UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RODRIGO SCHEER ROVERÉ

MÓDULO CONVERSOR DE SINAIS EM ETHERNET

Canoas, Julho de 2009.

RODRIGO SCHEER ROVERÉ

MÓDULO CONVERSOR DE SINAIS EM ETHERNET

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Departamento:

Engenharia Elétrica

Área de Concentração

Engenharia de Processamento Digital

Professor Orientador:

Eng. Eletr. Msc. Augusto Alexandre Durgante de Mattos – CREA-RS: 08.8003-D

Canoas

2009

FOLHA DE APROVAÇÃO

Nome do Autor: Rodrigo Scheer Roveré

Matrícula: 012101215-8

Título: Módulo Conversor de Sinais em Ethernet

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Professor Orientador:

Eng. Eletr. Msc. Augusto Alexandre Durgante de Mattos

CREA-RS: 08.8003-D

Banca Avaliadora:

Eng. Eletr. Msc. Dra. Marilia Amaral da Silveira

CREA-RS: 05.0909-D

Conceito Atribuído (A-B-C-D):

Eng. Eletr. Msc. Dr. João Carlos Vernetti dos Santos

CREA-RS: 04.5852-D

Conceito Atribuído (A-B-C-D):

Assinaturas:

Autor	Orientador
Rodrigo Scheer Roveré	Augusto Alexandre Durgante de Mattos
Avaliador	**Avaliador**
Marilia Amaral da Silveira	João Carlos Vernetti dos Santos

Relatório Aprovado em:

DEDICATÓRIA

Dedico a minha mãe Margarida e a minha esposa Cyntia pelo carinho, incentivo e apoio incondicional durante essa difícil jornada em busca do sonho da graduação.

AGRADECIMENTOS

A todos que colaboraram direta ou indiretamente na elaboração deste trabalho, o meu reconhecimento.

Ao Professor Augusto de Mattos pelo estímulo, dedicação e esforço pessoal proporcionado.

Aos colegas e amigos Nicholas Gauto e Ernani Ulsenheimer pelas sugestões e observações valiosas, além de uma grande amizade e companheirismo.

Aos colegas da empresa Coda Automação Industrial Ltda. por tantos anos de incentivo, ajuda e amizade durante o trajeto profissional e acadêmico.

Aos Professores Dalton Vidor e Marilia Amaral pelas valiosas contribuições e por serem, além de grandes mestres, exemplos pessoais a serem seguidos.

A todos os familiares e amigos que ajudaram e apoiaram essa jornada.

EPÍGRAFE

Hay que endurecer pero sin perder la ternura jamás...

RESUMO

ROVERÉ, Rodrigo Scheer. **Módulo Conversor de Sinais em Ethernet: TCP/IP**. Trabalho de Conclusão de Curso em Engenharia Elétrica - Departamento de Engenharia Elétrica. Universidade Luterana do Brasil. Canoas, RS. 2009.

O presente projeto descreve o desenvolvimento de um módulo para coleta de sinais digitais e analógicos via protocolo TCP/IP através de uma rede Ethernet de alta velocidade. Devido à grande aceitação desse protocolo em ambiente industrial e custo de implementação baixo, essa foi a solução que apresentou o melhor custo-benefício. O módulo em si, contém um microprocessador responsável pela coleta, conversão dos dados e pelo encapsulamento do pacote de dados dentro do protocolo TCP/IP. Esse pacote de dados é transmitido, processado e apresentado sob a forma de um supervisório para interação do usuário final, à distância, usando um microcomputador. Para tanto foi desenvolvido um software para o controle do módulo.

Palavras chave: Aquisição de dados. Ethernet. TCP/IP.

ABSTRACT

ROVERÉ, Rodrigo Scheer. Ethernet Signal Module Conversor: TCP/IP. Work of Conclusion of Course in Electrical Engineering - Electrical Engineering Department. Lutheran University of Brazil. Canoas, RS. 2009.

The follow project was develloped to build an Ethernet Signal Module Conversor. Today the default solution is read each signal point to point, this soluction despend a lot of time to repair and instal. A net solution uses less space than the convencional aplications. If the distances are big, the electrical signal can be lost because the low voltages used in the process. The module will have an Ethernet conversor to send the signals to the server with a fast and confiable situation. The microcontroller will read the signals and send with an embedded form by protocol TCP/IP. To control this module, will be developed a server software.

Keywords: Data acquisition. Ethernet. TCP/IP.

SUMÁRIO

1. INTRODUÇÃO

O presente trabalho descreve o desenvolvimento de um projeto de software com a finalidade de transformar um módulo microprocessado em um conversor de sinais elétricos em sinais em rede Ethernet baseado no protocolo TCP/IP. A metodologia explicando os procedimentos adotados nesse projeto é discutida no capítulo 2.

O foco de mercado seriam sistemas de rastreabilidade em indústrias e automações prediais. Em uma primeira análise, se tem a dificuldade de introduzir em ambientes industriais módulos que não sejam de marcas já consagradas o que não ocorre na automação predial. Além disso, essa modalidade de automação em prédios vem crescendo dia após dia e tornando-se cada vez um atrativo maior para o consumidor final de imóveis.

Ao analisar o procedimento para coleta de sinais elétricos dentro das indústrias e prédios, nota-se que na grande maioria das aplicações utiliza-se o método ponto a ponto, ou seja, para cada ponto de coleta analógico ou digital utiliza-se um cabo até o controlador. Essa solução apresenta diversos problemas, tais como queda de tensão devido às distâncias, espaço físico ocupado maior nas calhas, interferências eletromagnéticas externas, maior tempo de instalação e dificuldade na manutenção devido à grande quantidade de cabos. A presente proposta de trabalho constitui-se no desenvolvimento de um módulo, de baixo custo, que possibilite coletar os sinais analógicos e digitais no campo e transmitir essas informações de forma rápida e segura via rede. Para tanto, será utilizado um microprocessador que será responsável pela coleta e conversão dos dados, bem como, pelo encapsulamento desse pacote de dados dentro do protocolo TCP/IP e envio via rede Ethernet ao controlador. Por fim, esse pacote de dados será recebido, processado e convertido em sinais visuais em uma tela de supervisório para informação e interação com o usuário do sistema através de um microcomputador. De forma simplificada, será criado um software no microprocessador que funcionará como cliente e outro software no microcomputador que funcionará como servidor ou banco de dados. O capítulo 4 descreve passo a passo a aplicação e os resultados obtidos.

Com essa solução, as principais vantagens seriam diminuir a quantidade de cabos, aumentar a velocidade e confiabilidade de comunicação e transmissão de dados elétricos. Os gastos com calhas, cabos, terminais, identificações são altos em uma instalação ponto a ponto. Além disso, há um grande espaço físico utilizado e muitas horas/homem gastas tanto na instalação quanto na manutenção desse tipo de solução. O tempo de manutenção preventiva também é grande devido ao re-aperto de bornes e demais conexões. Sistemas com confiabilidade baixa podem causar um grande problema em lotes de produção e conseqüentes retrabalhos em campo após a venda do produto final. A solução proposta minimiza todos esses problemas, pois se trata de uma solução limpa, confiável e de baixo custo. No capítulo 3 pode-se ver o descritivo do *hardware* do módulo.

1.1. Justificativa

O presente projeto tem como principal justificativa a carência de produtos nacionais no que se refere aos módulos com interface Ethernet. Contribuir na melhoria contínua de sistemas de produção e automação predial.

A oportunidade do negócio permite fornecer o sistema com uma gama de serviços agregados, incluindo um pacote de controle que pode servir para atuar sobre todas as etapas de um processo dedicado e direcionado às necessidades de cada cliente. Além do produto em si, pode-se vender todo o tipo de serviços, desde projetos elétricos até a própria instalação fica do sistema.

1.2. Objetivos gerais

Este trabalho tem por objetivos gerais desenvolver um módulo para coleta de sinais elétricos e um software que sirva de interface e controle para esse módulo.

Criar um módulo de baixo custo, rápido e confiável que possa substituir as instalações ponto a ponto de forma limpa, prática e de baixa manutenção, além disso, deve possuir flexibilidade na coleta dos sinais e fácil interação com o usuário final.

1.3. Objetivos específicos

Esse trabalho tem como principais objetivos desenvolver um módulo de baixo custo com parcerias específicas para a produção em série da placa, coletar sinais elétricos, convertê-los em sinais digitais, encapsular em protocolo TCP/IP e transmiti-los via rede Ethernet. Substituir as soluções com excesso de cabos e baixa confiabilidade pelo módulo conversor, minimizar os problemas de baixa velocidade e quedas de tensão, criar um software amigável e interativo para controle

dos sinais elétricos coletados e minimizar horas de manutenção corretiva e preventiva.

1.4. Especificações Técnicas da Solução Escolhida

O módulo possuirá um microprocessador de 32 bits com conversores AD e bloco Ethernet para 10Mbit/s ou 100Mbit/s. Basicamente, o módulo possuirá 8 entradas e 8 saídas digitais. Além disso, possuirá 3 entradas analógicas. A interface de rede será Ethernet através de um conector RJ-45.

O projeto do supervisório será desenvolvido em linguagem Visual Basic através do *software Visual Studio Net* e terá a função de ler, tratar e controlar os sinais elétricos de entrada e saída e servir de interface entre o módulo e o usuário final.

O projeto apresenta grande complexidade em nível de software e proporcionará um bom produto para coleta de sinais elétricos, principalmente no que se refere à versatilidade. Além disso, vai ao encontro da idéia de que um projeto de engenharia deve ser desenvolvido para solucionar um problema. No caso do presente trabalho, o problema existe e a solução é satisfatória.

2. REFERENCIAL TEÓRICO

Nesse capítulo é descrito, de forma sucinta, como são montados os pacotes de dados, estrutura do protocolo TCP/IP e o meio físico Ethernet.

Os sistemas de comunicação para dados, em especial a Internet, estão cada vez mais se tornando uma ferramenta importantíssima para o homem atual, pois aumenta a eficiência de quem utiliza esses recursos com seriedade. Devido ao baixo custo de implementação, a utilização de pontos de acesso tanto à Internet quanto à ambientes dedicados para troca de informações é uma possibilidade quase obrigatória, em ambientes empresariais principalmente.

Até mesmo em computadores pessoais a Internet é fator importante para todo o tipo de comunicação, sendo que os sistemas operacionais desse tipo de máquina já vêm dedicados para a utilização dessa ferramenta. Esse sistema ainda será válido por muitos anos na grande maioria das aplicações de rede. Porém, o presente trabalho tem o objetivo de utilizar uma aplicação embarcada para Internet.

A Figura 2-1 mostra um sistema baseado no conceito cliente – servidor onde um microcomputador supervisiona todos os demais equipamentos interligados na rede. O presente projeto foi desenvolvido dentro desse mesmo conceito onde o módulo funcionará como cliente e um microcomputador como servidor.

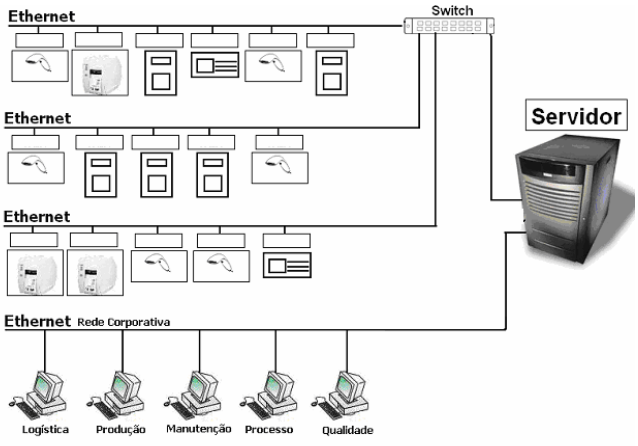


Figura 2-1 – Exemplo de rede Ethernet com módulos para conversão de sinais.

Essa nova tecnologia permite a criação de sistemas e produtos de baixo custo e grande interatividade com todo o tipo de consumidor. Desde brinquedos a sistemas de comunicação IP altamente complexos.

Para desenvolvimento desse projeto, foram trabalhados e estudados os conceitos de programação em C, programação em Visual Basic, conversão de sinais elétricos, tratamento de sinais elétricos, configurações de redes de comunicação, especificação e configuração de microprocessadores, protocolo TCP/IP.

2.1. Ferramenta para análise de protocolo

Para entender o protocolo e visualizar fisicamente seus eventos, utilizou-se uma ferramenta para análise de protocolo. A ferramenta escolhida foi a Ethereal que está disponível gratuitamente na Internet no endereço **www.ethereal.com**.

O Ethereal utiliza uma característica da placa de rede chamada como modo promíscuo, ou seja, repassa ao software todos os pacotes que trafegam na rede sem filtrar nada. Utilizando uma rede ethernet via hub ou switch, podemos visualizar todos os pacotes trocados nessa rede tanto do computador com o Ethereal quanto dos demais equipamentos interligados, conforme Figura 2-2.

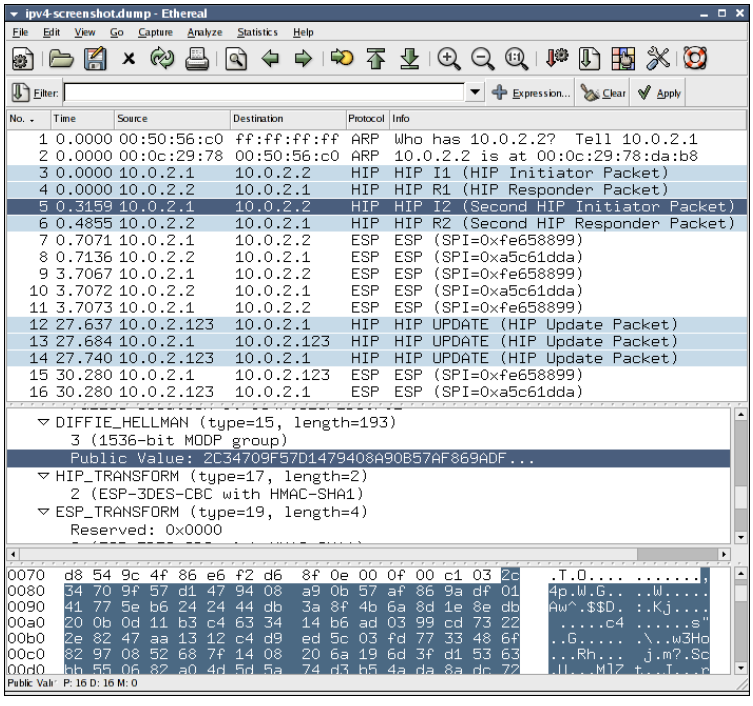


Figura 2-2 – Aparência do software Ethereal.

Ao capturar os pacotes em uma rede, podemos verificar os protocolos envolvidos nessa comunicação e os conteúdos ASCII e binários de cada pacote.

Podemos aplicar um filtro de pacote nessa rede e capturar apenas os pacotes trocados de um equipamento específico ligado à essa rede.

2.2. Ferramenta para análise de protocolo

TCP/IP é uma abreviatura para o termo *Transmission Control Protocol/Internet Protocol Suíte,* ou seja, é um conjunto de protocolos onde dois dos mais importantes (o TCP e o IP) deram os seus nomes a essa arquitetura. O protocolo IP, base da estrutura de comunicação da Internet, é baseado no princípio da troca de pacotes. Os protocolos TCP/IP podem ser utilizados sobre qualquer estrutura de rede, desde as mais simples do tipo ponto a ponto até uma rede de pacotes complexa do tipo telefonia via satélite. A arquitetura TCP/IP realiza a divisão das funções do sistema de comunicação em estrutura de camadas, sendo as quatro camadas divididas em Aplicação, Transporte, Internet e Rede (MOKARZEL, 2004). Abaixo a Figura 2-3 ilustra as camadas dessa arquitetura.

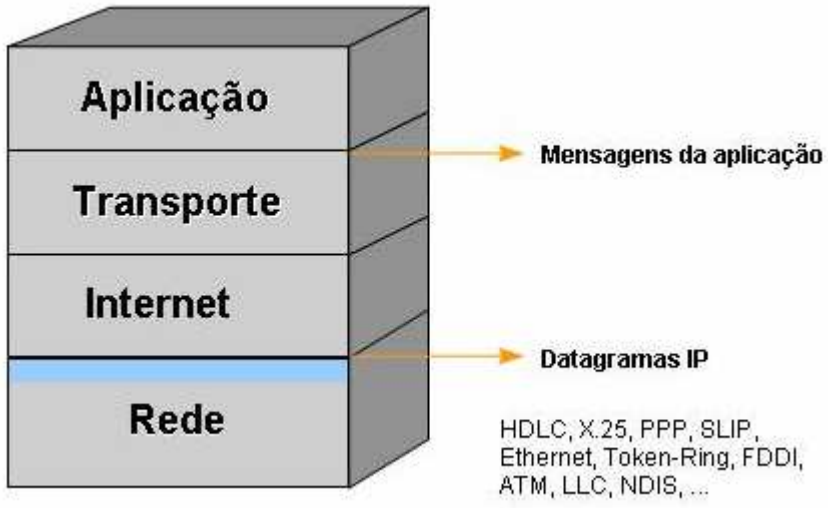


Figura 2-3 – Camadas do protocolo TCP/IP.

A Camada de Rede é responsável pelo envio de datagramas construídos pela Camada Internet. Essa camada realiza também o mapeamento entre um endereço de identificação de nível Internet (endereço IP) para um endereço físico ou lógico do nível de rede (*Mac Address*).

A Camada Internet é independente do nível de rede. Os protocolos deste nível possuem um esquema de identificação das máquinas interligadas por este protocolo. Por exemplo, cada máquina situada em uma rede Ethernet possui um identificador único chamado endereço MAC ou endereço físico que permite

distinguir uma máquina da outra, possibilitando o envio de mensagens específicas para cada uma delas.

A Camada Internet realiza a comunicação entre máquinas vizinhas através do protocolo IP. Para identificar cada máquina e a própria rede onde estas estão situadas, é definido um identificador, chamado de endereço IP, que é independente de outras formas de endereçamento que possam existir nos níveis inferiores.

O protocolo IP realiza a função mais importante desta camada que é a própria comunicação Internet. Para isto ele realiza a função de roteamento que consiste no transporte de mensagens entre redes e na decisão de qual rota uma mensagem deve seguir através da estrutura de rede para chegar ao destino. Esse protocolo também utiliza a própria estrutura de rede dos níveis inferiores para entregar uma mensagem destinada a uma máquina que está situada na mesma rede que a máquina origem. Para enviar mensagens para máquinas situadas em redes distintas, utiliza a função de roteamento IP. A mensagem é enviada para uma máquina com a função de *router* e essa, por sua vez, repassa para outros *routers* ou para a própria máquina de destino conforme Figura 2-4.

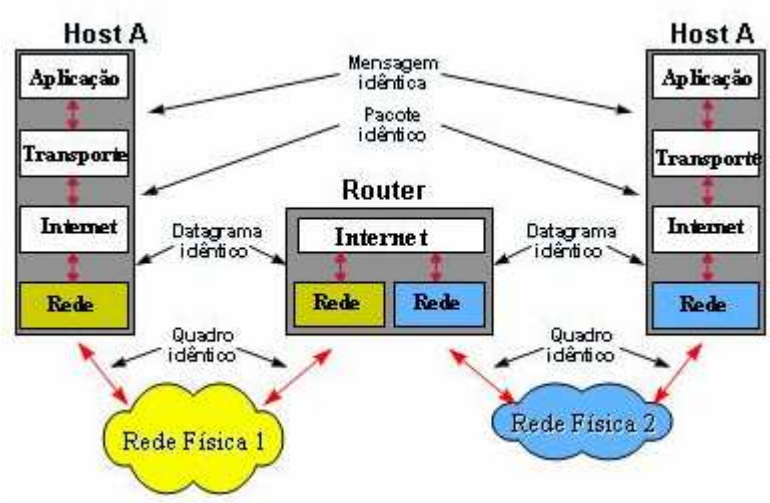


Figura 2-4 – Mapa de comunicação do protocolo IP.

A Camada de Transporte reúne os protocolos que realizam as funções de transportar os dados fim a fim, ou seja, considerando apenas a origem e o destino da comunicação, sem considerar os elementos intermediários. Essa camada possui dois protocolos, o UDP (*User Datagram Protocol)* e o TCP *(Transmission Control Protocol).* Essa camada permite acessar um conjunto de funções que possibilita criar e acessar um conjunto de aplicações para criar as interfaces (*sockets)* independentes do sistema operacional utilizado.

A camada de transporte TCP tem a função de dar robustez ao sistema e garantir que as mensagens enviadas serão recebidas. Para isso, o protocolo TCP tem como característica a capacidade de transferir um fluxo contínuo de octetos de uma estação a outra, empacotando esses octetos em grupos que são transportados pela camada Internet. O TCP cuida de recuperar dados que foram danificados, perdidos duplicados ou recebidos fora de ordem. Isso se dá pelo seqüenciamento dos pacotes e confirmação de recebimento. Além disso, há o controle de fluxo de envio através do retorno de uma "janela" em cada mensagem de confirmação.

O protocolo UDP realiza apenas a multiplexação para que várias aplicações possam aceder ao sistema de comunicação de forma coerente. Trata-se de sistema simplificado e sem confiabilidade. Já o protocolo TCP, além da multiplexação, realiza uma série de funções para tornar a comunicação entre origem e destino mais fiável. Dentre essas funções, destacamos o controle de fluxo, controle de erro, seqüência e multiplexação das mensagens.

A Camada de Aplicação reúne os protocolos que fornecem serviços de comunicação ao sistema ou ao utilizador. Podem-se separar os protocolos de aplicação em protocolos de serviços básicos ou protocolos para o utilizador. Esses protocolos fornecem serviços para atender as próprias necessidades do sistema de comunicação TCP/IP: DNS, BOOTP, DHCP.

2.3. Rede Ethernet

Ethernet (IEEE 802.3) é a camada de rede mais popular para redes LAN. Ela é facilmente reconhecida pelo cabo azul UTP categoria 5e conforme Figura 2-5 abaixo.

Figura 2-5– Cabos UTP categoria 5e com conectores RJ45.

O IEEE 802.3 é uma coleção de *Standards* que especificam as camadas física e sub-camada MAC da camada de ligação de dados do Modelo OSI para o

protocolo Ehternet, define a banda máxima possível, as interfaces físicas e o formato do frame. A rede Ethernet é subdividida em quatro subcamadas: Especificação de mídia, subcamada física (PHY), subcamada de controle de acesso à mídia (MAC 802.3), subcamada de controle lógico do link (LLC) (MOKARZEL, 2004).

As subcamadas físicas normalmente possuem um componente dedicado, também chamado de PHY. No caso desse projeto, foi utilizado o DP83848I da *National Semiconductor*.

Os dados que trafegam pela Ethernet são encapsulados da seguinte forma:

Preâmbulo: Constituído de uma seqüência de 64 bits com a função de iniciar a comunicação. Essa seqüência de bits é usada para ajustes dos circuitos, principalmente o sincronismo do PLL.

Destino: o endereço físico do receptor do pacote.

Origem: o endereço físico do remetente do pacote.

Tipo/Tamanho: normalmente informa o tipo do protocolo de nível mais alto transportado por esse pacote. Em alguns casos ele contém informação sobre o tamanho.

Dados: São os dados que devem ser transportados pela Ethernet e foram fornecidos pela camada Internet.

FCS: A seqüência de bits responsável pela verificação da integridade do quadro. É gerado pela aplicação de um polinômio padrão aos bits que compõem o quadro.

Ao receber um pacote, a camada Ethernet verifica o FCS e, caso ele esteja correto, verifica o endereço de destino que deve ser o de *Broadcast* (ff:ff:ff:ff:ff:ff) ou o seu próprio endereço. Então, baseado no Tipo/Tamanho, envia o pacote para a camada superior.

O protocolo de resolução de endereço ARP é normalmente empregado nas redes baseadas em Ethernet e traduz o endereço lógico IP para o endereço físico MAC.

Basicamente, sempre que uma estação sempre que uma estação envia uma mensagem para outra, o protocolo ARP é utilizado. Todas as estações dessa rede recebem e tratam essa mensagem, mas apenas a que possuir o endereço IP solicitado retorna outra mensagem com o seu endereço MAC.

O comando Ping é utilizado pelo protocolo ICMP para testar a conectividade entre dispositivos, desenvolvido para ser usado em redes com a pilha de protocolo TCP/IP, como a internet. Seu funcionamento consiste no envio de pacotes atraves do protocolo ICMP para o equipamento de destino e no recebimento da resposta do equipamento. Se o equipamento de destino estiver ativo, uma resposta Pong (analogia ao famoso jogo de ping-pong) é devolvida ao computador solicitante. Dessa forma, podemos testar se há conectividade com qualquer equipamento e até sites, pois estão alocados em um servidor(MOKARZEL, 2004). Na presente aplicação, esse comando foi utilizado para detectar a conectividade de módulo na rede, isso é possível em redes com *hubs* e *switches* ou diretamente na rede mundial de computadores. A Figura 2-6 mostra o comando sendo utilizado diretamente com o módulo através do endereço de IP que foi configurado no módulo, 192.168.0.100.

Figura 2-6– Tela da utilização do comando Ping.

2.4. Formação do Socket

Para entender o protocolo TCP, é necessário visualizar o IP também, pois grande parte das informações está contida ali. Cada conexão TCP entre um cliente e um servidor possui um sistema de controle conhecido como *socket*. Cada *socket* é formado por um par IP/Porta do cliente e IP/Porta do servidor. Por meio desse *socket,* o protocolo permite a multiplexação, pois o servidor pode atender diversos IPs ao mesmo tempo, alocando um *socket* para cada um, além disso, um mesmo IP (cliente) pode possuir diversas conexões com um servidor usando portas distintas (MOKARZEL, 2004).

Para permitir que uma única estação use diversas comunicações TCP simultâneas, o protocolo fornece uma numeração de porta, que combinada com a rede e o endereço IP, forma um *socket.* Cada *socket* é único permitindo assim a execução de múltiplas conexões.

O ISN (*Initial Sequence Number*) trata-se do número seqüencial para iniciar uma conexão. Para que as estações conheçam os ISN uma da outra, toda conexão lógica inicia com o comando de sincronização (SYN). Esse processo é conhecido como cumprimento de três vias e caracteriza o estabelecimento de uma conexão TCP. O fechamento de uma conexão também é parecido, porém com a utilização de um bit finalizador que deve ser respondido pelo cliente. Após o envio da mensagem de finalização, nenhuma outra mensagem será aceita pelo receptor.

2.5. Sistema Operacional de Tempo Real

No presente projeto é utilizado uma rotina onde se implementa um Sistema Operacional de Tempo Real, RTOS da sigla anglo-saxônica *Real Time Operationing System,* é um sistema operacional/operativo destinado à execução de múltiplas tarefas onde o tempo de resposta a um evento, externo ou interno, é pré-definido não importando se a velocidade de resposta é elevada ou não. Esse tempo de resposta é chamado de prazo da tarefa e a perda de um prazo, isto é, o não cumprimento de uma tarefa dentro do prazo esperado, caracteriza uma falha do sistema. Outra característica dos sistemas de tempo real é a sua interação com o meio ao redor. Os STRs tem que reagir, dentro de um prazo pré-definido, à um estímulo do meio. Por exemplo, em um hospital, o sistema que monitora os batimentos cardíacos de um paciente deve alarmar os médicos caso haja alteração nos batimentos. Outro aspecto importante dos STRs é a previsibilidade. O sistema é considerado previsível quando podemos antecipar seu comportamento independentemente de falhas, sobrecargas e variações de hardware.

Uma implementação de software de um RTOS facilita a concepção de um sistema em tempo real, mas não garante que o resultado final seja um sistema de tempo real, para tal é necessário que o programa nele implementado tenha sido corretamente desenvolvido. Um RTOS não tem que ter necessariamente um elevado número de saídas, no entanto, tem que garantir que certas tarefas sejam executadas em um determinado intervalo de tempo. Um RTOS é mais eficaz e é mais valorizado pela forma previsível e rápida na resposta a um evento, do que pela quantidade de dados que processa. Os fatores chave em um STR são, então, fornecer latências de interrupções e de alternância de tarefas mínimas.

3. DESCRIÇÃO DO MÓDULO

Os sinais elétricos a serem lidos são dos tipos digitais e analógicos. No caso dos digitais serão provenientes de botões, sensores, dispositivos específicos, relés e toda sorte de equipamentos que possibilitem o envio de um sinal elétrico para montar uma informação e enviar. Para os sinais analógicos, tem-se a possibilidade de ler tensões variáveis de 0 a 10 Vcc podendo ser provenientes de diversos equipamentos, tais como servomotores, inversores de freqüência, células de carga e torque, transdutores lineares e potenciométricos e qualquer outro tipo de equipamento que envie esse tipo de sinal. Os comandos digitais enviados ao módulo possibilitam acionar relés, dispositivos, sinaleiros e toda a sorte de cargas dentro do limite de capacidade das saídas.

Para efetuar a alimentação do módulo, utilizou-se uma fonte retificadora chaveada, marca Murr com alimentação de entrada de 90 a 240 Vac. Na saída dessa fonte temos 24 Vcc e uma corrente de até 2,5 A. Para proteção do sistema, tanto na entrada quanto na saída da fonte foram utilizados disjuntores termomagnéticos, curva C, com 2 e 4 A, respectivamente. Para acionar as entradas digitais foram utilizados 8 botões com contatos normalmente abertos. Por questões de proteção, todas as saídas digitais acionam relés opto acopladores marca Phoenix Contact cujo acionamento é realizado com 24 Vcc. As cargas ligadas a estes relés são sirenes e sinaleiros para indicações sonoras e visuais. A Figura 3-1 mostra a foto do módulo.

Posteriormente é feito o tratamento e o encapsulamento desses sinais para, através do protocolo Ethernet serem transportados com rapidez e confiabilidade entre cliente e servidor. O objetivo maior é possibilitar a interação entre o *software* desenvolvido no módulo e o *software* desenvolvido no computador que servirá como servidor para, dessa forma, ler as entradas e acionar as saídas remotamente.

Figura 3-1– Foto do módulo CP-FLEX

3.1. Especificações técnicas

Inicialmente foi feita a escolha do microprocessador. Essa escolha se baseou nas características técnicas que atenderam de forma completa todas as necessidades para a aplicação desejada além de permitir futuras melhorias em nível de redes e aplicativos.

O microprocessador escolhido foi o LPC2368, conforme Figura 3-2 abaixo, da Philips, padrão ARM7 e com Ethernet incorporada. Esse chip possui um processador que roda acima de 72 MHz, memória flash de programação de 512 kB nos sistemas ISP e IAP, 32 kB de SRAM sobre o barramento ARM diretamente para permitir a CPU com alta performance, 16 kB de RAM estática para a interface Ethernet, 8 kB de RAM estática para a interface USB, sistema duplo AHB para atuação simultânea do USB DMA e do Ethernet DMA, controle avançado para interrupção de até 32 vetores, interface serial, interface Ethernet MAC com controle DMA associado, interface USB 2.0, porta direta para PHY, dois canais de interface CAN, controle SPI, possui interface para acréscimo de cartão SD, pinos de entradas e saídas configuráveis, 8 entradas conversoras da analógico para digital de 10 bits, quatro entradas de contagem rápida, um bloco para PWM/Timer, relógio em tempo real com pinos separados, temporizador do tipo *watchdog*, alimentação de 3,0 a 3,6 Volts, cristal de 1 a 24 MHz, oscilador interno, além de muitas outras funções.

Figura 3-2– Microprocessador LPC 2368.

Após a definição do microprocessador, buscou-se um módulo pronto devido à dificuldade de aquisição dos componentes eletrônicos no mercado nacional e ao tipo de solda ser SMD o que inviabiliza esse procedimento manual, pois se corre muitos riscos de erro. Depois de uma análise envolvendo custos e benefícios, chegamos ao produto CP-FLEX, conforme Figura 3-3 abaixo, fabricado pela empresa MCU. Esse módulo possui todo o *hardware* necessário e vem pronto para criar a aplicação de *software*.

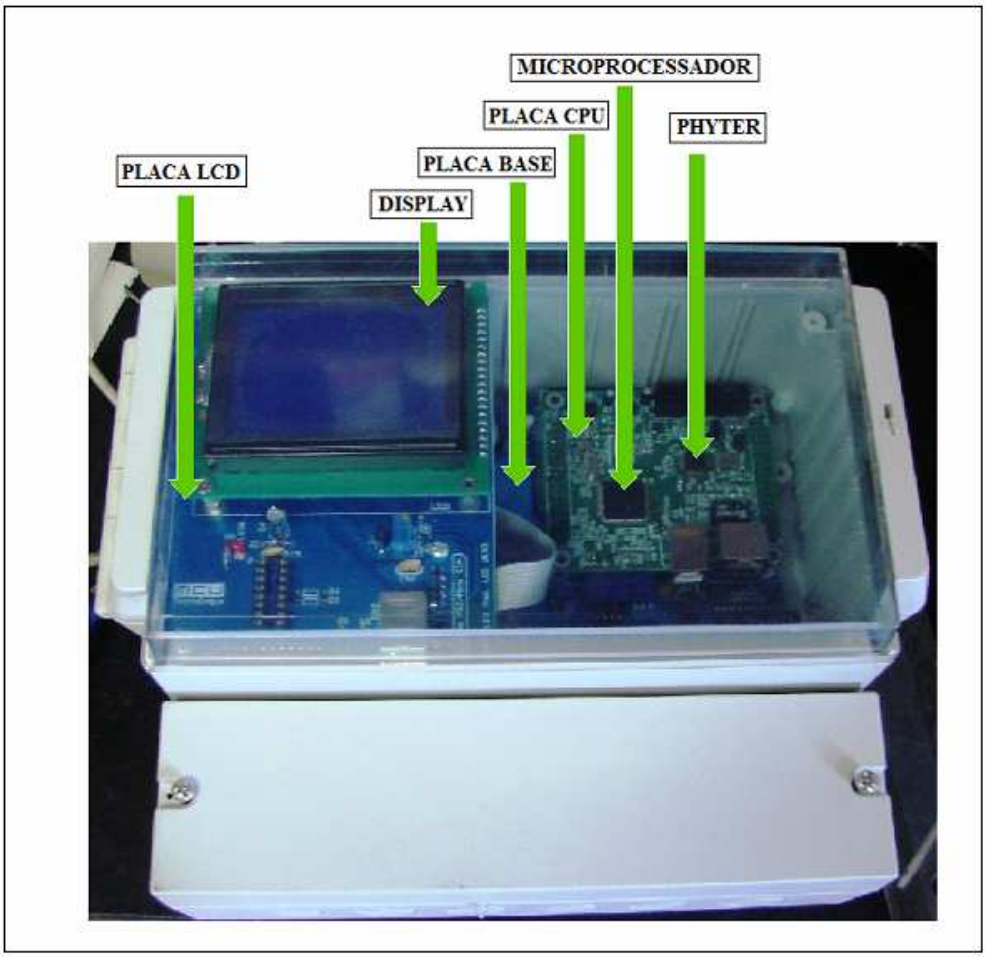


Figura 3-3 – Foto do módulo CP-FLEX.

O módulo escolhido é de um equipamento de controle programável, possui uma CPU microcontrolada, com um processador de 32 bits baseado na arquitetura ARM7 LPC2368.

Constituído por um display LCD 128 x 64 pixels, teclado de membrana, porta CAN, 8 saídas digitais a relé, 8 saídas digitais optoisoladas, 8 entradas digitais optoisoladas e 3 entradas analógicas não isoladas, 1 saída analógica configurada em 0 a 10 Vcc, porta Ethernet 10/100Mbps, porta USB, 2 portas seriais, 1 porta RS485 configurável como mestre ou escrava, entrada de teclado padrão PS2 e entrada para cartão SD.

O *hardware* do CP-FLEX está dividido em 3 placas, sendo a placa base onde estão os relés, bornes e acopladores óticos. Temos também a placa LCD que é a placa superior onde se encontra o display e a placa CPU onde se encontra o microprocessador LPC 2368 e todos os periféricos específicos para ela, além dos conectores RJ45 para a porta Ethernet, USB tipo B e para o cartão SD.

A ligação dos sinais no CP-FLEX é feita através dos bornes localizados na tampa inferior da caixa. Existem duas fileiras de bornes, fileira K com bornes verdes de encaixe e fileira Z com bornes azuis fixos conforme Figura 3-4.

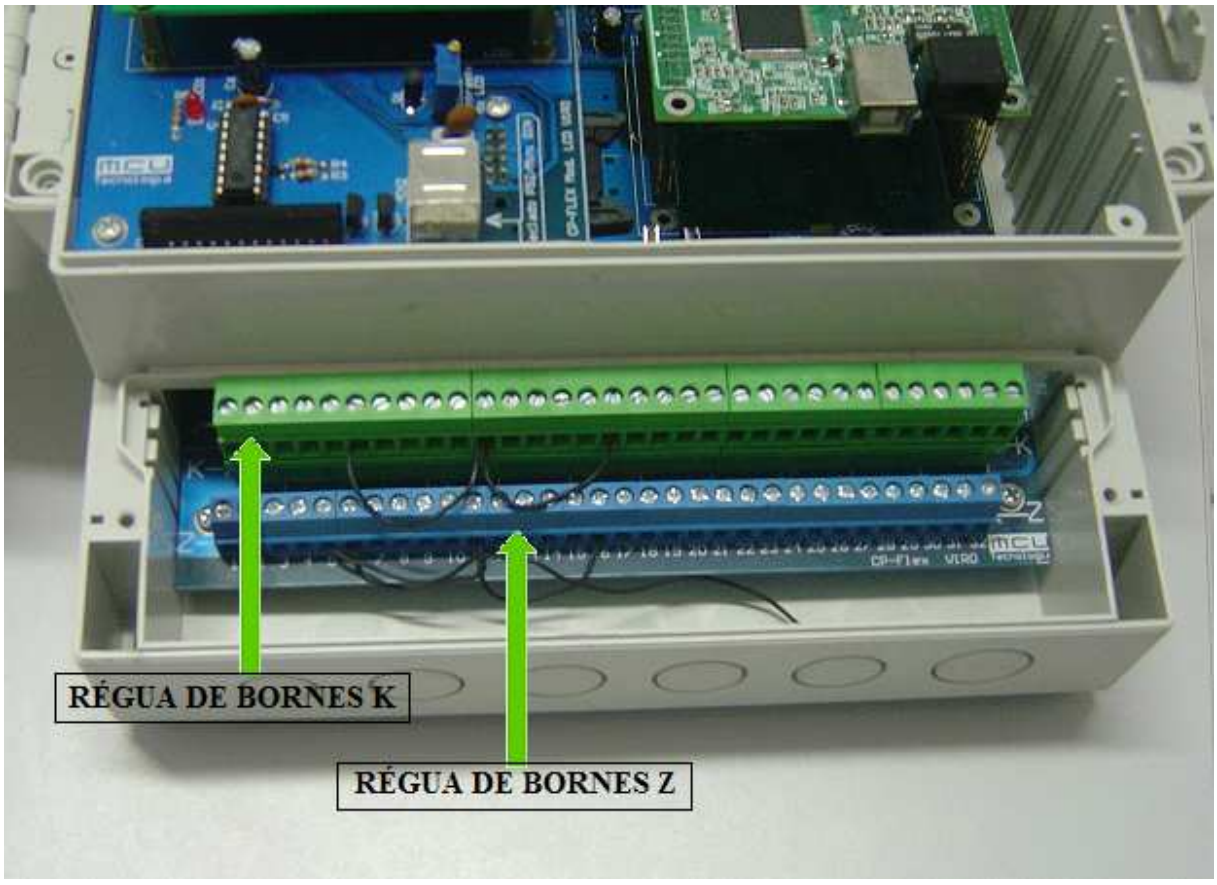


Figura 3-4 – Teclado de membrana do módulo não utilizado nesse projeto.

A ligação do teclado PS2 é feita abrindo a tampa superior e conectando o teclado na placa superior onde se encontra o display de LCD.

As ligações da porta Ethernet, USB e cartão de memória SD são feitas na placa da CPU, que está conectada à placa base, via barra de pinos.

O consumo do módulo, alimentado em 24Vcc, com todas as saídas desligadas é 140 mA e com as saídas ligadas é 320 mA.

As saídas do produto possuem um sistema chamado *Watchdog* que protege os acionamentos indesejados durante o reset ou durante falha de energia. Esse sistema interrompe o processador no caso do software trancar. Todas as saídas são desligadas automaticamente caso o programa trave por algum motivo.

Seu teclado de membrana possui 28 teclas, conforme Figura 3-5, que podem ser usadas para a entrada de números ou textos. Ambos os teclados podem ser utilizados ao mesmo tempo. Esse teclado não está sendo utilizado nesse projeto.

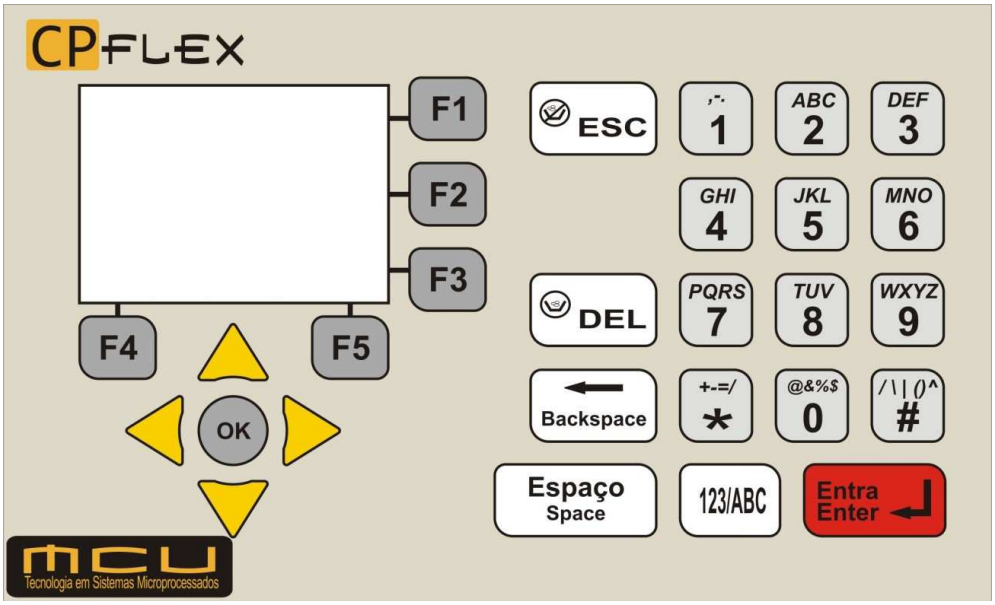


Figura 3-5 – Teclado de membrana do módulo não utilizado nesse projeto.

3.2. *Entradas e saídas digitais*

O CP-FLEX possui 8 saídas digitais. Essas saídas estão divididas em dois grupos de 4 saídas, ou seja, pode-se trabalhar com duas fontes diferentes sendo uma em cada grupo. Essa possibilidade é de grande valia para cargas com acionamentos em tensões diferentes.

Se o sistema não precisa de fonte isolada, ou seja, se for utilizada a mesma fonte de alimentação do CP-FLEX para ativar as cargas, pode-se utilizar os conectores existentes próximo aos bornes de saída digital para ligar o comum de

uma das saídas ao de ambas à alimentação positiva do CP-FLEX, dessa forma não se utiliza o borne comum.

Se o conector JP4 estiver inserido, o comum das saídas 0 até 3 é ligado às entradas positivas de alimentação do CP-FLEX. Se o conector for removido, o comum não fica ligado em nada e precisa de uma referência externa.

Se o conector JP5 estiver inserido, o comum das saídas 4 até 7 é ligado às entradas positivas de alimentação do CP-FLEX. Se o conector for removido, o comum não fica ligado em nada e precisa de uma referência externa.

O sinal comum das saídas deverá ser positivo, por exemplo, 12 ou 24 Vcc. E quando a saída for ativada, esse sinal será utilizado para alimentar a carga que estiver ligada na saída, ou seja, será enviado um sinal positivo para as saídas ativadas. Essa ligação é mostrada na Figura 3-7.

O CP-FLEX possui 8 entradas digitais. Essas entradas estão divididas em dois grupos de 4 saídas, ou seja, pode-se trabalhar com duas fontes diferentes sendo uma em cada grupo. Essa possibilidade é de grande valia para dispositivos de entrada com acionamentos em tensões diferentes.

Se o sistema não precisa de fonte isolada, ou seja, se for utilizada a mesma fonte de alimentação do CP-FLEX para os sinais de entrada, pode-se utilizar os conectores existentes próximo aos bornes das entradas digitais para ligar o comum de uma das entradas ao de ambas à alimentação negativa, denominada GND, do CP-FLEX, dessa forma não se utiliza o borne comum.

Se o conector JP6 estiver inserido, o comum das entradas 0 até 3 é ligado às entradas negativas de alimentação do CP-FLEX. Se o conector for removido, o comum não fica ligado em nada e precisa de uma referência externa.

Se o conector JP7 estiver inserido, o comum das entradas 4 até 7 é ligado às entradas negativas de alimentação do CP-FLEX. Se o conector for removido, o comum não fica ligado em nada e precisa de uma referência externa.

O sinal comum das entradas poderá ser negativo e o sinal deverá ser positivo do tipo 12 ou 24 Vcc. Esse sinal nunca pode ser inferior a 7 Vcc ou superior a 24Vcc. O esquema de ligação é mostrado na Figura 3-6.

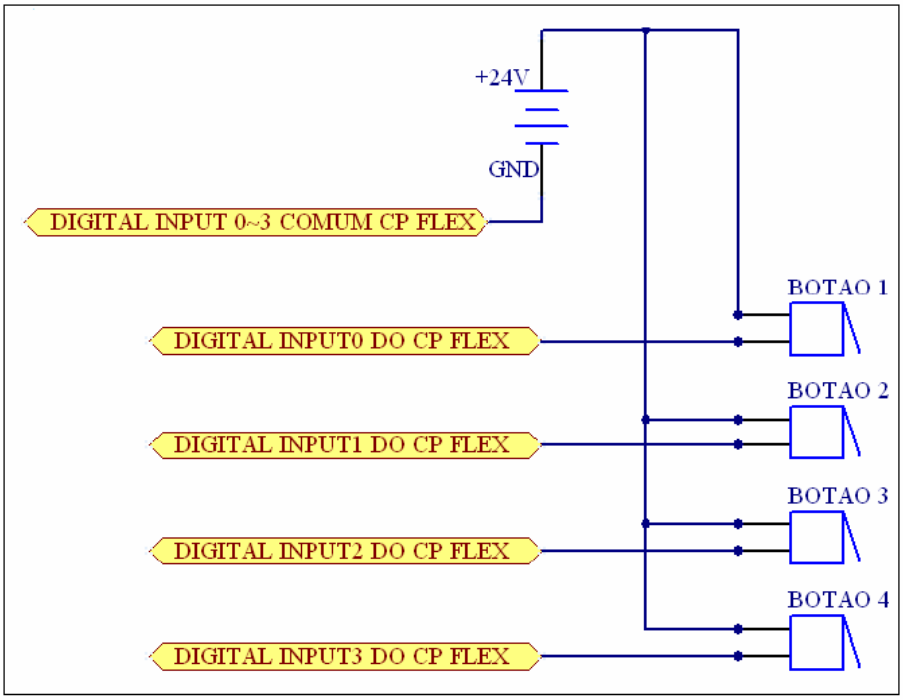


Figura 3-6 – Esquema de ligação para as entradas do CP-FLEX

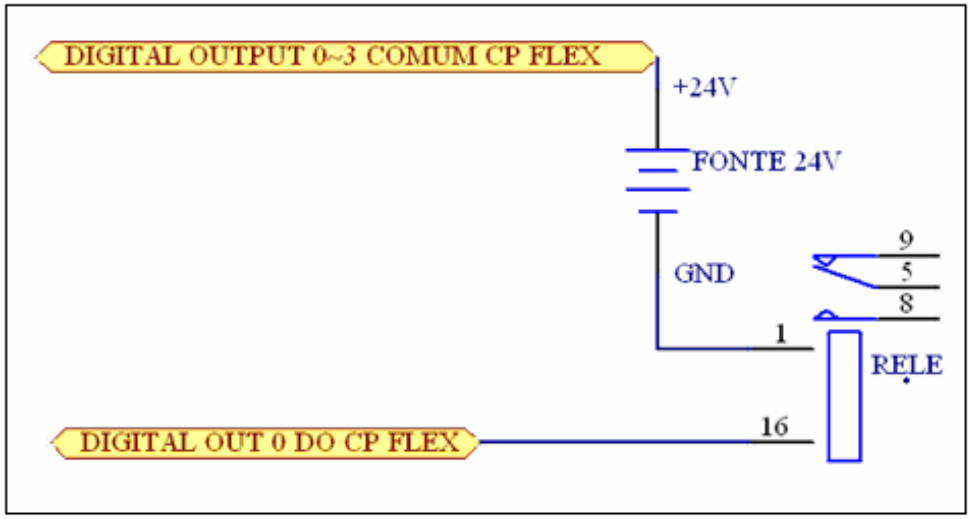


Figura 3-7 – Esquema de ligação para as saídas do CP-FLEX.

4. DESENVOLVIMENTO DA APLICAÇÃO

Para demonstrar o funcionamento do sistema, uma aplicação simulando a realidade foi desenvolvida. Fundamentado no conceito cliente-servidor, onde o servidor é o computador e o cliente seria o módulo, criou-se o sistema a seguir descrito. Trata-se da leitura dos sinais analógicos e da resposta através de uma sinalização visual e sonora. Ao inserir os parâmetros com os valores máximos e mínimos de cada entrada analógica criou-se uma janela para cada leitura. Se o valor lido for maior que o máximo programado, um sinal sonoro é emitido, acionado através de uma das saídas digitais. Caso o valor fique abaixo, um sinaleiro vermelho é acionado. Por fim, caso o valor esteja dentro da janela, um sinaleiro verde é acionado. Essas respostas também serão mostradas na tela do servidor, tanto as saídas e entradas digitais quanto a leitura das entradas analógicas. Nessa mesma tela, além de mostrar os níveis lógicos de cada saída, ainda poderemos acionar as saídas do módulo individualmente.

4.1. Servidor

Para o servidor do sistema, foi criada uma aplicação em Visual Basic. Trata-se de uma linguagem de programação orientada a eventos. Isto significa que todas as ações que ocorrem durante a execução do programa são estruturadas nos eventos dos objetos. Os comandos usados no Visual Basic são basicamente os mesmos usados no Basic, com a diferença de que foram ampliados para satisfazer as necessidades de uma aplicação voltada para ambientes gráficos.

Dessa forma, o objetivo seria criar uma aplicação interagindo com o módulo através de um arquivo executável que abriria uma janela de interface com os sinais do módulo. Dessa forma poder-se-ia interagir com o módulo lendo e escrevendo rotinas a fim de gerar eventos pré-definidos. A Figura 4-1 mostra a tela inicial do Visual Basic 2008 versão SP1.

Figura 4-1– Tela inicial do Visual Basic.

No servidor foi criada uma aplicação onde são enviados e recebidos pacotes vai rede Ethernet, esses pacotes são encapsulados dentro do protocolo TCP/IP. Essa troca de informações acontece devido ao envio de *strings*, que são um conjunto de caracteres ordenados, solicitando informações entre os sistemas. A figura 4-2 mostra a tela inicial de apresentação desse projeto.

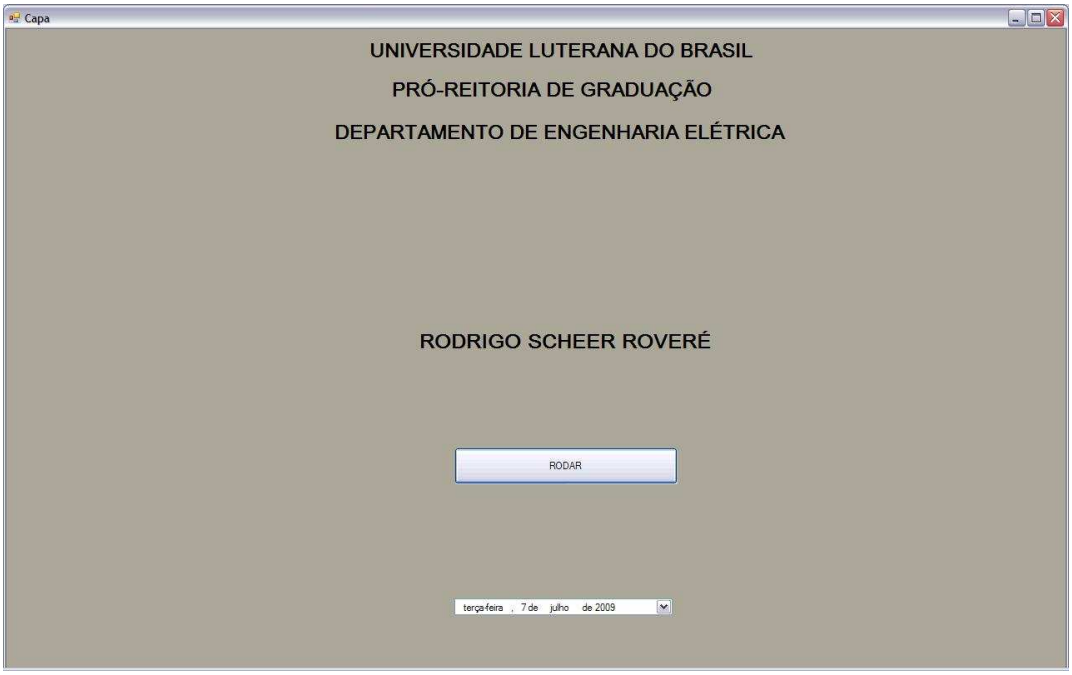


Figura 4-2 – Tela inicial de apresentação da aplicação.

A tela principal do projeto é mostrada na Figura 4-3. Na parte superior esquerda tem-se um campo onde é possível inserir o endereço IP no qual se deseja

conectar. Logo ao lado há o botão Conectar que faz a conexão do servidor com o cliente. Logo abaixo, existe uma fileira com oito botões que servem para acionar e desacionar as saídas a relé do módulo. A segunda fileira de oito botões possibilita acionar ou desacionar as saídas digitais transistorizadas. Logo abaixo, tem-se uma fileira que serve como indicadora para as oito entradas digitais, conforme as entradas são acionadas a cor muda. Na parte inferior da tela temos uma listagem com os eventos do sistema, tais como comandos, conexões e desconexões. Ao lado, um botão permite apagar essa listagem.

No lado direito da tela são mostradas as três leituras analógicas do módulo, os campos editáveis permitem que sejam inseridos os limites inferior e superior permitidos de cada leitura, sinalizadores de atenção, perigo e leitura boa completam a tela da aplicação.

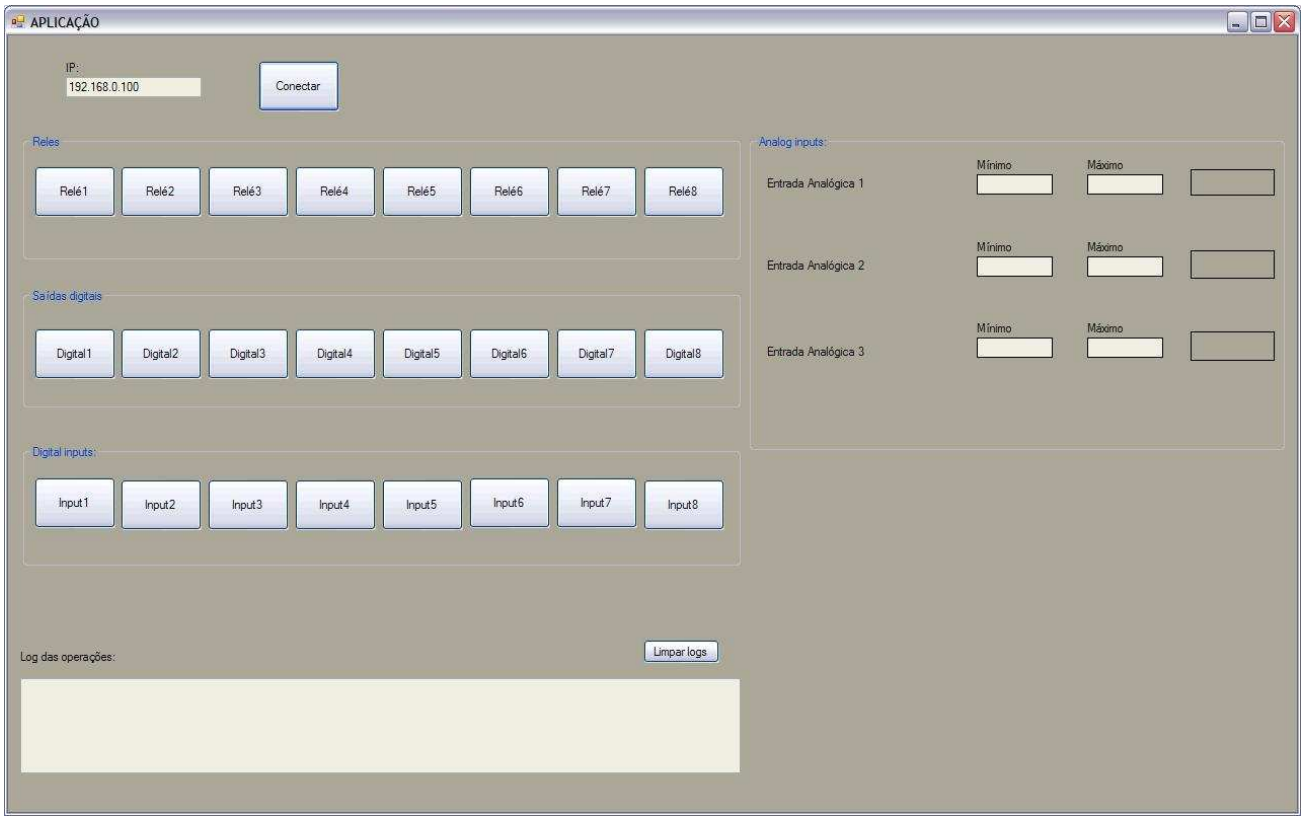


Figura 4-3 – Tela principal da aplicação

Ao acionar as teclas no supervisório, uma *string* é criada e enviada ao cliente. Esse sinal é tratado e interpretado pelo cliente e a ação solicitada é realizada. O formato dessa *string* é dividido em objeto, número do objeto e estado. Por exemplo, ao acionar o botão Relé 1, envia-se a seguinte mensagem para o cliente: RL;1;1. Isto quer dizer que se quer acionar o relé 01. O RL chama um relé, o primeiro 1 chama o

número desse relé e o segundo número chama o estado que é ON = 1. Para desligar o mesmo relé, ao precionar-se o mesmo botão novamente a *string* muda para RL;1;0, pois OFF = 0. As informações sempre são transmitidas separadas por ponto e vírgula (;). O mesmo acontece para as leituras e demais escritas.

4.2. Cliente

Para programar o módulo, foi desenvolvida uma aplicação em linguagem C dentro de um compilador da empresa Keil chamado µVison, versão 3.20, conforme Figura 4-4. Trata-se de uma linguagem de programação compilada de propósito geral, estruturada, procedural, de alto nível e padronizada, criada em 1972, por Dennis Ritchie, no AT & T Bell Labs, para desenvolver o sistema operacional UNIX, que foi originalmente escrito em linguagem Assembly. A linguagem C é classificada de alto nível pela própria definição desse tipo de linguagem. A programação em linguagem de alto nível tem como característica não ser necessário conhecer o processador, ao contrário das linguagens de baixo nível. As linguagens de baixo nível estão fortemente ligadas ao processador. A linguagem C permite acesso de baixo nível com a utilização do código Assembly no meio do código fonte. Assim, o baixo nível é realizado por Assembly e não C. Desde então, espalhou-se por muitos outros sistemas, e tornou-se uma das linguagens de programação mais utilizadas influenciando muitas outras linguagens, especialmente C++, que foi originalmente desenvolvida como uma extensão para o C.

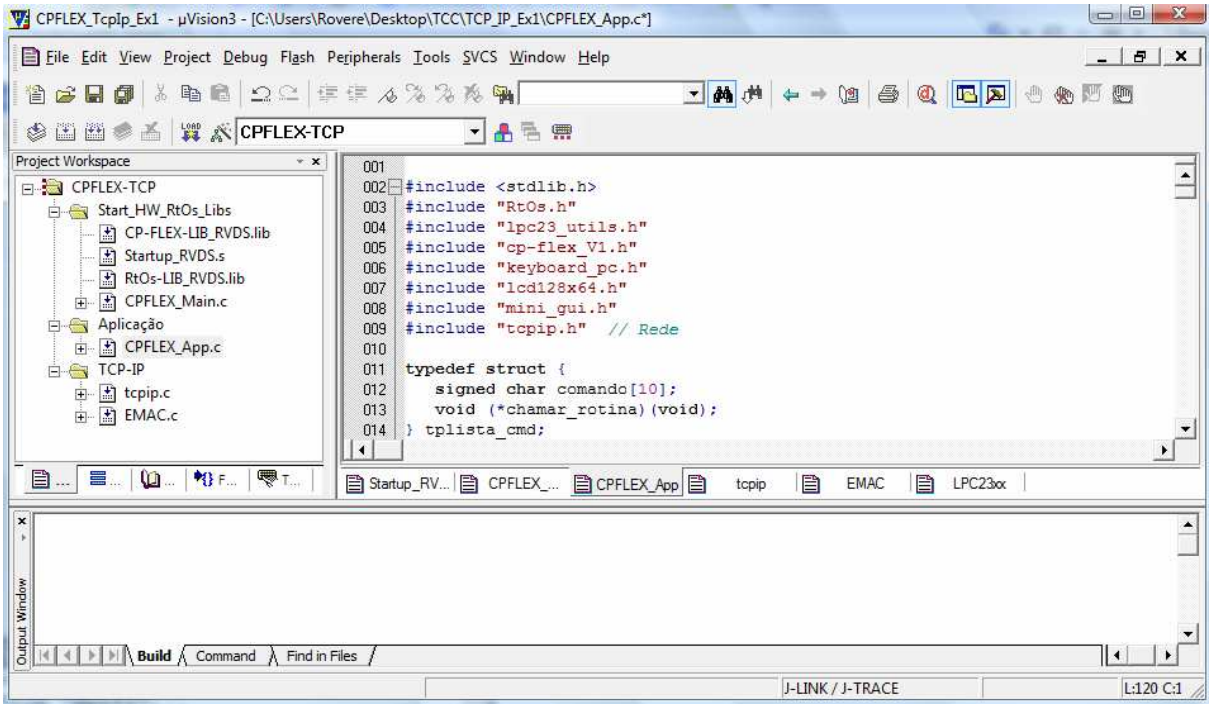


Figura 4-4 - Tela de apresentação do µVision.

Na figura acima é demonstrado que o software foi estruturado em diretórios e arquivos específicos. O diretório principal é denominado de CPFLEX-TCP e trata-se do nome geral do projeto. Dentro dessa pasta devem estar todos os demais arquivos para o correto funcionamento da aplicação.

A primeira pasta dentro do diretório principal é a Start_HW_RtOs_Libs e é onde se encontram todos os arquivos necessários para a inicialização do módulo. Pode-se notar que há duas bibliotecas facilmente identificadas pela extensão .lib. Essas duas bibliotecas são responsáveis pelo carregamento dos arquivos que inicializam a aplicação. As bibliotecas CP-FLEX-LIB_RVDS.lib e RtOs-LIB_RVDS.lib são responsáveis pela padronização de aplicativos que podem ser utilizados em outras aplicações. No caso da primeira biblioteca, são aplicativos comuns ao módulo CP-FLEX e no caso da segunda, são aplicativos comuns para a utilização de sistemas RTOS.

A seguir, temos o arquivo Startup_RVDS.s que nada mais é que uma rotina em Assembler que inicializa o microprocessador. Ela acerta vários parâmetros de velocidade, PLLs e espaço para a pilha.

O próximo arquivo é o programa CPFLEX_Main.C. Essa rotina e seus arquivos secundários são responsáveis pela inicialização do processador, controle dos relés, controle das saídas digitais, controle das entradas digitais, leitura dos sinais analógicos, rotinas de controle do display gráfico como desenhos e fontes, pilha TCP/IP com Sockets, rotinas de configuração das portas de comunicação serial e RS-485, rotinas de geração de pulsos nas saídas digitais e rotinas de contagem de pulsos das entradas digitais.

A seguir, temos uma pasta chamada Aplicação que contém o programa da aplicação desenvolvida para esse trabalho. Dentro da rotina principal, CPFLEX_APP.c, temos diversos aplicativos que chamam as rotinas principais para execução da aplicação. Essas rotinas são responsáveis pela interface entre o servidor e o cliente, criação de buffer para armazenagem dos dados, verificação do tamanho do dado recebido, lê as portas do CP-FLEX e devolve em um formato separado por ponto e vírgula, aplica a rotina dos sinais digitais e analógicos, executa o pacote de dados, separa os argumentos, inicializa o TCP/IP, verifica se tem dados no buffer, esvazia o buffer e efetua o fluxo dos sinais trocados entre cliente e servidor.

Por fim, temos a pasta TCP-IP que possui duas rotinas para a comunicação. A primeira delas é a tcpip.c que possui alguns aplicativos responsáveis por incluir

as bibliotecas de rede, traduzir os pacotes do *hardware* para TCP/IP e permitir a aceitação de apenas uma comunicação por vez. Trata-se de uma pilha TCP simples e de protocolo aberto.

Dentro da rotina EMAC.c existem alguns aplicativos que, dentre outras coisas, traduz a camada MAC para o microprocessador LPC2368. Trata-se de uma pilha MAC simples e de protocolo aberto.

4.3. Testes para validação da aplicação

Para a validação do sistema foram realizados diversos tipos de testes individuais tanto em rede quanto diretamente nas entradas e saídas do módulo. Por fim foi desenvolvida uma aplicação onde são inseridos valores máximos e mínimos para as leituras analógicas criando, desse modo, janelas para cada uma das três entradas analógicas. Para os valores que ficam fora dessas janelas, sinaleiros vermelhos e um sinalizador sonoro são acionados para avisar ao operador. Para os valores considerados aceitáveis, sinalizadores luminosos verdes são acionados para aviso ao operador.

A parte de leitura de sinais no módulo cliente apresentou um comportamento dentro do esperado, pois no servidor podemos ler os valores analógicos e o nível lógico das entradas digitais de acordo com qual delas foi acionada. O mesmo ocorreu para as saídas quando o comando dado era para acionar qualquer uma delas, ao precionar-se qualquer dos botões os relés e trnsistores operaram como desejado.

Os testes realizados nas partes separadamente possibilitaram a validação de todos componentes e partes do sistema e ao rodar a aplicação principal todas essas partes interagiram a contento.

Por fim, para validação do sistema, ao colocar a ferramenta Ethereal na rede podemos verificar os pacotes trocados conforme Figura 4-5. Observa-se que a janela está dividida em três partes maiores. A primeira contém os diversos pacotes que passaram pela rede durante a captura, a segunda traduz os pacotes trocados, podendo ser visulizados um por um bastando selecionar a linha. Por fim, a terceira mostra o conteúdo binário e o ASCII do pacote. As três partes são integradas e sempre que se altera uma delas as outras refletem a alteração.

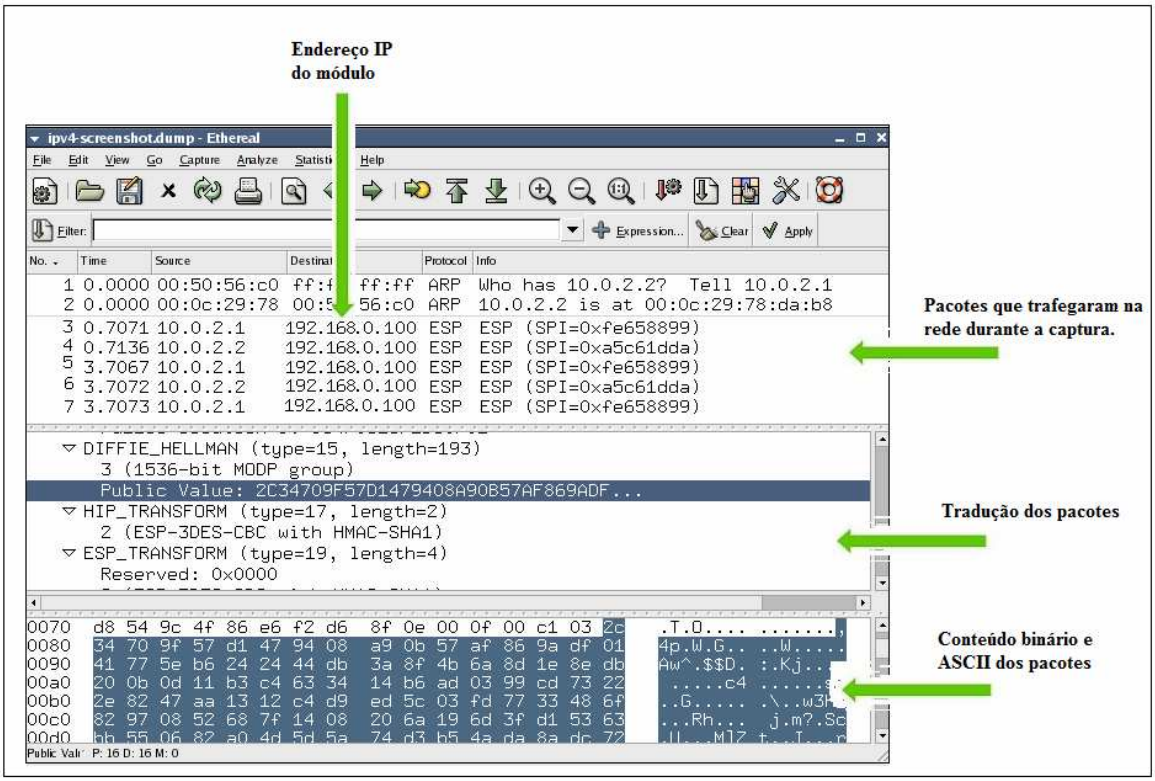


Figura 4-5 – Troca de pacotes visualizadas pelo software Ethereal.

5. CONCLUSÃO

O sistema desenvolvido funcionou da maneira esperada visto que tanto os sinais digitais quanto analógicos foram enviados e recebidos através da rede de forma rápida e confiável. Cabe ressaltar a necessidade de melhorias nos tempos de leitura e escrita a fim de tornar o sistema mais confiável para aplicações onde leituras instantâneas são importantes, além da inclusão das demais funcionalidades do módulo, tais como os teclados, cartão de memória e display gráfico que não foram utilizados no presente projeto.

Em face da pesquisa e dos experimentos realizados, foi possível alcançar conhecimentos nas áreas de redes, especificamente na rede Ethernet focada no protocolo TCP/IP. Em conjunto com desenvolvimento dos conhecimentos básicos ao longo do curso de Engenharia Elétrica. Conhecimentos que possibilitaram a manipulação das informações de maneira mais confiável e na obtenção de conclusões que fundamentam tais informes.

Como todo processo tecnológico necessita de constante aprimoramento, o projeto apresentado também pretende evoluir dentro de um contexto técnico-econômico visando aperfeiçoamentos para o desenvolvimento de um produto ou serviço de melhor qualidade. Com o foco em aplicações prediais e industriais visando o controle de máquinas, linhas de montagem, salas e prédios além de servir de interface para coleta de dados indicadores de produção visando fornecer índices de qualidade, o presente projeto sofrerá adequações de acordo com cada aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

MOKARZEL, Marcos Perez **Internet Embedded**: TCP/IP para Microcontroladores 1ª. Ed. – Érica, 2004

COCIAN, Luis Fernando Espinosa **Manual da Linguagem C** – 1ª. Ed. – Editora da Ulbra, 2004

HUBBARD, John R. **Theory and Problems of Programming With C** - 1a. Ed. – McGraw-Hill, 1996

SCHILDT, Herbert **C Completo e Total** – 3a. Ed. – Makron Books, 1997

KIMMEL, Paul **Advanced C Programming** – 2a. Ed. – Macgraw Hill, 2003

PHILIPS, **LPC 2364/6/8 User manual** – Rev. 01 – 2006

MCU, **KIT 2368 Manual do usuário**– Rev. 02 – 2008

MCU, **CP-FLEX Manual do usuário**– Rev. 03 – 2009

APÊNDICES:

APÊNDICE A – GRAVAÇÃO DO PROGRAMA NO MÓDULO

Para gravar o software desenvolvido no compilador, é necessária uma ferramenta de gravação. No presente projeto se escolheu o software Flash Magic, devido à interface amigável e a facilidade de utilização. Trata-se de um software desenvolvido pela empresa NXP Semiconductors baseado no ambiente Windows e dedicado para sistemas embarcados.

Seu princípio de funcionamento é bastante simples, no primeiro momento apaga a memória Flash totalmente ou por blocos conforme a escolha do usuário, após essa etapa há a gravação do novo programa na memória, há uma leitura da memória para verificação de possíveis erros durante a gravação.

Para efetuar a gravação, basta apertar dois botões dentro do módulo, conectar a placa ao computador usando um cabo serial DB9 macho para DB9 fêmea e conectar na serial 0 do módulo. Abre-se então o programa Flash Magic, no menu Options abre-se Advanced Options e depois o Hardware Config e configura-se conforme a Figura A-1 abaixo. Dessa forma configura-se os tempos de comunicação.

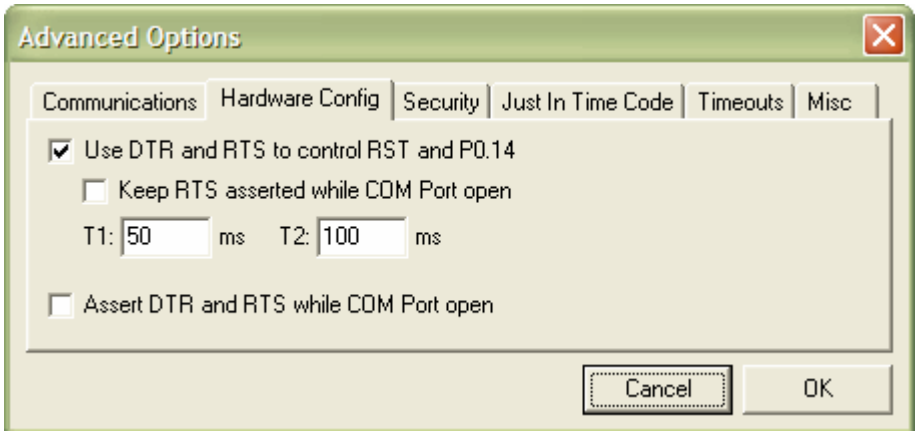


Figura A-1 – Configuração dos tempos de comunicação.

Em seguida seleciona-se a porta de comunicação COM onde o módulo está conectado. Parametriza-se a velocidade máxima em 57600, seleciona-se o cristal de 12 MHz e Device = LPC2368. Para gravar o programa acesso o Browse selecionando, em seguida, o arquivo HEX e depois acione o Start. Conforme a Figura A-2 abaixo.

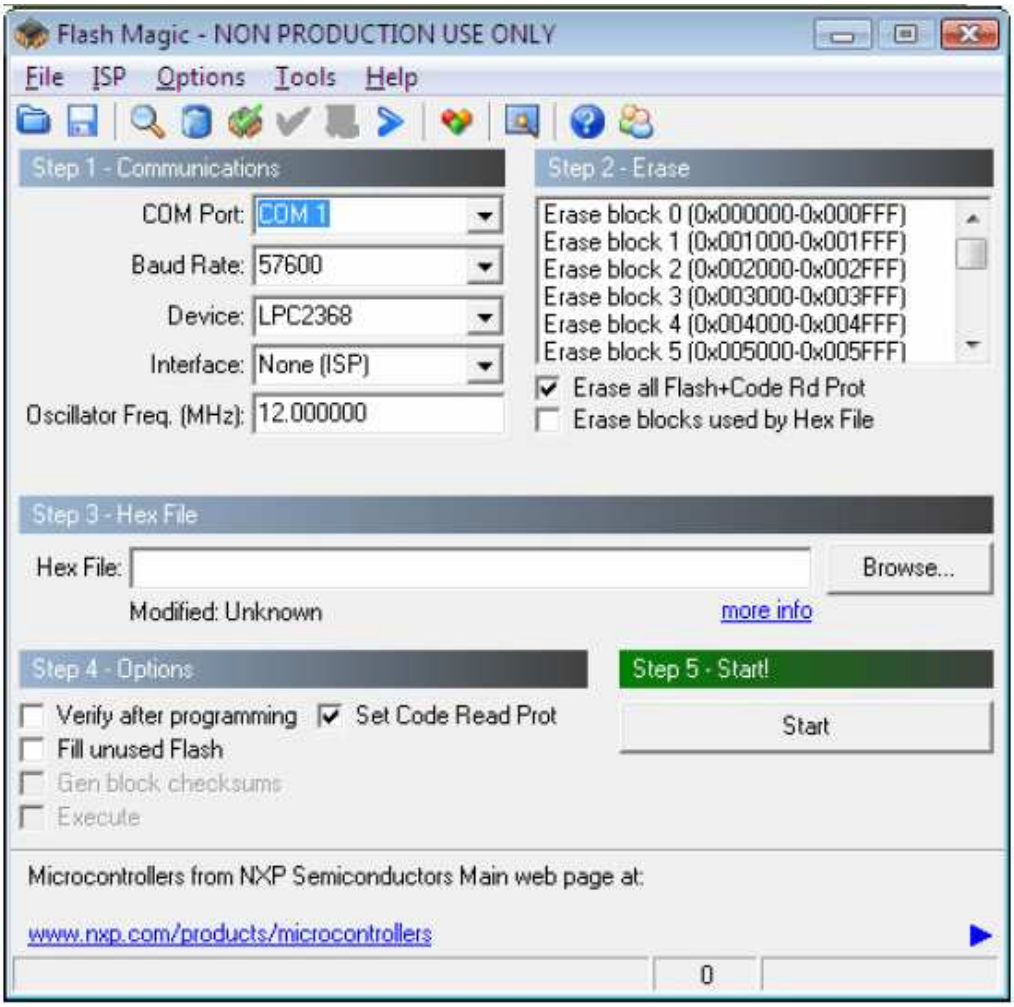


Figura A-2 – Tela do aplicativo Flash Magic.

APÊNDICE B – SOFTWARE DO SERVIDOR

```
Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.Text
Imports System.IO

Public Class frmEx2TCP
    Structure tpTCP
        Dim Conectado As Boolean
        Dim tc As TcpClient
        Dim ns As NetworkStream
        Dim sw As StreamWriter
        Dim sr As StreamReader
        Dim Sair As Boolean
        Dim Recebeu As Boolean
        ' Pilha de dados
        Dim FilaRx As Queue(Of String)
    End Structure
    Dim TCP As tpTCP
    WithEvents Timer_Dados_Recebidos As New Timer
    Dim th_Trata_Dado As System.Threading.Thread

    Sub thTrataDado_Rotina()
        Dim Vetor(1024) As Byte
        Dim N As Integer, Size As Integer
        Dim Str As String = ""
        Do
            If TCP.ns.DataAvailable Then
                Size = TCP.ns.Read(Vetor, 0, 1024)
                ' exibo
                Str = ""
                For N = 0 To Size - 1
                    Str = Str & Chr(Vetor(N))
                Next
                TCP.FilaRx.Enqueue(Str)
            End If
            Threading.Thread.Sleep(1)
        Loop While TCP.Conectado
    End Sub

    Private Sub frmEx2TCP_FormClosing(ByVal sender As Object, ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles
Me.FormClosing
        Desconectar()
    End Sub

    Private Sub frmEx2TCP_MouseMove(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.MouseMove
        If TCP.Conectado = True Then
            Enviar("READ PORTS")
        End If
    End Sub

    Private Sub frmEx2TCP_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    End Sub

    Sub Adicionar_log(ByVal str As String)
        txtLog.Text = str & "  -  " & Now & vbNewLine & txtLog.Text
    End Sub
    Private Sub cmdConectar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdConectar.Click
        If TCP.Conectado = False Then
            Conectar()
            If TCP.tc.Connected Then
                cmdConectar.Text = "Desconectar"
            End If

        Else
            Desconectar()
            If Not TCP.tc.Connected Then
                cmdConectar.Text = "Conectar"
            End If
        End If
```

```
    End Sub

    Sub Conectar()
        TCP.tc = New TcpClient()
        'Inicia a conexão TCP com o servidor na porta referenciada
        Try
            ' Abro a conexao
            TCP.tc.ReceiveTimeout = 0
            TCP.tc.Connect(txtIP.Text, Val(txtPorta.Text)) 'Colocar TRY
            'Cria um Stream de rede
            TCP.ns = TCP.tc.GetStream()
        Catch ex As Exception
            Adicionar_log("Erro ào conectar: " & ex.Message)
            TCP.Conectado = False
            Exit Sub
        End Try

        'Objeto de leitura da resposta do servidor através da Rede
        'TCP.sr = New StreamReader(TCP.ns)
        TCP.FilaRx = New Queue(Of String)
        TCP.Conectado = True

        ' ativo a thread
        th_Trata_Dado = New System.Threading.Thread(AddressOf thTrataDado_Rotina)
        With th_Trata_Dado
            .IsBackground = True
            .Priority = System.Threading.ThreadPriority.Normal
            .Start()
        End With

        ' ligo o timer
        Timer_Dados_Recebidos.Interval = 10
        Timer_Dados_Recebidos.Enabled = True

    End Sub

    Sub Desconectar()

        Timer_Dados_Recebidos.Enabled = False
        Try
            th_Trata_Dado.Abort()
        Catch

        End Try
        If TCP.Conectado Then
            TCP.Conectado = False
            TCP.tc.Close()
        End If
    End Sub

    Private Sub Timer_Dados_Recebidos_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Timer_Dados_Recebidos.Tick
        Dim str As String
        Dim args() As String

        If TCP.FilaRx.Count > 0 Then
            str = TCP.FilaRx.Dequeue

            ' separo os argumentos
            args = str.Split(";")
            ' verifico o argumento 0 é o COMANDO
            If (args(0) = "READ PORTS") Then
                If args(2) = "0" Then Input1.BackColor = Color.Red Else Input1.BackColor = Color.Green
                If args(3) = "0" Then Input2.BackColor = Color.Red Else Input2.BackColor = Color.Green
                If args(4) = "0" Then Input3.BackColor = Color.Red Else Input3.BackColor = Color.Green
                If args(5) = "0" Then Input4.BackColor = Color.Red Else Input4.BackColor = Color.Green
                If args(6) = "0" Then Input5.BackColor = Color.Red Else Input5.BackColor = Color.Green
                If args(7) = "0" Then Input6.BackColor = Color.Red Else Input6.BackColor = Color.Green
                If args(8) = "0" Then Input7.BackColor = Color.Red Else Input7.BackColor = Color.Green
                If args(9) = "0" Then Input8.BackColor = Color.Red Else Input8.BackColor = Color.Green

                lblAN0.Text = "AN0=" & args(11)
                lblAN1.Text = "AN1=" & args(12)
                lblAN2.Text = "AN2=" & args(13)


                If args(11) >= Min_AN0.Text Then
```

```
            If args(11) <= Max_AN0.Text Then
                Enviar("DOUT;4;1")
                Enviar("DOUT;1;0")
                Status_AN0.Text = "OK"
                Status_AN0.BackColor = Color.Green
            End If
            If args(11) < Min_AN0.Text Then
                Enviar("DOUT;4;0")
                Enviar("DOUT;1;1")
                Status_AN0.Text = "Cuidado"
                Status_AN0.BackColor = Color.Yellow
            End If
            If args(11) > Max_AN0.Text Then
                Enviar("DOUT;0;1")
                Enviar("DOUT;4;0")
                Enviar("DOUT;1;1")
                Status_AN0.Text = "Perigo"
                Status_AN0.BackColor = Color.Red

            End If

            If args(12) >= Min_AN1.Text Then
                If args(12) <= Max_AN1.Text Then
                    Enviar("DOUT;5;1")
                    Status_AN1.Text = "OK"
                    Status_AN1.BackColor = Color.Green
                End If
                If args(12) < Min_AN1.Text Then
                    Enviar("DOUT;5;0")
                    Status_AN1.Text = "Cuidado"
                    Status_AN1.BackColor = Color.Yellow
                End If
                If args(12) > Max_AN1.Text Then
                    Enviar("DOUT;5;0")
                    Enviar("DOUT;0;1")
                    Status_AN1.Text = "Perigo"
                    Status_AN1.BackColor = Color.Red
                End If
            End If


            If args(13) >= Min_AN2.Text Then
                If args(13) <= Max_AN2.Text Then
                    Enviar("DOUT;6;1")
                    Status_AN2.Text = "OK"
                    Status_AN2.BackColor = Color.Green
                End If
                If args(13) > Min_AN2.Text Then
                    Enviar("DOUT;6;0")
                    Status_AN2.Text = "Cuidado"
                    Status_AN2.BackColor = Color.Yellow
                End If
                If args(13) > Max_AN2.Text Then
                    Enviar("DOUT;6;0")
                    Enviar("DOUT;0;1")
                    Status_AN2.Text = "Perigo"
                    Status_AN2.BackColor = Color.Red
                End If
            End If

        End If

        Adicionar_log("PORTAS LIDAS, Tela atualizada!")
    End If

  End If

End Sub

Private Sub txtDadosRecebidos_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)

End Sub

Sub Enviar(ByVal str As String)
  Try
    Dim N As Integer
    Dim vetor() As Byte
```

```
            ReDim vetor(str.Length)
            For N = 0 To str.Length - 1
                vetor(N) = Convert.ToByte(str(N))
            Next
            TCP.ns.Write(vetor, 0, vetor.Length)
        Catch ex As Exception
            Adicionar_log("Erro: " & ex.Message)
        End Try

    End Sub

    Private Sub Liga1_AutoSizeChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles Liga1.AutoSizeChanged

    End Sub

    Private Sub Liga1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga1.Click
        If Liga1.Bottom Then
            Enviar("RL;0;1") ' mando ligar
            Liga1.Text = "Desligado"
            Liga1.BackColor = Color.Red
        Else
            If Not Liga1.Bottom Then
                Enviar("RL;0;0") ' mando ligar
                Liga1.Text = "Ligado"
                Liga1.BackColor = Color.Green
            End If
        End If
    End Sub


    Private Sub Liga2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga2.Click
        If Liga2.Bottom Then
            Enviar("RL;0;0") ' mando ligar
            Liga2.Text = "Ligado"
            Liga2.BackColor = Color.Red
        End If
    End Sub
    Private Sub Liga3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga3.Click
        If Liga3.Enabled Then
            Enviar("RL;2;1") ' mando ligar
            Liga3.Text = "Ligado"
            Liga3.BackColor = Color.Red
        Else
            Enviar("RL;2;0") ' mando desligar
            Liga3.Text = "Desligado"
            Liga3.BackColor = Color.MediumBlue
        End If
    End Sub
    Private Sub Liga4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga4.Click
        If Liga4.Bottom Then
            Enviar("RL;3;1") ' mando ligar
            Liga4.Text = "Ligado"
            Liga4.BackColor = Color.Red
        Else
            Enviar("RL;3;0") ' mando desligar
            Liga4.Text = "Desligado"
            Liga4.BackColor = Color.MediumBlue
        End If
    End Sub
    Private Sub Liga5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga5.Click
        If Liga5.Bottom Then
            Enviar("RL;4;1") ' mando ligar
            Liga5.Text = "Ligado"
            Liga5.BackColor = Color.Red
        Else
            Enviar("RL;4;0") ' mando desligar
            Liga5.Text = "Desligado"
            Liga5.BackColor = Color.MediumBlue
        End If
    End Sub
    Private Sub Liga6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga6.Click
        If Liga6.Bottom Then
            Enviar("RL;5;1") ' mando ligar
            Liga6.Text = "Ligado"
            Liga6.BackColor = Color.Red
        Else
            Enviar("RL;5;0") ' mando desligar
```

```
        Liga6.Text = "Desligado"
        Liga6.BackColor = Color.MediumBlue
    End If
End Sub
Private Sub Liga7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga7.Click
    If Liga7.Bottom Then
        Enviar("RL;6;1") ' mando ligar
        Liga7.Text = "Ligado"
        Liga7.BackColor = Color.Red
    Else
        Enviar("RL;6;0") ' mando desligar
        Liga7.Text = "Desligado"
        Liga7.BackColor = Color.MediumBlue
    End If
End Sub
Private Sub Liga8_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Liga8.Click
    If Liga8.Bottom Then
        Enviar("RL;7;1") ' mando ligar
        Liga8.Text = "Ligado"
        Liga8.BackColor = Color.Red
    Else
        Enviar("RL;7;0") ' mando desligar
        Liga8.Text = "Desligado"
        Liga8.BackColor = Color.MediumBlue
    End If
End Sub

Private Sub SaídaDigital1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital1.Click
    If SaídaDigital1.Bottom Then
        Enviar("DOUT;0;1") ' mando ligar
        SaídaDigital1.Text = "Ligado"
        SaídaDigital1.BackColor = Color.MediumPurple
    Else
        Enviar("DOUT;0;0") ' mando desligar
        SaídaDigital1.Text = "Desligado"
        SaídaDigital1.BackColor = Color.LightGreen
    End If
End Sub
Private Sub SaídaDigital2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital2.Click
    If SaídaDigital2.Bottom Then
        Enviar("DOUT;1;1") ' mando ligar
        SaídaDigital2.Text = "Ligado"
        SaídaDigital2.BackColor = Color.MediumPurple
    Else
        Enviar("DOUT;1;0") ' mando desligar
        SaídaDigital2.Text = "Desligado"
        SaídaDigital2.BackColor = Color.LightGreen
    End If
End Sub
Private Sub SaídaDigital3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital3.Click
    If SaídaDigital3.Bottom Then
        Enviar("DOUT;2;1") ' mando ligar
        SaídaDigital3.Text = "Ligado"
        SaídaDigital3.BackColor = Color.MediumPurple
    Else
        Enviar("DOUT;2;0") ' mando desligar
        SaídaDigital3.Text = "Desligado"
        SaídaDigital3.BackColor = Color.LightGreen
    End If
End Sub
Private Sub SaídaDigital4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital4.Click
    If SaídaDigital4.Bottom Then
        Enviar("DOUT;3;1") ' mando ligar
        SaídaDigital4.Text = "Ligado"
        SaídaDigital4.BackColor = Color.MediumPurple
    Else
        Enviar("DOUT;3;0") ' mando desligar
        SaídaDigital4.Text = "Desligado"
        SaídaDigital4.BackColor = Color.LightGreen
    End If
End Sub
Private Sub SaídaDigital5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital5.Click
    If SaídaDigital5.Bottom Then
        Enviar("DOUT;4;1") ' mando ligar
        SaídaDigital5.Text = "Ligado"
        SaídaDigital5.BackColor = Color.MediumPurple
    Else
```

```
            Enviar("DOUT;4;0") ' mando desligar
            SaídaDigital5.Text = "Desligado"
            SaídaDigital5.BackColor = Color.LightGreen
        End If
    End Sub
    Private Sub SaídaDigital6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital6.Click
        If SaídaDigital6.Bottom Then
            Enviar("DOUT;5;1") ' mando ligar
            SaídaDigital6.Text = "Ligado"
            SaídaDigital6.BackColor = Color.MediumPurple
        Else
            Enviar("DOUT;5;0") ' mando desligar
            SaídaDigital6.Text = "Desligado"
            SaídaDigital6.BackColor = Color.LightGreen
        End If
    End Sub
    Private Sub SaídaDigital7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital7.Click
        If SaídaDigital7.Bottom Then
            Enviar("DOUT;6;1") ' mando ligar
            SaídaDigital7.Text = "Ligado"
            SaídaDigital7.BackColor = Color.MediumPurple
        Else
            Enviar("DOUT;6;0") ' mando desligar
            SaídaDigital7.Text = "Desligado"
            SaídaDigital7.BackColor = Color.LightGreen
        End If
    End Sub
    Private Sub SaídaDigital8_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaídaDigital8.Click
        If SaídaDigital8.Bottom Then
            Enviar("DOUT;7;1") ' mando ligar
            SaídaDigital8.Text = "Ligado"
            SaídaDigital8.BackColor = Color.MediumPurple
        Else
            Enviar("DOUT;7;0") ' mando desligar
            SaídaDigital8.Text = "Desligado"
            SaídaDigital8.BackColor = Color.LightGreen
        End If
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
        txtLog.Text = ""
    End Sub


     End Class
```

APÊNDICE C – ROTINA DE INICIALIZAÇÃO DO SISTEMA

```
//Rotina que inicia o LPC2368. Inicializa o hardware e chama a rotina principal.
Mode_USR        EQU     0x10
Mode_FIQ        EQU     0x11
Mode_IRQ        EQU     0x12
Mode_SVC        EQU     0x13
Mode_ABT        EQU     0x17
Mode_UND        EQU     0x1B
Mode_SYS        EQU     0x1F

I_Bit           EQU     0x80            ; quando o bit I é setado, IRQ é desabilitado
F_Bit           EQU     0x40            ; quando o bit F é setado, FIQ é desabilitado

UND_Stack_Size  EQU     0x00000000
SVC_Stack_Size  EQU     0x00000080
ABT_Stack_Size  EQU     0x00000000
FIQ_Stack_Size  EQU     0x00000000
IRQ_Stack_Size  EQU     0x00000080
USR_Stack_Size  EQU     0x00000080

ISR_Stack_Size  EQU     (UND_Stack_Size + SVC_Stack_Size + ABT_Stack_Size + \
                 FIQ_Stack_Size + IRQ_Stack_Size)

            AREA    STACK, NOINIT, READWRITE, ALIGN=3

Stack_Mem       SPACE   USR_Stack_Size
__initial_sp    SPACE   ISR_Stack_Size
Stack_Top
Heap_Size       EQU     0x00000010
            AREA    HEAP, NOINIT, READWRITE, ALIGN=3
__heap_base
Heap_Mem        SPACE   Heap_Size
__heap_limit
; Bloco do Sistema de controle (SCB). Definições:
SCB_BASE        EQU     0xE01FC000      ; SCB Base Address
PLLCON_OFS      EQU     0x80            ; PLL Control Offset
PLLCFG_OFS      EQU     0x84            ; PLL Configuration Offset
PLLSTAT_OFS     EQU     0x88            ; PLL Status Offset
PLLFEED_OFS     EQU     0x8C            ; PLL Feed Offset
CCLKCFG_OFS     EQU     0x104           ; CPU Clock Divider Reg Offset
USBCLKCFG_OFS   EQU     0x108           ; USB Clock Divider Reg Offset
CLKSRCSEL_OFS   EQU     0x10C           ; Clock Source Sel Reg Offset
SCS_OFS         EQU     0x1A0           ; Sys Control and Status Reg Offset
```

```
PCLKSEL0_OFS    EQU     0x1A8          ; Periph Clock Sel Reg 0 Offset
PCLKSEL1_OFS    EQU     0x1AC          ; Periph Clock Sel Reg 0 Offset
; Constantes
OSCRANGE        EQU     (1<<4)         ; Oscillator Range Select
OSCEN           EQU     (1<<5)         ; Main oscillator Enable
OSCSTAT         EQU     (1<<6)         ; Main Oscillator Status
PLLCON_PLLE     EQU     (1<<0)         ; PLL Enable
PLLCON_PLLC     EQU     (1<<1)         ; PLL Connect
PLLSTAT_M       EQU     (0x7FFF<<0)    ; PLL M Value
PLLSTAT_N       EQU     (0xFF<<16)     ; PLL N Value
PLLSTAT_PLOCK   EQU     (1<<26)        ; PLL Lock Status
CLOCK_SETUP     EQU     0
SCS_Val         EQU     0x00000020
CLKSRCSEL_Val   EQU     0x00000001
PLLCFG_Val      EQU     0x0000000B
CCLKCFG_Val     EQU     0x00000005
USBCLKCFG_Val   EQU     0x00000005
PCLKSEL0_Val    EQU     0x00000000
PCLKSEL1_Val    EQU     0x01000000
 ; Módulo Acelerador de Memória (MAM) definições:
MAM_BASE        EQU     0xE01FC000     ; MAM Base Address
MAMCR_OFS       EQU     0x00           ; MAM Control Offset
MAMTIM_OFS      EQU     0x04           ; MAM Timing Offset
MAM_SETUP       EQU     0
MAMCR_Val       EQU     0x00000002
MAMTIM_Val      EQU     0x00000004
; O código de início precisa ser linkado com um endereço que espera o sistema rodar.
            AREA    RESET, CODE, READONLY
            ARM
Vectors       LDR     PC, Reset_Addr
            LDR     PC, Undef_Addr
            LDR     PC, SWI_Addr
            LDR     PC, PAbt_Addr
            LDR     PC, DAbt_Addr
            NOP                     ; Reserved Vector
            LDR     PC, [PC, #-0x0120]    ; Vector from VicVectAddr
            LDR     PC, FIQ_Addr

                    IMPORT   SWI_Handler           ;  RTOS
Reset_Addr      DCD     Reset_Handler
Undef_Addr      DCD     Undef_Handler
SWI_Addr        DCD     SWI_Handler
PAbt_Addr       DCD     PAbt_Handler
DAbt_Addr       DCD     DAbt_Handler
            DCD     0               ; Reserved Address
IRQ_Addr        DCD     IRQ_Handler
FIQ_Addr        DCD     FIQ_Handler
```

```
Undef_Handler   B       Undef_Handler
PAbt_Handler    B       PAbt_Handler
DAbt_Handler    B       DAbt_Handler
IRQ_Handler     B       IRQ_Handler
FIQ_Handler     B       FIQ_Handler
; Reset Handler
                EXPORT  Reset_Handler
Reset_Handler
; Setup Clock
                IF      CLOCK_SETUP != 0
                LDR     R0, =SCB_BASE
                MOV     R1, #0xAA
                MOV     R2, #0x55
;  Configure and Enable PLL
                LDR     R3, =SCS_Val          ; Enable main oscillator
                STR     R3, [R0, #SCS_OFS]
                IF      (SCS_Val:AND:OSCEN) != 0
OSC_Loop        LDR     R3, [R0, #SCS_OFS]    ; Wait for main osc stabilize
                ANDS    R3, R3, #OSCSTAT
                BEQ     OSC_Loop
                ENDIF
                LDR     R3, =CLKSRCSEL_Val    ; Select PLL source clock
                STR     R3, [R0, #CLKSRCSEL_OFS]
                LDR     R3, =PLLCFG_Val
                STR     R3, [R0, #PLLCFG_OFS]
                STR     R1, [R0, #PLLFEED_OFS]
                STR     R2, [R0, #PLLFEED_OFS]
                MOV     R3, #PLLCON_PLLE
                STR     R3, [R0, #PLLCON_OFS]
                STR     R1, [R0, #PLLFEED_OFS]
                STR     R2, [R0, #PLLFEED_OFS]
;  Wait until PLL Locked
PLL_Loop        LDR     R3, [R0, #PLLSTAT_OFS]
                ANDS    R3, R3, #PLLSTAT_PLOCK
                BEQ     PLL_Loop
M_N_Lock        LDR     R3, [R0, #PLLSTAT_OFS]
                LDR     R4, =(PLLSTAT_M:OR:PLLSTAT_N)
                AND     R3, R3, R4
                LDR     R4, =PLLCFG_Val
                EORS    R3, R3, R4
                BNE     M_N_Lock
;  Setup CPU clock divider
                MOV     R3, #CCLKCFG_Val
                STR     R3, [R0, #CCLKCFG_OFS]
;  Setup USB clock divider
                LDR     R3, =USBCLKCFG_Val
                STR     R3, [R0, #USBCLKCFG_OFS]
```

```
; Setup Peripheral Clock
        LDR     R3, =PCLKSEL0_Val
        STR     R3, [R0, #PCLKSEL0_OFS]
        LDR     R3, =PCLKSEL1_Val
        STR     R3, [R0, #PCLKSEL1_OFS]
; Switch to PLL Clock
        MOV     R3, #(PLLCON_PLLE:OR:PLLCON_PLLC)
        STR     R3, [R0, #PLLCON_OFS]
        STR     R1, [R0, #PLLFEED_OFS]
        STR     R2, [R0, #PLLFEED_OFS]
        ENDIF   ; CLOCK_SETUP
; Setup MAM
        IF      MAM_SETUP != 0
        LDR     R0, =MAM_BASE
        MOV     R1, #MAMTIM_Val
        STR     R1, [R0, #MAMTIM_OFS]
        MOV     R1, #MAMCR_Val
        STR     R1, [R0, #MAMCR_OFS]
        ENDIF   ; MAM_SETUP
; Memory Mapping (when Interrupt Vectors are in RAM)
MEMMAP  EQU     0xE01FC040      ; Memory Mapping Control
        IF      :DEF:REMAP
        LDR     R0, =MEMMAP
        IF      :DEF:RAM_MODE
        MOV     R1, #2
        ELSE
        MOV     R1, #1
        ENDIF
        STR     R1, [R0]
        ENDIF
; Initialise Interrupt System
; Setup Stack for each mode
        LDR     R0, =Stack_Top
;  Enter Undefined Instruction Mode and set its Stack Pointer
        MSR     CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit
        MOV     SP, R0
        SUB     R0, R0, #UND_Stack_Size
;  Enter Abort Mode and set its Stack Pointer
        MSR     CPSR_c, #Mode_ABT:OR:I_Bit:OR:F_Bit
        MOV     SP, R0
        SUB     R0, R0, #ABT_Stack_Size
;  Enter FIQ Mode and set its Stack Pointer
        MSR     CPSR_c, #Mode_FIQ:OR:I_Bit:OR:F_Bit
        MOV     SP, R0
        SUB     R0, R0, #FIQ_Stack_Size
;  Enter IRQ Mode and set its Stack Pointer
        MSR     CPSR_c, #Mode_IRQ:OR:I_Bit:OR:F_Bit
```

```
                MOV     SP, R0
                SUB     R0, R0, #IRQ_Stack_Size
;  Enter Supervisor Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit
                MOV     SP, R0
                SUB     R0, R0, #SVC_Stack_Size
;  Enter User Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_USR
                IF      :DEF:__MICROLIB
                EXPORT __initial_sp
                ELSE
                MOV     SP, R0
                SUB     SL, SP, #USR_Stack_Size
                ENDIF
; Enter the C code
                IMPORT  __main
                LDR     R0, =__main
                BX      R0
                IF      :DEF:__MICROLIB
                EXPORT  __heap_base
                EXPORT  __heap_limit
                ELSE
; User Initial Stack & Heap
                AREA    |.text|, CODE, READONLY
                IMPORT  __use_two_region_memory
                EXPORT  __user_initial_stackheap
__user_initial_stackheap
                LDR     R0, =  Heap_Mem
                LDR     R1, =(Stack_Mem + USR_Stack_Size)
                LDR     R2, = (Heap_Mem +      Heap_Size)
                LDR     R3, = Stack_Mem
                BX      LR
                ENDIF
                END
```

APÊNDICE D – ROTINA DE INICIALIZAÇÃO DO MÓDULO

```
#include <stdlib.h>
#include "RtOs.h"
#include "lpc23_utils.h"
#include "cp-flex_V1.h"
#include "keyboard_pc.h"
#include "lcd128x64.h"
#include "mini_gui.h"
extern UINT8 CPFLEX_App_Init ( void );
static void tk_Sistema( void *pvParameters );
// Buffer da pilha usado pelo RTOS - 1024 caracteres e multiplica por 4. Área de memória que pode ser usada pelo RtOs
UINT8 RTOS_Heap_Buffer[ (1024*4) ];
/*
	Rotina Principal
*/
int main ( void ) {
	UINT32 volatile start;
	UINT8 resp;
	for ( start=0x00;start<1E3;start++ ) {
	}
	/* Inicializa a CPU, com todos os pinos configurados */
	CPU_Init( CPU_CRYSTAL_SUGGESTED,
					CPU_MAM_TIM_SUGGESTED,
					CPU_MAM_CTRL_SUGGESTED,
					CPU_N_SEL_SUGGESTED,
					CPU_M_SEL_SUGGESTED,
					CPU_CLOCK_SOURCE_SUGGESTED
					 );
	CP_FLEX_Init();			// Inicializa os periféricos da CP FLEX
	/*
	Informar à lib do CP FLEX a rotina que é usada para delay em ms
		Pois ela é usada dentro da biblioteca CP FLEX
	*/
	CP_FLEX_Client_Delay_ms=vTaskDelay;
	/*
		Área de memória que pode ser usada pelo RtOs
	*/
	RTOS_Heap_Init( sizeof(RTOS_Heap_Buffer),RTOS_Heap_Buffer );
	/*
		RtOs
		Inicializo primeiro Rotinas do Sistema e FIFOs
		Esta rotina é responsável pelas rotinas de sistema:
	*/
	resp=RTOS_CreateTask( tk_Sistema,
		"tk_Sistema",
```

```
                RTOS_MINIMAL_Stack,
                NULL,
                ( RTOS_TASK_LOW_PRIORITY ),
                NULL
                );
        if ( resp!=RESULT_OK ) {
                for ( ;; ) {
                }
        }
        CPFLEX_App_Init();
        // Agora manda o RTOS Rodar, nunca mais volta.
        RTOS_Init( 0x00 );
        for ( ;; ) {
        }
        return 0;
}
/*
        void tk_Sistema( void *pvParameters )
        Rotina do sistema.
                Varre o teclado, e empilha na Fifo
                Mantém os I/Os. do CP FLEX ativados, através do Output WatchDog
*/
static void tk_Sistema( void *pvParameters ) {
        BYTE n,b;
        ( void ) pvParameters;/* pára de dar Warning */
        n=0;
        b=0;
        for ( ;; ) {
                KEYBOARD_PC_Poll();          // chamo a rotina que verifica os dados do teclado
                if ( KEYBOARD.event ) {// verifico se recebeu uma tecla
                        // Coloco a tecla na fila
                        if ( FIFO_Send ( &xTECLADO, &KEYBOARD )==FIFO_OK ) {      // Coloco o dado na fifo
                                // colocado na fila de espera limpo a tecla.
                                KEYBOARD.event=0;
                                if ( KEYBOARD.membrane ) {
                                        CP_FLEX_Buzzer( CP_FLEX_STATE_ON );
                                        n=0;
                                        b=1;
                                }
                        }
                }
                vTaskDelay(5);
                n++;
                if ( n>=30 ) {
                        // a cada 100mS executa as instruções abaixo
                        n=0x00;
                WDT_Clear();          // Reseto o WDT para não resetar o Software
```

```
                    if ( b==1 ) {           // aumentei o tempo do buzzer
                        // A cada 100mS executa as rotinas abaixo
                        CP_FLEX_Buzzer( CP_FLEX_STATE_OFF );
                b=0;
                    }
                }
                #if defined ( CP_FLEX_USE_WATCH_DOG_OUTPUT_ENABLE )
                // Este recurso só pode ser usado com o HW V2 do CP FLEX
                    // Mantém o OE ativado. Chamar esta rotina a cada no máxio 5mS para que as saídas se mantenham ativas
                        CP_FLEX_Outputs_WatchDog();
                #endif
            }
        }
```

APÊNDICE E – ROTINA PRINCIPAL DA APLICAÇÃO

```
#include <stdlib.h>
#include "RtOs.h"
#include "lpc23_utils.h"
#include "cp-flex_V1.h"
#include "keyboard_pc.h"
#include "lcd128x64.h"
#include "mini_gui.h"
#include "tcpip.h"  // Rede
typedef struct {
  signed char comando[10];
  void (*chamar_rotina)(void);
} tplista_cmd;
typedef struct {
signed char *args[10];// podemos passar até 10 argumentos nesta variável
        signed char buffer[1024];          // buffer que armazena os dados
        UINT8 argumentos;                  // total de argumentos
        UINT16 size;                       // tamanho do dado recebido
} tp_dadorecebido;
tp_dadorecebido DADO_RECEBIDO;
signed char string_temp[20];    // string temporária usada somente para conversão de números
// Rotina que trata o Comando PING
void cmd_ping ( void ){
        utils_memcpy(TCP_TX_BUF,"PING: OK",8);          // Copio a string "PING: OK" para o buffer de transmissão
        TCPIP.TCPTxDataCount =8;
        TCPTransmitTxBuffer();          // send last segment
}
/*
        Le todas as Portas do CP FLEX e devolve assim:
        Comando READ PORTS tem o seguinte formato:
        READ PORTS. devolve:
        din                             dout
        READ PORTS;DIN;DI7;DI6;DI5;DI4;DI3;DI2;DI1;DI0;ANALOG;AN0;AN1;AN2;DOUTS;DO7;DO6;DO5;DO4;DO3;DO2;DO1;DO0;RELAYS;RL7;RL6;RL5;RL4;RL3;RL2;RL1;RL0
*/
void cmd_read_ports ( void ){
        UINT8 n,n2;
        UINT16 i;
        i=14;
        // Primeiro monto as leituras dos DIN
        utils_memcpy( TCP_TX_BUF,"READ PORTS;DI;",i );          // Copio a string "PING: OK" para o buffer de transmissão
        for ( n=0x00;n<CP_FLEX_Digital_Input_Count();n++ ) {
                //  Leito a Porta de entrada 0, e devolvo o estado: 1=ligado, 0= desligado
```

```
                if ( CP_FLEX_Digital_Input_Read(n)==CP_FLEX_STATE_ON ) {
                        utils_memcpy( &TCP_TX_BUF[i],"1;",2 );          // Copio a string "PING: OK" para o buffer de transmissão
                } else {
                        utils_memcpy( &TCP_TX_BUF[i],"0;",2 );          // Copio a string "PING: OK" para o buffer de transmissão
                }
                i+=2;
        }
        // Leitura dos sinais analogicos
        utils_memcpy( &TCP_TX_BUF[i],"ANALOG;",7 );             // Copio a string "PING: OK" para o buffer de transmissão
        i+=7;
        for ( n=0x00;n<ANALOG_Channel_Count();n++ ) {
                utils_itoa(ANALOG_Input_Read( n ),string_temp, 10);
                n2=utils_strlen(string_temp);   // n2= tamanho da string
                // adiciono ; no fim do número que acabou de ser convertido
                string_temp[n2]=';';
                n2++;
                // agora copio o número convertido e que tem ; no final, para o buffer de transmissao
                utils_memcpy( &TCP_TX_BUF[i],string_temp,n2 );          // Copio a string string_temp para o buffer tx
                i+=n2; // proxima posicao do buffer de transmissao onde deverao ser inseridos dados
        }
        // Leitura das saídas digitais DOUT
        utils_memcpy( &TCP_TX_BUF[i],"DOUT;",5 );               // Copio a string "PING: OK" para o buffer de transmissão
        i+=5;
        for ( n=0x00;n<CP_FLEX_Digital_Output_Count();n++ ) {
                //  Leito a Porta de entrada 0, e devolvo o estado: 1=ligado, 0= desligado
                if ( CP_FLEX_Digital_Output_State(n)==CP_FLEX_STATE_ON ) {
                        utils_memcpy( &TCP_TX_BUF[i],"1;",2 );          // Copio a string "PING: OK" para o buffer de transmissão
                } else {
                        utils_memcpy( &TCP_TX_BUF[i],"0;",2 );          // Copio a string "PING: OK" para o buffer de transmissão
                }
                i+=2;
        }
        // Leitura das saídas digitais RELES
        utils_memcpy( &TCP_TX_BUF[i],"RELAY;",6 );              // Copio a string "PING: OK" para o buffer de transmissão
        i+=6;
        for ( n=0x00;n<CP_FLEX_Relay_Count();n++ ) {
                //  Leito a Porta de entrada 0, e devolvo o estado: 1=ligado, 0= desligado
                if ( CP_FLEX_Relay_State(n)==CP_FLEX_STATE_ON ) {
                        utils_memcpy( &TCP_TX_BUF[i],"1;",2 );          // Copio a string "PING: OK" para o buffer de transmissão
                } else {
                        utils_memcpy( &TCP_TX_BUF[i],"0;",2 );          // Copio a string "PING: OK" para o buffer de transmissão
```

```
                }
                i+=2;
        }
        TCPIP.TCPTxDataCount =i;
        TCPTransmitTxBuffer();          //Envia o último pedaço da mensagem
}
/*
        formato do comando DOUT:
        DOUT;porta;estado
        estado=1 liga
        estado=0 desliga
        Exemplo para ligar a saída digital 1, faça: DOUT;1;1
        Exemplo para ligar a saída digital 2, faça: DOUT;2;1

        Exemplo para desligar a saída digital 1, faça: DOUT;1;0
        Exemplo para desligar a saída digital 2, faça: DOUT;2;0
*/
void cmd_dout( void ) {
        UINT8 porta,estado;
        porta=utils_atoi( DADO_RECEBIDO.args[1] );
        estado=     DADO_RECEBIDO.args[2][0];  // pego o primeiro caractere do argumento 2
        if ( estado=='1' ) {
                CP_FLEX_Digital_Output ( porta,CP_FLEX_STATE_ON );
        } else {
                CP_FLEX_Digital_Output ( porta,CP_FLEX_STATE_OFF );
        }
        utils_memcpy( TCP_TX_BUF,"OK",2 );                    // Copio a string "PING: OK" para o buffer de transmissão
        TCPIP.TCPTxDataCount =2;
        TCPTransmitTxBuffer();              // send last segment
}
/*
        formato do comando RL:
        RL;numero do rele;estado
        estado=1 liga
        estado=0 desliga
        Exemplo para ligar o rele 1, faça: RL;1;1
        Exemplo para ligar o rele 2, faça: RL;2;1
        Exemplo para desligar o rele 1, faça: RL;1;0
        Exemplo para desligar o rele 2, faça: RL;2;0
*/
void cmd_rl( void ) {
        UINT8 porta,estado;
        porta=utils_atoi( DADO_RECEBIDO.args[1] );
        estado=     DADO_RECEBIDO.args[2][0];  // pego o primeiro caractere do argumento 2
        if ( estado=='1' ) {
                CP_FLEX_Relay ( porta,CP_FLEX_STATE_ON );
        } else {
```

```
                CP_FLEX_Relay ( porta,CP_FLEX_STATE_OFF );
        }
        utils_memcpy( TCP_TX_BUF,"OK",2 );                    // Copio a string "PING: OK" para o buffer de transmissão
        TCPIP.TCPTxDataCount =2;
        TCPTransmitTxBuffer();              // send last segment
}
/*
        Lista de comandos
*/
const tplista_cmd comandos[] = {
        "READ PORTS",cmd_read_ports,
        "PING",cmd_ping,
        "DOUT",cmd_dout,
        "RL",cmd_rl,
};
/*
        Esta rotina, Compara o dado recebido com a Lista de comandos informada neste arquivo.
        Se encontrar um comando, executa a rotina associada ào comando
        Neste exemplo, consideramos que o dado vai vir no seguinte formato:
                par1;par2;par3;par4;
        ou seja, os dados vem separados por ;
        o primeiro argumento, é o comando, e o restante são parametros
*/
void Trata_Dados ( void ) {
        UINT8 n;
        UINT16 i;
        // Imprimo os dados recebidos
        LCD.fonte=FONTE_7;
        l_puts("\fRECEBI:\r\n");
        l_puts( DADO_RECEBIDO.buffer );
        lcdg_atualiza();
        // separo os argumentos
        DADO_RECEBIDO.argumentos=utils_split_string_2 ( DADO_RECEBIDO.buffer,';',DADO_RECEBIDO.args );
        // Dados separados
        // DADO_RECEBIDO.args[0] = comando
        for ( n = 0; n < ( sizeof(comandos)/sizeof(tplista_cmd) ); n++) {
                if ( utils_strcmp ( DADO_RECEBIDO.args[0], (signed char *)&comandos[n].comando ) ) {
                        continue;
                }
                comandos[n].chamar_rotina(); // executa o comando
                break;
        }
}
/*
        Esta rotina fica lendo o Buffer de dados para ver se recebeu dado TCP-IP
*/
void tk_TrataDadoTCP ( void *pvParameters ){
```

```
		( void ) pvParameters;/* pára de dar Warning */
		LCD.fonte=FONTE_7;
		l_puts("\fCP FLEX\r\nEx1. TCP IP");
		lcdg_atualiza();
		// Inicializo a TCPIP
		TCPIP.TCPLocalPort = 550;
		TCPIP_Init();
		for ( ;; ) {	// Esse comando cria um laço de repetição
			TCPIP_Poll();// NetWork - Trata todo o nível baixo da Ethernet
			/*
				Tratamento de dados
			*/
			if ( TCPIP.sock_connected )		// Verifica se tem algo conectado no TCP
			{
				// Verifico se tem dado TCP
			if ( TCPIP.sock_data_available ) {
				TCPIP.sock_data_available=0;	// libero para receber mais pacotes, ou seja apago o buffer.
					// copio o dado recebido para o buffer BUFFER_Rx_TCP
					utils_memcpy ( DADO_RECEBIDO.buffer, TCPIP._RxTCPBuffer,TCPIP.sock_data_available_lenght );
					DADO_RECEBIDO.size=TCPIP.sock_data_available_lenght;
					l_puts(DADO_RECEBIDO.buffer);
					Trata_Dados();
				}
			if ( TCPIP.SocketStatus & SOCK_TX_BUF_RELEASED)	// Verifica se o buffer está livre para o TX
			{
					// Conectou
			}
			}
		}
	}
	/*
	*/
	UINT8 CPFLEX_App_Init ( void ) {
		UINT8 resp;
		// Tarefa que trata os dados recebidos via TCP-IP
		resp=RTOS_CreateTask( tk_TrataDadoTCP,
			"tk_TrataDadoTCP",
			100,
			NULL,
			( RTOS_TASK_LOW_PRIORITY ),
			NULL
			);
		if ( resp!=RESULT_OK ) return resp;
```

APÊNDICE F – ROTINA DE TRADUÇÃO PARA TCP/IP

```
//Inclui as bibliotecas
//Traduz os pacotes do hardware para TCP/IP. Trata-se de uma pilha simples e aberta
//Só aceita uma conexão
#include <LPC23xx.h>
#include "EMAC.h"
#include <string.h>
#define  __tcp_ip_c
#include "tcpip.h"
#include "utils.h"
void TCPIP_Poll(void);
void ProcessEthBroadcastFrame(void);
void ProcessEthIAFrame(void);
void ProcessICMPFrame(void);
void ProcessTCPFrame(void);
void PrepareARP_REQUEST(void);
void PrepareARP_ANSWER(void);
void PrepareICMP_ECHO_REPLY(void);
void PrepareTCP_FRAME(UINT16 TCPCode);
void PrepareTCP_DATA_FRAME(void);
void SendFrame1(void);
void SendFrame2(void);
void TCPStartRetryTimer(void);
void TCPStartTimeWaitTimer(void);
void TCPRestartTimer(void);
void TCPStopTimer(void);
void TCPHandleRetransmission(void);
void TCPHandleTimeout(void);
UINT16 CalcChecksum(void *Start, UINT16 Count, UINT8 IsTCP);
UINT16 SwapBytes(UINT16 Data);
void WriteWBE(UINT8 *Add, UINT16 Data);
void WriteDWBE(UINT8 *Add, UINT32 Data);
void TCPPassiveOpen(void);
void TCPActiveOpen(void);
void TCPClose(void);
void TCPReleaseRxBuffer ( void );
void TCPTransmitTxBuffer( void );
/*
        Início das rotinas
*/
//Conecção fechada
void TCPIP_Init(void)
{
  Init_EMAC();
  TCPIP.TransmitControl = 0;
```

```
  TCPIP.TCPFlags = 0;
  TCPIP.TCPStateMachine = CLOSED;
  TCPIP.SocketStatus = 0;
  TCPIP.sock_connected=0;
  TCPIP.sock_data_available=0;
 }
 //Passa o comunicar
 void TCPPassiveOpen( void ) {
  if ( TCPIP.TCPStateMachine == CLOSED ) {
    TCPIP.TCPFlags &= ~TCP_ACTIVE_OPEN;
    TCPIP.TCPStateMachine = LISTENING;
    TCPIP.SocketStatus = SOCK_ACTIVE;
  }
 }
 //Tenta estabelecer  conexão
 void TCPActiveOpen(void)
 {
  if ((TCPIP.TCPStateMachine == CLOSED) || (TCPIP.TCPStateMachine == LISTENING))
  {
    TCPIP.TCPFlags |= TCP_ACTIVE_OPEN;
    TCPIP.TCPFlags &= ~IP_ADDR_RESOLVED;
    PrepareARP_REQUEST();
    TCPIP.LastFrameSent = ARP_REQUEST;
    TCPStartRetryTimer();
    TCPIP.SocketStatus = SOCK_ACTIVE;
  }
 }
 // Lógica para fechar a conecção
 void TCPClose(void)
 {
  switch ( TCPIP.TCPStateMachine )
  {
    case LISTENING :
    case SYN_SENT :
    {
      TCPIP.TCPStateMachine = CLOSED;
      TCPIP.TCPFlags = 0;
      TCPIP.SocketStatus = 0;
      break;
    }
    case SYN_RECD :
    case ESTABLISHED :
    {
      TCPIP.TCPFlags |= TCP_CLOSE_REQUESTED;
      break;
    }
          default:
```

```
                    break;
  }
}
//Se estiver estabilizado, faz um armazenamento de dados em um buffer se preciso
void TCPTransmitTxBuffer(void)
{
  if ( (TCPIP.TCPStateMachine == ESTABLISHED) || (TCPIP.TCPStateMachine == CLOSE_WAIT) )
    if ( TCPIP.SocketStatus & SOCK_TX_BUF_RELEASED )
    {
      TCPIP.SocketStatus &= ~SOCK_TX_BUF_RELEASED;
      TCPIP.TCPUNASeqNr += TCPIP.TCPTxDataCount;
      TCPIP.TxFrame1Size = ETH_HEADER_SIZE + IP_HEADER_SIZE + TCP_HEADER_SIZE + TCPIP.TCPTxDataCount;
      TCPIP.TransmitControl |= SEND_FRAME1;
      TCPIP.LastFrameSent = TCP_DATA_FRAME;
      TCPStartRetryTimer();
    }
}
// Lê o tamanho dos dados recebidos via Ethernet e verifica o destino da mensagem
unsigned int IsBroadcast(void) {
  UINT16 RecdDestMAC[3];
  TCPIP.RecdFrameLength = StartReadFrame();
  CopyFromFrame_EMAC(&RecdDestMAC,  6);
  CopyFromFrame_EMAC(&TCPIP.RecdFrameMAC, 6);
  if ((RecdDestMAC[0] == 0xFFFF) &&
      (RecdDestMAC[1] == 0xFFFF) &&
      (RecdDestMAC[2] == 0xFFFF)) {
    return(1);
  } else {
    return (0);
  }
}
void TCPIP_Poll(void)
{
        // Timer usado na TCP
        static unsigned short div;
        div++;
        if ( div>250 ) {
                TCPIP.ISNGenHigh++;
                TCPIP.TCPTimer++;
                div=0x00;
        }
        if ( !(TCPIP.SocketStatus & SOCK_ACTIVE) ) {
                TCPPassiveOpen();
        }
        if (CheckFrameReceived())
        {
                if (IsBroadcast()) {
```

```
                ProcessEthBroadcastFrame();
        } else {
                ProcessEthIAFrame();
        }
        EndReadFrame();
}
if ( TCPIP.TCPFlags & TCP_TIMER_RUNNING){
        if (TCPIP.TCPFlags & TIMER_TYPE_RETRY)
        {
                if ( TCPIP.TCPTimer > RETRY_TIMEOUT)
                {
                        TCPRestartTimer();
                        if ( TCPIP.RetryCounter )
                        {
                                TCPHandleRetransmission();
                                TCPIP.RetryCounter--;
                        } else  {
                                TCPStopTimer();
                                TCPHandleTimeout();
                        }
                }
        } else if ( TCPIP.TCPTimer > FIN_TIMEOUT ) {
                TCPIP.TCPStateMachine = CLOSED;
                TCPIP.TCPFlags = 0;
                TCPIP.sock_data_available=0;
        }
}
switch (TCPIP.TCPStateMachine)
{
        case CLOSE_WAIT :
        {
                // Essa lógica Fecha o PROTOCOLO, atua de forma direta no botão conectar da aplicação
                if (!(TCPIP.TransmitControl & (SEND_FRAME2 | SEND_FRAME1)))
                if ( TCPIP.TCPSeqNr == TCPIP.TCPUNASeqNr )
                {
                        TCPIP.TCPUNASeqNr++;
                        PrepareTCP_FRAME(TCP_CODE_FIN | TCP_CODE_ACK);
                        TCPIP.LastFrameSent = TCP_FIN_FRAME;
                        TCPStartRetryTimer();
                        TCPIP.TCPStateMachine = LAST_ACK;
        TCPClose();
                        TCPIP.TransmitControl = 0;
                TCPIP.TCPFlags = 0;
                        TCPIP.TCPStateMachine = CLOSED;
                        TCPIP.SocketStatus = 0;
                }
        break;
```

```
                }
                default:
                break;
        }
        if (TCPIP.TransmitControl & SEND_FRAME2)
        {
                RequestSend(TCPIP.TxFrame2Size);
                if (Rdy4Tx())
                        SendFrame2();
                else {
                        TCPIP.TCPStateMachine = CLOSED;
                        TCPIP.SocketStatus = SOCK_ERR_ETHERNET;
                        TCPIP.TCPFlags = 0;
                }
                TCPIP.TransmitControl &= ~SEND_FRAME2;
}
        if (TCPIP.TransmitControl & SEND_FRAME1)
        {
                PrepareTCP_DATA_FRAME();
                RequestSend( TCPIP.TxFrame1Size );
                if (Rdy4Tx())
                        SendFrame1();
                else {
                        TCPIP.TCPStateMachine = CLOSED;
                        TCPIP.SocketStatus = SOCK_ERR_ETHERNET;
                        TCPIP.TCPFlags = 0;
                }
                TCPIP.TransmitControl &= ~SEND_FRAME1;
        }
}
void ProcessEthBroadcastFrame(void)
{
  UINT16 TargetIP[2];
  if (ReadFrameBE_EMAC() == FRAME_ARP)
    if (ReadFrameBE_EMAC() == HARDW_ETH10)
      if (ReadFrameBE_EMAC() == FRAME_IP)
        if (ReadFrameBE_EMAC() == IP_HLEN_PLEN)
          if (ReadFrameBE_EMAC() == OP_ARP_REQUEST)
          {
            DummyReadFrame_EMAC(6);
            CopyFromFrame_EMAC( &TCPIP.RecdFrameIP, 4);
            DummyReadFrame_EMAC(6);
            CopyFromFrame_EMAC(&TargetIP, 4);
            if (!memcmp(&MyIP, &TargetIP, 4))
              PrepareARP_ANSWER();
          }
}
```

```
//Aqui é feito uma verificação sobre o endereço de IP                    void ProcessEthIAFrame(void)
{
        UINT16 TargetIP[2];
        UINT8 ProtocolType;
        switch (ReadFrameBE_EMAC())
        {
  case FRAME_IP :
  {
   if ((ReadFrameBE_EMAC() & 0xFF00 ) == IP_VER_IHL)
   {
    TCPIP.RecdIPFrameLength = ReadFrameBE_EMAC();
    ReadFrameBE_EMAC();
    if (!(ReadFrameBE_EMAC() & (IP_FLAG_MOREFRAG | IP_FRAGOFS_MASK)))
    {
     ProtocolType = ReadFrameBE_EMAC() & 0xFF;
     ReadFrameBE_EMAC();
     CopyFromFrame_EMAC(&TCPIP.RecdFrameIP, 4);
     CopyFromFrame_EMAC(&TargetIP, 4);
     if ( !memcmp(&MyIP, &TargetIP, 4) )
                switch (ProtocolType) {
                  case PROT_ICMP : { ProcessICMPFrame(); break; }
                  case PROT_TCP  : { ProcessTCPFrame(); break; }
                  case PROT_UDP  : break;
                }
    }
   }
   break;
  }
 }
}
void ProcessICMPFrame(void)
{
}
void ProcessTCPFrame(void)
{
 UINT16 TCPSegSourcePort;
 UINT16 TCPSegDestPort;
 UINT32 TCPSegSeq;
 UINT32 TCPSegAck;
 UINT16 TCPCode;
 UINT8 TCPHeaderSize;
 UINT16 NrOfDataBytes;
 TCPSegSourcePort = ReadFrameBE_EMAC();
 TCPSegDestPort = ReadFrameBE_EMAC();
 if ( TCPSegDestPort != TCPIP.TCPLocalPort) return;
 TCPSegSeq = (UINT32)ReadFrameBE_EMAC() << 16;
 TCPSegSeq |= ReadFrameBE_EMAC();
```

```
TCPSegAck = (UINT32)ReadFrameBE_EMAC() << 16;
TCPSegAck |= ReadFrameBE_EMAC();
TCPCode = ReadFrameBE_EMAC();
TCPHeaderSize = (TCPCode & DATA_OFS_MASK) >> 10;
NrOfDataBytes = TCPIP.RecdIPFrameLength - IP_HEADER_SIZE - TCPHeaderSize;
if (NrOfDataBytes > MAX_TCP_RX_DATA_SIZE) return;
if (TCPHeaderSize > TCP_HEADER_SIZE)
 DummyReadFrame_EMAC(TCPHeaderSize - TCP_HEADER_SIZE);
switch (TCPIP.TCPStateMachine)
 {
 case LISTENING :
 {
  if (!(TCPCode & TCP_CODE_RST))
  {
    TCPIP.TCPRemotePort = TCPSegSourcePort;
    utils_memcpy(&TCPIP.RemoteMAC, &TCPIP.RecdFrameMAC, 6);
    utils_memcpy(&TCPIP.RemoteIP, &TCPIP.RecdFrameIP, 4);
     if (TCPCode & TCP_CODE_ACK)
    {
     TCPIP.TCPSeqNr = TCPSegAck;
     PrepareTCP_FRAME(TCP_CODE_RST);
    }
    else if (TCPCode & TCP_CODE_SYN)
    {
         TCPIP.TCPAckNr = TCPSegSeq + 1;
                                  TCPIP.TCPSeqNr = ((UINT32)TCPIP.ISNGenHigh << 16);
     TCPIP.TCPUNASeqNr = TCPIP.TCPSeqNr + 1;
     PrepareTCP_FRAME(TCP_CODE_SYN | TCP_CODE_ACK);
     TCPIP.LastFrameSent = TCP_SYN_ACK_FRAME;
     TCPStartRetryTimer();
     TCPIP.TCPStateMachine = SYN_RECD;
    }
  }
 }
 default :
 {
 if ( TCPSegSeq != TCPIP.TCPAckNr ) break;
 if ( TCPSegAck == TCPIP.TCPUNASeqNr )
  {
    TCPStopTimer();
    TCPIP.TCPSeqNr = TCPIP.TCPUNASeqNr;
    switch (TCPIP.TCPStateMachine)
    {
     case SYN_RECD :
     {
       TCPIP.TCPStateMachine = ESTABLISHED;
       TCPIP.sock_connected=1;
```

```
                    break;
                   }
                   case FIN_WAIT_1 : { TCPIP.TCPStateMachine = FIN_WAIT_2; break; }
                   case CLOSING :    { TCPIP.TCPStateMachine = TIME_WAIT; break; }
                   case LAST_ACK :
                                default:
                  }
                   if ( TCPIP.TCPStateMachine == ESTABLISHED )
                   TCPIP.SocketStatus |= SOCK_TX_BUF_RELEASED;
                 }
                if ( ( TCPIP.TCPStateMachine == ESTABLISHED) || ( TCPIP.TCPStateMachine == FIN_WAIT_1) || ( TCPIP.TCPStateMachine == FIN_WAIT_2) )
                   if (NrOfDataBytes)
                   if ( !TCPIP.sock_data_available )
                   {
                    DummyReadFrame_EMAC( 6 );
                    CopyFromFrame_EMAC( RxTCPBuffer, NrOfDataBytes );
                    TCPIP.TCPRxDataCount = NrOfDataBytes;
                    TCPIP.TCPAckNr += NrOfDataBytes;
                    PrepareTCP_FRAME( TCP_CODE_ACK );
                                        // Passo as informações para a variável
                                        TCPIP.sock_data_available_lenght=NrOfDataBytes;
                                        TCPIP.sock_data_available=1;
                   }
                  }
                 }
                }
                void PrepareARP_REQUEST(void)
                {
                }
                void PrepareARP_ANSWER(void)
                {
                 // Ethernet
                   utils_memcpy(&TxFrame2[ETH_DA_OFS], &TCPIP.RecdFrameMAC, 6);
                   utils_memcpy(&TxFrame2[ETH_SA_OFS], &MyMAC, 6);
                  *(UINT16 *)&TxFrame2[ETH_TYPE_OFS] = SWAPB(FRAME_ARP);
                 // ARP
                 *(UINT16 *)&TxFrame2[ARP_HARDW_OFS] = SWAPB(HARDW_ETH10);
                 *(UINT16 *)&TxFrame2[ARP_PROT_OFS] = SWAPB(FRAME_IP);
                 *(UINT16 *)&TxFrame2[ARP_HLEN_PLEN_OFS] = SWAPB(IP_HLEN_PLEN);
                 *(UINT16 *)&TxFrame2[ARP_OPCODE_OFS] = SWAPB(OP_ARP_ANSWER);
                  utils_memcpy(&TxFrame2[ARP_SENDER_HA_OFS], &MyMAC, 6);
                  utils_memcpy(&TxFrame2[ARP_SENDER_IP_OFS], &MyIP, 4);
                  utils_memcpy(&TxFrame2[ARP_TARGET_HA_OFS], &TCPIP.RecdFrameMAC, 6);
                  utils_memcpy(&TxFrame2[ARP_TARGET_IP_OFS], &TCPIP.RecdFrameIP, 4);
                  TCPIP.TxFrame2Size = ETH_HEADER_SIZE + ARP_FRAME_SIZE;
                  TCPIP.TransmitControl |= SEND_FRAME2;
```

```
}
void PrepareICMP_ECHO_REPLY(void)
{
 UINT16 ICMPDataCount;
}


void PrepareTCP_FRAME(UINT16 TCPCode)
{
 // Ethernet
 utils_memcpy(&TxFrame2[ETH_DA_OFS], &TCPIP.RemoteMAC, 6);
 utils_memcpy(&TxFrame2[ETH_SA_OFS], &MyMAC, 6);
 *(UINT16 *)&TxFrame2[ETH_TYPE_OFS] = SWAPB(FRAME_IP);
 // IP
 *(UINT16 *)&TxFrame2[IP_VER_IHL_TOS_OFS] = SWAPB(IP_VER_IHL | IP_TOS_D);
 if (TCPCode & TCP_CODE_SYN)
  *(UINT16 *)&TxFrame2[IP_TOTAL_LENGTH_OFS] = SWAPB(IP_HEADER_SIZE + TCP_HEADER_SIZE + TCP_OPT_MSS_SIZE);
 else
  *(UINT16 *)&TxFrame2[IP_TOTAL_LENGTH_OFS] = SWAPB(IP_HEADER_SIZE + TCP_HEADER_SIZE);
 *(UINT16 *)&TxFrame2[IP_IDENT_OFS] = 0;
 *(UINT16 *)&TxFrame2[IP_FLAGS_FRAG_OFS] = 0;
 *(UINT16 *)&TxFrame2[IP_TTL_PROT_OFS] = SWAPB((DEFAULT_TTL << 8) | PROT_TCP);
 *(UINT16 *)&TxFrame2[IP_HEAD_CHKSUM_OFS] = 0;
 utils_memcpy(&TxFrame2[IP_SOURCE_OFS], &MyIP, 4);
 utils_memcpy(&TxFrame2[IP_DESTINATION_OFS], &TCPIP.RemoteIP, 4);
 *(UINT16 *)&TxFrame2[IP_HEAD_CHKSUM_OFS] = CalcChecksum(&TxFrame2[IP_VER_IHL_TOS_OFS], IP_HEADER_SIZE, 0);
 // TCP
 WriteWBE(&TxFrame2[TCP_SRCPORT_OFS], TCPIP.TCPLocalPort);
 WriteWBE(&TxFrame2[TCP_DESTPORT_OFS], TCPIP.TCPRemotePort);
 WriteDWBE(&TxFrame2[TCP_SEQNR_OFS], TCPIP.TCPSeqNr);
 WriteDWBE(&TxFrame2[TCP_ACKNR_OFS], TCPIP.TCPAckNr);
 *(UINT16 *)&TxFrame2[TCP_WINDOW_OFS] = SWAPB(MAX_TCP_RX_DATA_SIZE);
 *(UINT16 *)&TxFrame2[TCP_CHKSUM_OFS] = 0;
 *(UINT16 *)&TxFrame2[TCP_URGENT_OFS] = 0;
 if (TCPCode & TCP_CODE_SYN)
 {
  *(UINT16 *)&TxFrame2[TCP_DATA_CODE_OFS] = SWAPB(0x6000 | TCPCode);
  *(UINT16 *)&TxFrame2[TCP_DATA_OFS] = SWAPB(TCP_OPT_MSS);
  *(UINT16 *)&TxFrame2[TCP_DATA_OFS + 2] = SWAPB(MAX_TCP_RX_DATA_SIZE);
  *(UINT16 *)&TxFrame2[TCP_CHKSUM_OFS] = CalcChecksum(&TxFrame2[TCP_SRCPORT_OFS], TCP_HEADER_SIZE + TCP_OPT_MSS_SIZE, 1);
  TCPIP.TxFrame2Size = ETH_HEADER_SIZE + IP_HEADER_SIZE + TCP_HEADER_SIZE + TCP_OPT_MSS_SIZE;
 }
 else
 {
  *(UINT16 *)&TxFrame2[TCP_DATA_CODE_OFS] = SWAPB(0x5000 | TCPCode);
```

```
        *(UINT16 *)&TxFrame2[TCP_CHKSUM_OFS] = CalcChecksum(&TxFrame2[TCP_SRCPORT_OFS], TCP_HEADER_SIZE, 1);
        TCPIP.TxFrame2Size = ETH_HEADER_SIZE + IP_HEADER_SIZE + TCP_HEADER_SIZE;
      }
      TCPIP.TransmitControl |= SEND_FRAME2;
    }
    void PrepareTCP_DATA_FRAME(void)
    {
      // Ethernet
      utils_memcpy(&TxFrame1[ETH_DA_OFS], &TCPIP.RemoteMAC, 6);
      utils_memcpy(&TxFrame1[ETH_SA_OFS], &MyMAC, 6);
      *(UINT16 *)&TxFrame1[ETH_TYPE_OFS] = SWAPB(FRAME_IP);
      // IP
      *(UINT16 *)&TxFrame1[IP_VER_IHL_TOS_OFS] = SWAPB(IP_VER_IHL | IP_TOS_D);
      WriteWBE(&TxFrame1[IP_TOTAL_LENGTH_OFS], IP_HEADER_SIZE + TCP_HEADER_SIZE + TCPIP.TCPTxDataCount);
      *(UINT16 *)&TxFrame1[IP_IDENT_OFS] = 0;
      *(UINT16 *)&TxFrame1[IP_FLAGS_FRAG_OFS] = 0;
      *(UINT16 *)&TxFrame1[IP_TTL_PROT_OFS] = SWAPB((DEFAULT_TTL << 8) | PROT_TCP);
      *(UINT16 *)&TxFrame1[IP_HEAD_CHKSUM_OFS] = 0;
      utils_memcpy(&TxFrame1[IP_SOURCE_OFS], &MyIP, 4);
      utils_memcpy(&TxFrame1[IP_DESTINATION_OFS], &TCPIP.RemoteIP, 4);
      *(UINT16 *)&TxFrame1[IP_HEAD_CHKSUM_OFS] = CalcChecksum(&TxFrame1[IP_VER_IHL_TOS_OFS], IP_HEADER_SIZE, 0);
      // TCP
      WriteWBE(&TxFrame1[TCP_SRCPORT_OFS], TCPIP.TCPLocalPort);
      WriteWBE(&TxFrame1[TCP_DESTPORT_OFS], TCPIP.TCPRemotePort);
      WriteDWBE(&TxFrame1[TCP_SEQNR_OFS], TCPIP.TCPSeqNr );
      WriteDWBE(&TxFrame1[TCP_ACKNR_OFS], TCPIP.TCPAckNr );
      *(UINT16 *)&TxFrame1[TCP_DATA_CODE_OFS] = SWAPB(0x5000 | TCP_CODE_ACK);
      *(UINT16 *)&TxFrame1[TCP_WINDOW_OFS] = SWAPB(MAX_TCP_RX_DATA_SIZE);
      *(UINT16 *)&TxFrame1[TCP_CHKSUM_OFS] = 0;
      *(UINT16 *)&TxFrame1[TCP_URGENT_OFS] = 0;
      *(UINT16 *)&TxFrame1[TCP_CHKSUM_OFS] = CalcChecksum(&TxFrame1[TCP_SRCPORT_OFS], TCP_HEADER_SIZE + TCPIP.TCPTxDataCount, 1);
    }
    UINT16 CalcChecksum(void *Start, UINT16 Count, UINT8 IsTCP)
    {
      UINT32 Sum = 0;
      UINT16 * piStart;
      if (IsTCP) {
        Sum += MyIP[0];
        Sum += MyIP[1];
        Sum += TCPIP.RemoteIP[0];
        Sum += TCPIP.RemoteIP[1];
        Sum += SwapBytes(Count);
        Sum += SWAPB(PROT_TCP);
      }
```

```
  piStart = Start;
  while (Count > 1) {
    Sum += *piStart++;
    Count -= 2;
  }
  if (Count)
    Sum += *(UINT8 *)piStart;
  while (Sum >> 16)
    Sum = (Sum & 0xFFFF) + (Sum >> 16);
  return ~Sum;
}
void TCPStartRetryTimer(void)
{
}
void TCPStartTimeWaitTimer(void)
{
}
void TCPRestartTimer(void)
{
  TCPIP.TCPTimer = 0;
}
void TCPStopTimer(void)
{
  TCPIP.TCPFlags &= ~TCP_TIMER_RUNNING;
}
void TCPHandleRetransmission(void)
{
}
void TCPHandleTimeout(void)
{
}
void SendFrame1(void)
{
  CopyToFrame_EMAC( TxFrame1, TCPIP.TxFrame1Size ) ;
}
void SendFrame2(void)
{
  CopyToFrame_EMAC( TxFrame2, TCPIP.TxFrame2Size);
}
void WriteWBE(UINT8 *Add, UINT16 Data)
{
  *Add++ = Data >> 8;
  *Add = (char)Data;
}
 void WriteDWBE(UINT8 *Add, UINT32 Data)
{
  *Add++ = Data >> 24;
```

```
  *Add++ = Data >> 16;
  *Add++ = Data >> 8;
  *Add = (UINT8)Data;
}
UINT16 SwapBytes(UINT16 Data)
{
  return (Data >> 8) | (Data << 8);
}
```

APÊNDICE G – ROTINA DE TRADUÇÃO PARA EMAC

```
//Camada Mac para o chip LPC2368. Protocolo aberto
#include <LPC23xx.h>
#include "EMAC.h"
#include "tcpip.h"
#include "types.h"
static UINT16 *rptr;
static UINT16 *tptr;
void write_PHY (int PhyReg, int Value)
{
  UINT32 tout;
  MAC_MADR = DP83848C_DEF_ADR | PhyReg;
  MAC_MWTD = Value;
  tout = 0;
  for (tout = 0; tout < MII_WR_TOUT; tout++) {
    if ((MAC_MIND & MIND_BUSY) == 0) {
      break;
    }
  }
}
UINT16 read_PHY (UINT8 PhyReg)
{
  UINT32 tout;
  MAC_MADR = DP83848C_DEF_ADR | PhyReg;
  MAC_MCMD = MCMD_READ;
  tout = 0;
  for (tout = 0; tout < MII_RD_TOUT; tout++) {
    if ((MAC_MIND & MIND_BUSY) == 0) {
      break;
    }
  }
  MAC_MCMD = 0;
  return (MAC_MRDD);
}
void rx_descr_init (void)
{
  UINT16 i;
  for (i = 0; i < NUM_RX_FRAG; i++) {
    RX_DESC_PACKET(i)  = RX_BUF(i);
    RX_DESC_CTRL(i)    = RCTRL_INT | (ETH_FRAG_SIZE-1);
    RX_STAT_INFO(i)    = 0;
    RX_STAT_HASHCRC(i) = 0;
  }
  MAC_RXDESCRIPTOR    = RX_DESC_BASE;
  MAC_RXSTATUS        = RX_STAT_BASE;
```

```
  MAC_RXDESCRIPTORNUM = NUM_RX_FRAG-1;
  MAC_RXCONSUMEINDEX  = 0;
}
void tx_descr_init (void) {
  UINT16 i;
  for (i = 0; i < NUM_TX_FRAG; i++) {
    TX_DESC_PACKET(i) = TX_BUF(i);
    TX_DESC_CTRL(i)   = 0;
    TX_STAT_INFO(i)   = 0;
  }
  MAC_TXDESCRIPTOR    = TX_DESC_BASE;
  MAC_TXSTATUS        = TX_STAT_BASE;
  MAC_TXDESCRIPTORNUM = NUM_TX_FRAG-1;
  MAC_TXPRODUCEINDEX  = 0;
}
void Init_EMAC(void)
{
  UINT16 regv,id1,id2;
  UINT32 tout;
   PCONP |= 0x40000000;
  if (MAC_MODULEID == OLD_EMAC_MODULE_ID) {
    PINSEL2 = 0x50151105;
  }
  else {
    PINSEL2 = 0x50150105;
  }
  PINSEL3 = (PINSEL3 & ~0x0000000F) | 0x00000005;
  MAC_MAC1 = MAC1_RES_TX | MAC1_RES_MCS_TX | MAC1_RES_RX | MAC1_RES_MCS_RX |
             MAC1_SIM_RES | MAC1_SOFT_RES;
  MAC_COMMAND = CR_REG_RES | CR_TX_RES | CR_RX_RES;
  for (tout = 100; tout; tout--);
  MAC_MAC1 = MAC1_PASS_ALL;
  MAC_MAC2 = MAC2_CRC_EN | MAC2_PAD_EN;
  MAC_MAXF = ETH_MAX_FLEN;
  MAC_CLRT = CLRT_DEF;
  MAC_IPGR = IPGR_DEF;
  MAC_COMMAND = CR_RMII | CR_PASS_RUNT_FRM;
  MAC_SUPP = SUPP_RES_RMII;
  for (tout = 100; tout; tout--);
  MAC_SUPP = 0;
  write_PHY (PHY_REG_BMCR, 0x8000);
  for (tout = 0; tout < 0x100000; tout++) {
    regv = read_PHY (PHY_REG_BMCR);
    if (!(regv & 0x8000)) {
      break;
    }
  }
```

```
  id1 = read_PHY (PHY_REG_IDR1);
  id2 = read_PHY (PHY_REG_IDR2);
  if (((id1 << 16) | (id2 & 0xFFF0)) == DP83848C_ID) {
    write_PHY (PHY_REG_BMCR, PHY_AUTO_NEG);
    for (tout = 0; tout < 0x100000; tout++) {
      regv = read_PHY (PHY_REG_BMSR);
      if (regv & 0x0020) {
        break;
      }
    }
  }
  for (tout = 0; tout < 0x10000; tout++) {
    regv = read_PHY (PHY_REG_STS);
    if (regv & 0x0001) {
      break;
    }
  }
  if (regv & 0x0004) {
    MAC_MAC2    |= MAC2_FULL_DUP;
    MAC_COMMAND |= CR_FULL_DUP;
    MAC_IPGT     = IPGT_FULL_DUP;
  }
  else {
    MAC_IPGT = IPGT_HALF_DUP;
  }
  if (regv & 0x0002) {
    MAC_SUPP = 0;
  }
  else {
    MAC_SUPP = SUPP_SPEED;
  }
  MAC_SA0 = (MYMAC_6 << 8) | MYMAC_5;
  MAC_SA1 = (MYMAC_4 << 8) | MYMAC_3;
  MAC_SA2 = (MYMAC_2 << 8) | MYMAC_1;
  rx_descr_init ();
  tx_descr_init ();
  MAC_RXFILTERCTRL = RFC_BCAST_EN | RFC_PERFECT_EN;
  MAC_INTENABLE = INT_RX_DONE | INT_TX_DONE;
  MAC_INTCLEAR  = 0xFFFF;
  MAC_COMMAND  |= (CR_RX_EN | CR_TX_EN);
  MAC_MAC1     |= MAC1_REC_EN;
}
UINT16 ReadFrame_EMAC(void)
{
  return (*rptr++);
}
UINT16 ReadFrameBE_EMAC(void)
```

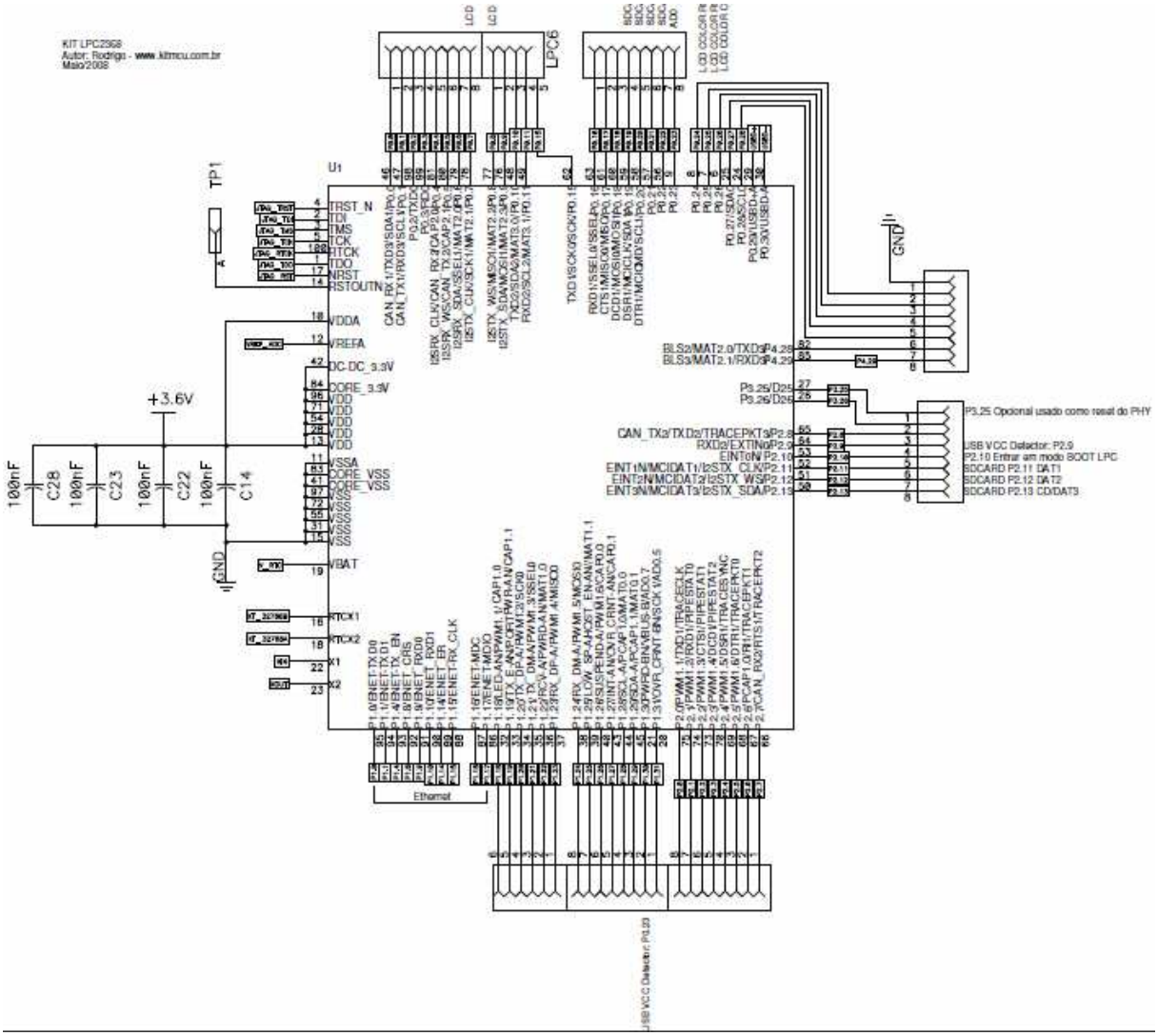
```
{
  UINT16 ReturnValue;
  ReturnValue = SwapBytes (*rptr++);
  return (ReturnValue);
}
void CopyFromFrame_EMAC(void *Dest, UINT16 Size)
{
  UINT16 * piDest;
  piDest = Dest;
  while (Size > 1) {
    *piDest++ = ReadFrame_EMAC();
    Size -= 2;
  }
  if (Size) {
    *(UINT8 *)piDest = (char)ReadFrame_EMAC();
  }
}
void DummyReadFrame_EMAC(UINT16 Size)
{
  while (Size > 1) {
    ReadFrame_EMAC();
    Size -= 2;
  }
}
UINT16 StartReadFrame(void) {
  UINT16 RxLen;
  UINT16 idx;
  idx = MAC_RXCONSUMEINDEX;
  RxLen = (RX_STAT_INFO(idx) & RINFO_SIZE) - 3;
  rptr = (UINT16 *)RX_DESC_PACKET(idx);
  return(RxLen);
}
void EndReadFrame(void) {
  UINT16 idx;
  idx = MAC_RXCONSUMEINDEX;
  if (++idx == NUM_RX_FRAG) idx = 0;
  MAC_RXCONSUMEINDEX = idx;
}
UINT8 CheckFrameReceived(void) {
  if (MAC_RXPRODUCEINDEX != MAC_RXCONSUMEINDEX)
    return(1);
  else
    return(0);
}
void RequestSend(UINT16 FrameSize)
{
  UINT16 idx;
```

```
  idx  = MAC_TXPRODUCEINDEX;
  tptr = (UINT16 *)TX_DESC_PACKET(idx);
  TX_DESC_CTRL(idx) = FrameSize | TCTRL_LAST;
}
unsigned int Rdy4Tx(void)
{
  return (1);
}
void WriteFrame_EMAC(UINT16 Data)
{
  *tptr++ = Data;
}
void CopyToFrame_EMAC(void *Source, unsigned int Size)
{
  UINT16 * piSource;
  UINT16 idx;
  piSource = Source;
  Size = (Size + 1) & 0xFFFE;
  while (Size > 0) {
    WriteFrame_EMAC(*piSource++);
    Size -= 2;
  }
  idx = MAC_TXPRODUCEINDEX;
  if (++idx == NUM_TX_FRAG) idx = 0;
  MAC_TXPRODUCEINDEX = idx;
}
```

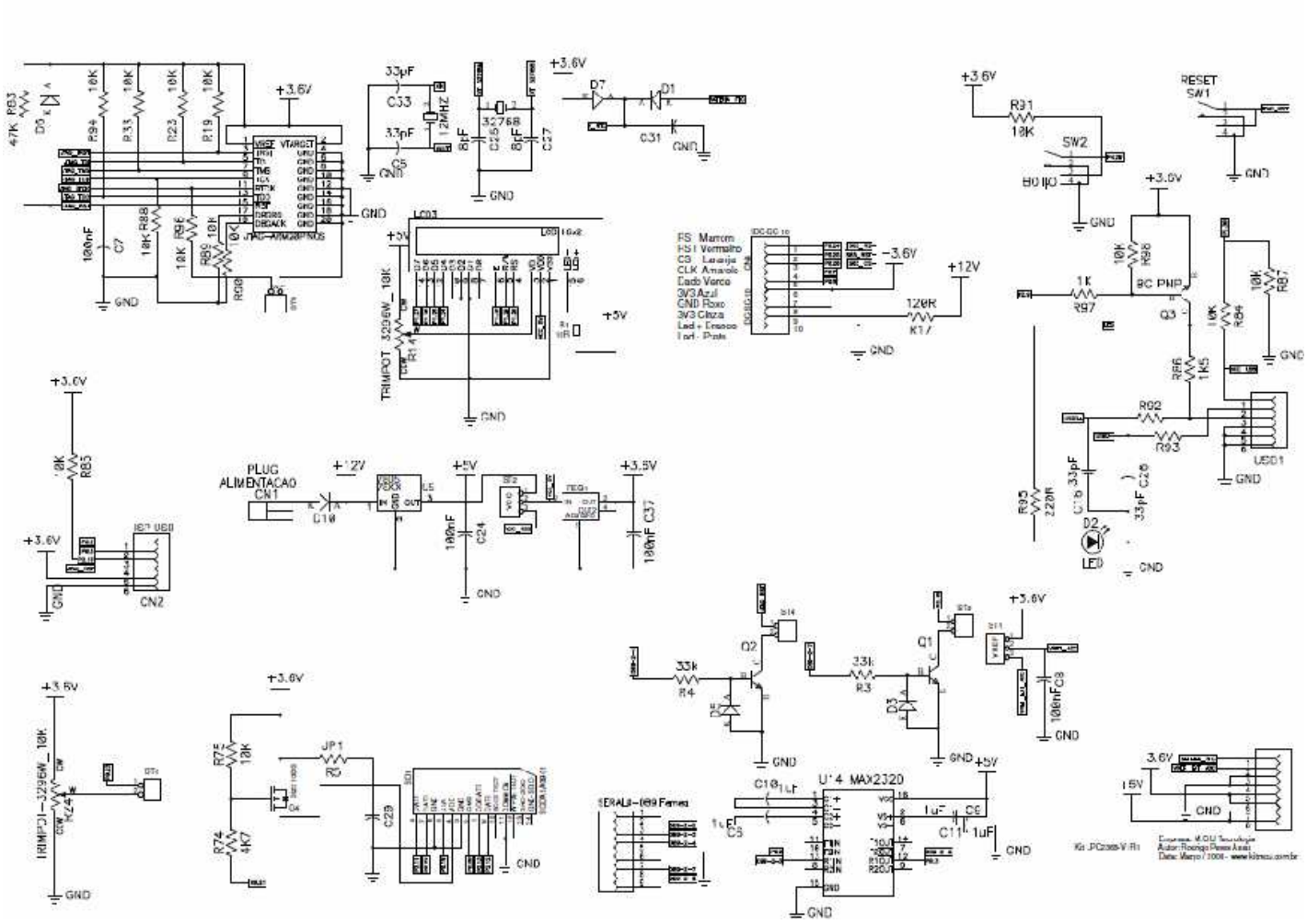
ANEXOS

ANEXO A– ESQUEMA ELÉTRICO CP-FLEX

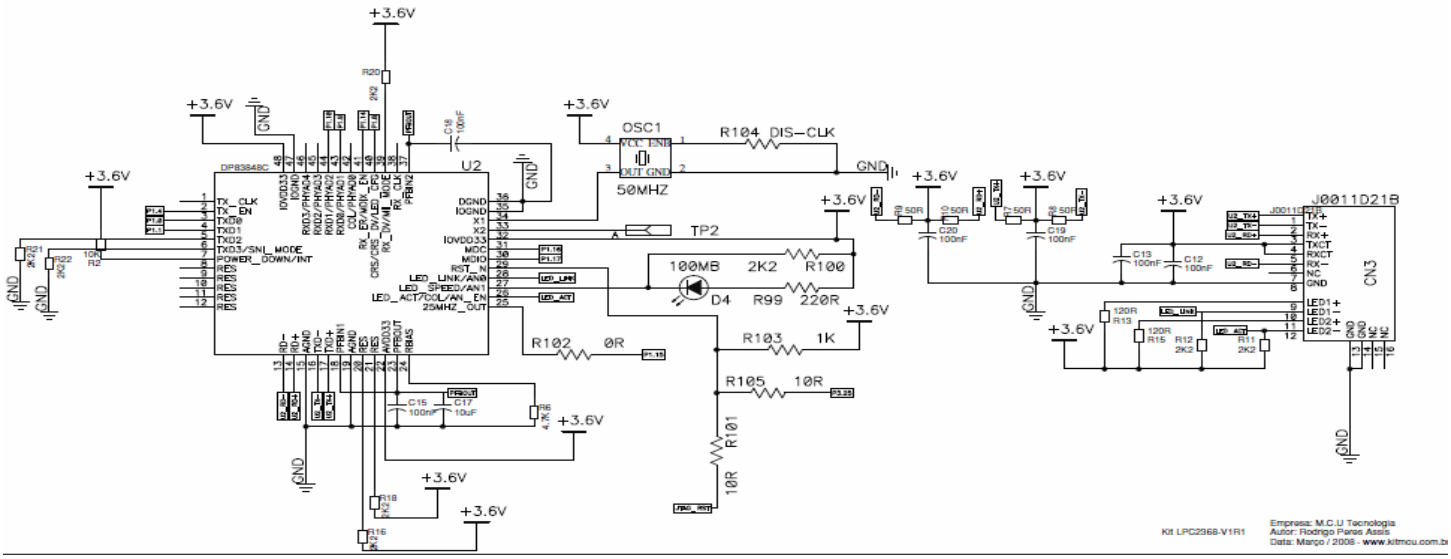
MICROPROCESSADOR:



PERIFÉRICOS:



PHYTER:



ANEXO B – PINAGEM DO CP-FLEX

K1	Alimentação: 12VDC à 24VDC
K2	Alimentação GND
K3	GND
K4	12V Saída
K5	GND
K6	GND
K7	RS232-RX0
K8	RS232-TX0
K9	RS485 B
K10	RS485 A
K11	Digital INPUTS(0~3) Comum
K12	INPUT0
K13	INPUT1
K14	INPUT2
K15	INPUT3
K16	Digital Outputs(0~3) Comum (TIPO N)
K17	OUTPUT0
K18	OUTPUT1
K19	OUTPUT2
K20	OUTPUT3
K21	RELE2 NA
K22	RELE2 CM
K23	RELE2 NF
K24	RELE4 NA
K25	RELE4 CM
K26	RELE4 NF
K27	RELE6 NA
K28	RELE6 CM
K29	RELE6 NF
K30	RELE8 NA
K31	RELE8 CM
K32	RELE8 NF

Z1	AN0 – Entrada analógica 0
Z2	AN1 – Entrada analógica 1
Z3	AN0 – Entrada analógica 2
Z4	Não conectado. Não usado
Z5	GND
Z6	GND
Z7	RS232-RX1
Z8	RS232-TX1
Z9	CANL
Z10	CANH
Z11	Digital INPUTS(4~7) Comum
Z12	INPUT4
Z13	INPUT5
Z14	INPUT6
Z15	INPUT7
Z16	Digital Outputs(4~7) Comum (TIPO N)
Z17	OUTPUT4
Z18	OUTPUT5
Z19	OUTPUT6
Z20	OUTPUT7
Z21	RELE1 NA
Z22	RELE1 CM
Z23	RELE1 NF
Z24	RELE3 NA
Z25	RELE3 CM
Z26	RELE3 NF
Z27	RELE5 NA
Z28	RELE5 CM
Z29	RELE5 NF
Z30	RELE7 NA
Z31	RELE7 CM
Z32	RELE7 NF