponível em: <http://www.sindominio.net/biblioweb/telematica/open-sources-html/node86.html>.

11 ASSIS, A. C.; BARROS, M. A. uclinux: o linux dos pequenos. *Revista do Linux*, 2003.

12 DRABIK, J. uclinux: World's most popular embedded linux distro? *LinuxDevices*, 2002. Disponível em: <http://www.linuxdevices.com/articles/AT3267251481.html>.

13 MCCULLOUGH, D. What is the difference between uc-libc and uclibc. 2001. Disponível em: <http://www.ucdot.org/article.pl?sid=02/08/21/1124218mode=thread>.

14 GILLHAM, M. uclinux and linux set to merge. 2002. Disponível em: <http://www.snapgear.com/tb20021115.html>.