THIAGO ALVES DE ARAÚJO

SGM-W: SISTEMA PARA GERENCIAMENTO DE MICROCONTROLADORES VIA INTERFACE *WEB*

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como exigência do curso de Ciência da Computação para a obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS – BRASIL
2007

THIAGO ALVES DE ARAÚJO

SGM-W: SISTEMA PARA GERENCIAMENTO DE MICROCONTROLADORES VIA INTERFACE *WEB*

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como exigência do curso de Ciência da Computação para a obtenção do título de Bacharel em Ciência da Computação.

Área de concentração:
Sistemas Embarcados

Orientador:
Prof. Wilian Soares Lacerda

LAVRAS
MINAS GERAIS – BRASIL
2007

Ficha Catalográfica preparada pela Divisão de Processos Técnico da Biblioteca Central da UFLA

Araújo, Thiago Alves de

SGM-W: Sistema para Gerenciamento de Microcontroladores via Interface *Web* / Thiago Alves de Araújo. Lavras – Minas Gerais, 2007. 185p.: il.

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação

1. Automação. 2. Sistemas Embarcados. 3. Tecnologia de Microeletrônica. I. ARAUJO, T. A. II. Universidade Federal de Lavras. III. Título.

THIAGO ALVES DE ARAÚJO

SGM-W: SISTEMA PARA GERENCIAMENTO DE MICROCONTROLADORES VIA INTERFACE *WEB*

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como exigência do curso de Ciência da Computação para a obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 14 de janeiro de 2008

_____

Prof. Giovanni Francisco Rabelo

_____

Prof. João Carlos Giacomin

_____

Prof. Wilian Soares Lacerda
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

Dedico este trabalho a toda minha família, especialmente a meus pais Maria e Vantuil, pelo intenso apoio, motivação e confiança. A meus irmãos João Victor, Guilherme e Diogo, que esta conquista possa servir como referência de caminho a ser percorrido. A minha companheira Darlene que sempre esteve ao meu lado. E aos meus avós por sempre estarem torcendo por mim.

Agradecimentos

Aos meus pais, pela confiança, incentivo e apoio dedicado a esta minha batalha, não basta apenas agradecer. Sempre serão parte do mérito dessa conquista.

A Darlene, mais que minha companheira, agradeço por sempre estar ao meu lado, pela paciência e amor compartilhado comigo.

As minhas avós Ilda e Gasparina, agradeço pelas orações dispensadas a mim, que sempre foram válidas.

Ao meu avô Baltazar, sou muito grato pelo apoio e grande verdadeiro amigo que sempre foi.

Ao meu avô Joaquim, também agradeço pela amizade e confiança.

Aos demais familiares que fielmente me desejaram o bem, sou carinhosamente muito grato pelo imenso apoio.

Ao meu orientador e amigos que colaboraram para esta realização, meu sincero agradecimento.

A todos que realmente acreditaram na minha determinação e competência em alcançar mais esta realização, muito obrigado.

RESUMO

SGM-W: SISTEMA PARA GERENCIAMENTO DE MICROCONTROLADORES VIA INTERFACE *WEB*

Este trabalho apresenta um sistema de comunicação para monitoramento e controle de microcontroladores, utilizando uma comunicação remota entre o usuário e o microcontrolador através da rede mundial de computadores, a Internet. A interface do sistema é feita por uma página *Web*, acessada por um *browser* em um computador remoto, onde o usuário pode verificar os estados dos periféricos do microcontrolador ou solicitar a alteração dos mesmos em diversas aplicações que necessitem de acesso remoto.

Palavras-chave: Sistema Embarcado, Microcontrolador, Sistema para Gerenciamento.

ABSTRACT

SGM-W: MANAGEMENT SYSTEM OF MICROCONTROLLERS ON WEB INTERFACE

This work shows a system of communication for monitoring and control of microcontrollers, using a remote communication between the user and the microcontroller through the world network of computers, the Internet. The interface of the system is made for a Web page, accessed by a browser on a remote computer, where the user be able to verify the states of the peripherals of the microcontroller or to request them in several applications that require remote access.

Keywords: Embedded System, Microcontroller, Management System.

SUMÁRIO

LISTA DE FIGURAS.....iv

1. INTRODUÇÃO.....1
1.1. Motivação.....1
1.2. Objetivos.....1
1.3. Estrutura do trabalho.....2

2. MICROCONTROLADORES.....4
2.1. Definição.....4
2.2. Microcontrolador PIC 18F458.....5
2.3. Programando o PIC.....8

3. KIT PICLAB 4C.....13
3.1. Componentes básicos.....13
3.2. Outros componentes do kit.....14

4. KIT KPME-10.....17
4.1. Introdução.....17
4.2. O servidor HTTP da Microchip.....18

5. COMUNICAÇÃO DE DADOS.....23
5.1. Comunicação serial.....23
5.2. Redes de computadores.....24
5.3. Redes de computadores.....25
5.3.1. Introdução.....25
5.3.2. HTTP: O protocolo padrão da Web.....26
5.3.3. Páginas da Web com a HTML.....28

6. DESENVOLVIMENTO.....30
6.1. Tipo da pesquisa.....30
6.2. Sistema SGM-W.....30
6.2.1. Introdução.....30
6.2.2. Protocolo de comandos.....32
6.2.2.1. Comandos recebidos no PIC gerenciado.....33
6.2.2.2. Comandos recebidos no servidor Web.....37
6.2.3. Software do servidor Web.....38
6.2.4. Software do microcontrolador PIC gerenciado.....42

7. RESULTADOS E DISCUSSÃO.....46

8. CONCLUSÕES.....54

REFERÊNCIAS BIBLIOGRÁFICAS.....56

ANEXO I – CÓDIGO FONTE DO MÓDULO DE COMUNICAÇÃO SGM-W DO PIC MONITORADO.....58

ANEXO II – CÓDIGO FONTE DO SERVIDOR WEB.....75

ANEXO III – PÁGINAS WEB ARMAZENADAS NO SERVIDOR.....158

LISTA DE FIGURAS

Figura 2.2.1 - Diagrama de pinos do PIC 18F458 PDIP.....8

Figura 2.3.1 - Ambiente de desenvolvimento “PCW C Compiler IDE”, da CCS.....11

Figura 2.3.2 - IC-Prog, software para gravação de circuitos integrados programáveis seriais 12

Figura 3.1.1 - Kit Piclab 4C.....13

Figura 3.2.1 - Circuito mínimo com o PIC 18F458.....14

Figura 4.1.1 - Kit KPME-10 para desenvolvimento de sistemas com conectividade Ethernet/Internet.....17

Figura 4.2.1 - Implementação da função HTTPGetVar.....21

Figura 4.2.2 - Conteúdo da página Web para alteração de variáveis.....21

Figura 4.2.3 - Código da página Web para alteração de variáveis.....22

Figura 4.2.4 - Exemplo de implementação da função HTTPExecCmd.....22

Figura 5.1.1: Sinal de dados na comunicação serial assíncrona.....24

Figura 5.3.2.1: Requisição-resposta no protocolo HTTP.....26

Figura 6.2.1.1 - Conexões entre os dispositivos envolvidos no SGM-W.....31

Figura 6.2.3.1 - Página Web principal.....40

Figura 6.2.3.2 - Página de monitoramento de pinos.....40

Figura 6.2.3.3 - Frame para envio de comandos ao PIC monitorado.....40

Figura 6.2.3.4 - Página de monitoramento de serviços.....40

Figura 6.2.3.5 - Página de monitoramento da memória EEPROM.....41

Figura 6.2.3.6 - Página de monitoramento da memória RAM.....41

Figura 7.1 - Conexão dos dispositivos do sistema SGM-W.....46

Figura 7.2 - Tela inicial da página Web do sistema.....47

Figura 7.3 - Microcontrolador gerenciado ainda sem conexão com o servidor.....47

Figura 7.4 - Envio de comando de pedido de conexão ao microcontrolador gerenciado.…48

Figura 7.5 - Microcontrolador gerenciado após pedido de conexão.....48

Figura 7.6 - Página de monitoramento dos pinos após conexão estabelecida.....48

Figura 7.7 - Página de monitoramento de periféricos.....49

Figura 7.8 - Página de monitoramento da memória EEPROM.....49

Figura 7.9 - Página de monitoramento da memória RAM.....49

Figura 7.10 - Envio de comando para alteração da porta D.....50

Figura 7.11 - Página de monitoramento dos pinos, atualizada após alteração no PIC gerenciado.....50

Figura 7.12 - Envio de comando de ativação e alteração do duty cicle do gerador de PWM 51

Figura 7.13 - Página de monitoramento dos periféricos, atualizada após alteração no PWM do PIC gerenciado.....52

Figura 7.14 - Envio de comando para alteração do valor da memória EEPROM.....53

Figura 7.15 - Página de monitoramento da memória EEPROM, atualizada após sua alteração no PIC gerenciado.....53

1. INTRODUÇÃO

1.1. Motivação

Nos dias de hoje, é muito comum nos depararmos com algum dispositivo eletrônico que possua um microcontrolador embutido. Aparelhos como celulares, tocadores de DVDs, tocadores de MP3, centrais de injeção eletrônica de automóveis, têm, dentro de si, um *chip* com um microprocessador e seus periféricos.

Esses dispositivos estão cada vez mais presentes nos produtos eletrônicos que utilizamos, como fornos de microondas e aparelhos de TV, a fim de facilitar a vida dos usuários e aumentar as possibilidades de desenvolvimento de um produto que possua inúmeras funcionalidades.

Para uma maior comodidade em determinadas aplicações, o gerenciamento remoto de um microcontrolador fixado em seu ambiente de atuação, utilizando um sistema de comunicação que possa ser acessado através de diversos dispositivos diferentes espalhados pelo mundo ofereceria muitos benefícios, tais como a economia de tempo e recursos. Desta forma, através de diferentes tipos de elementos processados como *handhelds*, celulares e microcomputadores, cada vez mais comuns nos dias atuais, seria possível visualizar as tarefas desempenhadas pelo microcontrolador e enviar solicitações de execução de tarefas pelo mesmo, tudo de maneira remota e a partir de diferentes meios.

1.2. Objetivos

Este trabalho tem como propósito geral o desenvolvimento de um sistema de monitoramento e controle de microcontroladores via interface *Web*, ou seja, utilizando a rede mundial de computadores e um browser de navegação.

Com o sistema é possível, no monitoramento, verificar os estados dos periféricos do

microcontrolador em tempo real e, no controle, enviar comandos de pedidos de realização de tarefas pelo dispositivo. No monitoramento, por exemplo, são mostrados os estados dos pinos, estado de ativação das interfaces de comunicação e ocorrência de interrupções no microcontrolador. No controle é possível, por exemplo, ativar um serviço de comunicação, acionar dispositivos externos ligados ao microcontrolador e gravar dados nas memórias do microcontrolador. Todas essas tarefas são realizadas remotamente, através de uma página *Web*.

1.3. Estrutura do trabalho

Primeiramente, nos capítulos de 2 a 6, este documento apresenta uma base teórica para um melhor entendimento do trabalho pelo leitor. Posteriormente, nos capítulos de 7 a 9, são mostrados o sistema desenvolvido, os resultados obtidos e as conclusões do trabalho, respectivamente. No final estão anexados materiais de apoio para complementar o entendimento do trabalho.

No capítulo 2 são apresentados os principais conceitos sobre microcontroladores. Na sua primeira seção é definido o que são os microcontroladores. A seguir é apresentado um modelo específico destes dispositivos, o PIC 18F458, com uma descrição de suas características e demonstração de seus principais periféricos disponíveis. Assim que definidas e explicadas as principais características, é mostrado como é o processo de programação dos microcontroladores a fim de executarem as tarefas desejadas pelo usuário.

Os capítulos 3 e 4 descrevem dois kits de placas eletrônicas didáticas, Piclab 4C da Vidal Microcontroladores (SILVA JR., 2006) e KPME-10 da 2EI (2EI, 2007), utilizadas no trabalho para o desenvolvimento do sistema.

Após demonstrados os microcontroladores e o kit de utilização dos mesmos, no capítulo 5 é apresentado um meio para promover a comunicação destes dispositivos com outros elementos eletrônicos de processamento, especificamente a comunicação serial

padrão RS-232. Também comenta sobre as redes de computadores e sua principal aplicação, a *Web* na Internet. Inicialmente, a *Web* é definida para que sejam apresentados o principal protocolo utilizado na mesma, o HTTP, e a maneira básica de implementação de uma página *Web*, utilizando a linguagem HTML.

Logo após expostas as bases teóricas para o melhor entendimento do trabalho, no capítulo 6 é especificado o tipo de metodologia adotada para a execução do trabalho, bem como os procedimentos para a realização do mesmo.

O capítulo 7 apresenta testes efetuados no sistema desenvolvido, bem como a descrição de utilização do mesmo.

No capítulo 8 estão as conclusões do trabalho.

2. MICROCONTROLADORES

2.1. Definição

Segundo Souza (2003), um microcontrolador é um pequeno componente eletrônico com inteligência programável utilizado para controle de processos lógicos. O autor explica que o controle de processos lógicos diz respeito ao controle de periféricos como LEDs, botões, *displays*, sensores, dentre outros, onde suas operações dependem do estado de tais periféricos. Os microcontroladores possuem inteligência programável, pois toda a lógica de operação dita é codificada em forma de um programa e o mesmo é gravado no dispositivo. Os microcontroladores são pequenos componentes, pois possuem todos os periféricos necessários para o controle de processos em uma única placa de silício (ou *chip*).

Para Silva Jr. (1997), um microcontrolador é um componente eletrônico que possui um microprocessador e todos os seus periféricos embutidos em uma só pastilha, facilitando o desenvolvimento de sistemas pequenos e baratos, embora complexos e sofisticados. O autor já chegou a definir um microcontrolador como um microcomputador de um só *chip*.

O funcionamento básico de um microcontrolador é bastante semelhante ao de um microcomputador. Tem-se um *chip* com um microprocessador, memórias, registradores e diversos outros periféricos necessários para que ele processe informações e forneça respostas, semelhante ao funcionamento básico de um computador PC.

Hoje existem diversos modelos de microcontroladores disponíveis no mercado, que vão desde aqueles com apenas 6 pinos, minúsculos, ideais em aplicações de inteligência artificial para dispositivos mecatrônicos e até *chips* maiores, mais completos, que já possuem diversos periféricos inclusos como interfaces seriais RS232, CAN, USB, conversores analógico/digitais, gerador de sinal PWM, dentre outros. Devido a essa grande variedade de *chips* microcontroladores, temos diversas opções de escolha que vão desde o

mais simples ao mais completo e do menor ao maior, que devem ser analisados para serem utilizados nas diversas aplicações que mais lhes convêm.

2.2. Microcontrolador PIC 18F458

Entre os diversos modelos de microcontroladores disponíveis no mercado, nesta seção é apresentado um em especial, uma versão de microcontrolador fabricado pela Microchip, o PIC 18F458. É um *chip* de 40 pinos (Figura 2.2.1), sendo um dos mais completos atualmente no que diz respeito aos periféricos inclusos. Ele possui instruções de 16 bits, o que garante uma boa eficiência a um custo baixo.

Dos 40 pinos existentes no 18F458, 33 são para entrada ou saída de informações, os quais são divididos em 5 portas (A, B, C, D, E) que podem ser acessadas diretamente. As características de cada porta, serão explicadas nas próximas seções.

As principais características do PIC 18F458 são (MICROCHIP, 2004):

Quanto ao desempenho:

- Capacidade de executar até 10 MIPS (milhões de instruções por segundo);
- Possibilidade de sinal de *clock* até 10 MHz;
- *Clock* interno de até 40 MHz para as instruções (4 x frequência de *clock*);
- Instruções de 16 bits, acesso a dados de 8 bits;
- Implementação de níveis de prioridade para interrupções;
- Hardware de multiplicação 8 bits x 8 bits em apenas um ciclo de máquina;

Quanto aos periféricos disponíveis:

- Capacidade de obter/oferecer 25 mA de corrente;
- 3 pinos para interrupções externas;
- Módulo Timer0: temporizador/contador de 8 e 16 bits com *prescaler* ajustável;
- Módulo Timer1: temporizador/contador de 16 bits;

- Módulo Timer2: temporizador/contador de 8 bits com registrador de período de 8 bits (utilizado também como base de tempo para PWM);
- Módulo Timer3: temporizador/contador de 16 bits;
- Opção de oscilador para sinal de *clock* secundário – Timer1/Timer3;
- Módulo CCP (Capture/Compare/PWM), onde seus pinos podem ser configurados como:
  - Entrada de captura: 16 bits com resolução máxima de 6,25 ns;
  - Comparador: 16 bits com resolução máxima de 100 ns (Tcy);
  - Saída de sinal PWM: resolução de 1 a 10 bits, com frequência máxima de 156 kHz em 8 bits e 39 kHz em 10 bits.
- Interface de comunicação serial SPI 3-*Wire*;
- Interface de comunicação serial I2C;
- Interface de comunicação serial padrão RS232;
- Módulo de comunicação serial CAN;
- Módulo endereçável USART;

Características Analógicas Avançadas:

- Conversores analógico/digital de 8 bits e 10 bits de resolução, com até 8 entradas;
- Módulo comparador de tensão a partir de uma referência;
- Módulo programável para detecção de baixa tensão, suportando interrupção – *Low-Voltage Detection* (LVD);
- *Brown-out Reset* (BOR) programável;

Características Especiais:

- *Power-on Reset* (POR), *Power-up Timer* (PWRT) e *Oscillator Start-up Timer* (OST);
- *Watchdog Timer* (WDT);
- Proteção programável para leitura de código;
- Modo *Sleep*, para economia de energia;
- Suporte a programação do *chip* no mesmo circuito de execução – *In-Circuit Serial Programming* (ICSP) – através de dois pinos, um de *clock* e outro de dados;

Capacidades:

- 32 kB de memória para instruções, sendo suportado 16 k de instruções (2 bytes por instrução);
- 1536 bytes de memória RAM para dados de programa;
- 256 bytes de memória EEPROM de dados;
- 33 pinos para entrada/saída;
- 2 comparadores de tensão;

Seus pinos podem ser observados na Figura 2.2.1.

Uma descrição detalhada de cada pino, bem como de cada periférico, poderá ser vista no *data sheet* do 18F458, disponível no *site* da Microchip, referenciado no final deste documento.

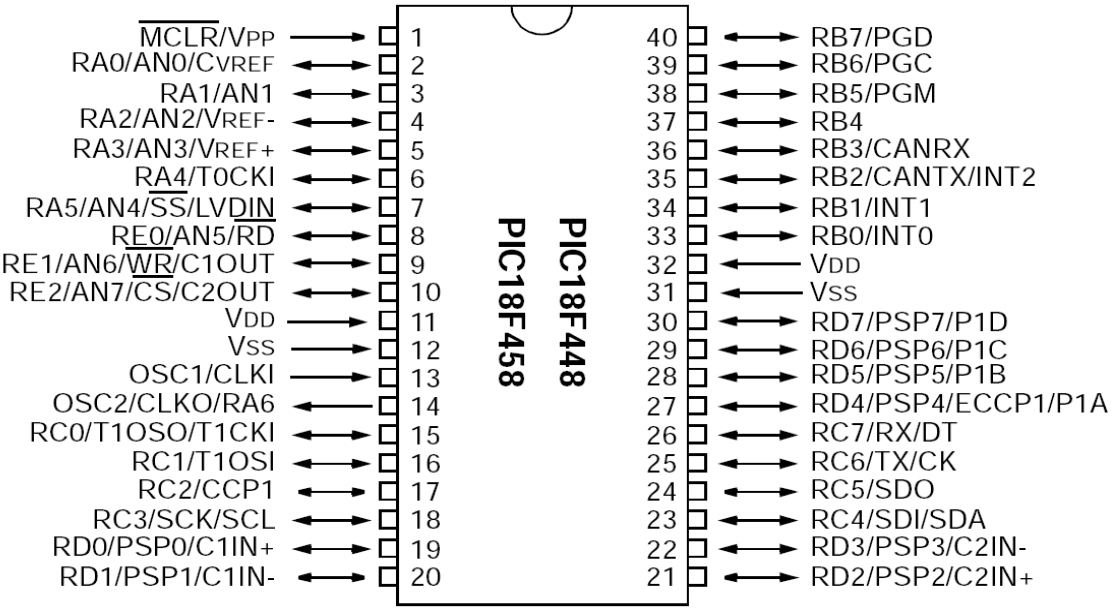


Figura 2.2.1 - Diagrama de pinos do PIC 18F458 PDIP

2.3. Programando o PIC

Para que um processador possa executar instruções, obviamente elas precisam ser escritas e depois colocadas em alguma unidade de armazenamento para que o processador possa lê-las e processá-las. Portanto, nesta seção é descrito como é feito esse desenvolvimento e gravação de instruções no PIC para que possam ser executadas.

No PIC, as instruções são armazenadas em uma memória especial, específica para o seu armazenamento. Esta memória tem a característica de não ser volátil, ou seja, mesmo quando está sem alimentação de energia, as instruções continuam armazenadas e intactas para que possam ser executadas pelo processador. Portanto, esta memória serve para armazenar apenas instruções, e não dados temporários.

Como o processador lê as instruções como uma seqüência de bits, seria muito custoso para o desenvolvedor ter que criar seus programas diretamente em linguagem de máquina, utilizando diretamente os bits. No entanto, conforme Pereira (2003), atualmente essa prática não é utilizada. Hoje tem-se a opção de desenvolver os programas em uma

linguagem muito próxima do hardware, mas de uma maneira mais legível ao desenvolvedor. Para tal, podemos utilizar a linguagem Assembly, que possui uma representação do conjunto de instruções de um processador específico. Com o Assembly, o desenvolvedor estará confeccionando o programa utilizando as próprias instruções do processador, mas sem precisar escrevê-la em forma de bits, deixando a cargo do montador efetuar tal conversão e, no máximo, algumas pequenas modificações no código. Portanto, mesmo assim, não ficaria nada trivial desenvolver programas mais sofisticados, exigindo grande esforço do programador e deixando o software mais propenso a erros de programação.

Portanto, conforme Silva Jr. (2006) o interessante seria desenvolver os programas de uma maneira mais inteligível ao programador de maneira mais eficiente possível. Os microcontroladores PIC, nativamente, são programáveis em Assembly, mas podem ser programados utilizando uma linguagem de alto nível. Será utilizada a linguagem C neste projeto, devido à facilidade para geração dos programas, sendo necessário apenas o conhecimento do funcionamento básico de cada periférico do PIC, visto que o compilador irá se encarregar de traduzir as funções do código em linguagem de máquina.

O código abaixo em assembly efetua uma escrita na EEPROM do PIC 18F458:

```
MOVLW EE_ADDRESS
MOVWF EEADR
MOVLW EE_DATA
MOVWF EEDATA
BCF EECON1, EEPGD
BCF EECON1, CFGS
BSF EECON1, WREN
BCF INTCON, GIE
MOVLW 55h
MOVWF EECON2
MOVLW 0AAh
MOVWF EECON2
```

```
BSF EECON1, WR
BSF INTCON, GIE
```

e em linguagem C:

```
write_eeprom(ee_address, ee_data);
```

Como se pode observar nos dois trechos de código para escrita em EEPROM, o uso de uma linguagem de alto nível facilita e simplifica muito o desenvolvimento de um programa. A utilização da linguagem C para programação do PIC também pode trazer algumas desvantagens em casos específicos. Por exemplo, em sistemas críticos onde se necessita realizar uma rotina de execução de forma precisa obtendo resposta rápida e correta, conforme Pereira (2003) a programação em assembly, desde que o programador saiba o que está fazendo, seria mais adequada para o caso. Portanto, para esses casos específicos, pode-se desenvolver códigos em assembly dentro do programa escrito em linguagem C. Para isto, basta colocar as instruções em assembly entre as diretivas #asm e #endasm. É uma característica muito útil, pois além da programação em uma linguagem de alto nível, é possível desenvolver um conjunto de instruções assembly para o trecho de código que realiza tarefas críticas a fim de otimizar sua execução e obter uma maior precisão. Claro que o programador precisa ter uma certa experiência com o conjunto de instruções do PIC para que seja possível efetuar tal otimização, caso contrário só iria piorar o desempenho, visto que os compiladores atuais de linguagens de alto nível conseguem otimizar consideravelmente o código.

Para o desenvolvimento dos programas, existe um ambiente integrado chamado MpLab, da Microchip. Ele já possui total integração com um compilador C da CCS e permite desenvolver e compilar os programas no mesmo ambiente. Mas, como alternativa, pode ser utilizado um outro ambiente chamado “PCW C Compiler IDE” (Figura 2.3.1), da própria CCS, desenvolvedora do compilador C para PIC. É um ambiente similar ao MpLab

e também com integração ao compilador para PIC da CCS. Será apresentado nesta seção, o "PCW C Compiler IDE".

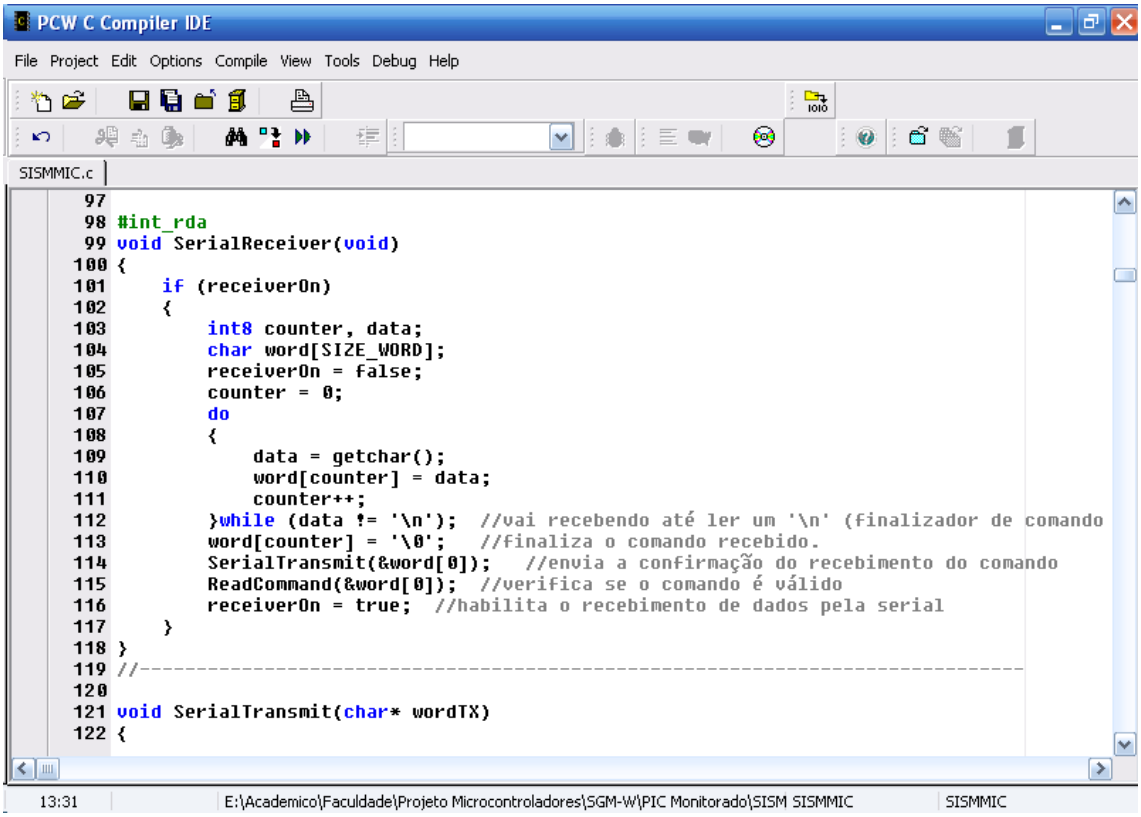


Figura 2.3.1 - Ambiente de desenvolvimento "PCW C Compiler IDE", da CCS

Conforme Silva Jr. (2006), o processo completo de programação do PIC começa com o desenvolvimento do programa em linguagem C. Logo após, o programa final é compilado pelo "CCS C Compiler", que já vem integrado ao ambiente PCW. Até este ponto, tudo é feito no ambiente de desenvolvimento da CCS, o PCW. Após a compilação, o CCS irá criar um arquivo com extensão .hex, um arquivo hexadecimal que contem as instruções do PIC, juntamente com algumas informações adicionais ao gravador.

Com o arquivo .hex gerado, é necessário gravar as instruções na memória de programas do PIC. Para isso, foi utilizado o programa IC-Prog (Figura 2.3.2), que permite gravar diversos modelos de microcontroladores. O software é gratuito, podendo ser

baixado pela internet, através do *site* do desenvolvedor, referenciado no final deste documento.

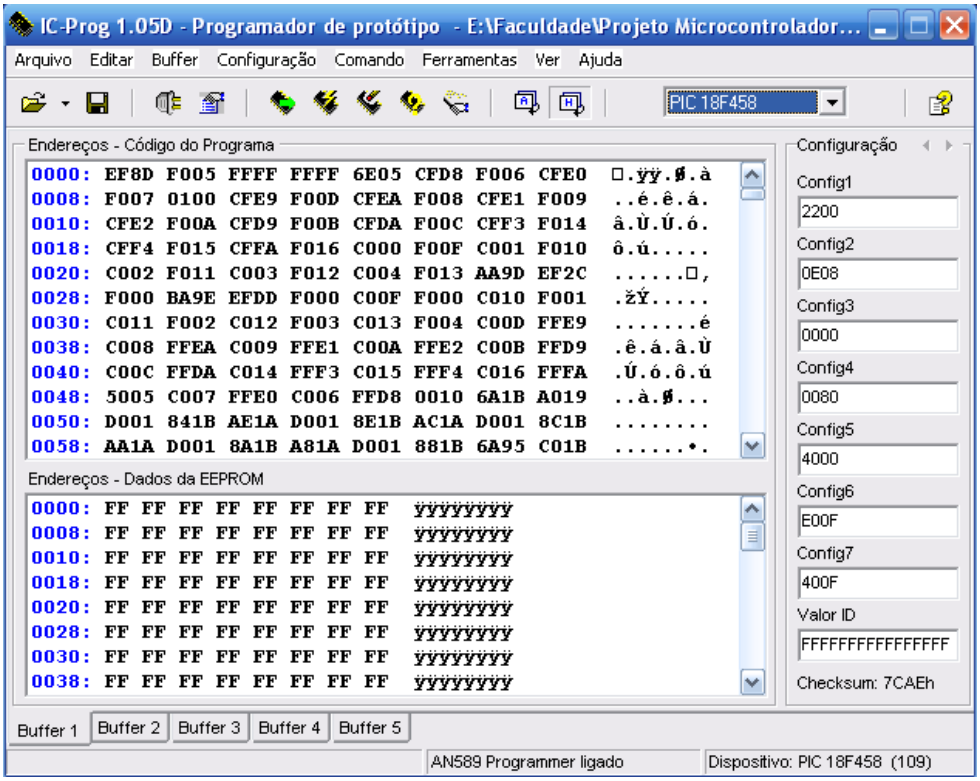


Figura 2.3.2 - IC-Prog, software para gravação de circuitos integrados programáveis seriais

Em pesquisa feita em www.ic-prog.com, *site* do desenvolvedor IC-Prog, constatou-se que existem diversos padrões de gravação que podem ser escolhidos no IC-Prog. Cada padrão representa um modo diferente de envio de dados do programa gravador ao PIC, podendo ser utilizada a porta de comunicação serial ou paralela do microcomputador. Os padrões mais utilizados são o “JDM Programmer”, utilizando interface serial, e o “AN589 Programmer”, utilizando interface paralela. Após ter conectado o computador ao microcontrolador, através do tipo de conexão escolhida, e colocado o pino 1 (MCLR) do PIC em tensão +13V para ativar o modo de gravação, pode-se então enviar as instruções do programa desenvolvido para serem armazenadas no PIC e futuramente serem processadas pelo microcontrolador.

Após isso, o PIC já poderá executar o programa desenvolvido.

3. KIT PICLAB 4C

3.1. Componentes básicos

O Piclab 4C (Figura 3.1.1) é uma placa eletrônica didática desenvolvida pela Vidal Microcontroladores, empresa especializada em treinamento e projetos com microcontroladores, que contém um microcontrolador PIC modelo PIC 18F458 fabricado pela Microchip e alguns outros elementos eletrônicos para a realização de tarefas.

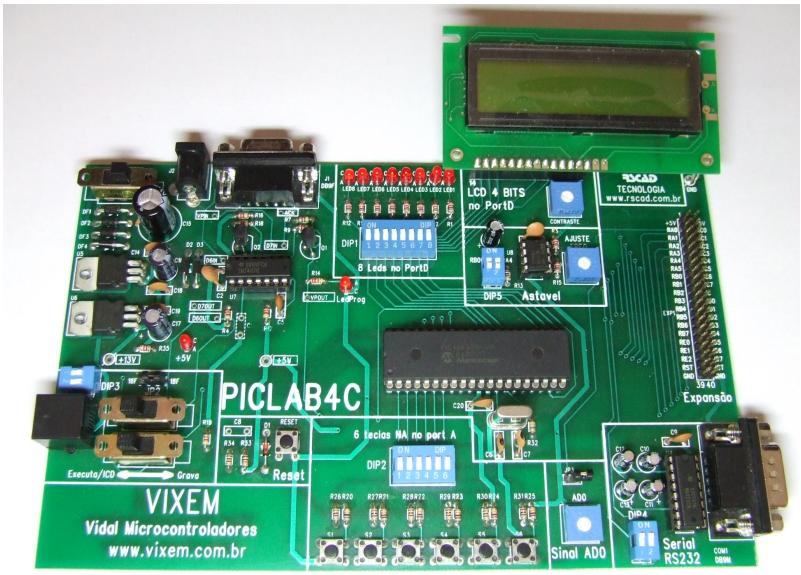


Figura 3.1.1 - Kit Piclab 4C

Além do *chip* PIC, os outros elementos eletrônicos presentes no kit são: um cristal para geração de *clock* de 10 MHz para o PIC, alguns *push-buttons* para interface com o usuário, um LCD e 8 leds para apresentação de resultados, uma interface para comunicação serial RS232 e outra paralela, uma interface ICD para *debugger* de execução de código, um gerador de *clock* para contadores utilizando o CI 555, um gerador de sinal analógico para teste de conversor A/D presente no microcontrolador e fonte de alimentação do kit.

O microcontrolador PIC 18F458, presente no kit, já foi apresentado na seção 2.1.

Os demais periféricos presentes no Piclab 4C, são comentados na próxima seção.

3.2. Outros componentes do kit

O componente principal do kit é o microcontrolador. Mas, ele sozinho não é capaz de receber dados e processá-los, a fim de fornecer resultados. Portanto, no kit ainda existem outros componentes eletrônicos, incluindo alguns deles que são imprescindíveis para compor o circuito mínimo com o 18F458, conforme Silva Jr. (2006), apresentado na Figura 3.2.1.

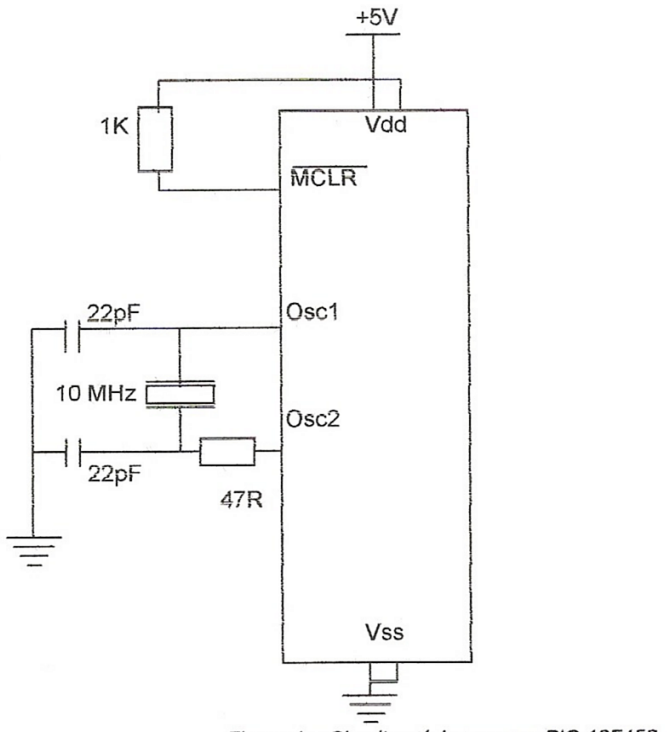


Figura 3.2.1 - Circuito mínimo com o PIC 18F458

Como pode-se ver na figura, o circuito mínimo com o PIC 18F458 é composto de, além do *chip* microcontrolador, um resistor ligando a alimentação ao pino MCLR (1), pinos VDD (11 e 32) alimentados em +5V, pinos VSS (12 e 31) em 0V e pinos Osc1 e Osc2 ligados ao oscilador de *clock* utilizando dois capacitores e um resistor. Ou seja, com

poucos componentes eletrônicos e baixo custo é possível colocar o microcontrolador em funcionamento. Além do circuito mínimo, tem-se os circuitos complementares, que diferenciam uns dos outros em cada aplicação, dependendo do ambiente, tarefas a serem executadas, necessidades de leitura de dados e acesso a outros dispositivos elétricos externos pelo PIC.

No caso do kit em questão, seu objetivo é oferecer uma forma de aprendizagem mais didática e simples possível, ao mesmo tempo que ofereça uma diversidade de uso de recursos presentes no microcontrolador. Portanto, o kit possui, além do 18F458, os componentes abaixo:

- Um LCD na porta D: utilizado para impressão de caracteres em sua tela, possibilita uma visualização dos resultados obtidos através do processamento de dados pelo PIC de uma maneira mais próxima da nossa linguagem habitual;
- 8 leds na porta D para visualizar o estado dos pinos, muito útil para realizar as experiências no kit;
- Um CI 555 no pino RA4 para geração de *clock* externo, usado como base de contagem para o *timer* 0 e no pino RB0 para geração de interrupções sucessivas com período determinado pelo ajuste do CI;
- Um oscilador de cristal de10 MHz para geração de *clock* ao processador;
- 6 teclas do tipo *push-button* na porta A, para alteração do estado de seus pinos, utilizadas para realizar uma interação entre o PIC e o usuário do mesmo;
- Tecla para *reset* do PIC;
- Interface ICD com conector RJ45 para efetuar *debugger* de código;
- Chaveador para execução/ICD e gravação do *chip*;
- Interface serial padrão RS232 com CI MAX232 ligado aos pinos de RX e TX do PIC;

- Interface paralela utilizada para gravação do *chip* através da porta paralela de um microcomputador;
- Uma entrada para alimentação de 13V, que é transformada em outro circuito de 5V para alimentação do PIC quando entra em operação. O circuito de 13V é necessário para alternar o estado do PIC, através do pino 1 (MCLR), de execução de instruções (quando em 5V) para gravação de programa (quando em 13V). São utilizados dois reguladores de tensão e 2 diodos para obter tais tensões.

Com todos esses componentes presentes no kit, juntamente com o microcontrolador PIC, é possível usufruir de quase todas as funcionalidades oferecidas pelo microcontrolador. Ao mesmo tempo que o kit possa ser utilizado como uma plataforma experimental, também temos a possibilidade de utilizá-lo para efetuar gravações de programas no PIC sem precisar retirar o *chip* do circuito para realizar tal tarefa.

4. KIT KPME-10

4.1. Introdução

O KPME-10 (Figura 4.1.1) é um kit, disponibilizado pela empresa 2EI, contendo placas eletrônicas e materiais de apoio para o desenvolvimento de sistemas embarcados com conectividade Ethernet/Internet.

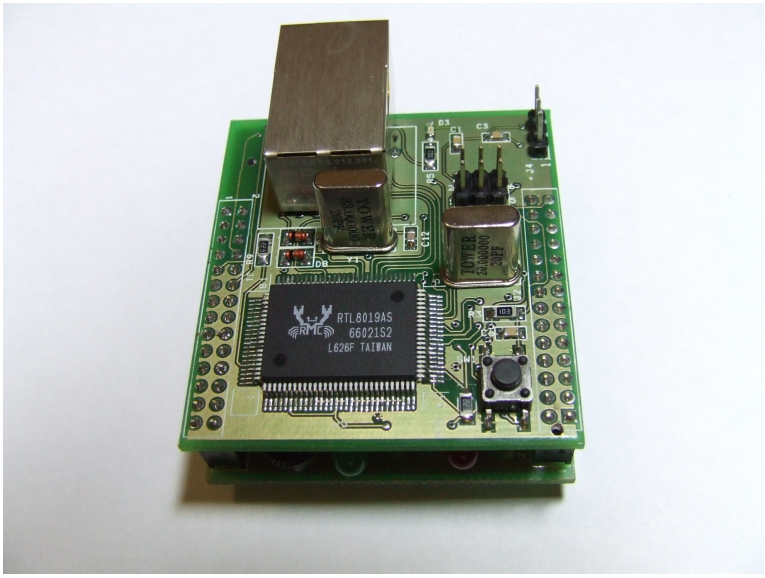


Figura 4.1.1 - Kit KPME-10 para desenvolvimento de sistemas com conectividade Ethernet/Internet

Os ítens presentes no kit incluem:

1 – Placa eletrônica PME-10 contendo um microcontrolador PIC 18F8722, um controlador Ethernet Realtek RTL8019AS e demais componentes para o circuito;

2 – Placa filha FD01, com alimentação de +5VDC, chave *push-button*, led e potenciômetro para testes do kit;

3 – Programador de microcontroladores padrão JDM, para a gravação de programas na memória flash do PIC;

4 – Material de apoio contendo a pilha TCP/IP para microcontroladores PIC, implementada pela Microchip, e exemplos de utilização.

O principal componente do kit é a placa PME-10, que contém o microcontrolador e

o controlador ethernet da Realtek. O controlador RTL8019AS está ligado ao microcontrolador que possui a implementação da pilha TCP/IP para conectividade ethernet e internet e promove uma comunicação com o controlador ethernet para transferência de pacotes TCP/IP.

A pilha TCP/IP para microcontroladores pode ser obtida gratuitamente no *site* da Microchip (AN833). Ela possui a implementação dos principais serviços oferecidos pelo protocolo como conexões UDP e TCP, ICMP (*ping*), com opção de utilização de DHCP para obtenção automática de IP para o microcontrolador. Além destes serviços, está disponível uma implementação de um servidor HTTP, que está comentado na próxima seção.

A placa filha FD01 é encarregada de fornecer a alimentação de 5VDC ao PIC, através de um regulador de tensão. Além disso, possui componentes para interação com o microcontrolador para a execução de testes.

Para gravar o programa a ser rodado no microcontrolador, o kit inclui um programador padrão JDM (JDM Programmer). Este padrão de programadores PIC foi desenvolvido por Jens Dyekjær Madsen e utiliza a porta serial do computador para o processo de gravação. O *layout* da placa padrão JDM pode ser obtida gratuitamente através do *site* JDM Programmer.

4.2. O servidor HTTP da Microchip

Como foi mencionado, o kit KPME-10 possui uma implementação de servidor HTTP. Esta implementação se encontra no arquivo “HTTP.C”, juntamente com duas funções a serem implementadas pela aplicação do usuário, HTTPGetVar e HTTPExecCmd, que serão descritas adiante. A própria Microchip disponibiliza um exemplo de programa principal implementado fazendo uso do servidor HTTP. Esta implementação está no arquivo “Websrvr.c”.

Segundo 2EI (2007), o servidor HTTP da Microchip implementa as principais

funcionalidades dos documentos RFC, referência importante para HTTP, suficientes para o desenvolvimento de páginas *Web* em sistemas embarcados, sendo possível a adição de novas funcionalidades ao aplicativo. O servidor HTTP suporta múltiplas conexões HTTP, o que pode ser limitado através do arquivo de implementação do servidor.

Para o armazenamento das páginas *Web* no PIC, existe uma implementação de sistema de arquivos fornecido pela Microchip, o MPFS (Microchip *File System*). Deste modo, as páginas a serem armazenadas no servidor HTTP, após serem desenvolvidas, devem ser convertidas para o formato MPFS. Essa conversão para o formato de sistema de arquivos da Microchip é feitas pelo executável "MPFS.exe", através da chamada:

```
$ mpfs WebPages MPFSImg.c /c
```

onde WebPages é a pasta que possui os arquivos a serem convertidos. Após a conversão, será gerado um arquivo no formato da linguagem "C" que deverá ser incorporado ao projeto do sistema e compilado junto dos outros arquivos fonte. Assim que requisitado uma conexão HTTP com o servidor, o mesmo irá buscar o arquivo "index.htm" e irá retorná-lo ao cliente que o requisitou. Se necessário, esta página inicial pode ser alterada através da definição `HTTP_DEFAULT_FILE_STRING` no arquivo de implementação "http.c".

Existe uma restrição quanto aos conteúdos das páginas *Web*. Não deverá existir nenhum dos caracteres ', ", <, >, #, %, {, }, [,], |, \, ~, ^ ou :. Isso se aplica a apenas o texto a ser apresentado na página *Web* e não inclui as *tags* html. Se uma das páginas tiver algum desses caracteres, ela poderá se tornar inacessível.

O servidor HTTP da Microchip, por padrão, suporta arquivos nos formatos ".txt", ".htm", ".gif", ".cgi", ".jpg", ".cla" e ".wav", mas outros poderão ser acrescentados na enumeração "httpContents" no arquivo "http.c".

Para o desenvolvimento de páginas *Web* com conteúdo dinâmico, o servidor HTTP suporta páginas CGI (*Common Gateway Interface*), que deverão estar armazenadas no formato ".cgi". Para a alteração das páginas de modo dinâmico, são utilizadas variáveis na página CGI, precedidas pelo caracter "%" e seguidas pelo seu identificador de dois dígitos

(“%xx”, onde xx é o identificador que deve estar entre 00 e 99). Ao encontrar um caracter “%”, o servidor HTTP retira o “%” e chama a função `HTTPGetVar(byte var, Word ref, Byte *val)`, uma das duas funções mencionadas no início deste tópico. Essa função deve ser implementada no programa principal e possui os seguintes parâmetros:

var: entrada com o identificador da variável requisitada pela página CGI;

ref: entrada usada para controle durante a passagem do valor da variável;

val: saída da função que retorna 1 byte do valor da variável requisitada.

A função HTTPGetVar pode ser chamada várias vezes para uma mesma variável requisitada, pois só é retornado 1 byte do valor da variável de cada vez. Portanto, se o valor de uma variável tiver 2 ou mais bytes, cada byte será transferido a cada chamada da função até que toda a *string* seja obtida. Para o controle de qual byte da *string* está sendo transferido, a entrada ref é utilizada. A variável ref pode ter os valores HTTP_START_OF_VAR e valores inteiros que indicam a posição da *string* a ser transferida. Na primeira chamada à função, o valor de ref é HTTP_START_OF_VAR e a cada chamada esse valor é atualizado com a posição corrente da *string* a ser transferida até que o conteúdo da variável finalmente é totalmente transferido.

A função HTTPGetVar retorna um valor para o servidor, indicando quando deverá parar de efetuar chamadas à função. Enquanto toda a *string* ainda não foi transferida, o valor retornado pela função é a posição corrente. Quando a *string* é totalmente lida, o valor de retorno da função é HTTP_END_OF_VAR, sinalizando ao servidor HTTP a finalização da *string*. Veja um exemplo de implementação da função HTTPGetVar, que contém os casos de um e vários bytes a serem tranferidos na Figura 4.2.1.

A outra função mencionada no início deste tópico refere-se como alterar valores em um navegador *Web* no cliente e transferí-los para o microcontrolador. Esta função tem a sintaxe `HTTPExecCmd(BYTE *argv, BYTE argc)`, onde:

argv: argumentos passados pelo navegador *Web* em forma de *strings*;

argc: número de argumentos passados como parâmetro.

Para entender melhor seu funcionamento, pode-se observar um exemplo de página *Web* com requisição de alteração de variáveis do microcontrolador na Figura 4.2.2 e Figura 4.2.3.

```
WORD HTTPGetVar(BYTE var, WORD ref, BYTE *val)
{
    switch(var)
    {
             case 4: //retorna o estado do pino RB5
                 // retorna '1' se RB5 estiver em nível lógico 1
            if ( PORTBbits.RB5 )
                *val = '1';
            else
                *val = 0;
            // Reporta ao HTTP que este foi o último byte do
            // valor da variável
            return HTTP_END_OF_VAR;
        break;
        case 5: //retorna o número serial
            //primeira chamada
            if (ref == HTTP_START_OF_VAR)
            {
                // inicializa índice para o vetor a ser
                // transferido
                ref = (BYTE)0;
            }
            //Acessa o byte no índice corrente e salva no
            //buffer
            *val = SerailNumberStr[(BYTE)ref];
            //Se é final da string
            if (*val=='\0')
            {
                //sinaliza ao servidor que o último byte foi
                //transferido
                return HTTP_END_OF_VAR;
            }
            //caso contrário, incrementa o índice do vetor e
            //retorna para o servidor HTTP
            (BYTE)ref++;
            // desde que não é o fim da string, retorna ref
            return ref;
        break;
    }
}
```

Figura 4.2.1 - Implementação da função HTTPGetVar

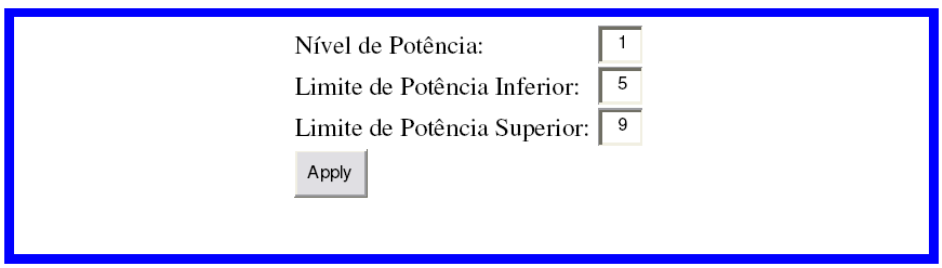


Figura 4.2.2 - Conteúdo da página Web para alteração de variáveis

```
<html>

<body><center>
<FORM METHOD=Get action=Power.cgi>
<table>
<tr><td>Nível de Potência:</td>
<td><input type=text size=2 maxlength=1 name=P></td></tr>
 <tr><td>Limite de Potência Inferior:</td>
<td><input type=text size=2 maxlength=1 name=L ></td></tr>
<tr><td>Limite de Potência Superior:</td>
<td><input type=text size=2 maxlength=1 name=H ></td></tr>
<tr><td><input type=submit name=B value=Apply></td></tr>
</table>
</body>
</html>
```

Figura 4.2.3 - Código da página *Web* para alteração de variáveis

Na página *Web* do exemplo, o usuário entra com os valores de potência nas caixas de texto e, ao clicar no botão, o navegador envia uma requisição HTTP para o servidor com a *string* "Power.cgi?P=5&L=1&H=9". Ao ser atendida, o servidor HTTP chama a função HTTPExecCmd com os seguintes parâmetros: `argv[0]="Power.cgi"`, `argv[1]="P"`, `argv[2]="5"`, `argv[3]="L"`, `argv[4]="1"`, `argv[5]="H"`, `argv[6]="9"` `argc=7`. A implementação da função HTTPExecCmd deverá estar na aplicação principal, semelhante ao código do exemplo que pode ser visto na Figura 4.2.4.

```
void HTTPExecCmd(BYTE *argv, BYTE argc)
{
        BYTE i;
        // Varre todos os parâmetros
        for (i=1; i < argc;i++)
        {
                // Identifica parâmetros
                if ( argv[i][0] == 'P')          // É nível de potência?
                {
                        PowerLevel = atoi(argv[++i]);
                }
                else if ( argv[i][0] == 'L' )    // É limite inferior de potência?
                        LowPowerSetting = atoi (argv[++i]);
                else if (argv[i][0] == 'H')      // É limite superior de potência?
                        HighPowerSeting = atoi (argv[++i]);
        }
        // se outra página é para ser mostrada como resultado deste comando
        // copie o nome em maiúsculo para argv[0]
        // strcpy(argv[0], "RESULTS.CGI");
}
```

Figura 4.2.4 - Exemplo de implementação da função HTTPExecCmd

5. COMUNICAÇÃO DE DADOS

5.1. Comunicação serial

Em algumas situações, é necessário promover uma comunicação do microcontrolador com algum outro dispositivo externo. Existem diversas formas de promover tal comunicação, dependendo de diversos fatores como a localização do dispositivo externo, distância entre os equipamentos, ambiente onde estão inseridos, dentre outros.

Segundo Pereira (2003), existem diversas diferenças entre os sistemas de comunicação como: velocidade, imunidade a ruídos, custo, etc. A escolha de um irá depender de como e para que será utilizado. Dentro dos sistemas de comunicação, o autor divide-os em duas grandes categorias: comunicação serial e comunicação paralela.

Na comunicação serial, as mensagens são divididas em pequenas partes e transmitidas em forma de bits, onde são recebidas pelo receptor um após o outro, de forma serial. São exemplos de comunicação serial as interfaces RS-232, USB, SPI, I2C, dentre outros.

Já na comunicação paralela, os bits de cada fatia da mensagem são transmitidos ao mesmo tempo e paralelamente. Dentre as interfaces de comunicação paralela, podemos exemplificar os barramentos PCI e ISA, comunicação paralela de impressoras de microcomputadores, interface IDE de alguns discos rígidos, dentre outros.

A comunicação serial padrão RS-232 é do tipo serial assíncrona onde, como explica Pereira (2003), existem marcadores de início e fim de transmissão. A transmissão começa por um sinal de início (*start*), seguido pelos bits de dados (geralmente 8 bits) e finalizada pelo bit de parada (*stop*). Na Figura 5.1.1, podemos observar a transmissão de um sinal serial assíncrono do caractere 'A'.

No caso do microcontrolador PIC, especificamente no PIC 18F458, tem-se a

comunicação serial assíncrona padrão RS-232 como uma das formas de troca de dados com outros dispositivos externos.

Conforme Silva Jr. (2006), para transmitir um byte, o transmissor interno do PIC sinaliza um envio de novos dados aplicando um nível de tensão em 0V na linha de dados (*start*) no pino TX, inicialmente em nível de tensão alto (5V), durante um tempo dT. Logo após, os bits de dados vão sendo transmitidos um a um em intervalos de tempo dT, começando pelo bit menos significativo. Após enviados os dados, o pino TX é colocado novamente em nível alto de tensão (5V), sinalizando o fim da transmissão.

Na recepção dos dados, assim que o receptor interno do PIC detecta a descida do sinal no pino RX (tensão de 5V para 0V), inicia-se o processo de varredura automática neste pino, copiando para um registrador interno o estado do pino (0 ou 1), até que se obtenham todos os 8 bits de dados e o *Stop* Bit (nível 1). A comunicação ocorrendo com sucesso, o byte recebido é gravado no registro RCREG do microcontrolador e sinaliza um pedido de interrupção.

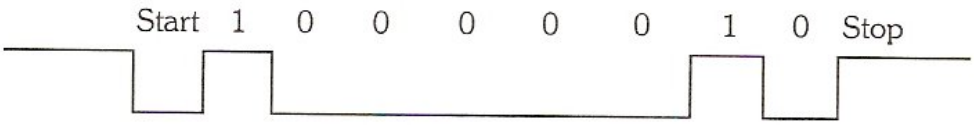


Figura 5.1.1: Sinal de dados na comunicação serial assíncrona

Para a utilização de comunicação serial RS-232 no PIC, existem funções em linguagem C pré-desenvolvidas para tratamento das interrupções para a comunicação.

5.2. Redes de computadores

Conforme Tanembaum (1996), redes de computadores estão relacionadas a um conjunto de computadores autônomos interconectados. Dois computadores estão interconectados quando podem trocar informações e autônomos quando um não pode iniciar, encerrar ou controlar o outro.

Tanembaum (1996) divide as redes de computadores em quatro grupos: LAN,

MAN, WAN e a inter-rede. O autor descreve as redes locais, chamadas de LAN's, como redes privadas localizadas em um prédio ou em um campus universitário com alguns quilômetros de extensão. As redes metropolitanas, ou MAN, são redes privadas ou públicas que abrangem um grupo de escritórios vizinhos ou uma cidade inteira, mas com tecnologias utilizadas semelhantes as das LAN's. Já a rede geograficamente distribuída, ou WAN, abrange uma grande área como um país ou até um continente. Por fim, a internet seria uma inter-rede, que interliga o mundo inteiro através da conexão entre diferentes tipos de redes dentre as três anteriores.

Conforme Kurose e Ross (2003), a internet é uma rede mundial de equipamentos interligados, desde PCs tradicionais até celulares, *pagers*, torradeiras, dentre outros dispositivos. Cada vez mais, vemos inúmeros equipamentos incomuns conectados à internet. Existem diversos serviços disponíveis na internet para serem utilizados pelos dispositivos conectados, mas, como afirmam os autores, o WWW, que é mais detalhado na próxima seção, é o mais utilizado.

5.3. Redes de computadores

5.3.1. Introdução

Kurose e Ross (2003) lembram que até a década de 90, a internet era utilizada basicamente por acadêmicos e universitários. Ela era usada, principalmente, como ferramenta de troca de arquivos, notícias e mensagens eletrônicas. Após a década de 90 a WWW entrou em cena, chamando a atenção dos usuários da rede e alterando a forma como as pessoas interagem.

A *World Wide Web*, conforme Tanembaum (1996, pág.776, tradução da editora), pode ser definida como "[...] a estrutura arquitetônica que permite o acesso a documentos vinculados espalhados por milhares de máquinas na Internet". A *Web* oferece uma interface gráfica colorida e de fácil navegação. Seu conteúdo não se limita a textos e

hipertextos, ainda oferece ícones, desenhos de linhas, mapas, fotografias, etc. Conforme o autor, também existem os conteúdos hipermídia, quando tem-se páginas de hipertexto misturadas a sons e vídeos. Os conteúdos da *Web* podem ser visualizados através de um *browser*, que verifica os arquivos e exibe a página na tela.

Segundo Kurose e Ross (2003), um *browser* é um aplicativo que exibe as páginas *Web* solicitadas e que também implementa o lado cliente do protocolo HTTP, o protocolo padrão da *Web* que será comentado na próxima seção. Já um servidor *Web* é aquele que armazena as páginas *Web* e que implementa o lado servidor do protocolo HTTP.

5.3.2. HTTP: O protocolo padrão da Web

O protocolo padrão de transferência na *Web* é o HTTP (*HyperText Transfer Protocol*). Conforme Tanembaum (1996) e Kurose e Ross (2003), o protocolo HTTP consiste basicamente em requisições do *browser* (navegador *Web*) no formato ASCII e respostas do servidor *Web*, como pode ser exemplificado na Figura 5.3.2.1.

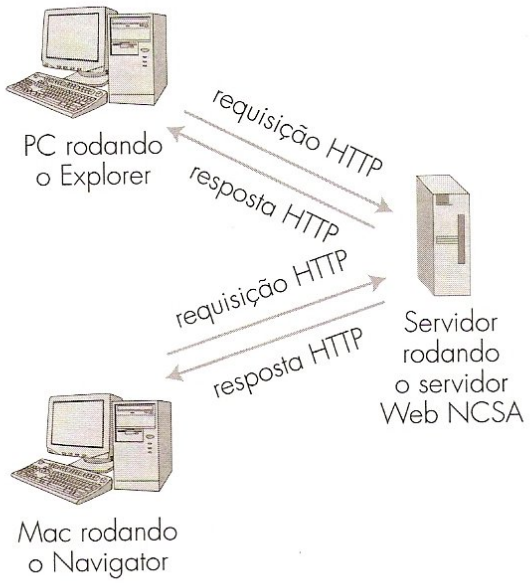


Figura 5.3.2.1: Requisição-resposta no protocolo HTTP

De acordo com Kurose e Ross (2003) uma solicitação do *browser* consiste em uma

mensagem constituída por várias linhas separadas pelos caracteres “\r” e “\n” (equivalente ao “*enter*” no Windows) e finalizada com um “\r” e um “\n” adicionais. A primeira linha da mensagem de requisição HTTP é chamada **linha de requisição** e as linhas seguintes são as **linhas de cabeçalho**, que contêm informações adicionais. Para exemplificar uma destas mensagens, na solicitação do conteúdo de uma página utiliza-se o comando GET, seguido do endereço requisitado e a versão do HTTP na linha de requisição e outras informações nas linhas de cabeçalho como navegador utilizado, idioma, domínio do *host*, dentre outras. Veja o exemplo dado por Kurose e Ross (2003):

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
```

O servidor, como resposta ao exemplo de requisição, envia a mensagem HTTP:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

(conteúdo da página)
```

Na mensagem de resposta HTTP acima enviada pelo servidor *Web*, tem-se a **linha de status** seguida por seis **linhas de cabeçalho** e o **corpo da entidade**. A linha de status possui a versão do protocolo HTTP, uma codificação para o estado da conexão e sua mensagem correspondente, que no caso do exemplo ocorreu tudo OK e o conteúdo será enviado ao cliente. Nas linhas de cabeçalho, são enviadas algumas informações do servidor

como versão do aplicativo servidor, data da última modificação, tamanho do objeto que está sendo enviado, dentre outras. E, por fim, no corpo da entidade está a informação que realmente se deseja, o conteúdo da página *Web* requisitado.

5.3.3. Páginas da *Web* com a HTML

Conforme Tanembaum (1996), a HTML (*HyperText Markup Language*) é uma linguagem para o desenvolvimento das páginas *Web*. Com a HTML é possível criar páginas da *Web* com textos, gráficos e ponteiros para outras páginas. Os ponteiros são como atalhos para o acesso a outras páginas *Web*.

As páginas *Web* padronizadas possuem um cabeçalho e um corpo entre as *tags* (comandos de formatação) <HTML> e </HTML>. O cabeçalho é delimitado pelas *tags* <HEAD> e </HEAD> e o corpo pelas *tags* <BODY> e </BODY>, ou seja, tudo o que está entre as *tags* <HEAD> e </HEAD> é considerado como cabeçalho da página e o mesmo ocorre para o corpo.

No cabeçalho podem ser colocadas diversas informações como, por exemplo, o título da página que é delimitado pelas *tags* <TITLE> e </TITLE>. Tudo o que está no cabeçalho não aparece explicitamente na página visualizada, mas é utilizado pelo navegador para identificar a maneira como o corpo deverá ser apresentado.

No corpo está o conteúdo da página *Web* propriamente dita, onde são colocados o texto, as imagens, atalhos para outros endereços da *Web*, dentre outros elementos. Por exemplo, se deseja-se colocar uma imagem na tela, no corpo da página terá, da maneira mais simples de apresentação, algo semelhante com:

<IMG SRC="imagem.jpg">

Como pode-se observar, os conteúdos das páginas *Web* são identificados através das *tags*. Cada *tag* terá dentro de si as diretivas, que são os comandos relacionados a um determinado conteúdo. A maioria das *tags* da HTML são utilizadas de forma a indicar o início de uma área da página, com o formato <COMANDO>, e o final da mesma, com o

formato </COMANDO>. Em alguns casos, só é utilizado o primeiro formato descrito, como no exemplo da inserção de uma imagem. Dentro das *tags*, poderão existir outros parâmetros juntos aos comandos, o que também pode ser exemplificado no caso da imagem.

São diversas *tags* que podem ser utilizadas na página e as mesmas podem ser combinadas. Para colocar o texto "Veja esta imagem e/ou clique nela para ampliar:", para ser apresentado em negrito e logo após ser exibida uma imagem que se for clicada levará a uma outra página que contém uma ampliação da mesma, o código HTML poderia ser escrito da forma:

```
<B>Veja esta imagem e/ou clique nela para ampliar:</B>
<A HREF="http://www.paginaexemplo.com.br/imgAmpliada.html">
<IMG SRC="imgPeq.jpg"></A>
```

Vale lembrar que as páginas *Web* são desenvolvidas utilizando também diversas outras linguagens como Java, PHP, ASP, dentre outras, que podem ser combinadas. A escolha de um método ou outro vai depender das necessidades, podendo ser implementada com a HTML pura ou combinada com uma ou mais das outras linguagens.

6. DESENVOLVIMENTO

6.1. Tipo da pesquisa

Conforme Jung (2004) e a partir de observações das características do trabalho, esta pesquisa científica é de natureza tecnológica, com objetivos de caráter exploratório, utilizando procedimentos experimentais e com um trabalho de pesquisa em campo.

Quanto à natureza, é tecnológica por objetivar a aplicação dos conhecimentos básicos obtidos para gerar novas tecnologias e conhecimentos.

Em relação aos objetivos, são de caráter exploratório por visar a descoberta de novos fenômenos para originar um novo produto ou inovação a partir de combinações de conhecimentos já assimilados.

Esta pesquisa científica utiliza procedimentos experimentais por objetivar a descoberta de novos materiais e métodos para a geração de um novo produto tecnológico.

Sobre o local da realização do trabalho, o mesmo é de caráter de uma pesquisa em campo, por ser realizada no ambiente real onde a nova tecnologia será aplicada. Sendo assim, a partir de comportamentos apresentados pelo sistema, o mesmo pode ser aperfeiçoado a medida que dificuldades vêm surgindo.

6.2. Sistema SGM-W

6.2.1. Introdução

Agora será apresentado o sistema desenvolvido durante a execução do trabalho.

O sistema desenvolvido tem por objetivo promover um monitoramento e controle dos periféricos de um microcontrolador utilizando uma interface através de uma página *Web*. Portanto, além do microcontrolador a ser gerenciado, existe um segundo microcontrolador (presente no placa PME-10) atuando como um servidor *Web* para

atender requisições HTTP e promover a interface com o sistema de modo a haver uma comunicação com o microcontrolador gerenciado. O modo de interligação entre os dispositivos envolvidos no sistema SGM-W pode ser visto na Figura 6.2.1.1.

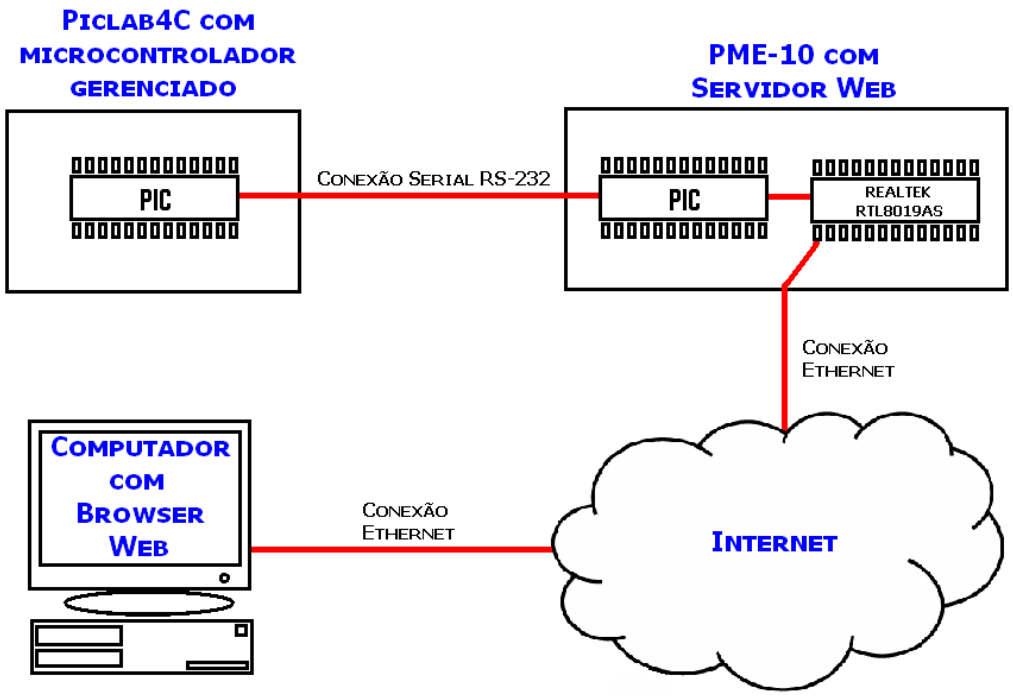


Figura 6.2.1.1 - Conexões entre os dispositivos envolvidos no SGM-W

Para representar o microcontrolador gerenciado foi utilizada a placa eletrônica didática Piclab4c, mencionada no capítulo 3, que possui o PIC 18F458 e outros componentes para teste. Para a implementação do servidor *Web* foi utilizada a placa eletrônica PME-10 presente no kit KPME-10, mencionada no capítulo 4, que possui um microcontrolador PIC 18F8722, um controlador ethernet RTL8019AS e outros componentes auxiliares. As placas Piclab4c e PME-10 estão conectadas através de suas interfaces seriais padrão RS232, meio onde serão feitas as comunicações necessárias para requisições de alteração dos estados de periféricos e informações de eventos ocorridos.

No microcontrolador PIC 18F8722 da placa PME-10 está a implementação da pilha TCP/IP e servidor *Web* da Microchip e as funções desenvolvidas para o funcionamento do sistema. Neste há uma página *Web* armazenada que faz a interface com o usuário e exibe os estados dos periféricos do PIC gerenciado. A partir de requisições do

usuário através de um navegador, o servidor *Web* envia comandos pela porta serial para o microcontrolador gerenciado, que então realizará as tarefas solicitadas.

No microcontrolador gerenciado PIC 18F458 da placa Piclab4c está um módulo do sistema que fica encarregado de manter o servidor *Web* sempre informado dos eventos ocorridos e receber comandos que possam ser lidos, interpretados e convertidos em ações específicas a serem executados pelo próprio PIC.

Para exemplificar o funcionamento do processo de comunicação entre as duas placas eletrônicas, pode-se enviar um comando do servidor *Web* para o PIC solicitando que o pino RA0 seja colocado em nível alto. Todos esses comandos são formados através da página *Web*. Sendo possível realizar a ação, o PIC imediatamente irá atender o pedido e informará ao servidor *Web* tal evento ocorrido através do envio de um outro comando de informação. Sendo assim, o servidor *Web* irá receber o comando, lê-lo e interpretá-lo como do tipo informativo de evento ocorrido no PIC. Desta maneira, na página Web será apresentado o pino RA0 em nível alto. Portanto, na comunicação entre os dois dispositivos existe um protocolo de comandos.

Vale salientar que "PIC gerenciado" ou "microcontrolador gerenciado" está relacionado ao microcontrolador PIC 18F458 presente na placa Piclab4c e "servidor *Web*" está relacionado ao microcontrolador PIC 18F8722 presente na placa PME-10.

Antes de mostrar os dois programas desenvolvidos para o servidor *Web* e o PIC gerenciado, será mostrado o protocolo de comandos elaborado que estabelece uma linguagem pela qual os dois dispositivos "conversam".

6.2.2. Protocolo de comandos

Como foi apresentado, o sistema desenvolvido consiste em um conjunto formado pelo microcontrolador gerenciado e um servidor *Web* embarcado, onde ambos possuem programas que "conversam" entre si de modo a desempenhar funções de monitoramento e

controle do microcontrolador. Mas, para que esses dispositivos troquem informações, é necessário estabelecer uma linguagem a ser utilizada por ambos. Portanto, para tal, foi desenvolvido um protocolo de comandos que será mostrado neste tópico.

O protocolo de comandos é dividido em dois conjuntos: um que contém os comandos a serem reconhecidos pelo microcontrolador gerenciado e outro que possui os comandos a serem reconhecidos pelo servidor *Web*. Os comandos recebidos pelo microcontrolador gerenciado faz solicitações de alterações nos estados dos periféricos do PIC e os recebidos pelo servidor *Web* informam eventos ocorridos nos periféricos do microcontrolador gerenciado. Os dois conjuntos de comandos são explicados nos próximos sub-tópicos.

6.2.2.1. Comandos recebidos no PIC gerenciado

Basicamente, o conjunto de comandos do microcontrolador gerenciado especifica ações a serem executadas pelo mesmo. Esses comandos são enviados pelo servidor *Web*. Desta forma, o servidor *Web* estará enviando instruções para o PIC interpretar e, caso seja identificado dentro do seu conjunto de comandos, desempenhar a ação solicitada. Os comandos são de 4 tipos:

CMD:W <periférico> <opção> <valor>[\r][\n]: Altera o valor de uma porta ou memória do PIC ou ajusta o período do sinal de PWM.

CMD:R <periférico>[\r][\n]: Efetua a leitura do valor de uma porta ou memória ou verifica o estado de um periférico do PIC.

CMD:A <periférico> <estado> <opção>[\r][\n]: Ativa ou desativa um serviço de um periférico do PIC.

CMD:Q <periférico>[\r][\n]: Verifica o estado de ativação de um serviço de um periférico do PIC.

Abaixo está o conjunto de comandos a serem recebidos pelo microcontrolador

gerenciado e suas descrições de forma mais detalhada, considerando que os caracteres minúsculos deverão ser substituídos pela opção desejada e caracteres minúsculos precedidos de “\” são caracteres especiais:

Altera o estado de uma porta do PIC.

“p”: porta do PIC, variando de A até E.

“xxx”: valor entre 000 e o maior valor decimal da porta.

Altera o estado de um pino do PIC.

“p”: porta do PIC, variando de A até E.

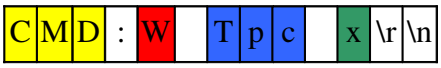
“c”: canal da porta “p”, variando de 0 até o último canal da porta.

“x”: valor do pino, podendo ser 0 (nível baixo) ou 1 (nível alto).

Altera o tris (sentido do fluxo de sinal) de uma porta do PIC.

“p”: porta do PIC, variando de A até E.

“xxx”: valor de tris entre 000 e o maior valor decimal da porta.



Altera o tris (sentido do fluxo de sinal) de um pino do PIC.

“p”: porta do PIC, variando de A até E.

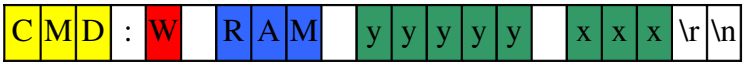
“c”: canal da porta “p”, variando de 0 até o último canal da porta.

“x”: valor do tris, podendo ser 0 (saída) ou 1 (entrada).

Grava um caracter em uma posição da EEPROM.

“yyy”: posição da EEPROM onde será gravado o dado.

“xxx”: valor em decimal da tabela ascii a ser gravado na EEPROM.



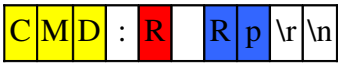
Grava um caracter em uma posição da memória RAM.

“yyyyy”: posição da RAM onde será gravado o dado.

“xxx”: valor em decimal da tabela ascii a ser gravado na RAM.

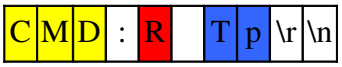
Ativa o gerador de sinal PWM caso esteja desativado e ajusta o período do sinal.

“xxx”: período do sinal PWM.



Efetua a leitura do estado de uma porta do PIC.

“p”: porta do PIC, variando de A até E.

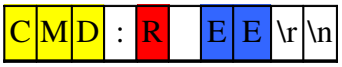


Efetua a leitura do tris de uma porta do PIC.

“p”: porta do PIC, variando de A até E.

Efetua a leitura de uma posição da memória EEPROM.

“xxx”: endereço da EEPROM do PIC, variando de 000 até o último endereço.



Efetua a leitura de todas as posições da memória EEPROM.

C	M	D	:	R		R	A	M		x	x	x	x	x	\r	\n

Efetua a leitura de uma posição da memória RAM.

“xxxxx”: endereço da RAM do PIC, variando de 0000 até o último endereço.

C	M	D	:	R		R	A	M	\r	\n

Efetua a leitura de todas as posições da memória RAM.

C	M	D	:	R		A	D	C	x	\r	\n

Efetua a leitura do resultado da conversão de um dos conversores analógico/digital.

“x”: número do conversor, variando de 0 ao último conversor.

C	M	D	:	R		T	M	x	\r	\n

Efetua a leitura do valor inicial de um dos contadores *timer*.

“x”: número do *timer*, variando de 0 ao último contador.

C	M	D	:	R		P	W	M	\r	\n

Efetua a leitura do período do gerador de sinal PWM.

C	M	D	:	A		s	s	s		e	\r	\n

Ativa ou desativa um serviço de um periférico do PIC.

“sss”: periférico do PIC, podendo assumir os valores ADCx (conversor AD), EXTx (interrupção externa), PWM (gerador de sinal PWM), CAN (comunicação CAN), 232 (comunicação RS232), I2C (comunicação I2C) ou SPI (comunicação SPI).

“e”: estado de ativação, podendo ser 0 (desativar) ou 1 (ativar).

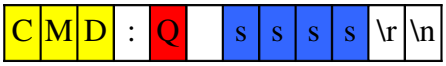
C	M	D	:	A		T	M	x		e		x	x	x	\r	\n

Ativa ou desativa um contador *timer*.

“x”: número do contador, variando de 0 ao último contador.

“e”: estado de ativação, podendo ser 0 (desativar) ou 1 (ativar).

“xxx”: valor inicial do contador *timer*.

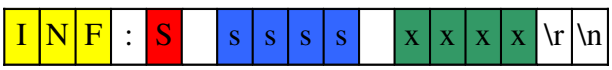


Efetua a leitura do estado de ativação (ativado ou desativado) de um periférico do PIC.

“ssss”: periférico do PIC, podendo assumir os valores ADCx (conversor AD), EXTx (interrupção externa), PWM (gerador de sinal PWM), CAN (comunicação CAN), 232 (comunicação RS232), I2C (comunicação I2C), TMx (contador *timer*) ou SPI (comunicação SPI).

6.2.2.2. Comandos recebidos no servidor *Web*

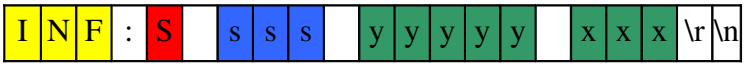
Como foi dito anteriormente, o servidor *Web* envia comandos para o microcontrolador PIC para que o mesmo execute tarefas ou efetue leituras de seus estados. Nos dois casos, é necessário uma resposta do microcontrolador ao servidor sobre a execução desses comandos. Para tal, o software do PIC fica encarregado de, periodicamente e/ou ao recebimento de um comando, verificar alterações nos estados de seus periféricos e, assim, enviar comandos ao servidor *Web* que informem tais eventos ocorridos. Desta forma o servidor *Web* irá receber esses comandos de resposta, interpretá-los e traduzí-los em informações na página *Web* sobre os estados do PIC. Esses comandos de resposta enviados pelo microcontrolador ao servidor *Web* são os apresentados neste tópico.



Informa o valor de um periférico do PIC, podendo ser um valor da porta, memória, variáveis de controle, resultados de conversões, dentre outros.

“ssss”: periférico do PIC, podendo assumir os valores Rpc (valor de um pino do PIC), Rp (valor de um canal do PIC), Tp (valor do tris de uma porta), ADCx (conversor AD), PWM (período do sinal PWM) ou TMx (contador timer).

“xxxx”: valor de uma variável do periférico. Varia de 0 até o maior valor do mesmo.

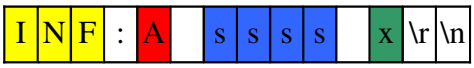


Informa o valor armazenado em uma das memórias do PIC, sendo elas a EEPROM e a RAM.

“sss”: memória do PIC, podendo assumir os valores EE (EEPROM) ou RAM.

“yyyyy”: endereço da memória, variando de 00000 ao maior endereço da RAM ou de 000 ao maior endereço da EEPROM.

“xxx”: valor em decimal da tabela ascii do espaço de memória.



Informa o estado de ativação (desativado ou ativado) de um periférico do PIC.

“ssss”: periférico do PIC, podendo assumir os valores ADCx (conversor AD), EXTx (interrupção externa), PWM (gerador de sinal PWM), CAN (comunicação CAN), 232 (comunicação RS232), I2C (comunicação I2C), TMx (contador *timer*) ou SPI (comunicação SPI).

“x”: estado de ativação do periférico, podendo ser 0 (desativado) ou 1 (ativado).

6.2.3. Software do servidor *Web*

Agora que foi mostrado o protocolo de comandos do sistema SGM-W, será apresentado o software desenvolvido para o servidor *Web*, para que no próximo tópico seja

mostrado o software do microcontrolador PIC gerenciado.

O software do servidor *Web* possui, principalmente, a pilha TCP/IP da Microchip, o sistema de arquivos MPFS da Microchip, uma implementação de um servidor *Web*, páginas *Web* armazenadas na memória para interface com o usuário e funções específicas para o tratamento de comandos a serem enviados e recebidos pela interface serial RS232.

Como já foi descrito, a pilha TCP/IP da Microchip possui os principais protocolos implementados para que seja possível estabelecer conexões à rede mundial de computadores, a internet.

O sistema de arquivos MPFS da Microchip é uma implementação necessária para o armazenamento dos arquivos que contêm as páginas *Web* a serem acessadas pelos usuários do sistema.

A implementação de servidor *Web* atende a pedidos de conexão e requisições HTTP para que as páginas *Web* possam ser exibidas no navegador utilizado pelo usuário.

As páginas *Web* armazenadas possuem implementações CGI para possibilitar ao usuário efetuar pedidos de envio de comandos para o PIC gerenciado e permitir a exibição de variáveis do servidor *Web*, que representam os estados dos periféricos do microcontrolador monitorado em tempo real. Como já foi explicado no capítulo 4, os pedidos de envio de comandos através da página *Web* são interpretados pela função HTTPExecCmd() do servidor *Web* e a busca dos valores das variáveis a serem exibidas nas páginas é feita pela função HTTPGetVar() toda vez que um caracter '%' é encontrado nas páginas *Web*.

A página *Web* principal do sistema é mostrada na Figura 6.2.3.1. É a partir dela que pode-se acessar as outras páginas para monitoramento de pinos (Figura 6.2.3.2), serviços (Figura 6.2.3.4), memória EEPROM (Figura 6.2.3.5) e memória RAM (Figura 6.2.3.6), bem como o *frame* para controle dos periféricos do PIC gerenciado através do envio de comandos (Figura 6.2.3.3).

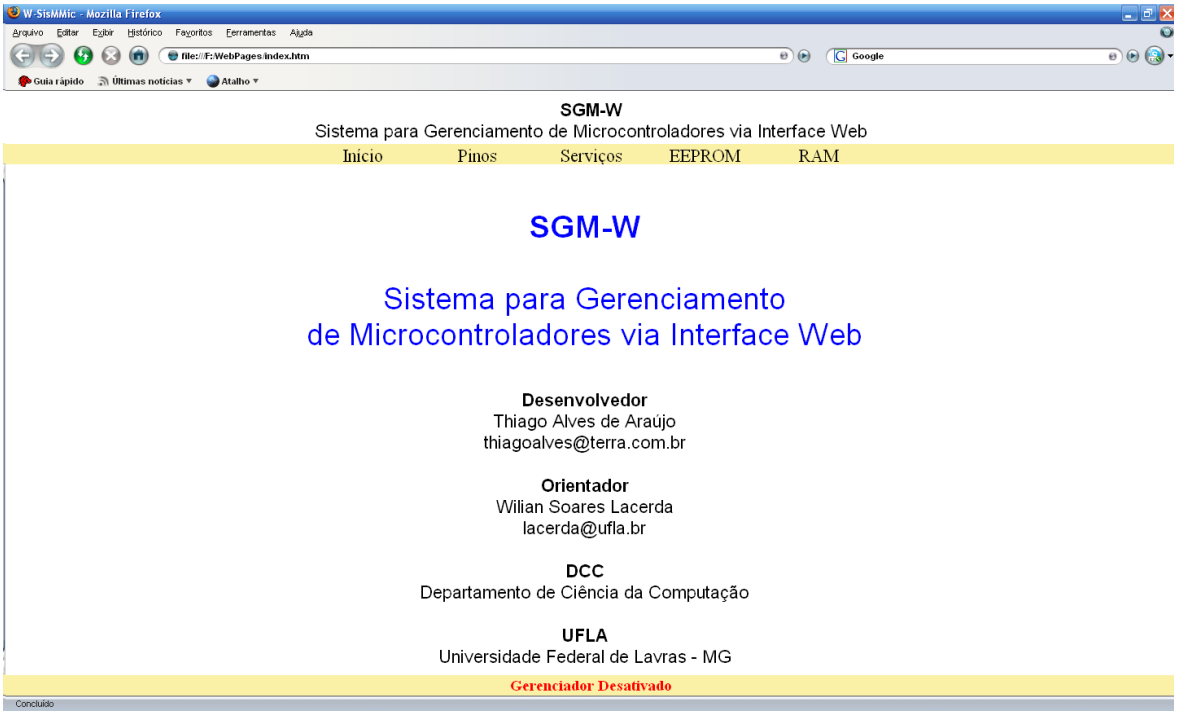


Figura 6.2.3.1 - Página *Web* principal

Figura 6.2.3.2 - Página de monitoramento de pinos

Figura 6.2.3.3 - *Frame* para envio de comandos ao PIC monitorado

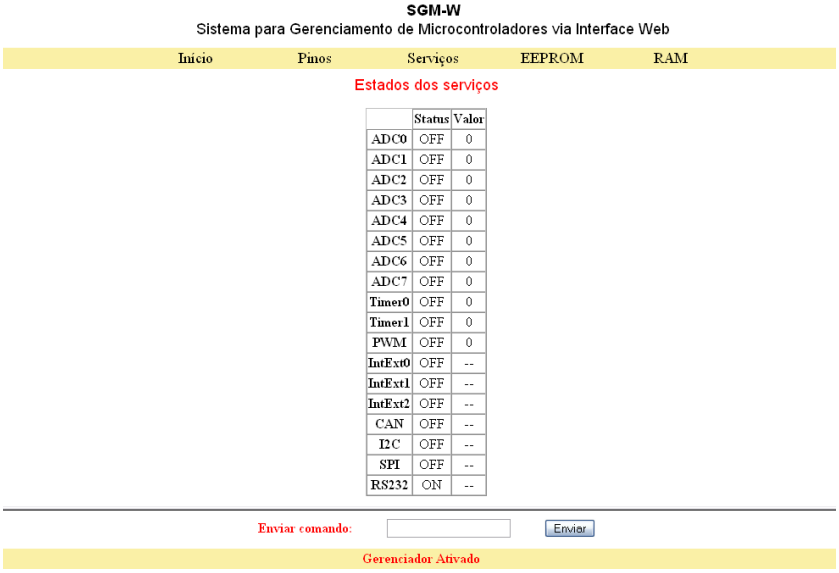


Figura 6.2.3.4 - Página de monitoramento de serviços

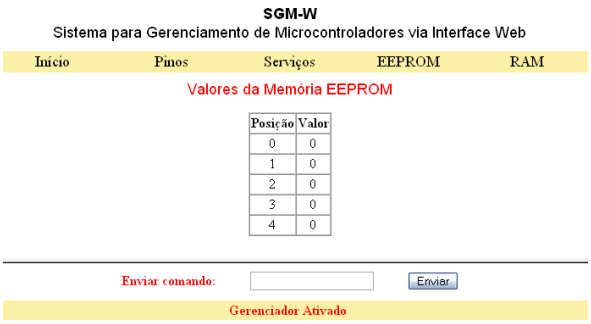


Figura 6.2.3.5 - Página de monitoramento da memória EEPROM

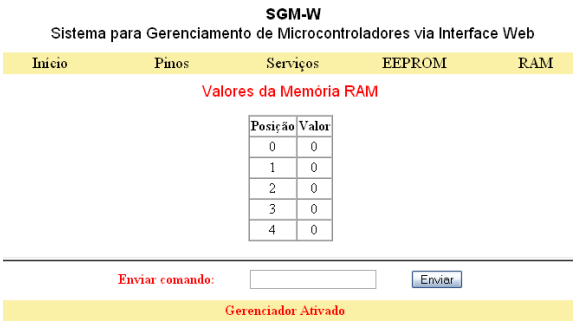


Figura 6.2.3.6 - Página de monitoramento da memória RAM

Para utilizar o sistema, primeiramente é necessário estabelecer uma conexão entre o servidor *Web* e o PIC monitorado através da porta serial. Para isso, depois de conectados pelo cabo serial, basta enviar um comando "CONNECT" através do *frame* de controle dos periféricos.

A cada alteração de seus estados, o microcontrolador monitorado enviará comandos ao servidor *Web*, para que o mesmo possa mostrar tais eventos na área de visualização dos estados nas páginas de monitoramento.

As funções específicas desenvolvidas, que tratam da interpretação de comandos recebidos, atendimento a pedidos de leitura de variáveis pela página *Web*, solicitação de envio de comandos para o PIC monitorado, dentre outras, são as seguintes:

- **#pragma interruptlow SerialISR void SerialISR(void):** Trata do recebimento de um comando pela interface serial, utilizando interrupção. Após recebê-lo, o comando é interpretado pela função ReadCommand().

- **void ReadCommand(char* command):** Interpreta o comando recebido de forma a atualizar os valores das variáveis que representam os estados dos periféricos do microcontrolador gerenciado.

- **WORD HTTPGetVar(unsigned char var, WORD ref, unsigned char* val):** Retorna para a página *Web* o valor obtido de um periférico do PIC monitorado. É ativada por uma requisição CGI quando encontrado o caracter '%' na página *Web*.

- **void HTTPExecCmd(unsigned char** argv, unsigned char argc):** Envia comandos para o PIC gerenciado. Esta função é chamada pelo servidor *Web* e ativada pela página *Web* quando ocorre uma requisição CGI com o conteúdo do comando a ser enviado.

- **void CommandSplit(char* word, char* oper, char* peripheral, char* value1, char* value2):** Divide o comando recebido de modo a separar identificadores de comandos, identificadores de periféricos, endereços de memória e valores de forma a facilitar a interpretação do comando como um todo.

6.2.4. Software do microcontrolador PIC gerenciado

Para completar o sistema SGM-W, o software em execução no microcontrolador PIC possui duas partes: uma relacionada ao sistema desenvolvido em si e outra à execução de tarefas para sua intervenção no meio em que está inserido. Em outras palavras, a primeira seria um módulo do *firmware* (programa do microcontrolador) do PIC que entra em execução assim que é estabelecido uma conexão entre o microcontrolador e o servidor através da página *Web*. Esse módulo do *firmware* é o módulo de comunicação SGM-W do PIC, que é executado sem interferir na segunda parte do programa do microcontrolador que desempenha funções dependentes do ambiente onde o *chip* está inserido.

Portanto, o programa principal do PIC precisa incluir o módulo de comunicação SGM-W para que o sistema funcione em modo de monitoramento e controle. A partir disso, um desenvolvedor do PIC poderá escrever seus programas normalmente para que o microcontrolador execute as tarefas desejadas sem estar interferindo no módulo de comunicação ou o inverso.

Agora, em se tratando do código-fonte do programa, a linguagem de programação escolhida foi a C para PIC. Existem bibliotecas escritas em linguagem C que implementam e definem algumas funções do microcontrolador da família PIC. Portanto, fica muito mais

intuitivo utilizar os recursos do microcontrolador, uma vez que na maioria das vezes é necessária a chamada de apenas um procedimento através de uma linha para executar tarefas mais avançadas que exigiriam um grande esforço do programador em assembly. O programa é modularizado em procedimentos pertencentes ao módulo de comunicação, que irão desempenhar cada tarefa necessária. Esses procedimentos do módulo de comunicação do SGM-W no PIC estão agrupados em um único arquivo de código-fonte, que deverá ser incluído no código do programa principal. Vejamos agora os principais procedimentos do módulo de comunicação:

void MonitorInit(): Este procedimento é encarregado de ativar o modo de gerenciamento do PIC. Deverá ser chamado pelo programa principal, de modo a ativar o estado quando desejar.

void MonitorTerminate(): Desativa o modo de gerenciamento do PIC. Também deverá ser chamado pelo programa principal, de modo a desativar o monitoramento do PIC.

#int_rda void SerialReceiver(): Procedimento que implementa a ação a ser feita quando no recebimento de algum comando pela porta serial RS232 do PIC. Ao receber um comando, o procedimento verifica se é um comando válido e, se positivo, interpreta e desempenha as tarefas solicitadas no comando.

void SerialTransmit(char* wordTX): Envia um comando pela porta serial RS232 do PIC. É neste procedimento onde ocorre a informação das alterações dos estados dos periféricos do microcontrolador gerenciado para o servidor *Web*.

void ConnectHost(): Estabelece uma conexão entre o microcontrolador gerenciado e o servidor *Web*. É chamado no recebimento de um comando pela serial solicitando o estabelecimento da conexão.

void DisconnectHost(): Interrompe a conexão estabelecida entre os dispositivos.

void MonitorVerifyAlterations(): Este procedimento executa a verificação constante dos estados dos periféricos e, caso tenha alguma alteração de seus valores, chama o procedimento de envio de comandos de informação ao servidor *Web*.

void ReadCommand(char* word): Irá processar o comando recebido pela serial, chamando os métodos necessários para a interpretação do mesmo.

void CommandSplit(char* word, char* oper, char* peripheral, char* value1, char* value2): Divide o comando recebido de modo a separar identificadores de comandos, identificadores de periféricos, endereços de memória e valores a serem atribuídos de forma a facilitar a interpretação do comando como um todo.

void InterpretCommand(char* oper, char* peripheral, char* value1, char* value2): É neste procedimento que ocorre realmente a interpretação do comando recebido. Ao ser chamado, o comando recebido já deverá estar todo dividido e preparado para ser identificado pelo procedimento.

void SendInfo(char typeInfo, char* peripheral, int16 value1, int16 value2, int8 sizeValue1, int8 sizeValue2): É o procedimento responsável pelo envio de comandos de informação dos estados dos periféricos do PIC através da porta serial.

Basicamente, o programa do PIC estará sempre em execução normal até que ocorra alguma alteração de seus estados e assim o ocorrido seja informado ao módulo de comunicação, para que o mesmo envie o devido comando de informação ao servidor *Web*. Assim que enviado, o módulo de comunicação devolve o controle ao programa principal do PIC.

No momento do recebimento de algum comando vindo do servidor *Web* ocorre uma interrupção externa para o recebimento do comando através do módulo de comunicação. Caso o comando seja reconhecido pelo PIC, ele será interpretado e traduzido em ações, sejam elas um uso de algum componente ou uma solicitação da informação de

seus estados atuais. Assim que o comando é atendido, como na primeira situação, o módulo de comunicação devolve o controle ao programa principal.

Portanto, o módulo de comunicação SGM-W do PIC deverá estar sempre em execução, de maneira a não interferir no programa principal, executando tarefas solicitadas pelo servidor *Web* através do usuário e mantendo o servidor sempre informado das alterações ocorridas em seus periféricos.

7. RESULTADOS E DISCUSSÃO

Agora que o SGM-W já foi apresentado e seu funcionamento explicado, esta seção apresentará os resultados obtidos pelo uso do sistema, através de exemplos de utilização do mesmo.

Antes de qualquer uso do sistema, deve-se ter certeza de que o microcontrolador gerenciado e o servidor *Web* estejam conectados pelas suas respectivas portas seriais padrão RS232. Também, o servidor *Web* deverá estar ligado a algum ponto de rede disponível no local, com um IP disponível e conhecido (Figura 7.1). Nos exemplos será utilizado o IP 192.168.254.254 e máscara 255.255.255.0 para o servidor *Web*.

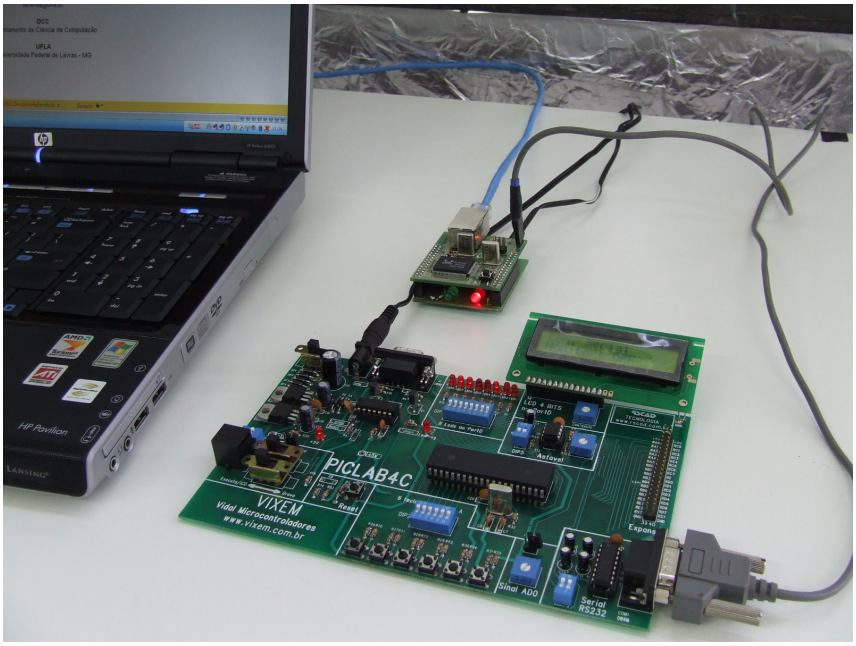


Figura 7.1 - Conexão dos dispositivos do sistema SGM-W

Assim que todos os procedimentos de conexão são feitos, o sistema SGM-W pode ser acessado através de um navegador *Web* de qualquer dispositivo, desde computadores do tipo PC até celulares, que esteja em qualquer localidade do mundo, desde que tenha uma conexão com a rede internet.

Acessando o sistema pela primeira vez, a partir de um *browser* de computador PC, tem-se uma página inicialmente sem os estados atuais do microcontrolador (Figura

7.2) devido a conexão ainda não estabelecida entre o servidor *Web* e o microcontrolador gerenciado (Figura 7.3).

Figura 7.2 - Tela inicial da página *Web* do sistema

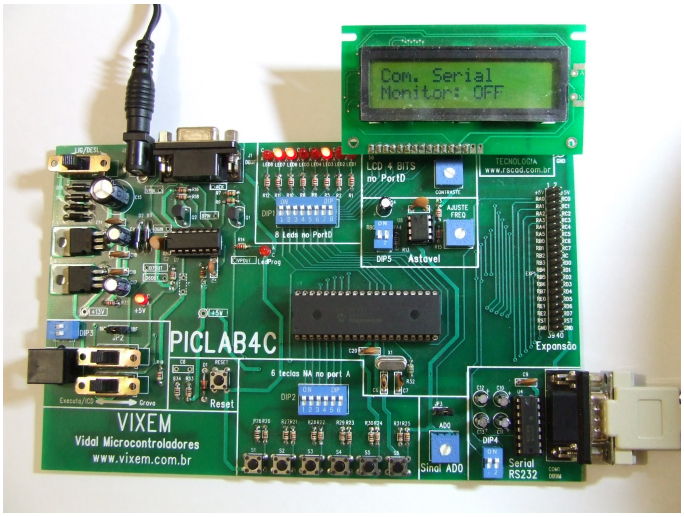


Figura 7.3 - Microcontrolador gerenciado ainda sem conexão com o servidor

Para efetuar a conexão entre os dispositivos, deve-se enviar um comando de ativação de estado de monitoramento do PIC através da caixa “Enviar comando:” existente em qualquer uma das páginas de monitoramento: “Pinos”, “Serviços”, “EEPROM” ou “RAM” (Figura 7.4). Assim que for efetuada a conexão, o microcontrolador gerenciado envia automaticamente comandos de informação dos estados de seus periféricos e a página Web é atualizada pelo servidor com os valores recebidos (Figura 7.5 e Figura 7.6). Maiores detalhes de outros periféricos do microcontrolador poderão ser vistos através das

outras páginas de monitoramento acessíveis pelo menu, as quais informam, além dos estados dos pinos de suas portas, os estados e valores dos serviços oferecidos por cada porta (Figura 7.7) e os valores de suas memórias (Figura 7.8 e Figura 7.9).

Figura 7.4 - Envio de comando de pedido de conexão ao microcontrolador gerenciado

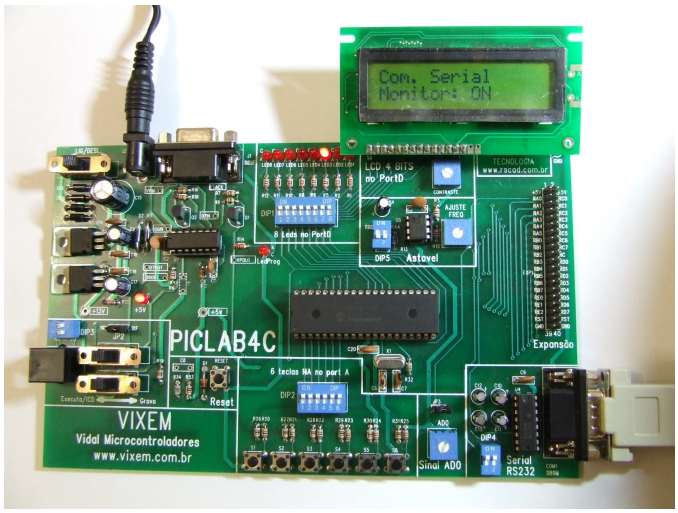


Figura 7.5 - Microcontrolador gerenciado após pedido de conexão

Figura 7.6 - Página de monitoramento dos pinos após conexão estabelecida

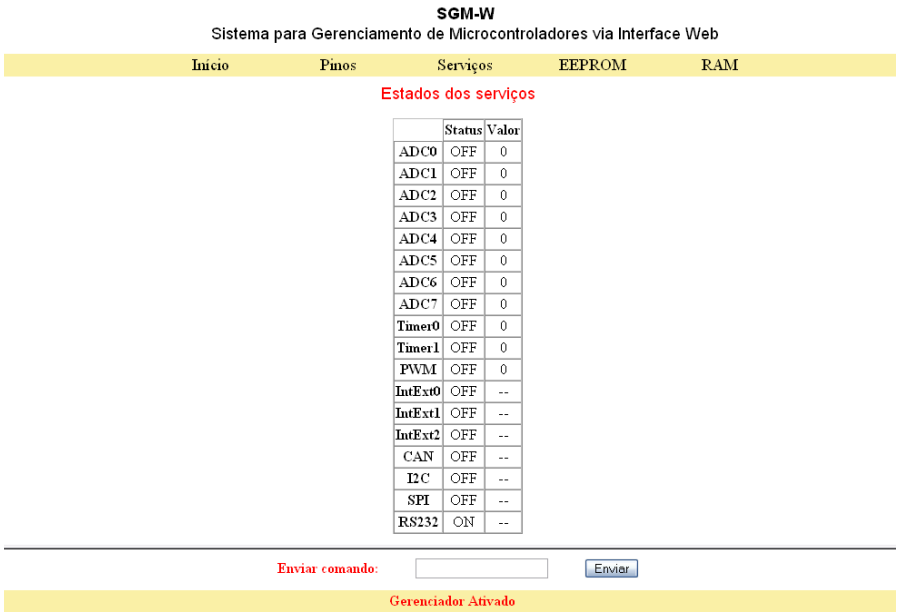


Figura 7.7 - Página de monitoramento de periféricos

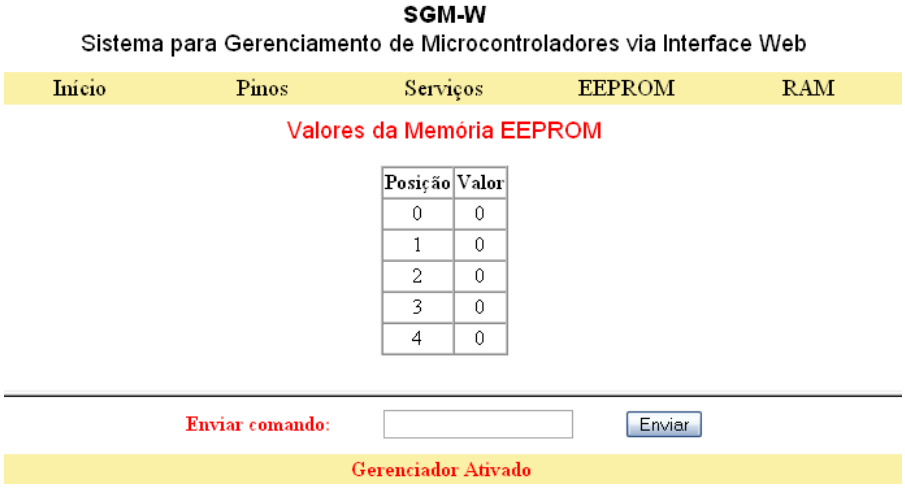


Figura 7.8 - Página de monitoramento da memória EEPROM

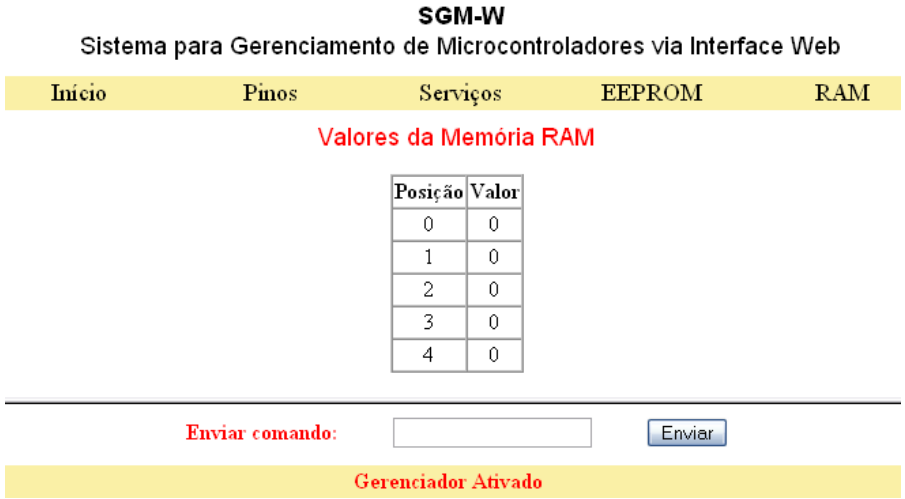


Figura 7.9 - Página de monitoramento da memória RAM

Assim, feita a conexão, o microcontrolador estará sendo monitorado constantemente e esperando por recebimento de comandos para execução de tarefas. Como exemplo, ao enviar um comando para alteração do valor de saída da porta D para 114 em decimal (Figura 7.10) o PIC monitorado recebe o comando, o interpreta, altera o estado da porta requisitada e envia um comando de informação do estado da porta D para o servidor *Web*. Assim que recebido, o servidor *Web* se encarrega de interpretar o comando e colocar na página de monitoramento dos pinos o resultado das alterações (Figura 7.11).

Figura 7.10 - Envio de comando para alteração da porta D

Figura 7.11 - Página de monitoramento dos pinos, atualizada após alteração no PIC gerenciado

Um outro tipo de comando a ser enviado ao PIC, seria a ativação de um serviço, como a geração de sinal PWM. O sinal PWM possui uma parcela de seu período de sinal de oscilação em que fica em nível lógico alto (+5V). Essa parte em que o sinal PWM fica em nível alto é chamado de *duty cicle*. O *duty cicle* é a relação T_{on}/T, onde T_{on} é o tempo em que o sinal fica em nível lógico alto (ligado) e T é o tempo total do sinal. Portanto, ao enviar um comando para alterar o *duty cicle* do PWM para 127 (Figura 7.12), o PIC recebe o comando, interpreta e, caso o PWM esteja desativado, o mesmo é ativado e seu *duty cicle* é alterado para o valor desejado. Como anteriormente, o PIC irá enviar um comando de informação do estado do gerador de sinal PWM ao servidor *Web* para que o mesmo possa mostrar os resultados na página de monitoramento dos periféricos (Figura 7.13).

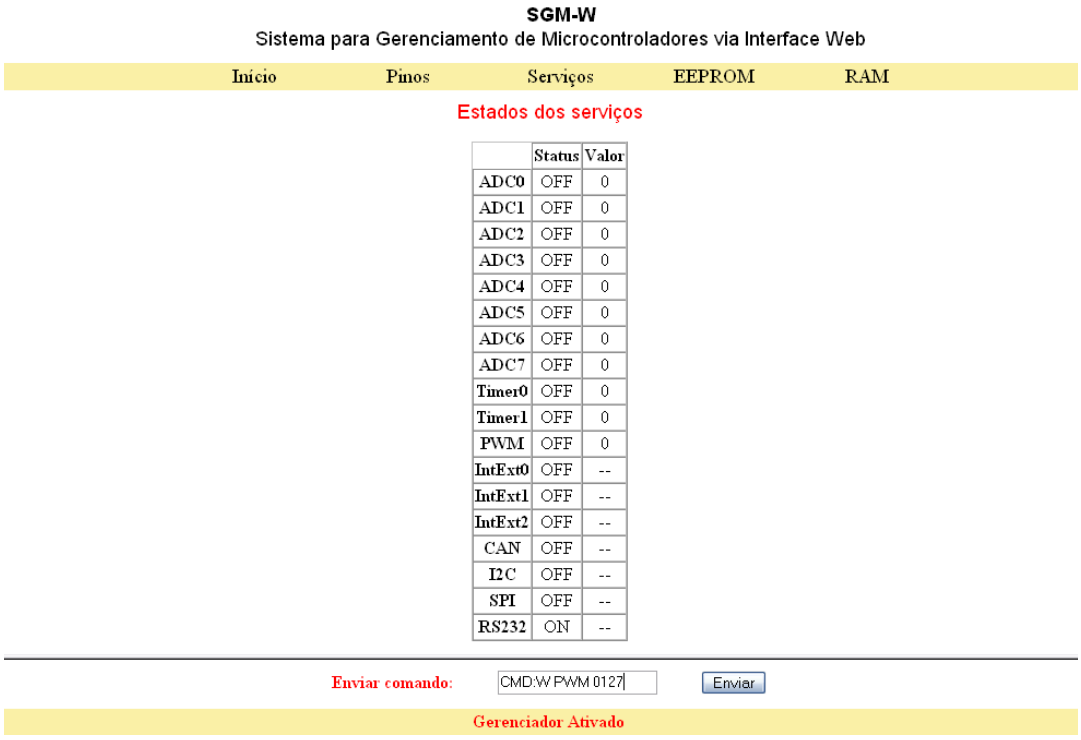


Figura 7.12 - Envio de comando de ativação e alteração do *duty cicle* do gerador de PWM

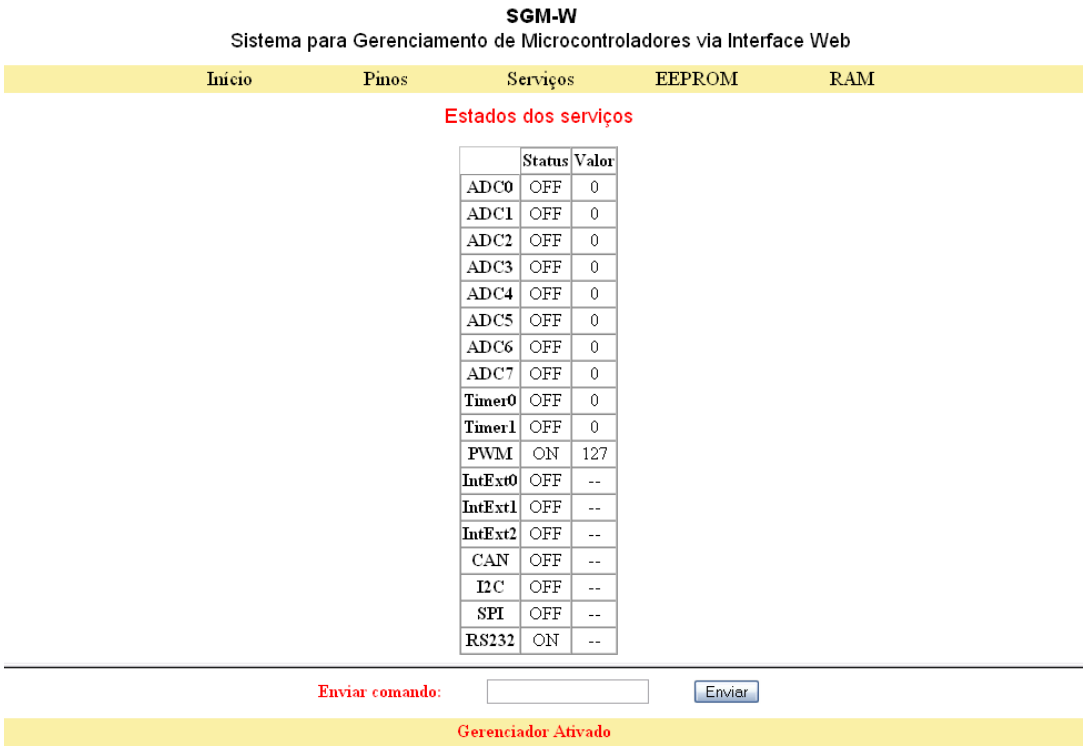


Figura 7.13 - Página de monitoramento dos periféricos, atualizada após alteração no PWM do PIC gerenciado

Nas páginas de monitoramento das memórias EEPROM e RAM, têm-se os valores das últimas 5 posições das memórias requisitadas pelo servidor *Web*. É possível ler e alterar seus valores através do envio de comandos. Ao enviar um comando para escrita em memória EEPROM (Figura 7.14), especifica-se o endereço e o valor a ser atribuído, no caso deste exemplo, a posição 10 com o valor 56. Assim que o PIC gerenciado recebe o comando para escrita em memória, o mesmo interpreta e altera o valor da posição desejada, desde que ela exista. Assim que alterado, o PIC envia um comando de informação do valor alterado na memória ao servidor *Web*. Desta maneira, o servidor *Web* atualiza, na respectiva posição, o valor da memória (Figura 7.15).

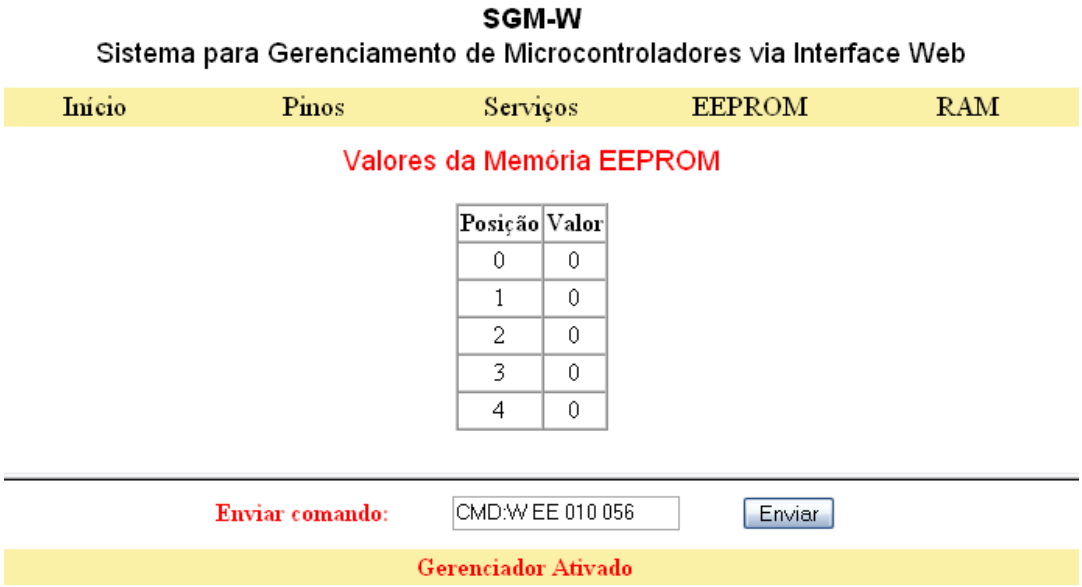


Figura 7.14 - Envio de comando para alteração do valor da memória EEPROM

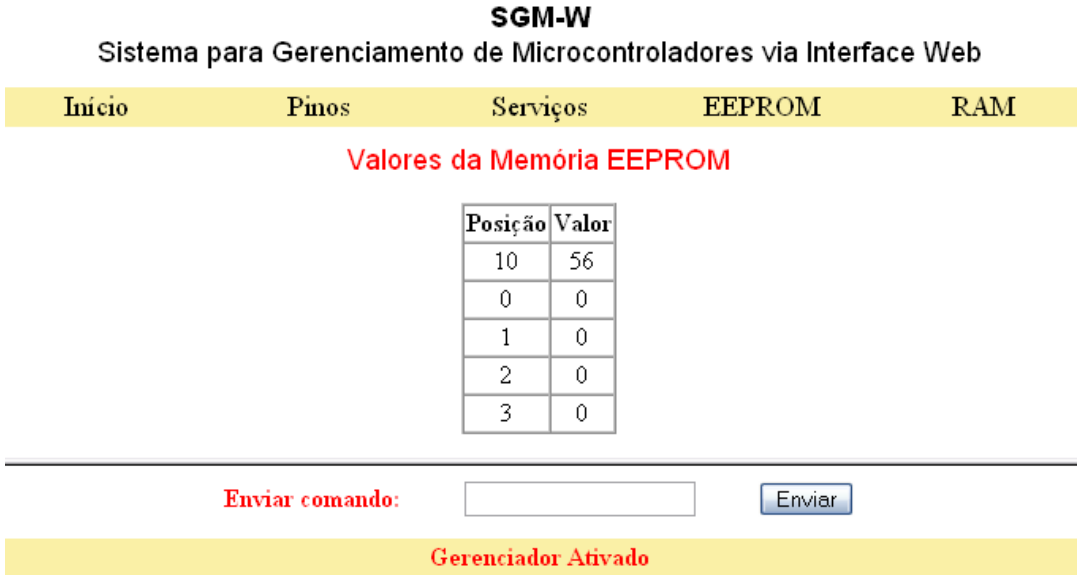


Figura 7.15 - Página de monitoramento da memória EEPROM, atualizada após sua alteração no PIC gerenciado

Portanto, com o sistema SGM-W, pode-se ter um controle e monitoramento dos periféricos do microcontrolador através do envio e recebimento de comandos pelo servidor *Web* acessado através de um navegador. Assim que um comando para alterar um estado é recebido pelo PIC gerenciado, o mesmo retorna uma resposta caso tenha sido alterado com sucesso.

8. CONCLUSÕES

Com o sistema SGM-W pode-se ter um controle e, ao mesmo tempo, um monitoramento de eventos ocorridos em um microcontrolador através de uma interface *Web* utilizando um navegador que poderá estar desde em um computador PC até celulares. Com isso, o sistema pode ser utilizado para que os recursos do microcontrolador sejam utilizados de forma remota sem a necessidade da manipulação direta do dispositivo microcontrolador. Pode-se programar o PIC gerenciado para efetuar tarefas específicas, com a obtenção de dados ou variáveis, de forma a retornar ao servidor *Web* os resultados obtidos com sua execução. Desta forma, o servidor *Web* poderá avaliar esses resultados, tomar decisões e comandar o PIC, tudo de modo automático e de qualquer parte do mundo através do usuário do sistema em um navegador *Web.*

Uma característica importante é a independência de outros dispositivos para que o sistema funcione. Basta o servidor *Web* estar conectado a qualquer ponto de rede que o SGM-W pode ser acessado através de qualquer navegador *Web*.

Pode-se observar nos exemplos que o sistema é bastante genérico quanto à monitoramento e controle dos recursos do microcontrolador. Quando o SGM-W for utilizado para gerenciar um microcontrolador que esteja realizando tarefas específicas basta efetuar algumas alterações para um melhor entendimento dos eventos ocorridos. Um exemplo seria, em um controle de irrigação, colocar identificadores nas páginas *Web* dos atuadores e sensores ligados ao PIC, como motores de pivôs e sensores de umidade e pressão.

Propostas de continuidade

Uma proposta futura para o projeto é a utilização de uma comunicação entre o servidor *Web* e o ponto de rede através de uma conexão sem fio. Isso poderia ser

desenvolvido utilizando um servidor *Web* com a implementação da pilha TCP/IP semelhante ao do sistema, mas com algumas implementações dos protocolos envolvidos em uma conexão *wireless*. Uma possível solução é a utilização do protocolo MIWI desenvolvido e disponibilizado pela Microchip que implementa as características necessárias para a rede *wireless* funcionar.

REFERÊNCIAS BIBLIOGRÁFICAS

2EI. **Eletrônica embarcada para internet**. Versão 1.10. Águas de Lindóia: Autor-editor independente, 2007. 64 p. Disponível em: <http://www.2ei.com.br/documentacao/AN001.PDF>. Acessado em: 26 de novembro 2007.

AN833. **Microchip TCP/IP Stack Application Note.** Disponível em: <http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en011993>. Acessado em 26 de novembro 2007.

IC-PROG PROTOTYPE PROGRAMMER. Disponível em: <http://www.ic-prog.com>. Acessado em: 14 de agosto 2007.

JDM PROGRAMMER. **Pic-Programmer 2 for PIC16C84 etc.** Disponível em: <http://www.jdm.homepage.dk/newpic.htm>. Acessado em: 26 de novembro 2007.

JUNG, Carlos Fernando. **Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos.** Rio de Janeiro: Axcel Books do Brasil Editora LTDA, 2004. 328 p.

KUROSE, James F.; ROSS, Keith W.. **Redes de computadores e a internet: uma nova abordagem**. 1ª ed. São Paulo: Addison Wesley, 2003. 548 p.

MICROCHIP. **PIC 18FXX8 Data Sheet**. U.S.A.: Microchip Technology Inc, 2004. 402p. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/41159e.pdf>. Acessado em: 10 de maio 2007.

PEREIRA, Fábio. **Microcontroladores PIC: programação em C**. 1ª ed. São Paulo: Editora Érica, 2003. 358 p.

SILVA JR., Vidal Pereira da. **Microcontroladores PIC: teoria e prática**. 1ª ed. São Paulo: Autor-editor independente, 1997. 140 p.

SILVA JR., Vidal Pereira da. **Aprenda Rápido PIC 18F Direto em Linguagem 'C'.** Guarulhos: Autor-editor independente, 2006. 121 p.

SOUZA, David José de. **Desbravando o PIC: ampliado e atualizado para PIC 16F628A**. 6ª ed. São Paulo: Editora Érica, 2003. 268 p.

TANEMBAUM, Andrew S. **Redes de computadores**. 3ª ed. Rio de Janeiro: Editora Campus, 1997. 923 p.

ANEXO I – CÓDIGO FONTE DO MÓDULO DE COMUNICAÇÃO SGM-W DO PIC MONITORADO

Arquivo: "SGM-W.c"

```
#include <string.h>
#include <stdlib.h>
#use rs232(BAUD=9600, XMIT=PIN_C6, RCV=PIN_C7,BITS=8)

#byte PortA = 0xF80 //Porta A
#byte PortB = 0xF81 //Porta B
#byte PortC = 0xF82 //Porta C
#byte PortD = 0xF83 //Porta D
#byte PortE = 0xF84 //Porta E
#byte TrisA = 0xF92 //Tris porta A
#byte TrisB = 0xF93 //Tris porta B
#byte TrisC = 0xF94 //Tris porta C
#byte TrisD = 0xF95 //Tris porta D
#byte TrisE = 0xF96 //Tris porta E
#define SIZE_EEPROM 256
#define SIZE_RAM 4096
#define SIZE_DATA_RAM 1536
#define MAX_VALUE_EEPROM 255
#define MAX_VALUE_RAM 255
#define MAX_VALUE_PORT_A 63
#define MAX_VALUE_PORT_B 255
#define MAX_VALUE_PORT_C 255
#define MAX_VALUE_PORT_D 255
#define MAX_VALUE_PORT_E 7
#define NUM_PINS_PORT_A 6
#define NUM_PINS_PORT_B 8
#define NUM_PINS_PORT_C 8
#define NUM_PINS_PORT_D 8
#define NUM_PINS_PORT_E 3
#define NUM_DIV_PWM 1024

//ativação de monitoramento dos pinos
#define RA_ON true
#define RB_ON false
#define RC_ON false
#define RD_ON true
#define RE_ON false
//tamanho das strings:
#define SIZE_WORD 21    //tamanho máximo dos comandos tipo CMD:
#define SIZE_OPER 6
#define SIZE_PERIPHERAL 4
#define SIZE_VALUE1 5
#define SIZE_VALUE2 4
#define SIZE_MAX_VALUE 5    //maior valor de SIZE_VALUE1 e SIZE_VALUE2
#define SIZE_MAX_INFO 22    //tamanho máximo dos comandos tipo INF:
//numero de dígitos dos valores
#define NUM_DIGITS_PORTS 3
#define NUM_DIGITS_ADDRESS_EE 3
#define NUM_DIGITS_ADDRESS_RAM 4
#define NUM_DIGITS_CYCLE_PWM 4

//variáveis
boolean receiverOn, monitorOn;
boolean PWMOn = false;
int8 ra, rb, rc, rd, re, ta, tb, tc, td, te;

//procedimentos
void MonitorInit();
```

```
void MonitorTerminate();
#int_rda
void SerialReceiver();
void SerialTransmit(char* wordTX);
void ConnectHost();
void DisconnectHost();
void MonitorVerifyAlterations();
void ReadCommand(char* word);
void CommandSplit(char* word, char* oper, char* peripheral, char* value1,
char* value2);
void InterpretCommand(char* oper, char* peripheral, char* value1, char*
value2);
void SendInfo(char typeInfo, char* peripheral, int16 value1, int16
value2, int8 sizeValue1, int8 sizeValue2);
void CmdWrtPort(char port, unsigned int8 value);
void CmdWrtPin(char port, char channelPort, int8 value);
void CmdWrtTris(char port, unsigned int8 value);
void CmdWrtTrisPin(char port, char channelPort, int8 value);
void CmdReadPort(char port);
void CmdReadTris(char port);
void IntToString(int16 data, char* dataChar, int8 sizeDataChar);
unsigned int16 StringToInt16(char* valueChar);

//-----------------------------------------------------------------------

void MonitorInit()
{
    receiverOn = true;
    monitorOn = false;

      enable_interrupts(int_rda);
      enable_interrupts(global);
}
//-----------------------------------------------------------------------

void MonitorTerminate()
{
    disable_interrupts(int_rda);
    monitorOn = false;
}
//-----------------------------------------------------------------------

#int_rda
void SerialReceiver(void)
{
    if (receiverOn)
    {
        int8 counter, data;
        char word[SIZE_WORD];
        receiverOn = false;
        counter = 0;
        do
        {
            data = getchar();
            word[counter] = data;
            counter++;
        }while (data != '\n');  //vai recebendo até ler um '\n'
(finalizador de comando)
        word[counter] = '\0';   //finaliza o comando recebido.
        SerialTransmit(&word[0]);   //envia a confirmação do recebimento
```

```
do comando
        ReadCommand(&word[0]);  //verifica se o comando é válido
        receiverOn = true;  //habilita o recebimento de dados pela serial
    }
}
//---------------------------------------------------------------------

void SerialTransmit(char* wordTX)
{
    int8 counter;
      for (counter = 0; wordTX[counter]; counter++)
      {
            putchar(wordTX[counter]);
      }
}
//---------------------------------------------------------------------

void ConnectHost()
{
    unsigned int8 data;
    unsigned int16 address; //int16 para não dar overflow na compara ão
do "for" abaixo
    char peripheral[3];

    ra = PortA; rb = PortB; rc = PortC; rd = PortD; re = PortE; ta =
TrisA; tb = TrisB; tc = TrisC; td = TrisD; te = TrisE;
    monitorOn = true;
    peripheral[0] = 'R'; peripheral[2] = '\0';
    peripheral[1] = 'A';
    SendInfo('S', &peripheral[0], PortA, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'B';
    SendInfo('S', &peripheral[0], PortB, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'C';
    SendInfo('S', &peripheral[0], PortC, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'D';
    SendInfo('S', &peripheral[0], PortD, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'E';
    SendInfo('S', &peripheral[0], PortE, 0, NUM_DIGITS_PORTS, 0);

    peripheral[0] = 'T';
    peripheral[1] = 'A';
    SendInfo('S', &peripheral[0], TrisA, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'B';
    SendInfo('S', &peripheral[0], TrisB, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'C';
    SendInfo('S', &peripheral[0], TrisC, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'D';
    SendInfo('S', &peripheral[0], TrisD, 0, NUM_DIGITS_PORTS, 0);
    peripheral[1] = 'E';
    SendInfo('S', &peripheral[0], TrisE, 0, NUM_DIGITS_PORTS, 0);

    peripheral[0] = 'E'; peripheral[1] = 'E';
    for (address = 0; address < SIZE_EEPROM; address++)
    {
        data = Read_Eeprom(address);
        if (data != MAX_VALUE_EEPROM)
            SendInfo('S', &peripheral[0], address, data,
NUM_DIGITS_ADDRESS_EE, 3);
    }
}
```

```
//---------------------------------------------------------------------

void DisconnectHost()
{
    monitorOn = false;
}
//---------------------------------------------------------------------

void MonitorVerifyAlterations()
{
    char peripheral[3];
    peripheral[2] = '\0';
    if (monitorOn)
    {
        if((RA_ON)&&(ra != PortA))
        {
            ra = PortA;
            peripheral[0] = 'R';    peripheral[1] = 'A';
            SendInfo('S', &peripheral[0], PortA, 0, NUM_DIGITS_PORTS, 0);
        }
        if((RB_ON)&&(rb != PortB))
        {
            rb = PortB;
            peripheral[0] = 'R';    peripheral[1] = 'B';
            SendInfo('S', &peripheral[0], PortB, 0, NUM_DIGITS_PORTS, 0);
        }
        if((RC_ON)&&(rc != PortC))
        {
            rc = PortC;
            peripheral[0] = 'R';    peripheral[1] = 'C';
            SendInfo('S', &peripheral[0], PortC, 0, NUM_DIGITS_PORTS, 0);
        }
        if((RD_ON)&&(rd != PortD))
        {
            rd = PortD;
            peripheral[0] = 'R';    peripheral[1] = 'D';
            SendInfo('S', &peripheral[0], PortD, 0, NUM_DIGITS_PORTS, 0);
        }
        if((RE_ON)&&(re != PortE))
        {
            re = PortE;
            peripheral[0] = 'R';    peripheral[1] = 'E';
            SendInfo('S', &peripheral[0], PortE, 0, NUM_DIGITS_PORTS, 0);
        }
        if(ta != TrisA)
        {
            ta = TrisA;
            peripheral[0] = 'T';  peripheral[1] = 'A';
            SendInfo('S', &peripheral[0], TrisA, 0, NUM_DIGITS_PORTS, 0);
        }
        if(tb != TrisB)
        {
            tb = TrisB;
            peripheral[0] = 'T';  peripheral[1] = 'B';
            SendInfo('S', &peripheral[0], TrisB, 0, NUM_DIGITS_PORTS, 0);
        }
        if(tc != TrisC)
        {
            tc = TrisC;
            peripheral[0] = 'T';  peripheral[1] = 'C';
```

```
            SendInfo('S', &peripheral[0], TrisC, 0, NUM_DIGITS_PORTS, 0);
        }
        if(td != TrisD)
        {
            td = TrisD;
            peripheral[0] = 'T';  peripheral[1] = 'D';
            SendInfo('S', &peripheral[0], TrisD, 0, NUM_DIGITS_PORTS, 0);
        }
        if(te != TrisE)
        {
            te = TrisE;
            peripheral[0] = 'T';  peripheral[1] = 'E';
            SendInfo('S', &peripheral[0], TrisE, 0, NUM_DIGITS_PORTS, 0);
        }
    }
}
//---------------------------------------------------------------------

void ReadCommand(char* word)
{
    char oper[SIZE_OPER], peripheral[SIZE_PERIPHERAL],
value1[SIZE_VALUE1], value2[SIZE_VALUE2], CONNECT[10], DISCONNECT[13];
    //"CONNECT\r\n"
    CONNECT[0] = 'C';    CONNECT[1] = 'O';    CONNECT[2] = 'N';
CONNECT[3] = 'N';    CONNECT[4] = 'E';    CONNECT[5] = 'C';    CONNECT[6]
= 'T';    CONNECT[7] = '\r';    CONNECT[8] = '\n';    CONNECT[9] = '\0';
    //"DISCONNECT\r\n"
    DISCONNECT[0] = 'D';    DISCONNECT[1] = 'I';    DISCONNECT[2] = 'S';
DISCONNECT[3] = 'C';    DISCONNECT[4] = 'O';    DISCONNECT[5] = 'N';
DISCONNECT[6] = 'N';    DISCONNECT[7] = 'E';    DISCONNECT[8] = 'C';
DISCONNECT[9] = 'T';    DISCONNECT[10] = '\r';    DISCONNECT[11] = '\n';
DISCONNECT[12] = '\0';
    if (strcmp(word, &CONNECT[0]) == 0)
    {
        ConnectHost();
    }else if (strcmp(word, &DISCONNECT[0]) == 0)
    {
        DisconnectHost();
    }else
    {
        CommandSplit(&word[0], &oper[0], &peripheral[0], &value1[0],
&value2[0]);   //separa as partes do comando
        InterpretCommand(&oper[0], &peripheral[0], &value1[0],
&value2[0]);   //interpreta comando
    }
}
//---------------------------------------------------------------------

void CommandSplit(char* word, char* oper, char* peripheral, char* value1,
char* value2)
{
    ////////////comando
    int8 counterWord, counter, data;
    oper[0] = '\0';
    peripheral[0] = '\0';
    value1[0] = '\0';
    value2[0] = '\0';
    counter = 0;
    counterWord = 0;
    while((counter<SIZE_OPER-
```

```
1)&&(word[counterWord])&&(word[counterWord]!=' '))
    {
        oper[counter] = word[counterWord];
        counter++;
        counterWord++;
    }
    oper[counter] = '\0';
    ///////////periférico do PIC
    if (word[counterWord] == ' ')
    {
        counterWord++;
        counter = 0;
        while((counter<SIZE_PERIPHERAL-1) && (word[counterWord]) &&
(word[counterWord]!=' ') && (word[counterWord]!='\r'))
        {
            peripheral[counter] = word[counterWord];
            counter++;
            counterWord++;
        }
        peripheral[counter] = '\0';
        //////////valor ou endere o  de uma porta ou registrador
        if (word[counterWord] == ' ')
        {
            counterWord++;
            counter = 0;
            while((counter<SIZE_VALUE1)&&(word[counterWord])&&(word[count
erWord]!=' ')&&(word[counterWord]!='\r'))
            {
                value1[counter] = word[counterWord];
                counter++;
                counterWord++;
            }
            value1[counter] = '\0';
            //////////valor de uma porta ou registrador
            if (word[counterWord] == ' ')
            {
                counterWord++;
                counter = 0;
                while((counter<SIZE_VALUE2)&&(word[counterWord])&&(word[c
ounterWord]!=' ')&&(word[counterWord]!='\r'))
                {
                    value2[counter] = word[counterWord];
                    counter++;
                    counterWord++;
                }
                value2[counter] = '\0';
            }
        }
    }
}
//---------------------------------------------------------------------

//Exemplo: CMD:W EE 123 456
//oper = CMD:W         peripheral = EE          value1 = 123      value2
= 456
void InterpretCommand(char* oper, char* peripheral, char* value1, char*
value2)
{
    char cmdType[6];
    //cmdType[6] = "CMD:W"
```

```
    cmdType[0] = 'C';    cmdType[1] = 'M';    cmdType[2] = 'D';
cmdType[3] = ':';    cmdType[4] = 'W';    cmdType[5] = '\0';

    //"CMD:W ..." -> comando de escrita
    if (strcmp(oper, &cmdType[0]) == 0)
    {
        //"CMD:W RAM xxxx xxx" -> alteração de RAM
        if ((peripheral[0] == 'R') && (peripheral[1] == 'A')&&
(peripheral[2] == 'M'))
        {
            unsigned int16 address;
            unsigned int8 data;
            address = StringToInt16(value1);
            data = atoi(value2);
            if ((address < SIZE_RAM) && (data <= MAX_VALUE_RAM))
            {
                int8* value;
                value = address;
                *value = data;
                SendInfo('S', &peripheral[0], value, *value,
NUM_DIGITS_ADDRESS_RAM, 3);
            }
        //"CMD:W Rxx xxx" -> alteração de porta
        }else if (peripheral[0] == 'R')
        {
            unsigned int8 data;
            data = atoi(value1);
            if ((peripheral[2] >= 48) &&(peripheral[2] <= 57))   //se
((peripheral[2] >= '0')&&(peripheral[2] <= '9'))
            {
                CmdWrtPin(peripheral[1], peripheral[2], data);
            }else
            {
                CmdWrtPort(peripheral[1], data);
            }
        //"CMD:W Txx xxx" -> alteração de Tris
        }else if (peripheral[0] == 'T')
        {
            unsigned int8 data;
            data = atoi(value1);
            if ((peripheral[2] >= 48) &&(peripheral[2] <= 57))   //se
((peripheral[2] >= '0')&&(peripheral[2] <= '9'))
            {
                CmdWrtTrisPin(peripheral[1], peripheral[2], data);
            }else
            {
                CmdWrtTris(peripheral[1], data);
            }
        //"CMD:W EE xxx xxx" -> alteração de EEPROM
        }else if ((peripheral[0] == 'E') && (peripheral[1] == 'E'))
        {
            unsigned int16 address;
            unsigned int8 data;
            address = StringToInt16(value1);
            data = atoi(value2);
            if ((address < SIZE_EEPROM) && (data <= MAX_VALUE_EEPROM))
            {
                Write_Eeprom(address, data);
                SendInfo('S', &peripheral[0], address, data,
NUM_DIGITS_ADDRESS_EE, 3);
```

```
            }
        //"CMD:W PWM xxxx" -> altera ão  de saída do sinal PWM
        }else if ((peripheral[0] == 'P') && (peripheral[1] == 'W')&&
(peripheral[2] == 'M'))
        {
            unsigned int16 dutyCycle;
            dutyCycle = StringToInt16(value1);
            if ((dutyCycle >= 0)&&(dutyCycle < NUM_DIV_PWM))
            {
                if (!PWMOn)
                {
                    setup_timer_2(T2_DIV_BY_16,248,1);
                    setup_ccp1(ccp_pwm);
                    PWMOn = true;
                }
                set_pwm1_duty(dutyCycle);
                SendInfo('S', &peripheral[0], dutyCycle, 0,
NUM_DIGITS_CYCLE_PWM, 0);
            }
        }
    //"CMD:R ..." -> comando de leitura
    }else if ((cmdType[4] = 'R')&&(strcmp(oper, &cmdType[0]) == 0))
    {
        //"CMD:R Rx" -> operacao de leitura de porta
        if ((peripheral[0] == 'R')&&(peripheral[2] == '\0'))
        {
            CmdReadPort(peripheral[1]);
        //"CMD:R Tx" -> operacao de leitura de Tris
        }else if (peripheral[0] == 'T')
        {
            CmdReadTris(peripheral[1]);
        //"CMD:R EE..." -> operacao de leitura de EEPROM
        }else if ((peripheral[0] == 'E') && (peripheral[1] == 'E'))
        {
            unsigned int8 data;
            unsigned int16 address;
            //"CMD:R EE" -> leitura de toda a EEPROM
            if (value1[0] == '\0')
            {
                for (address = 0; address < SIZE_EEPROM; address++)
                {
                    data = Read_Eeprom(address);
                    SendInfo('S', &peripheral[0], address, data,
NUM_DIGITS_ADDRESS_EE, 3);
                }
            //"CMD:R EE xxx" -> leitura de um endere o  da EEPROM
            }else
            {
                address = StringToInt16(value1);
                if (address < SIZE_EEPROM)
                {
                    data = Read_Eeprom(address);
                    SendInfo('S', &peripheral[0], address, data,
NUM_DIGITS_ADDRESS_EE, 3);
                }
            }
        //"CMD:R RAM..." -> opera ão  de leitura de mem ria  RAM
        }else if ((peripheral[0] == 'R') && (peripheral[1] == 'A') &&
(peripheral[2] == 'M'))
        {
```

```
            unsigned int8* value;
            unsigned int16 address;
            //"CMD:R RAM" -> leitura de toda a RAM
            if(value1[0] == '\0')
            {
                for (address = 0; address < SIZE_RAM; address++)
                {
                    value = address;
                    SendInfo('S', &peripheral[0], value, *value,
NUM_DIGITS_ADDRESS_RAM, 3);
                }
            //"CMD:R RAM xxxx" -> leitura de um endere o  da RAM
            }else
            {
                address = StringToInt16(value1);
                if(address < SIZE_RAM)
                {
                    value = address;
                    SendInfo('S', &peripheral[0], value, *value,
NUM_DIGITS_ADDRESS_RAM, 3);
                }
            }
        }
    }
}
//---------------------------------------------------------------------

void SendInfo(char typeInfo, char* peripheral, int16 value1, int16
value2, int8 numDigitsValue1, int8 numDigitsValue2)
{
    int8 counter, posWordTX;
    char wordTX[SIZE_MAX_INFO];
    char dataChar[SIZE_MAX_VALUE];
    wordTX[0] = 'I';
    wordTX[1] = 'N';
    wordTX[2] = 'F';
    wordTX[3] = ':';
    wordTX[4] = typeInfo;
    wordTX[5] = ' ';
    posWordTX = 6;
    for(counter = 0; peripheral[counter]; counter++)
    {
        wordTX[posWordTX] = peripheral[counter];
        posWordTX++;
    }
    wordTX[posWordTX] = ' ';
    posWordTX++;
    IntToString(value1, &dataChar[0], numDigitsValue1);    //converte de
inteiro para string
    for(counter = 0; dataChar[counter]; counter++)
    {
        wordTX[posWordTX] = dataChar[counter];
        posWordTX++;
    }
    if((numDigitsValue1 > 0)&&(numDigitsValue2 > 0))
    {
        wordTX[posWordTX] = ' ';
        posWordTX++;
    }
    IntToString(value2, &dataChar[0], numDigitsValue2);    //converte de
```

```
inteiro para string
    for(counter = 0; dataChar[counter]; counter++)
    {
        wordTX[posWordTX] = dataChar[counter];
        posWordTX++;
    }
    //fim da concatena ão
    //finalizando comando:
    wordTX[posWordTX] = '\r';
    wordTX[posWordTX+1] = '\n';
    wordTX[posWordTX+2] = '\0';
    SerialTransmit(&wordTX[0]); //envia status da porta pela serial
}
//---------------------------------------------------------------------

void CmdWrtPort(char port, unsigned int8 value)
{
    char peripheral[3];
    boolean sucess = true;
    switch (port)
    {
        case 'A':
            if (value <= MAX_VALUE_PORT_A)
            {
                PortA = value;
                ra = PortA;
            }
            break;
        case 'B':
            if (value <= MAX_VALUE_PORT_B)
            {
                PortB = value;
                rb = PortB;
            }
            break;
        case 'C':
            if (value <= MAX_VALUE_PORT_C)
            {
                PortC = value;
                rc = PortC;
            }
            break;
        case 'D':
            if (value <= MAX_VALUE_PORT_D)
            {
                PortD = value;
                rd = PortD;
            }
            break;
        case 'E':
            if (value <= MAX_VALUE_PORT_E)
            {
                PortE = value;
                re = PortE;
            }
            break;
        default:
            sucess = false;
            break;
    }
```

```
    if (sucess)
    {
        peripheral[0] = 'R'; peripheral[1] = port; peripheral[2] = '\0';
        SendInfo('S', &peripheral[0], value, 0, NUM_DIGITS_PORTS, 0);
    }
}
//---------------------------------------------------------------------

void CmdWrtPin(char port, char channelPort, int8 value)
{
    char peripheral[3];
    int8 channel;
    channel = channelPort - 48;
    if (channel >= 0)
    {
        boolean sucess = true;
        switch (port)
        {
            case 'A':
                if (channel < NUM_PINS_PORT_A)
                {
                    if (value == 0)
                        bit_clear(PortA, channel);
                    else
                        bit_set(PortA, channel);
                    value = PortA;
                    ra = PortA;
                }
                break;
            case 'B':
                if (channel < NUM_PINS_PORT_B)
                {
                    if (value == 0)
                        bit_clear(PortB, channel);
                    else
                        bit_set(PortB, channel);
                    value = PortB;
                    rb = PortB;
                }
                break;
            case 'C':
                if (channel < NUM_PINS_PORT_C)
                {
                    if (value == 0)
                        bit_clear(PortC, channel);
                    else
                        bit_set(PortC, channel);
                    value = PortC;
                    rc = PortC;
                }
                break;
            case 'D':
                if (channel < NUM_PINS_PORT_D)
                {
                    if (value == 0)
                        bit_clear(PortD, channel);
                    else
                        bit_set(PortD, channel);
                    value = PortD;
                    rd = PortD;
```

```
                }
                break;
            case 'E':
                if (channel < NUM_PINS_PORT_E)
                {
                    if (value == 0)
                        bit_clear(PortE, channel);
                    else
                        bit_set(PortE, channel);
                    value = PortE;
                    re = PortE;
                }
                break;
            default:
                sucess = false;
                break;
        }//fim switch
        if (sucess)
        {
            peripheral[0] = 'R'; peripheral[1] = port; peripheral[2] =
'\0';
            SendInfo('S', &peripheral[0], value, 0, NUM_DIGITS_PORTS, 0);
        }
    }
}
//---------------------------------------------------------------------

void CmdWrtTris(char port, unsigned int8 value)
{
    char peripheral[3];
    boolean sucess = true;
    switch (port)
    {
        case 'A':
            if (value <= MAX_VALUE_PORT_A)
            {
                TrisA = value;
                ta = TrisA;
            }
            break;
        case 'B':
            if (value <= MAX_VALUE_PORT_B)
            {
                TrisB = value;
                tb = TrisB;
            }
            break;
        case 'C':
            if (value <= MAX_VALUE_PORT_C)
            {
                TrisC = value;
                tc = TrisC;
            }
            break;
        case 'D':
            if (value <= MAX_VALUE_PORT_D)
            {
                TrisD = value;
                td = TrisD;
            }
```

```
            break;
        case 'E':
            if (value <= MAX_VALUE_PORT_E)
            {
                TrisE = value;
                te = TrisE;
            }
            break;
        default:
            sucess = false;
            break;
    }
    if (sucess)
    {
        peripheral[0] = 'T'; peripheral[1] = port; peripheral[2] = '\0';
        SendInfo('S', &peripheral[0], value, 0, NUM_DIGITS_PORTS, 0);
    }
}
//---------------------------------------------------------------------

void CmdWrtTrisPin(char port, char channelPort, int8 value)
{
    char peripheral[3];
    int8 channel;
    channel = channelPort - 48;
    if (channel >= 0)
    {
        boolean sucess = true;
        switch (port)
        {
            case 'A':
                if (channel < NUM_PINS_PORT_A)
                {
                    if (value == 0)
                        bit_clear(TrisA, channel);
                    else
                        bit_set(TrisA, channel);
                    value = TrisA;
                    ta = TrisA;
                }
                break;
            case 'B':
                if (channel < NUM_PINS_PORT_B)
                {
                    if (value == 0)
                        bit_clear(TrisB, channel);
                    else
                        bit_set(TrisB, channel);
                    value = TrisB;
                    tb = TrisB;
                }
                break;
            case 'C':
                if (channel < NUM_PINS_PORT_C)
                {
                    if (value == 0)
                        bit_clear(TrisC, channel);
                    else
                        bit_set(TrisC, channel);
                    value = TrisC;
```

```
                    tc = TrisC;
                }
                break;
            case 'D':
                if (channel < NUM_PINS_PORT_D)
                {
                    if (value == 0)
                        bit_clear(TrisD, channel);
                    else
                        bit_set(TrisD, channel);
                    value = TrisD;
                    td = TrisD;
                }
                break;
            case 'E':
                if (channel < NUM_PINS_PORT_E)
                {
                    if (value == 0)
                        bit_clear(TrisE, channel);
                    else
                        bit_set(TrisE, channel);
                    value = TrisE;
                    te = TrisE;
                }
                break;
            default:
                sucess = false;
                break;
        }//fim switch
        if (sucess)
        {
            peripheral[0] = 'T'; peripheral[1] = port; peripheral[2] =
'\0';
            SendInfo('S', &peripheral[0], value, 0, NUM_DIGITS_PORTS, 0);
        }
    }
}
//---------------------------------------------------------------------

void CmdReadPort(char port)
{
    char peripheral[3];
    int8 value;
    boolean sucess;
    sucess = true;
    switch(port)
    {
        case 'A':
            value = PortA;
            break;
        case 'B':
            value = PortB;
            break;
        case 'C':
            value = PortC;
            break;
        case 'D':
            value = PortD;
            break;
        case 'E':
```

```
            value = PortE;
            break;
        default:
            sucess = false;
            break;
    }
    if (sucess)
    {
        peripheral[0] = 'R'; peripheral[1] = port; peripheral[2] = '\0';
        SendInfo('S', &peripheral[0], value, 0, NUM_DIGITS_PORTS, 0);
    }
}
//---------------------------------------------------------------------

void CmdReadTris(char port)
{
    char peripheral[3];
    int8 value;
    boolean sucess;
    sucess = true;
    switch (port)
    {
        case 'A':
            value = TrisA;
            break;
        case 'B':
            value = TrisB;
            break;
        case 'C':
            value = TrisC;
            break;
        case 'D':
            value = TrisD;
            break;
        case 'E':
            value = TrisE;
            break;
        default:
            sucess = false;
            break;
    }
    if (sucess)
    {
        peripheral[0] = 'T'; peripheral[1] = port; peripheral[2] = '\0';
        SendInfo('S', &peripheral[0], value, 0, NUM_DIGITS_PORTS, 0);
    }
}
//---------------------------------------------------------------------

void IntToString(int16 data, char* dataChar, int8 numDigits)
{
    int8 i;
    for (i = 0; i < numDigits; i++)
        dataChar[i] = 48;
    dataChar[numDigits] = '\0';
    if (data > 0)
    {
        int8 counter, counterBuffer;
        char buffer[SIZE_MAX_VALUE];
        counter = 0;
```

```
        //vai armazenando a string em ordem inversa dentro de buffer
        while (data > 0)
        {
            buffer[counter] = (data % 10)+48;
            data = data / 10;
            counter++;
        }
        buffer[counter] = '\0';
        counterBuffer = numDigits - 1;
        //transfere string de buffer para dataChar com ordem correta
        for (counter = 0; buffer[counter]; counter++)
        {
            dataChar[counterBuffer] = buffer[counter];
            counterBuffer--;
        }
    }
}
//---------------------------------------------------------------------

unsigned int16 StringToInt16(char* valueChar)
{
    unsigned int8 pos, count, digit, num, sizeValueChar;
    unsigned int16 value, factor;
    sizeValueChar = 0;
    while(valueChar[sizeValueChar])
        sizeValueChar++;
    factor = 1;
    pos = sizeValueChar - 1;    //pos = posição do último dígito de
valueChar
    value = valueChar[pos] - 48;    //pegando o último dígito
    for(count = 0; count < 3, pos > 0; count++)  //for é executado no
máximo 3 vezes para não gerar overflow de factor
    {
        pos--;
        factor = factor * 10;
        digit = valueChar[pos] - 48;
        value = value + (digit * factor);
    }

    while(pos > 0)
    {
        pos--;
        //Multiplicação de factor por 10
        num = factor + factor;
        factor = num + num;
        factor = factor + factor;
        factor = factor + num;
        //Multiplicação do dígito a ser convertido por factor
        digit = valueChar[pos] - 48;  //armazenando temporariamente o
dígito
        num = 0;    //vai armazenar o resultado da multiplicação
        for (count = 0; count < digit; count++)
            num = num + factor;
        //Soma ao valor final da conversão
        value = value + num;
    }
    return value;
}
//---------------------------------------------------------------------
```

ANEXO II – CÓDIGO FONTE DO SERVIDOR WEB

Arquivo “websrv.c”

```
// v1.28 Servidor WAP e Porta HTTP
// v1.24 Time-out do Servidor TCP/IP foi retirado
// V1.23 linha 724, antes smStateEMAIL = SM_LISTEN_WAIT1;
//                  agora smState = SM_LISTEN_WAIT1;

// o teclado só funciona com o cabo do ICD desconectado, pois o mesmo
// usa RB6 e RB7. Não funciona também no modo de depuração. A define
// para usar o teclado chama-se "TECLADO".

// Informações sobre o correio eletronico
// IP do Servidor SMTP: 10.0.0.102
// MAC do Servidor SMTP: 0x00. 0x11. 0x09. 0x06. 0x80. 0xa1;
// via home-page quando navegador Web solicita enviar correio eletronico
// SMTP_Evento=1 e em SMTP_Init foi inicializado com zero
// em SMTP-Verificar(), qdo SMTP_Evento==1 e SMTP_enviando==0 implica em
SMTP_Evento=0 e SMTP_enviando==1
// e se SMTP_enviando==1 fica enviando email pela maquina de estado
SMTP_Envia()
// e se SMTP_enviando==0 zera time-out do correio eletronico

// Informações sobre a conexão TCPIP
// Porta: 49152
// Na inicialização indica que não deve transmitir dados para estação
central via conexão TCP
// e estado inicial da máquina de estado TCPIP
// Antes do loop principal reseta timeout
// No loop principal passa pela máquina de estado TCPIP
// O programa Cliente TCPIP do PC deve ser configurado com IP:10.0.0.101
e Porta 49152

// Informações sobre o Agent SNMP:
// Para ativar use SNMP em Build Options
//

/*********************************************************************
*
*       Example Web Server Application using Microchip TCP/IP Stack
*
*********************************************************************
* FileName:        WebSrvr.c
* Dependencies:    string.H
*                  usart.h
*                  StackTsk.h
*                  Tick.h
*                  http.h
*                  MPFS.h
* Processor:       PIC18
* Complier:        MCC18 v1.00.50 or higher
*                  HITECH PICC-18 V8.10PL1 or higher
* Company:         Microchip Technology, Inc.
*
* Software License Agreement
*
* The software supplied herewith by Microchip Technology Incorporated
* (the Company) for its PICmicro® Microcontroller is intended and
* supplied to you, the Companys customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
```

```
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are reserved.
* Any use in violation of the foregoing restrictions may subject the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN AS IS CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
*
* HiTech PICC18 Compiler Options excluding device selection:
*                   -FAKELOCAL -G -O -Zg -E -C
*
*
*
*
* Author               Date         Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Nilesh Rajbharti     4/19/01      Original (Rev. 1.0)
* Nilesh Rajbharti     2/09/02      Cleanup
* Nilesh Rajbharti     5/22/02      Rev 2.0 (See version.log for detail)
* Nilesh Rajbharti     7/9/02       Rev 2.1 (See version.log for detail)
* Nilesh Rajbharti     4/7/03       Rev 2.11.01 (See version log for
detail)
 ********************************************************************/

/*
 * Following define uniquely deines this file as main
 * entry/application In whole project, there should only be one such
 * definition and application file must define AppConfig variable as
 * described below.
 */

#define THIS_IS_STACK_APPLICATION
#define STACK_INCLUDE

#if defined(TEMPERATURA)
      #include "TC74.h"
      #include "I2C.h"
#endif

#include <string.h>
//#include "celular.h"
#include "serial.h"
#include "mac.h"
#include "tcp.h"
#include "ip.h"
#include "udp.h"
#include "stacktsk.h"
#include "tick.h"
#include "icmp.h"
#include "delay.h"
#include "mpfs.h"
#include "xlcd.h"
```

```
#if defined(SNMP)
      #include "snmp.h"
      #include "mib.h"
#endif

#if defined(STACK_USE_DHCP)
      ROM char DHCPMsg[] = "DHCP/Gleaning...";
      #include "dhcp.h"
#endif

#if defined(STACK_USE_ICMP)
    #include "icmp.h"
#endif

#if defined(STACK_USE_HTTP_SERVER)
      #include "http.h"
#endif

#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
      #include "ftp.h"
#endif

// Errata PIC18F8722: The RE4 pin latch remains at a logic level zero
when the ECCPMX Configuration bit is clear and
//selects PORTH.  Work around; This issue will be corrected in a future
revision of silicon.

#if defined(MCHP_C18) && defined(__18F8722)

      #pragma romdata CONFIG1L = 0x300000
      const rom unsigned char config1L = 0b00000000;      // não
implementado

      // IESO FCMEN X X FOSC2 FOSC1 FOSC0
      #pragma romdata CONFIG1H = 0x300001
      const rom unsigned char config1H = 0b10000010;

      // X X X BORV1 BORV0 BOREN1 BOREN0 PWRTEN
      #pragma romdata CONFIG2L = 0x300002
      const rom unsigned char config2L = 0b00000000;

      // X X X WDTPS3 WDTPS2 WDTPS1 WDTPS WDTEN
      #pragma romdata CONFIG2H = 0x300003
      const rom unsigned char config2H = 0b00011110;

      // WAIT BW ABW1 ABW0 X X PM1 PM0
      #pragma romdata CONFIG3L = 0x300004
      const rom unsigned char config3L = 0b00000011;

      // MCLRE X X X X LPT1OSC ECCPMX CCP2MX
      #pragma romdata CONFIG3H = 0x300005
      const rom unsigned char config3H = 0b10000011;

      // DEBUG XINST BBSIZ1 BBSIZ0 X LPV X STVREN
      #pragma romdata CONFIG4L = 0x300006
      const rom unsigned char config4L = 0b10000001;

      #pragma romdata CONFIG4H = 0x300007
      const rom unsigned char config4H = 0b00000000;
```

```
      // CP7 CP6 CP5 CP4 CP3 CP2 CP1 CP0
      #pragma romdata CONFIG5L = 0x300008
      const rom unsigned char config5L = 0b11111111;

      // CPD CPB X X X X X X
      #pragma romdata CONFIG5H = 0x300009
      const rom unsigned char config5H = 0b11000000;

      //WRT7 WRT6 WRT5 WRT4 WRT3 WRT2 WRT1 WRT0
      #pragma romdata CONFIG6L = 0x30000A
      const rom unsigned char config6L = 0b11111111;

      // WRTD WRTB WRTC X X X X X
      #pragma romdata CONFIG6H = 0x30000B
      const rom unsigned char config6H = 0b11100000;

      // EBTR7 EBTR6 EBTR5 EBTR4 EBTR3 EBTR2 EBTR1 EBTR0
      #pragma romdata CONFIG7L = 0x30000C
      const rom unsigned char config7L = 0b11111111;

      // X EBTRB X X X X X X
      #pragma romdata CONFIG7H = 0x30000D
      const rom unsigned char config7H = 0b01000000;

      #pragma romdata
#endif

#if defined(MCHP_C18) && defined(__18F8720)

      #pragma romdata CONFIG1L = 0x300000
      const rom unsigned char config1L = 0b00000000;      // não
implementado

      // X X OSCEN X X FOSC2 FOSC1 FOSC0
      #pragma romdata CONFIG1H = 0x300001
      const rom unsigned char config1H = 0b00100010;

      // X X X X BORV1 BORV0 BOREN PWRTEN
      #pragma romdata CONFIG2L = 0x300002
      const rom unsigned char config2L = 0b00000100;

      // X X X X WDTPS2 WDTPS1 WDTPS WDTEN
      #pragma romdata CONFIG2H = 0x300003
      const rom unsigned char config2H = 0b00000000;

      // WAIT X X X X X PM1 PM0
      #pragma romdata CONFIG3L = 0x300004
      const rom unsigned char config3L = 0b10000011;

      // X X X X X X X CCP2MX
      #pragma romdata CONFIG3H = 0x300005
      const rom unsigned char config3H = 0b00000001;

      // DEBUG X X X X LPV X STVREN
      #pragma romdata CONFIG4L = 0x300006
      const rom unsigned char config4L = 0b10000001;

      #pragma romdata CONFIG4H = 0x300007
      const rom unsigned char config4H = 0b00000000;
```

```
      // CP7 CP6 CP5 CP4 CP3 CP2 CP1 CP0
      #pragma romdata CONFIG5L = 0x300008
      const rom unsigned char config5L = 0b11111111;

      // CPD CPB X X X X X X
      #pragma romdata CONFIG5H = 0x300009
      const rom unsigned char config5H = 0b11000000;

      //WRT7 WRT6 WRT5 WRT4 WRT3 WRT2 WRT1 WRT0
      #pragma romdata CONFIG6L = 0x30000A
      const rom unsigned char config6L = 0b11111111;

      // WRTD WRTB WRTC X X X X X
      #pragma romdata CONFIG6H = 0x30000B
      const rom unsigned char config6H = 0b11100000;

      // EBTR7 EBTR6 EBTR5 EBTR4 EBTR3 EBTR2 EBTR1 EBTR0
      #pragma romdata CONFIG7L = 0x30000C
      const rom unsigned char config7L = 0b11111111;

      // X EBTRB X X X X X X
      #pragma romdata CONFIG7H = 0x30000D
      const rom unsigned char config7H = 0b01000000;

      #pragma romdata
#endif

#if defined(MCHP_C18) && defined(__18F8621)

      #pragma romdata CONFIG1L = 0x300000
      const rom unsigned char config1L = 0b00000000;      // não
implementado

      // X X OSCEN X FOSC3 FOSC2 FOSC1 FOSC0
      #pragma romdata CONFIG1H = 0x300001
      const rom unsigned char config1H = 0b00100010;

      // X X X X BORV1 BORV0 BOREN PWRTEN
      #pragma romdata CONFIG2L = 0x300002
      const rom unsigned char config2L = 0b00000100;

      // X X X WDTPS3 WDTPS2 WDTPS1 WDTPS WDTEN
      #pragma romdata CONFIG2H = 0x300003
      const rom unsigned char config2H = 0b00000000;

      // WAIT X X X X X PM1 PM0
      #pragma romdata CONFIG3L = 0x300004
      const rom unsigned char config3L = 0b10000011;

      // MCLRE X X X X X ECCPMX CCP2MX
      #pragma romdata CONFIG3H = 0x300005
      const rom unsigned char config3H = 0b10000001;

      // DEBUG X X X X LPV X STVREN
      #pragma romdata CONFIG4L = 0x300006
      const rom unsigned char config4L = 0b10000001;

      #pragma romdata CONFIG4H = 0x300007
      const rom unsigned char config4H = 0b00000000;
```

```
      // X X X X CP3 CP2 CP1 CP0
      #pragma romdata CONFIG5L = 0x300008
      const rom unsigned char config5L = 0b11111111;

      // CPD CPB X X X X X X
      #pragma romdata CONFIG5H = 0x300009
      const rom unsigned char config5H = 0b11000000;

      //X X X X WRT3 WRT2 WRT1 WRT0
      #pragma romdata CONFIG6L = 0x30000A
      const rom unsigned char config6L = 0b11111111;

      // WRTD WRTB WRTC X X X X X
      #pragma romdata CONFIG6H = 0x30000B
      const rom unsigned char config6H = 0b11100000;

      // EBTR7 EBTR6 EBTR5 EBTR4 EBTR3 EBTR2 EBTR1 EBTR0
      #pragma romdata CONFIG7L = 0x30000C
      const rom unsigned char config7L = 0b11111111;

      // X EBTRB X X X X X X
      #pragma romdata CONFIG7H = 0x30000D
      const rom unsigned char config7H = 0b01000000;

      #pragma romdata
#endif

#define DEMO_SNMP_MSG   "DemoSNMP 1.0"
ROM char StartupMsg[] = DEMO_SNMP_MSG;

ROM char SetupMsg[] = "Board Setup...";

#if defined(SNMP)
      // These are hardware I/O labels to ease access.
      #define SW_S1               PORTJ_RJ7
      #define LED_D2_CONTROL      LATA2
//    #define LED_D6_CONTROL      LATA3
      #define LCD_DISPLAY_LEN     (16)

      // Trap information.
      // This table maintains list of intereseted receivers
      // who should receive notifications when some interesting
      // event occurs.

      #define TRAP_TABLE_SIZE         (2)
      #define MAX_COMMUNITY_LEN       (8)

      typedef struct _TRAP_INFO
      {
      BYTE Size;
      struct
      {
            BYTE communityLen;
            char community[MAX_COMMUNITY_LEN];
            IP_ADDR IPAddress;
            struct
            {
                  unsigned int bEnabled : 1;
            } Flags;
      } table[TRAP_TABLE_SIZE];
```

```
      } TRAP_INFO;

      // Initialize trap table with no entries.

      TRAP_INFO trapInfo = { TRAP_TABLE_SIZE };

#endif

//                          1234567890123456
ROM char blankLCDLine[] = "                ";

// This is used by other stack elements.
// Main application must define this and initialize it with
// proper values.

APP_CONFIG AppConfig;

unsigned char myDHCPBindCount = 0;

#if defined(STACK_USE_DHCP)
    extern BYTE DHCPBindCount;
#else
    /*
     * If DHCP is not enabled, force DHCP update.
     */
    BYTE DHCPBindCount = 1;
#endif

//#define STARTUP_MSG "MpSta
ck 2.20.04"

//ROM char StartupMsg[] = STARTUP_MSG;
#if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
      ROM char DHCPMsg[] = "DHCP/Gleaning...";
#endif

/*
 * Isto é usado por outros elementos da pilha.
 * Main application must define this and initialize it with
 * proper values.
 */
APP_CONFIG AppConfig;



//---------------------------------------------------------------------------
-------
// Variáveis das Aplicações
//---------------------------------------------------------------------------
-------


/*********************************************************************
 * Minhas definições
 ********************************************************************/
#define MAX_SIZE_COMMAND 23 //tamanho maximo de um comando a ser enviado
pela serial
#define MAX_SIZE_VALUES 5
#define SIZE_OPER 6
#define SIZE_PERIPHERAL 5
```

```
#define SIZE_VALUE1 5
#define SIZE_VALUE2 5
#define SIZE_FIFO_EEPROM 5
#define SIZE_FIFO_RAM 5

char* pTemp;
static char temp2[MAX_SIZE_VALUES];
static const char ON[3] = "ON";
static const char OFF[4] = "OFF";
BOOL monitorOn = FALSE;

unsigned char RA = 0;  //010101
unsigned char RB = 0;   //00000000
unsigned char RC = 0; //11111111
unsigned char RD = 0; //10101010
unsigned char RE = 0;   //111
unsigned char TA = 0;
unsigned char TB = 0;
unsigned char TC = 0;
unsigned char TD = 0;
unsigned char TE = 0;
unsigned char statusADC = 0;
unsigned char statusTimer = 0;
unsigned char statusPWM = 0;
unsigned char statusIntExt = 0;
BOOL statusCAN = 0;
BOOL statusI2C = 0;
BOOL statusSPI = 0;
BOOL statusRS232 = 0;
unsigned char valueADC0 = 0;
unsigned char valueADC1 = 0;
unsigned char valueADC2 = 0;
unsigned char valueADC3 = 0;
unsigned char valueADC4 = 0;
unsigned char valueADC5 = 0;
unsigned char valueADC6 = 0;
unsigned char valueADC7 = 0;
unsigned char valueTimer0 = 0;
unsigned char valueTimer1 = 0;
unsigned char valuePWM = 0;
unsigned char addressEE[SIZE_FIFO_EEPROM] = {0, 0, 0, 0, 0};
unsigned char valueEE[SIZE_FIFO_EEPROM] = {0, 0, 0, 0, 0};
unsigned char addressRAM[SIZE_FIFO_RAM] = {0, 0, 0, 0, 0};
unsigned char valueRAM[SIZE_FIFO_RAM] = {0, 0, 0, 0, 0};

static unsigned char rec[20];                                          //
buffer de mensagem recebida serial - interrupção
static unsigned char USARTString_rec[20];                        //
buffer de mensagem recebida serial para mostrar no navegador
static unsigned char *p;
      // ponteiro para mensagem

void ReadCommand(char*);
void InterpretCommand(char*, char*, char*, char*);
void UpdPin(char, unsigned char);
void UpdTris(char, unsigned char);
void UpdEEPROM(unsigned char, unsigned char);
void UpdRAM(unsigned char, unsigned char);
void UpdADC(unsigned char, unsigned char);
void UpdTimer(unsigned char, unsigned char);
```

```
void UpdPWM(unsigned char);
void CommandSplit(char*, char*, char*, char*, char*);
void Concat(char*, char*, char*, char*, char*);
void IntToString(unsigned char, char*, unsigned char);
unsigned char Exp(unsigned char, unsigned char);

/*********************************************************************/


#if defined(TEMPERATURA)
      unsigned char temperatura;
#endif

static WORD_VAL AN0Value;
#if defined(SNMP)
      static WORD_VAL AN1Value;
      char LCDDisplayString[LCD_DISPLAY_LEN] = DEMO_SNMP_MSG;
      BYTE LCDDisplayStringLen = sizeof(DEMO_SNMP_MSG)-1;
#endif

// Porta HTTP
unsigned int porta_http; //2EI
unsigned char var_wml_cgi; //2EI

// Porta TCP/IP
#if defined (TCPIP)
static unsigned char Evento_tcp_ip;                                    //
Evento TCP_IP via navegador WEB
static unsigned char smState;                                          //
Estado Máquina de Estado relacionado aos protocolos TCP/IP
static TICK TCPIP_Time_Out;
      // Timeout para conexão TCP/IP
TCP_SOCKET TCPIPSocket;
      // Soquete para enviar dados para estação central
#endif

// EMAIL
#if defined (EMAIL)
      static TICK EMAIL_Time_Out;                                      //
Timeout para recebimento de dados serial
      unsigned char mensagem_email[35];                          //
buffer de recepção EMAIL
#endif

#if defined (EMAIL)
// Endereço IP do servidor de correio eletronico
static unsigned char IpE1;
static unsigned char IpE2;
static unsigned char IpE3;
static unsigned char IpE4;
// Endereço Mac do servidor de correio eletronico
static unsigned char Mac1;
static unsigned char Mac2;
static unsigned char Mac3;
static unsigned char Mac4;
static unsigned char Mac5;
static unsigned char Mac6;
static unsigned char SMTP_Evento;
static unsigned char SMTP_enviando;                        // Status de
envio do correio eletronico
```

```
unsigned char smStateEMAIL;                                          //
Estado Máquina de Estado relacionado ao e-mail
NODE_INFO RemoteNode;                                                //
Informaçõe do servidor de correio remoto
TCP_SOCKET EMAILSocket;                                              //
Soquete para enviar correio eletronico
#endif

#if defined (TECLADO)
      static unsigned char tecla;
      static unsigned char status_tecla;
      static TICK debounce1;
#endif


/*
 * Set configuration fuses for HITECH compiler.
 * For MCC18 compiler, separately linked config.asm file
 * will set the correct fuses.
 */
#if defined(HITECH_C18) && defined(__18F8621)
__CONFIG(1, UNPROTECT & HS);
__CONFIG(2, PWRTEN & BORDIS & WDTDIS);
#endif



//------------------------------------------------------------------------
-------
// Protótipo das Funções (Function Prototypes)
//------------------------------------------------------------------------
-------
#if defined (SNMP)
      static BOOL SendNotification(BYTE receiverIndex, SNMP_ID var,
SNMP_VAL val);
#endif

static void InitAppConfig(void);
static void InitializeBoard(void);
static void ProcessIO(void);
static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD);
void NotifyRemoteUser(void);
void SerialISR(void);
      // Tratamento da interrupção serial
void HighISR(void);
static void SetConfig(void);                                          //
Tratamento da interrupção do timer

#if defined (TECLADO)
void Teclado_Verifica(void);
#endif

#if defined (EMAIL)
void SMTP_Envia(void);
void SMTP_Verificar(void);
void SMTP_Init(void);
      // Inicialização do correio eletrônico
#endif

#if defined (TCPIP)
```

```
void TCPIP_Init(void);
      // Inicialização das variáveis conexão TCP/IP porta 49152
#endif

//*********************************************************************
*******
// Vetores de Interupção (Interrupt Vectors)
// 0x08 é o vetor de alta prioridade e 0x18 é o vetor de baixa prioridade
//*********************************************************************
*******

#if defined(MCHP_C18)
#pragma code highVector=0x08
void HighVector (void)
{
    _asm
      goto HighISR
      _endasm
}
#pragma code /* return to default code section */
#endif

#if defined(MCHP_C18)
#pragma code lowVector=0x18
void LowVector (void)
{
    _asm
      goto SerialISR
      _endasm
}
#pragma code /* return to default code section */
#endif


//*********************************************************************
*******
// Interrupt Service Routine
//*********************************************************************
*******

// NOTE: Several PICs, including the PIC18F4620 revision A3 have a RETFIE
FAST/MOVFF bug
// The interruptlow keyword is used to work around the bug when using C18
// To work around the bug on PICC-18, configure the compiler to Compile
for ICD

// We buffer the received packet in the interrupt handler, which requires
16-bit
// and 32-bit math.  Therefore, we must save the temporary registers and
the PROD register.
#if defined(MCHP_C18)
//    #pragma interruptlow HighISR
save=section(".tmpdata"),section("MATH_DATA"),PROD
    #pragma interruptlow HighISR
    void HighISR(void)
#elif defined(HITECH_C18)
    void interrupt HighISR(void)
#else
    void HighISR(void)
#endif
```

```
{
    TickUpdate();

#if defined(STACK_USE_SLIP)
    MACISR();
#endif
}



#define USARTIsGetReady()   (PIR1_RCIF)
#define USARTGet()          (RCREG)

#if defined (TCPIP)
// Usadas na Conexão TCP/IP com Cliente PC
#define SM_CONNECT_WAIT      0
#define SM_CONNECTED1        1
#define SM_CONNECTED2        2
#define SM_LISTEN            3
#define SM_LISTEN_WAIT1      4
#define SM_LISTEN_WAIT2      5
#endif

// PONTO DE ENTRADA PRINCIPAL * Main entry point.
void main(void)
{


#if defined (EMAIL)
      unsigned char buffer_rec[10];                                  //
buffer de recepção EMAIL
      unsigned char vv;                                                    //
uso no EMAIL
      unsigned char mensagem_email[35];                              //
buffer de recepção EMAIL
      ROM char* str;
      // ponteiro para receber mensagem da Flash, uso no EMAIL
#endif

static TICK t = 0;
static TICK t_cel = 0;
unsigned char a;

// definição da porta do servidor HTTP
porta_http = 80;

#if defined (TECLADO)
      tecla = 0x0;
      status_tecla=' ';
#endif

#if defined (EMAIL)
      SMTP_Init();
            // Inicialização das variáveis do protocolo SMTP
#endif

#if defined (TCPIP)
            // Inicializção das variáveis mensagens TCP/IP
      TCPIP_Init();
#endif
```

```
    // Inicializa qualquer aplicação específica de hardware
    InitializeBoard();

#if defined(TEMPERATURA)
      #if defined(__18F8720) || defined(__18F8621)
      I2C_Init(0x3F);                                          // Setup
I2C 100KHz para PIC18F8720 e 8621
      #endif
      #if defined (__18F8722)
      I2C_Init(0x73);                                          // Setup
I2C 100KHz para PIC18F8722
      #endif
      TC74_Init();                                             //
Inicializa sensor de temperatura
#endif

      // Inicializa todos os componentes relacionados ao stack
      // Os seguintes passos devem ser executados para todas as
aplicações
      // que usam pilha TCP/IP
    TickInit();

    MPFSInit();

      //Inicializa Stack e variaveis NV relacionadas as aplicações
    InitAppConfig();

      StackInit();

      #if defined(SNMP)
      SNMPInit();
      #endif

#if defined(STACK_USE_HTTP_SERVER)
    HTTPInit();
#endif

#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
    FTPInit();
#endif

#if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
   if ( AppConfig.Flags.bIsDHCPEnabled ) {
   #if defined (USE_LCD)
            XLCDGoto(1, 0);
            XLCDPutROMString(DHCPMsg);
   #endif
   }
   else {
/*
* Force IP address display update.
*/
      myDHCPBindCount = 1;
      #if defined(STACK_USE_DHCP)
         DHCPDisable();
      #endif
   }
#endif
```

```
      p = &rec[0];
                  // p aponta para início do endereço da mensagem a ser
ecebida
      *p = '\0';
                  // Termina mensagem com caracter null

      USARTString_rec[0]  = '\0';

      // Limpando buffer serial
      RCSTA_CREN=0x1;
                  // Habilita recepção, limpando FLAG de ERRO
      a=USARTGet();

      // Habilita interrupçaõ serial
      // RCIE: EUSART Receive Interrupt Enable bit
      // 1 = Enables the EUSART receive interrupt
      // 0 = Disables de EUSART receive interrupt
      // Habilita interrupção de recepção serial
      PIE1_RCIE = 1;

#if defined (TECLADO)
      // Habilita RB Port Change Interrupt Enable bit
      // INTCON bit 3 RBIE: RB Port Change Interrupt Enable bit
      // 1 = Enables the RB port change interrupt
      // 0 = disables the RB port change interrupt
      INTCONbits.RBIE = 1;

#endif

      // Uma vez que todos os itens foram inicializados, vai para um loop
infinito aonde as tarefas são executadas
      // Se uma aplicação necessita executar sua própria tarefa, ela
deverá ser colocada no final do loop
      // Note que estamos trabalhando com um kernel cooperativo de modo
que uma tarefa tomará o tempo necessário da CPU para sua execução
      // Se uma tarefa necesita de um tempo de execução maior para
realizar o seu trabalho
      // ela deve ser quebrada em tarefas menores de modo que outras
tarefas tenham acesso a CPU

      // Piscadela especial no Led só para verificar inicio do programa
      for (a=0; a< 5; a++) {
        LATA4 ^= 1;
        t = TickGet();
      while ( TickGetDiff(TickGet(), t) <= TICK_SECOND/8 );
      }

      #if defined(TCP_IP)
      TCPIP_Time_Out = TickGet();
      // Resetar time-out da conexão TCPIP
      #endif

    while(1) {

      #if defined (TECLADO)
      if ( TickGetDiff(TickGet(), debounce1) <= TICK_SECOND/4) {
            }
            else{
                  a = PORTB;
                  INTCONbits.RBIE = 1;
```

```
      // volta a habilitar interrupção porta B
            }
      #endif

        // Pisca LED a cada segundo
        if ( TickGetDiff(TickGet(), t) >= TICK_SECOND/2 ){
            t = TickGet();
            LATA4 ^= 1;
        }

            //Verifica se há mensagem de celular a cada 15 segundos
            #if defined(CELULAR_SMS)
            if ( TickGetDiff(TickGet(), t_cel) >= (15*TICK_SECOND) ){
                  t_cel = TickGet();
                        ler_mensagem_celular();
            }
            #endif

#if defined(TCPIP)
/*********************************************************************
***************
* COMUNICANDO COM O CLIENTE TCP/IP
*********************************************************************
**************/
            // Criação do soquete que envia dados para a estação remota
            if (smState==SM_LISTEN){
                  TCPIPSocket = TCPListen(49152);
                  if ( TCPIPSocket == INVALID_SOCKET ){
                        // Socket nâo está disponível
                  }
                  else{
                        smState = SM_LISTEN_WAIT1;
                  }
            }

            if (Evento_tcp_ip == 1) {

                  //if ( ( TickGetDiff(TickGet(),TCPIP_Time_Out ) >=
30*TICK_SECOND)  ){
                        //TCPDisconnect(TCPIPSocket);
                        //smState = SM_LISTEN_WAIT1;
                        //TCPIP_Time_Out = TickGet();
      // Resetar time-out do correio eletronico
                 //}

                  switch (smState){

                        case SM_LISTEN_WAIT1:
                              // Espera pela conexão
                              if ( TCPIsConnected(TCPIPSocket) ){
                                    smState = SM_CONNECTED1;
                              }
                        break;


                        case SM_CONNECTED1:

                              if ( TCPIsConnected(TCPIPSocket) ){
                                          if (TCPIsPutReady
(TCPIPSocket)){         // Soquete está livre para transmitir
```

```
                                        TCPPut(TCPIPSocket,'O');
                // envia mensagem
                                        TCPPut(TCPIPSocket,'l');

                                        TCPPut(TCPIPSocket,'a');

                                        TCPPut(TCPIPSocket, 0x0);
                // null
                                        TCPFlush(TCPIPSocket);
                // Transmite os dados
                                        Evento_tcp_ip=0x0;
                      // não há mais dados para transmitir
                                    }
                                    else{
                                        TCPFlush(TCPIPSocket);
                // Transmite os dados
                                        TCPIP_Time_Out =
TickGet();              // Resetar time-out do correio eletronico
                                    }
                               }
                               else{
                                    TCPDisconnect(TCPIPSocket);
                                    smState = SM_LISTEN_WAIT1;
                // fazer nova conexão
                               }
                      break;
                }
          }
//*****************************************************************
*********************************************************
#endif

          // Esta tarefa executa o stack incluindo checagem da chegada
de novos pacotes,
          // tipos de pacote e chamadas apropriadas a entidades do
stack para processamento
       StackTask();

          #if defined(SNMP)
               // This task performs common SNMP Agent tasks.  When
          // a request is received and processed,
          // it callbacks to application.
          SNMPTask();
          #endif

          #if defined(STACK_USE_HTTP_SERVER)
               // Esta é a aplicação TCP. Ela escuta a porta 80 TCP
com um ou mais soquetes
               // e responde a requisição remotas
               HTTPServer();
          #endif

          #if defined (EMAIL)
               SMTP_Verificar();
          #endif

          #if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
          FTPServer();
          #endif
```

```
        // Novas aplicações TCP/IP deverão ser adicionadas aqui

        // Adicione tarefas específicas de sua aplicação aqui.
        ProcessIO();

          #if defined(TEMPERATURA)
            temperatura = TC74_ProcessEvents();      // leitura da
temperatura do C.I TC74
            #endif

         // Para informações DHCP, mostramos quantas vezes a configuração
IP foi modificada desde o último RESET

        if ( DHCPBindCount != myDHCPBindCount ) {
               #if defined(USE_LCD)
                  DisplayIPValue(&AppConfig.MyIPAddr, TRUE);
               #endif
            myDHCPBindCount = DHCPBindCount;

            if ( AppConfig.Flags.bIsDHCPEnabled ){
                        #if defined (USE_LCD)
                              XLCDGoto(1, 14);
                              if ( myDHCPBindCount < 0x0a )
XLCDPut(myDHCPBindCount + '0');
                              else  XLCDPut(myDHCPBindCount + 'A');
                        #endif
            }
        }

    }
}

#if defined (SNMP)

static BOOL SendNotification(BYTE receiverIndex, SNMP_ID var, SNMP_VAL
val)
{
    static enum { SM_PREPARE, SM_NOTIFY_WAIT } smState = SM_PREPARE;
    IP_ADDR IPAddress;

    // Convert local to network order.
    IPAddress.v[0] = trapInfo.table[receiverIndex].IPAddress.v[3];
    IPAddress.v[1] = trapInfo.table[receiverIndex].IPAddress.v[2];
    IPAddress.v[2] = trapInfo.table[receiverIndex].IPAddress.v[1];
    IPAddress.v[3] = trapInfo.table[receiverIndex].IPAddress.v[0];

    switch(smState)
    {
    case SM_PREPARE:
        SNMPNotifyPrepare(&IPAddress,
                          trapInfo.table[receiverIndex].community,
                          trapInfo.table[receiverIndex].communityLen,
                          MICROCHIP,            // Agent ID Var
                          6,                    // Notification code
                          (DWORD)TickGet());
        smState = SM_NOTIFY_WAIT;

        break;

    case SM_NOTIFY_WAIT:
```

```
        if ( SNMPIsNotifyReady(&IPAddress) )
        {
            smState = SM_PREPARE;
            SNMPNotify(var, val, 0);
            return TRUE;
        }
    }
    return FALSE;
}
#endif


#if defined(SNMP)
BOOL SNMPValidate(SNMP_ACTION SNMPAction, char* community)
{
    return TRUE;
}
#endif

#if defined(SNMP)
// Only dynamic variables with ASCII_STRING or OCTET_STRING data type
// needs to be handled.
BOOL SNMPIsValidSetLen(SNMP_ID var, BYTE len)
{
    switch(var)
    {
    case TRAP_COMMUNITY:
        if ( len < MAX_COMMUNITY_LEN+1 )
            return TRUE;
        break;

    case LCD_DISPLAY:
        if ( len < LCD_DISPLAY_LEN+1 )
            return TRUE;
        break;
    }
    return FALSE;
}
#endif


#if defined(SNMP)
// Only dynamic read-write variables needs to be handled.
BOOL SNMPSetVar(SNMP_ID var, SNMP_INDEX index, BYTE ref, SNMP_VAL val)
{
    switch(var)
    {
    case LED_D2:
        LED_D2_CONTROL = val.byte;
        return TRUE;

//    case LED_D6:
//        LED_D6_CONTROL = val.byte;
//        return TRUE;

    case TRAP_RECEIVER_IP:
        // Make sure that index is within our range.
        if ( index < trapInfo.Size )
        {
            // This is just an update to an existing entry.
```

```
            trapInfo.table[index].IPAddress.Val = val.dword;
            return TRUE;
        }
        else if ( index < TRAP_TABLE_SIZE )
        {
            // This is an addition to table.
            trapInfo.table[index].IPAddress.Val = val.dword;
            trapInfo.table[index].communityLen = 0;
            trapInfo.Size++;
            return TRUE;
        }
        break;

    case TRAP_RECEIVER_ENABLED:
        // Make sure that index is within our range.
        if ( index < trapInfo.Size )
        {
            // Value of '1' means Enabled".
            if ( val.byte == 1 )
                trapInfo.table[index].Flags.bEnabled = 1;
            // Value of '0' means "Disabled.
            else if ( val.byte == 0 )
                trapInfo.table[index].Flags.bEnabled = 0;
            else
                // This is unknown value.
                return FALSE;
            return TRUE;
        }
        // Given index is more than our current table size.
        // If it is within our range, treat it as an addition to table.
        else if ( index < TRAP_TABLE_SIZE )
        {
            // Treat this as an addition to table.
            trapInfo.Size++;
            trapInfo.table[index].communityLen = 0;
        }

        break;

    case TRAP_COMMUNITY:
        // Since this is a ASCII_STRING data type, SNMP will call with
        // SNMP_END_OF_VAR to indicate no more bytes.
        // Use this information to determine if we just added new row
        // or updated an existing one.
        if ( ref ==  SNMP_END_OF_VAR )
        {
            // Index equal to table size means that we have new row.
            if ( index == trapInfo.Size )
                trapInfo.Size++;

            // Length of string is one more than index.
            trapInfo.table[index].communityLen++;

            return TRUE;
        }

        // Make sure that index is within our range.
        if ( index < trapInfo.Size )
        {
            // Copy given value into local buffer.
```

```
            trapInfo.table[index].community[ref] = val.byte;
            // Keep track of length too.
            // This may not be NULL terminate string.
            trapInfo.table[index].communityLen = (BYTE)ref;
            return TRUE;
        }
        break;

    case LCD_DISPLAY:
        // Copy all bytes until all bytes are transferred
        if ( ref != SNMP_END_OF_VAR )
        {
            LCDDisplayString[ref] = val.byte;
            LCDDisplayStringLen++;
        }
        else
        {
//            XLCDGoto(0, 0);
//            XLCDPutROMString(blankLCDLine);
//            XLCDGoto(0, 0);
//            XLCDPutString(LCDDisplayString);
        }

        return TRUE;

    }

    return FALSE;
}
#endif

#if defined(SNMP)
// Only sequence index needs to be handled in this function.
BOOL SNMPGetNextIndex(SNMP_ID var, SNMP_INDEX *index)
{
    SNMP_INDEX tempIndex;

    tempIndex = *index;

    switch(var)
    {
    case TRAP_RECEIVER_ID:
        // There is no next possible index if table itself is empty.
        if ( trapInfo.Size == 0 )
            return FALSE;

        // INDEX_INVALID means start with first index.
        if ( tempIndex == SNMP_INDEX_INVALID )
        {
            *index = 0;
            return TRUE;
        }
        else if ( tempIndex < (trapInfo.Size-1) )
        {
            *index = tempIndex+1;
            return TRUE;
        }
        break;
    }
    return FALSE;
```

```
}
#endif


#if defined(SNMP)
BOOL SNMPGetVar(SNMP_ID var, SNMP_INDEX index, BYTE *ref, SNMP_VAL* val)
{
    BYTE myRef;

    myRef = *ref;

    switch(var)
    {
    case SYS_UP_TIME:
        val->dword = TickGet();
        return TRUE;

    case LED_D2:
        val->byte = LED_D2_CONTROL;
        return TRUE;

//    case LED_D6:
//        val->byte = LED_D6_CONTROL;
//        return TRUE;

    case PUSH_BUTTON:
        // There is only one button - meaning only index of 0 is allowed.
        val->byte = SW_S1;
        return TRUE;

    case ANALOG_POT0:
        val->word = AN0Value.Val;
        return TRUE;

//    case ANALOG_POT1:
//        val->word = AN1Value.Val;
//        return TRUE;

    #if defined(TEMPERATURA)
    case TEMPERATURA_SNMP:
        val->word = temperatura;
        return TRUE;
      #endif

    case TRAP_RECEIVER_ID:
        if ( index < trapInfo.Size )
        {
            val->byte = index;
            return TRUE;
        }
        break;

    case TRAP_RECEIVER_ENABLED:
        if ( index < trapInfo.Size )
        {
            val->byte = trapInfo.table[index].Flags.bEnabled;
            return TRUE;
        }
        break;
```

```
    case TRAP_RECEIVER_IP:
        if ( index < trapInfo.Size )
        {
            val->dword = trapInfo.table[index].IPAddress.Val;
            return TRUE;
        }
        break;

    case TRAP_COMMUNITY:
        if ( index < trapInfo.Size )
        {
            if ( trapInfo.table[index].communityLen == 0 )
                *ref = SNMP_END_OF_VAR;
            else
            {
                val->byte = trapInfo.table[index].community[myRef];

                myRef++;

                if ( myRef == trapInfo.table[index].communityLen )
                    *ref = SNMP_END_OF_VAR;
                else
                    *ref = myRef;
            }
            return TRUE;
        }
        break;

    case LCD_DISPLAY:
        if ( LCDDisplayStringLen == 0 )
            myRef = SNMP_END_OF_VAR;
        else
        {
            val->byte = LCDDisplayString[myRef++];
            if ( myRef == LCDDisplayStringLen )
                myRef = SNMP_END_OF_VAR;
        }

        *ref = myRef;
        return TRUE;

    }

    return FALSE;
}
#endif


static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD)
{
    char IPDigit[8];

    if ( bToLCD ){
        // Limpa a segunda linha.
        XLCDGoto(1, 0);
        XLCDPutROMString(blankLCDLine);
    }

    // Re-escreve a segunda linha.
    XLCDGoto(1, 0);
```

```
    itoa(IPVal->v[0], IPDigit);
    if ( bToLCD ){
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else{
        ((unsigned char*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[1], IPDigit);
    if ( bToLCD ){
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else{
        USARTPutString((unsigned char*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[2], IPDigit);
    if ( bToLCD ){
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else{
        USARTPutString((unsigned char*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[3], IPDigit);
    if ( bToLCD )
        XLCDPutString(IPDigit);
    else
        USARTPutString((unsigned char*)IPDigit);
}



static char AN0String[8];
static char AN1String[8];

// Tecla F (L1,C1) = 0x11
// Tecla 3 (L2,C1) = 0x12
// Tecla 2 (L3,C1) = 0x13
// Tecla 1 (L4,C1) = 0x14
// Tecla E (L1,C2) = 0x21
// Tecla 6 (L2,C2) = 0x22
// Tecla 5 (L3,C2) = 0x23
// Tecla 4 (L4,C2) = 0x24
// Tecla D (L1,C3) = 0x41
// Tecla 9 (L2,C3) = 0x42
// Tecla 8 (L3,C3) = 0x43
// Tecla 7 (L4,C3) = 0x44
// Tecla C (L1,C4) = 0x81
// Tecla B (L2,C4) = 0x82
// Tecla 0 (L3,C4) = 0x83
// Tecla A (L4,C4) = 0x84
```

```
#define TECLA_F 0x11
#define TECLA_3 0x12
#define TECLA_2 0x13
#define TECLA_1 0x14
#define TECLA_E 0x21
#define TECLA_6 0x22
#define TECLA_5 0x23
#define TECLA_4 0x24
#define TECLA_D 0x41
#define TECLA_9 0x42
#define TECLA_8 0x43
#define TECLA_7 0x44
#define TECLA_C 0x81
#define TECLA_B 0x82
#define TECLA_0 0x83
#define TECLA_A 0x84

static void ProcessIO(void){

#if defined(SNMP)
    static BOOL lbNotify = FALSE;
    static BYTE i = 0;
    SNMP_VAL val;
#endif

      WORD_VAL ADCResult;

#if defined(SNMP)
    if ( PORTJ_RJ7 == 0 && !lbNotify )
        lbNotify = TRUE;

    if ( i == trapInfo.Size )
    {
        i = 0;
        lbNotify = FALSE;
    }

    if ( lbNotify )
    {
        if ( trapInfo.table[i].Flags.bEnabled )
        {
           val.byte = 0;
            if ( SendNotification(i, PUSH_BUTTON, val) )
                i++;
        }
        else
            i++;
    }

#endif

     // Seleciona canal AN0, Fosc/32 clock
    ADCON0      = 0b10000001;

    // Espera tempo de conversão.
    // Aqui um simples loop de espera é usado.
    ADCResult.v[0] = 100;
    while( ADCResult.v[0]-- );
```

```
    // Primeiramente converte canal AN0.
    // AN0 já está pronto para servir como entrada analógica.
    ADCON0_GO   = 1;

    // Espera conversão terminar
    while( ADCON0_GO );

    // Salva resultado.
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;

    AN0Value.v[0] = ADRESL;
    AN0Value.v[1] = ADRESH;

// teclado
//    ADCResult.v[0] = tecla;
//    ADCResult.v[0] = tecla;
//    ADCResult.v[1] = 0x0;

#if defined (TECLADO)
      Teclado_Verifica();
#endif

    // Converte valores 10-bit em String ASCII.
    itoa(ADCResult.Val, AN0String);

     // Agora converte canal AN1.
     // Em algumas placas (PICDEM.net), RA2 até RA7 deveriam ser digitais
ou caso contrário LED, LCD e NIC não operarão corretamente.
     // Desde que não há nenhum modo aonde somente AN0 e AN1 sejam
entradas analógicas enquanto o resto são pinos digitais,
     // nós temporariamente chavearemos o modo selecionado aonde RA2
torna-se uma entrada analógica enquanto fazemos a conversão
     // de RA1. Desde que a conversão tenha sido feita, nós converteremos
RA2 de volta para pino digital

      // cANAL 1 não será feito a conversão, pois a entrada está
flutuando

      /*
      ADCON1      = 0b10000100;

      // Seleciona canal AN1.
      ADCON0      = 0b10000001;

      // Espera tempo de conversão.
      ADCResult.v[0] = 100;
      while( ADCResult.v[0]-- );

      // Começa conversão.
      ADCON0_GO   = 1;

      // Espera conversão terminar.
      while( ADCON0_GO );

      // Salva resultado.
      ADCResult.v[0] = ADRESL;
      ADCResult.v[1] = ADRESH;
      */
```

```
      ADCResult.v[0] = 0;
      ADCResult.v[1] = 0;

    // Converte valor de 10 bits em string.
    itoa(ADCResult.Val, AN1String);

    /*
     * Reset RA2 pin back to digital output.
     */
    ADCON1      = 0b10001110;       // RA0 as analog input.

}


/*
 * CGI Command Codes.
 */
#define CGI_CMD_ENVIAR_CMD          (0x00)

#define VAR_CMD                      (0x00)
#define VAR_BUTTON_ENVIAR           (0x01)

/*
 * CGI Variable codes. - There could be 0-255 variables.
 */
#define CGI_VAR_GERENCIADORSTATUS (0x00)
#define CGI_VAR_RA0             (0x01)
#define CGI_VAR_RA1             (0x02)
#define CGI_VAR_RA2             (0x03)
#define CGI_VAR_RA3             (0x04)
#define CGI_VAR_RA4             (0x05)
#define CGI_VAR_RA5             (0x06)
#define CGI_VAR_RB0             (0x07)
#define CGI_VAR_RB1             (0x08)
#define CGI_VAR_RB2             (0x09)
#define CGI_VAR_RB3             (0x10)
#define CGI_VAR_RB4             (0x11)
#define CGI_VAR_RB5             (0x12)
#define CGI_VAR_RB6             (0x13)
#define CGI_VAR_RB7             (0x14)
#define CGI_VAR_RC0             (0x15)
#define CGI_VAR_RC1             (0x16)
#define CGI_VAR_RC2             (0x17)
#define CGI_VAR_RC3             (0x18)
#define CGI_VAR_RC4             (0x19)
#define CGI_VAR_RC5             (0x20)
#define CGI_VAR_RC6             (0x21)
#define CGI_VAR_RC7             (0x22)
#define CGI_VAR_RD0             (0x23)
#define CGI_VAR_RD1             (0x24)
#define CGI_VAR_RD2             (0x25)
#define CGI_VAR_RD3             (0x26)
#define CGI_VAR_RD4             (0x27)
#define CGI_VAR_RD5             (0x28)
#define CGI_VAR_RD6             (0x29)
#define CGI_VAR_RD7             (0x30)
#define CGI_VAR_RE0             (0x31)
#define CGI_VAR_RE1             (0x32)
#define CGI_VAR_RE2             (0x33)
#define CGI_VAR_TA              (0x34)
```

```
#define CGI_VAR_TB              (0x35)
#define CGI_VAR_TC              (0x36)
#define CGI_VAR_TD              (0x37)
#define CGI_VAR_TE              (0x38)
#define CGI_VAR_ADC0STATUS      (0x39)
#define CGI_VAR_ADC0VALOR       (0x40)
#define CGI_VAR_ADC1STATUS      (0x41)
#define CGI_VAR_ADC1VALOR       (0x42)
#define CGI_VAR_ADC2STATUS      (0x43)
#define CGI_VAR_ADC2VALOR       (0x44)
#define CGI_VAR_ADC3STATUS      (0x45)
#define CGI_VAR_ADC3VALOR       (0x46)
#define CGI_VAR_ADC4STATUS      (0x47)
#define CGI_VAR_ADC4VALOR       (0x48)
#define CGI_VAR_ADC5STATUS      (0x49)
#define CGI_VAR_ADC5VALOR       (0x50)
#define CGI_VAR_ADC6STATUS      (0x51)
#define CGI_VAR_ADC6VALOR       (0x52)
#define CGI_VAR_ADC7STATUS      (0x53)
#define CGI_VAR_ADC7VALOR       (0x54)
#define CGI_VAR_TIMER0STATUS    (0x55)
#define CGI_VAR_TIMER0VALOR     (0x56)
#define CGI_VAR_TIMER1STATUS    (0x57)
#define CGI_VAR_TIMER1VALOR     (0x58)
#define CGI_VAR_PWMSTATUS       (0x59)
#define CGI_VAR_PWMVALOR        (0x60)
#define CGI_VAR_INTEXT0STATUS   (0x61)
#define CGI_VAR_INTEXT1STATUS   (0x62)
#define CGI_VAR_INTEXT2STATUS   (0x63)
#define CGI_VAR_CANSTATUS       (0x64)
#define CGI_VAR_I2CSTATUS       (0x65)
#define CGI_VAR_SPISTATUS       (0x66)
#define CGI_VAR_RS232STATUS     (0x67)
#define CGI_VAR_EEPROMEND0      (0x70)
#define CGI_VAR_EEPROMVALOR0    (0x71)
#define CGI_VAR_EEPROMEND1      (0x72)
#define CGI_VAR_EEPROMVALOR1    (0x73)
#define CGI_VAR_EEPROMEND2      (0x74)
#define CGI_VAR_EEPROMVALOR2    (0x75)
#define CGI_VAR_EEPROMEND3      (0x76)
#define CGI_VAR_EEPROMVALOR3    (0x77)
#define CGI_VAR_EEPROMEND4      (0x78)
#define CGI_VAR_EEPROMVALOR4    (0x79)
#define CGI_VAR_RAMEND0         (0x80)
#define CGI_VAR_RAMVALOR0       (0x81)
#define CGI_VAR_RAMEND1         (0x82)
#define CGI_VAR_RAMVALOR1       (0x83)
#define CGI_VAR_RAMEND2         (0x84)
#define CGI_VAR_RAMVALOR2       (0x85)
#define CGI_VAR_RAMEND3         (0x86)
#define CGI_VAR_RAMVALOR3       (0x87)
#define CGI_VAR_RAMEND4         (0x88)
#define CGI_VAR_RAMVALOR4       (0x89)
#define CGI_VAR_SERIAL_IN           (0x91)
#define CGI_VAR_SERIAL_OUT           (0x92)

/*********************************************************************
************************
 * Function:        void HTTPExecCmd(unsigned char** argv, unsigned char
argc)
```

```
 *
 * Pré-Condição:    Nenhuma
 *
 * Entrada:         argv        - Lista de argumentos
 *                  argc        - Número de argumentos.
 *
 * Output:          Nenhuma
 *
 * Outros efeitos:  Nenhum
 *
 * Overview:        RECEBE uma lista de argumentos do navegador Web que
servem como COMANDOS.
 *                  argv[0]="0", argv[1]="name do botao", argc=3,
conforme nosso exemplo commands.cgi
 *                      argv[0] => 0 (action=0 do arquivo commands.cgi)
da linha
 *                                  <FORM METHOD=GET action=0>
 *                      argv[1] => qual botão foi pressionado(name=5 para
SMS e name=2 para LED A2)
 *                                  das linhas:
 *                                          <td><input type=submit
name=5 value="ENVIA SMS"></td>
 *                                          <td><input type=submit
name=1 value="LEDA2"></td>
 *
 * NOTA: Usar valores numéricos para "name" dos comandos simplifica o
nosso trabalho
 ************************************************************************
************************/
#if defined(STACK_USE_HTTP_SERVER)

ROM char COMMANDS_OK_PAGE[] = "CTRL.CGI";

// Copy string with NULL termination.
#define COMMANDS_OK_PAGE_LEN  (sizeof(COMMANDS_OK_PAGE))

ROM char CMD_UNKNOWN_PAGE[] = "CTRL.CGI";

// Copy string with NULL termination.
#define CMD_UNKNOWN_PAGE_LEN    (sizeof(CMD_UNKNOWN_PAGE))

unsigned char copiaCommand[MAX_SIZE_COMMAND] = "";

void HTTPExecCmd(unsigned char** argv, unsigned char argc)
{
      // Por exemplo, o navegador Web envia os seguintes argumentos para
o microcontrolador: 0?A=valor1&B=valor2&C=valor3
      // argv[0]  = "0"     Identificador do form. Definido no <form
METHOD=GET action=0>
      // argv[1]  = "A"     Variável definida no <select name=A>
      // argv[2]  = valor1  Valor atribuído a variável A
      // argv[3]  = "B"
      // argv[4]  = valor2
      // argv[5]  = "C"
      // argv[6]  =  valor3
    unsigned char command[MAX_SIZE_COMMAND]; //armazena comando a ser
enviado pela serial
    unsigned char form; //indica qual o form relacionado
    BOOL commandOK = TRUE; //sinaliza que comando foi executado
```

```
    // argumento de  <FORM METHOD=GET action=0>
    form = argv[0][0] - '0';//ja converte para inteiro

    switch(form)
    {
        case CGI_CMD_ENVIAR_CMD:
            Concat(&command[0], &argv[2][0], "\r\n", "", "");
            USARTPutString(command);  //envia comando pela serial
            strcpy(copiaCommand, command);
        break;
        default:
            commandOK = FALSE;
        break;
    }
    if(commandOK)
        memcpypgm2ram((void*)argv[0], (ROM void*)COMMANDS_OK_PAGE,
COMMANDS_OK_PAGE_LEN);
    else
        memcpypgm2ram((void*)argv[0], (ROM void*)CMD_UNKNOWN_PAGE,
CMD_UNKNOWN_PAGE_LEN);
}
#endif


/*****************************************************************
 * Function:        WORD HTTPGetVar(unsigned char var, WORD ref, unsigned
char* val)
 *
 * PreCondition:    None
 *
 * Input:           var         - Variable Identifier
 *                  ref         - Current callback reference with
 *                                respect to 'var' variable.
 *                  val         - Buffer for value storage.
 *
 * Output:          Variable reference as required by application.
 *
 * Side Effects:    None
 *
 * Overview:        This is a callback function from HTTPServer() to
 *                  main application.  (STATUS.CGI)
 *                  Whenever a variable substitution is required
 *                  on any html pages, HTTPServer calls this function
 *                  8-bit variable identifier, variable reference,
 *                  which indicates whether this is a first call or
 *                  not.  Application should return one character
 *                  at a time as a variable value.
 *
 * Note:            Since this function only allows one character
 *                  to be returned at a time as part of variable
 *                  value, HTTPServer() calls this function
 *                  multiple times until main application indicates
 *                  that there is no more value left for this
 *                  variable.
 *                  On begining, HTTPGetVar() is called with
 *                  ref = HTTP_START_OF_VAR to indicate that
 *                  this is a first call.  Application should
 *                  use this reference to start the variable value
 *                  extraction and return updated reference.  If
 *                  there is no more values left for this variable
```

```
 *                  application should send HTTP_END_OF_VAR.  If
 *                  there are any bytes to send, application should
 *                  return other than HTTP_START_OF_VAR and
 *                  HTTP_END_OF_VAR reference.
 *
 *                  THIS IS AN EXAMPLE CALLBACK.
 *                  MODIFY THIS AS PER YOUR REQUIREMENTS.
 ********************************************************************/
// MOSTRA VARIÁVEIS
// NO NAVEGADOR WEB
/*********************************************************************
*************/

#if defined(STACK_USE_HTTP_SERVER)
WORD HTTPGetVar(unsigned char var, WORD ref, unsigned char* val)
{
    switch(var)
    {
        case CGI_VAR_SERIAL_IN:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
                ref = (BYTE)0;
            *val = rec[(BYTE)ref];
            (BYTE)ref++;
            if(rec[(BYTE)ref] == '\0')  //se acabou de transferir o valor
para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_SERIAL_OUT:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
                ref = (BYTE)0;
            *val = copiaCommand[(BYTE)ref];
            (BYTE)ref++;
            if(copiaCommand[(BYTE)ref] == '\0')  //se acabou de
transferir o valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_GERENCIADORSTATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(monitorOn)
                {
                    temp2[0] = 'A';
                    temp2[1] = '\0';
                }
                else
                {
                    temp2[0] = 'D';
                    temp2[1] = 'e';
                    temp2[2] = 's';
                    temp2[3] = 'a';
                    temp2[4] = '\0';
```

```
                }
                ref = (BYTE)0;
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RA0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RA & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RA1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RA & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RA2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RA & 4)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
```

```
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RA3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RA & 8)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RA4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RA & 16)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RA5:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RA & 32)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
```

```
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RB0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RB1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RB2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 4)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
```

```
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RB3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 8)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RB4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 16)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RB5:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 32)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
```

```
        case CGI_VAR_RB6:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 64)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RB7:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RB & 128)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RC & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
```

```
            {
                if(RC & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RC & 4)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RC & 8)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RC & 16)
                    pTemp = ON;
```

```
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC5:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RC & 32)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC6:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RC & 64)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RC7:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RC & 128)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
```

```
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 4)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
```

```
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 8)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 16)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD5:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 32)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
```

```
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD6:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 64)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RD7:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RD & 128)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RE0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RE & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
```

```
        break;
        case CGI_VAR_RE1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RE & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RE2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(RE & 4)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_TA:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(TA, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref;
        break;
        case CGI_VAR_TB:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(TB, &temp2[0], 3);
```

```
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref;
        break;
        case CGI_VAR_TC:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(TC, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref;
        break;
        case CGI_VAR_TD:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(TD, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref;
        break;
        case CGI_VAR_TE:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(TE, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref;
        break;
        case CGI_VAR_ADC0STATUS:
            if(ref==HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 1)
                    pTemp = ON;
```

```
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC1STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC2STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 4)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC3STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 8)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
```

```
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC4STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 16)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC5STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 32)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC6STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 64)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
```

```
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC7STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusADC & 128)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_TIMER0STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusTimer & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_TIMER1STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusTimer & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
```

```
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_PWMSTATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusPWM & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_INTEXT0STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusIntExt & 1)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_INTEXT1STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusIntExt & 2)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
```

```
        break;
        case CGI_VAR_INTEXT2STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusIntExt & 4)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_CANSTATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusCAN)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_I2CSTATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusI2C)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_SPISTATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
```

```
requisição para a variável
            {
                if(statusSPI)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RS232STATUS:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                if(statusRS232)
                    pTemp = ON;
                else
                    pTemp = OFF;
                ref = (BYTE)0;
            }
            *val = pTemp[(BYTE)ref];
            (BYTE)ref++;
            if(pTemp[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC0VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueADC0, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC1VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueADC1, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
```

```
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC2VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueADC2, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC3VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueADC3, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC4VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueADC4, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC5VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
```

```
                ref = (BYTE)0;
                IntToString(valueADC5, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC6VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueADC6, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_ADC7VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueADC7, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_TIMER0VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueTimer0, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
```

```
        case CGI_VAR_TIMER1VALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueTimer1, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_PWMVALOR:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valuePWM, &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMEND0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressEE[0], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMEND1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressEE[1], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
```

```
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMEND2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressEE[2], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMEND3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressEE[3], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMEND4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressEE[4], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMVALOR0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueEE[0], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
```

```
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMVALOR1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueEE[1], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMVALOR2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueEE[2], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMVALOR3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueEE[3], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_EEPROMVALOR4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
```

```
                ref = (BYTE)0;
                IntToString(valueEE[4], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMEND0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressRAM[0], &temp2[0], 4);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMEND1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressRAM[1], &temp2[0], 4);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMEND2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressRAM[2], &temp2[0], 4);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
```

```
        case CGI_VAR_RAMEND3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressRAM[3], &temp2[0], 4);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMEND4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(addressRAM[4], &temp2[0], 4);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMVALOR0:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueRAM[0], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMVALOR1:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueRAM[1], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
```

```
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMVALOR2:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueRAM[2], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMVALOR3:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueRAM[3], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        case CGI_VAR_RAMVALOR4:
            if(ref == HTTP_START_OF_VAR)  //entra aqui na primeira
requisição para a variável
            {
                ref = (BYTE)0;
                IntToString(valueRAM[4], &temp2[0], 3);
            }
            *val = temp2[(BYTE)ref];
            (BYTE)ref++;
            if(temp2[(BYTE)ref] == '\0')  //se acabou de transferir o
valor para a variável
                return HTTP_END_OF_VAR; //servidor http pára de chamar
esta função para a variável
            return ref; //vai retornando valor da posição da variável
solicitada
        break;
        default:
            *val = '8';
            return HTTP_END_OF_VAR;
        break;
    }
    return HTTP_END_OF_VAR;
}
#endif
```

```
#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
ROM char FTP_USER_NAME[]    = "ftp";
#undef FTP_USER_NAME_LEN
#define FTP_USER_NAME_LEN   (sizeof(FTP_USER_NAME)-1)

ROM char FTP_USER_PASS[]    = "microchip";
#define FTP_USER_PASS_LEN   (sizeof(FTP_USER_PASS)-1)

BOOL FTPVerify(char *login, char *password)
{
    if ( !memcmppgm2ram(login, (ROM void*)FTP_USER_NAME,
FTP_USER_NAME_LEN) )
    {
        if ( !memcmppgm2ram(password, (ROM void*)FTP_USER_PASS,
FTP_USER_PASS_LEN) )
            return TRUE;
    }
    return FALSE;
}
#endif



/*********************************************************************
 * Function:        void InitializeBoard(void)
 *
 * Pré-Condição:    Nenhuma
 *
 * Entrada:         Nenhuma
 *
 * Saída:           Nenhuma
 *
 * Outros Efeitos:  Nenhum
 *
 * Função:          Initcializa hardware específico da placa.
 *
 * Notas:           Nenhuma
 ********************************************************************/
static void InitializeBoard(void){

      unsigned char a;

      // Prepara porta PORTA.RA0 como entrada analógica enquanto as
demais
      // são linhas de I/O digitais
      ADCON1  = 0b00001110;       // 00 - não implementado, 00 - AVdd,
1110 - RA0 como entrada analógica
      TRISA   = 0x03;                     // RA2 a RA7 como saídas, RA0 e
RA1 como entradas

      ADCON2=0xba;                        // Right Justified, 20 Tad,
Fosc/32

      // Desliga LED de teste
      LATA2 = 1;                          // Lembrar que é lógica
invertida
```

```
      // Desliga o led de atividade do programa
      LATA4 = 1;

      // PORTA B, RB7, RB6, RB5 e RB4 são entradas para teclado
      // RB7 como saída para LED
      // RB6 como saída para LED
      // RB5 como entrada para teclado
      // RB4 como entrada para teclado
      // RB3 como entrada, via foto acoplador
      // RB2 como entrada, via foto acoplador
      // RB1 livre
      // RB2 como entrada, via foto acoplador
      TRISB |= 0b00111101;
      TRISB &= 0b00111111;

      // Desliga o led de atividade XBEE
      LATB6 = 1;

      // Desliga o led de atividade do programa
      LATB7 = 1;

      // Prepara PORTA  C todos como saída exceção do RC7 que é entrada,
I2C configurado posteriormente
      // RC7 entrada da USART1
      // RC6 saída da USART1
      // RC5 livre
      // RC4 como como entrada I2C
      // RC3 como saída, I2C clock
      // RC2 livre
      // RC1 livre
      // RC0 livre
      TRISC |= 0b10010000;
      TRISC &= 0b10110111;

      // Prepara PORTA F
      // RF7 como saída, RESET do XBEE (RESET=0 ativo em baixo)
      // RF6 como saída, Sleep do XBEE (0 sem dormir)
      // RF5 livre
      // RF4 livre
      // RF3 livre
      // RF2 livre
      // RF1 livre
      // RF0 livre
      TRISF &= 0x3f;
      LATF7 = 0;
      LATF6 = 0;

      // Prepara porta G
      // RG4 como saída, RS do LCD
      // RG3 como saída, R/W do LCD
      // RG2 como entrada, Rx da serial
      // RG1 como saída, Tx da serial
      // RG0 como entrada, livre
      TRISG |= 0b00000101;
      TRISG &= 0b11100101;



      // Prepara porta H
```

```
// RH7 como saída, ativa relé
// RH6 livre
// RH5 livre
// RH4 livre
// RH3 como entrada, DB7 display de LCD
// RH2 como entrada, DB6 display de LCD
// RH1 como entrada, DB5 display de LCD
// Rh0 como entrada, DB4 display de LCD
TRISH |= 0b01000000;
TRISH &= 0b01111111;
// relé desativado
LATH7 = 0;


// Prepara porta J
// RJ7 como entrada, chave push-button
// RJ6 como saída, ativa relé RL3
// RJ5 como saída, ativa relé RL2
// RJ4 como saída, ativa sonalarme
// RJ3 como saída, teclado
// RJ2 como saída, teclado
// RJ1 como saída, teclado
// RJ0 como saída, teclado
TRISJ |= 0b10000000;
TRISJ &= 0b10000000;
// relé desativado

// não ativar RL3
LATJ6 = 0;
// não ativar RL2
LATJ5 = 0;
// não ativar sonalarme
LATJ4 = 0;

// não ativar linha do teclado
LATJ3 = 0;
// não ativar linha do teclado
LATJ2 = 0;

// não ativar linha do teclado
LATJ1 = 0;
      // Linha 1 do teclado para 0
// não ativar linha do teclado
LATJ0 = 0;
      // Linha 0 do teclado para 0


#if defined(USE_LCD)
      XLCDInit();
      XLCDGoto(0, 0);
      XLCDPutROMString(StartupMsg);
#endif


TXSTA = 0b00100000;     // Low BRG speed
RCSTA = 0b10010000;
SPBRG = SPBRG_VAL;            // BAUD serial

T0CON = 0;
```

```
      // IPEN: Interrupt Priority Enable bit
      // 1 = Enable priority levels on interrupts
      // 0 = Disables priority levels on interrupts (PIC 16XXX
Compatibility mode).
      RCONbits.IPEN=1;
            // Habilita prioridade de interrrupções

      // RCIP: EUSART Receive Interrupt Priority bit
      //    1 = High priority
      //  0 = Low priority
      IPR1bits.RCIP = 0;


      // INTCON2: Interrupt Control Register 2
      // bit 7 - RBPU: PORTB Pull-up Enable bit
      // 1 = All PORTB pull-ups are disabled
      // 0 = PORTB pull-ups are enabled by individual port latch values
      INTCON2bits.RBPU = 0;

      // INTCON2: Interrupt Control Register 2
      // bit 0 - RBIP: RB Port Change Interrupt Priority bit
      // 1 = High priority
      // 0 = low priority
      INTCON2bits.RBIP = 0;

      // GIE/GIEH: Global Interrupt Enable bit
      // When IPEN=1
      // 1 = Enables all high priority interrupts
      // 0 = Disable all interrupts
      // PEIE/GIEL: Peripheral Interrupt Enable bit
      // When IPEN=1
      // 1 = Enables all low priority peripheral interrupts
      // 0 = Disables all low priority peripheral interrupts
    // Enable global interrupts.

    INTCON_GIEL = 1;
            //Qdo IPEN=1, habilita todas as interrupções de baixa
prioridade
    INTCON_GIEH = 1;
            // Qdo IPEN=1, habilita todas as interrupções de alta
prioridade

}

/*********************************************************************
 * Function:        void InitAppConfig(void)
 *
 * PreCondition:    MPFSInit() is already called.
 *
 * Input:           None
 *
 * Output:          Write/Read non-volatile config variables.
 *
 * Side Effects:    None
 *
 * Overview:        None
 *
 * Note:            None
 ********************************************************************/
static void InitAppConfig(void)
```

```
{

    /*
     * Load default configuration into RAM.
     */
    AppConfig.MyIPAddr.v[0]     = MY_DEFAULT_IP_ADDR_BYTE1;
    AppConfig.MyIPAddr.v[1]     = MY_DEFAULT_IP_ADDR_BYTE2;
    AppConfig.MyIPAddr.v[2]     = MY_DEFAULT_IP_ADDR_BYTE3;
    AppConfig.MyIPAddr.v[3]     = MY_DEFAULT_IP_ADDR_BYTE4;

    AppConfig.MyMask.v[0]       = MY_DEFAULT_MASK_BYTE1;
    AppConfig.MyMask.v[1]       = MY_DEFAULT_MASK_BYTE2;
    AppConfig.MyMask.v[2]       = MY_DEFAULT_MASK_BYTE3;
    AppConfig.MyMask.v[3]       = MY_DEFAULT_MASK_BYTE4;

    AppConfig.MyGateway.v[0]    = MY_DEFAULT_GATE_BYTE1;
    AppConfig.MyGateway.v[1]    = MY_DEFAULT_GATE_BYTE2;
    AppConfig.MyGateway.v[2]    = MY_DEFAULT_GATE_BYTE3;
    AppConfig.MyGateway.v[3]    = MY_DEFAULT_GATE_BYTE4;

    AppConfig.MyMACAddr.v[0]    = MY_DEFAULT_MAC_BYTE1;
    AppConfig.MyMACAddr.v[1]    = MY_DEFAULT_MAC_BYTE2;
    AppConfig.MyMACAddr.v[2]    = MY_DEFAULT_MAC_BYTE3;
    AppConfig.MyMACAddr.v[3]    = MY_DEFAULT_MAC_BYTE4;
    AppConfig.MyMACAddr.v[4]    = MY_DEFAULT_MAC_BYTE5;
    AppConfig.MyMACAddr.v[5]    = MY_DEFAULT_MAC_BYTE6;

#if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
    AppConfig.Flags.bIsDHCPEnabled = TRUE;
#else
    AppConfig.Flags.bIsDHCPEnabled = FALSE;
#endif

}



BOOL StringToIPAddress(char *str, IP_ADDR *buffer)
{
    unsigned char v;
    char *temp;
    unsigned char byteIndex;

    temp = str;
    byteIndex = 0;

    while( v = *str )
    {
        if ( v == '.' )
        {
            *str++ = '\0';
            buffer->v[byteIndex++] = atoi(temp);
            temp = str;
        }
        else if ( v < '0' || v > '9' )
            return FALSE;

        str++;
    }
```

```
    buffer->v[byteIndex] = atoi(temp);

    return (byteIndex == 3);
}


void XLCDDelay15ms(void)
{
    DelayMs(15);
}
void XLCDDelay4ms(void)
{
    DelayMs(4);
}

void XLCDDelay100us(void)
{
    INTCON_GIEH = 0;
    Delay10us(1);
    INTCON_GIEH = 1;
}


// Tecla F (L1,C1) = 0x11
// Tecla 3 (L2,C1) = 0x12
// Tecla 2 (L3,C1) = 0x13
// Tecla 1 (L4,C1) = 0x14
// Tecla E (L1,C2) = 0x21
// Tecla 6 (L2,C2) = 0x22
// Tecla 5 (L3,C2) = 0x23
// Tecla 4 (L4,C2) = 0x24
// Tecla D (L1,C3) = 0x41
// Tecla 9 (L2,C3) = 0x42
// Tecla 8 (L3,C3) = 0x43
// Tecla 7 (L4,C3) = 0x44
// Tecla C (L1,C4) = 0x81
// Tecla B (L2,C4) = 0x82
// Tecla 0 (L3,C4) = 0x83
// Tecla A (L4,C4) = 0x84

#if defined (TECLADO)
void Teclado_Verifica(void) {

      switch(tecla){

      case TECLA_0:
                  tecla = 0x0;
                  status_tecla = '0';
                  #if defined (USE_LCD)
                  XLCDGoto(1,15);
            // Caracter T na última coluna da primeira linha
                  XLCDPutROMString((ROM char * const)"0");       //
indicando que esta estação remota está transmitindo p/ estação central
                  #endif
        break;

        case TECLA_1:
                  tecla = 0x0;
                  status_tecla = '1';
                  #if defined (USE_LCD)
```

```
                XLCDGoto(1,15);
            // Caracter T na última coluna da primeira linha
                XLCDPutROMString((ROM char * const)"1");       //
indicando que esta estação remota está transmitindo p/ estação central
                #endif
        break;

        case TECLA_2:
                tecla = 0x0;
                status_tecla = '2';
                #if defined (USE_LCD)
                XLCDGoto(1,15);
            // Caracter T na última coluna da primeira linha
                XLCDPutROMString((ROM char * const)"2");       //
indicando que esta estação remota está transmitindo p/ estação central
                #endif
        break;

        case TECLA_3:
                tecla = 0x0;
                status_tecla = '3';
                #if defined (USE_LCD)
                XLCDGoto(1,15);
            // Caracter T na última coluna da primeira linha
                XLCDPutROMString((ROM char * const)"3");       //
indicando que esta estação remota está transmitindo p/ estação central
                #endif
        break;

        case TECLA_4:
                tecla = 0x0;
                status_tecla = '4';
                #if defined (USE_LCD)
                XLCDGoto(1,15);
            // Caracter T na última coluna da primeira linha
                XLCDPutROMString((ROM char * const)"4");       //
indicando que esta estação remota está transmitindo p/ estação central
                #endif
        break;

      case TECLA_5:
                tecla = 0x0;
                status_tecla = '5';
                #if defined (USE_LCD)
                XLCDGoto(1,15);
            // Caracter T na última coluna da primeira linha
                XLCDPutROMString((ROM char * const)"5");       //
indicando que esta estação remota está transmitindo p/ estação central
                #endif
        break;

        case TECLA_6:
                tecla = 0x0;
                status_tecla = '6';
                #if defined (USE_LCD)
                XLCDGoto(1,15);
            // Caracter T na última coluna da primeira linha
                XLCDPutROMString((ROM char * const)"6");       //
indicando que esta estação remota está transmitindo p/ estação central
                #endif
```

```
        break;

        case TECLA_7:
                  tecla = 0x0;
                  status_tecla = '7';
                  #if defined (USE_LCD)
                  XLCDGoto(1,15);
              // Caracter T na última coluna da primeira linha
                  XLCDPutROMString((ROM char * const)"7");       //
indicando que esta estação remota está transmitindo p/ estação central
                  #endif
        break;

        case TECLA_8:
                  tecla = 0x0;
                  status_tecla = '8';
                  #if defined (USE_LCD)
                  XLCDGoto(1,15);
              // Caracter T na última coluna da primeira linha
                  XLCDPutROMString((ROM char * const)"8");       //
indicando que esta estação remota está transmitindo p/ estação central
                  #endif
        break;

        case TECLA_9:
                  tecla = 0x0;
                  status_tecla = '9';
                  #if defined (USE_LCD)
                  XLCDGoto(1,15);
              // Caracter T na última coluna da primeira linha
                  XLCDPutROMString((ROM char * const)"9");       //
indicando que esta estação remota está transmitindo p/ estação central
                  #endif
        break;

        case TECLA_A:
                  tecla = 0x0;
                  status_tecla = 'A';
                  #if defined (USE_LCD)
                  XLCDGoto(1,15);
              // Caracter T na última coluna da primeira linha
                  XLCDPutROMString((ROM char * const)"A");       //
indicando que esta estação remota está transmitindo p/ estação central
                  #endif
        break;

      case TECLA_B:
                  tecla = 0x0;
                  status_tecla = 'B';
                  #if defined (USE_LCD)
                  XLCDGoto(1,15);
              // Caracter T na última coluna da primeira linha
                  XLCDPutROMString((ROM char * const)"B");       //
indicando que esta estação remota está transmitindo p/ estação central
                  #endif
        break;

        case TECLA_C:
                  tecla = 0x0;
                  status_tecla = 'C';
```

```
                    #if defined (USE_LCD)
                    XLCDGoto(1,15);
                // Caracter T na última coluna da primeira linha
                    XLCDPutROMString((ROM char * const)"C");       //
indicando que esta estação remota está transmitindo p/ estação central
                    #endif
            break;

            case TECLA_D:
                    tecla = 0x0;
                    status_tecla = 'D';
                    #if defined (USE_LCD)
                    XLCDGoto(1,15);
                // Caracter T na última coluna da primeira linha
                    XLCDPutROMString((ROM char * const)"D");       //
indicando que esta estação remota está transmitindo p/ estação central
                    #endif
            break;

            case TECLA_E:
                    tecla = 0x0;
                    status_tecla = 'E';
                    #if defined (USE_LCD)
                    XLCDGoto(1,15);
                // Caracter T na última coluna da primeira linha
                    XLCDPutROMString((ROM char * const)"E");       //
indicando que esta estação remota está transmitindo p/ estação central
                    #endif
            break;

            case TECLA_F:
                    tecla = 0x0;
                    status_tecla = 'F';
                    #if defined (USE_LCD)
                    XLCDGoto(1,15);
                // Caracter T na última coluna da primeira linha
                    XLCDPutROMString((ROM char * const)"F");       //
indicando que esta estação remota está transmitindo p/ estação central
                    #endif
            break;

                default:
                break;
        }

}
#endif



#if defined (EMAIL)
/*********************************************************************
************************
*******************              CORREIO ELETRONICO - SMTP
**************************
*********************************************************************
************************/


// Usadas na conexão com o servidor de e-mail
```

```
#define SM_CONNECT                  0
#define SM_CONNECT_WAIT1      1
#define SM_CONNECTED          2
#define SM_MAIL_HELO          3
#define SM_MAIL_FROM          4
#define SM_MAIL_TO                  5
#define SM_MAIL_DATA          6
#define SM_MAIL_DATA1         7
#define SM_MAIL_DATA2         8
#define SM_MAIL_QUIT          9
#define SM_MAIL_DESCONECT     10
#define SM_MAIL_FIM                 11
#define SM_MAIL_HELO_A        12
#define SM_MAIL_FROM_A        13
#define SM_MAIL_TO_A          14
#define SM_MAIL_DATA_A        15
#define SM_MAIL_DATA1_A       16
#define SM_MAIL_RECEIVED      17
#define SM_MAIL_DESCONECT_A 18

//#define SM_MAIL_DATA_A1           19
//#define SM_MAIL_DATA_A2           20
//#define SM_MAIL_TO2               21
//#define SM_MAIL_TO3               22


ROM char EMAIL_LINHA1[] = "HELO nbk1\r\n";

ROM char EMAIL_LINHA2[] = "MAIL FROM: nbk1@nbk1\r\n";

ROM char EMAIL_LINHA3[] = "RCPT TO:a@guardian.ind.br\r\n";

ROM char EMAIL_LINHA4[] = "DATA\r\n";

ROM char EMAIL_LINHA5[] = "QUIT\r\n";

ROM char EMAIL_LINHA6[] = "From: \" nbk1 server\" <nbk1@nbk1>\r\n";

ROM char EMAIL_LINHA7[] = "To: <a@guardian.ind.br\r\n";

ROM char EMAIL_LINHA8[] = "Subject: Evento no-break \r\n";

ROM char EMAIL_LINHA9[] = "Content-type: text/plain\r\n\r\n";

//ROM char EMAIL_LINHA10[] = "Evento ocorrido no no-break: \r\n";
ROM char EMAIL_LINHA10[] = "Evento ocorrido no no-break:\r\n";
ROM char EMAIL_LINHA10a[] = "Atendimento ao cliente: (21)2501-6458\r\n";
ROM char EMAIL_LINHA10b[] = "cliente@guardian.ind.br\r\n";

ROM char EMAIL_LINHA11[] = ".\r\n";

/*****************************************************************************
*************************
*                            CORREIO ELETRONICO
*****************************************************************************
************************/
void SMTP_Envia(void){

      unsigned char buffer_rec[30];                         // buffer de
recepção EMAIL
```

```
      ROM char* str;                                                    //
ponteiro para receber mensagem da Flash, uso no EMAIL
      unsigned char vv;                                          // uso
no EMAIL

            //  Logo que entra no loop infinito tenta conectar com o
Servidor de E-mails
            switch (smStateEMAIL){
                  case SM_CONNECT:
                        // Conectando a um servidor de e-mail remoto. Com
TCP há uma requisição de conexão e a conexão
                        // é entre dois soquetes, com cada soquete
definido por um IP address e número da porta (processo)
                        EMAILSocket = TCPConnect(&RemoteNode, 25);
                        if (EMAILSocket == INVALID_SOCKET){
                              // Socket não está disponível
                              break;
                        }
                        else
                              smStateEMAIL = SM_CONNECT_WAIT1;
                  break;

                  case SM_CONNECT_WAIT1:
                        // Espera pela conexão
                        if ( TCPIsConnected(EMAILSocket)  ){
                              smStateEMAIL = SM_MAIL_HELO_A;
                        }
                  break;


                  // Deve esperar algo: 220 mta07-svc.abc.com ESMTP
server... ready Mon, 7 Jan 2002 18:43:43 +0000
                  case SM_MAIL_HELO_A:
                        // Espera pela conexão
                        if ( TCPIsGetReady(EMAILSocket) ){
                  // Se o socket contem dados recebidos
                              smStateEMAIL = SM_MAIL_HELO;
                              TCPGetArray(EMAILSocket,buffer_rec,28);
                  // Busca dos dados
                              TCPDiscard(EMAILSocket);
                        // Limpa o Buffer de recepção
                        }
                  break;

                  case SM_MAIL_HELO:
                        // ROM char EMAIL_LINHA1[] = "HELO nbk1\r\n";
                        // nbk1 é o nome do identificador remetente
                        if (TCPIsPutReady (EMAILSocket) )  {
                              smStateEMAIL = SM_MAIL_FROM_A;
                              str = (ROM char* ) EMAIL_LINHA1;
                              while( vv = *str++ ) TCPPut(EMAILSocket,
vv);        // Até aqui foi HELO
                        }
                        TCPFlush(EMAILSocket);
                  break;

                  // Deve esperar algo: 250 mta07-svc.abc.com
                  case SM_MAIL_FROM_A:
                        if ( TCPIsGetReady(EMAILSocket) ){
                  // Se o socket contem dados recebidos
```

```
                              smStateEMAIL = SM_MAIL_FROM;
                              TCPGetArray(EMAILSocket, buffer_rec, 28);
            // Busca dos dados
                              TCPDiscard(EMAILSocket);
                        // Limpa o Buffer de recepção
                        }
                  break;

                  //ROM char EMAIL_LINHA2[] = "MAIL FROM: nbk1@nbk1\r\n";
                  case SM_MAIL_FROM:
                        if (TCPIsPutReady (EMAILSocket) )  {
                              smStateEMAIL = SM_MAIL_TO_A;
                              str = (ROM char* ) EMAIL_LINHA2;
                              while( vv = *str++ ) TCPPut(EMAILSocket,
vv);
                        }
                        TCPFlush(EMAILSocket);
                  break;

                  // Deve esperar algo: 250 Sender <jpb@xyz.co.uk>
                  case SM_MAIL_TO_A:
                        if ( TCPIsGetReady(EMAILSocket) ){
                  // Se o socket contem dados recebidos
                              smStateEMAIL = SM_MAIL_TO;
                              TCPGetArray(EMAILSocket, buffer_rec, 28);
            // Busca dos dados
                              TCPDiscard(EMAILSocket);
                        // Limpa o Buffer de recepção
                        }
                  break;

                  //ROM char EMAIL_LINHA3[] = "RCPT
TO:a@guardian.ind.br\r\n";
                  case SM_MAIL_TO:
                              // Para o destinatário
                              if (TCPIsPutReady (EMAILSocket) )  {
                                    smStateEMAIL = SM_MAIL_DATA_A;
                                    str = (ROM char* ) EMAIL_LINHA3;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);
                              }
                              TCPFlush(EMAILSocket);

                  break;

                  // Deve esperar algo: 250 Recipient <jeremy@uvw.co.uk>
                  case SM_MAIL_DATA_A:
                        if ( TCPIsGetReady(EMAILSocket) ){  // Se o socket
contem dados recebidos
                              smStateEMAIL = SM_MAIL_DATA;
                              TCPGetArray(EMAILSocket, buffer_rec, 28);
                  // Busca dos dados
                              TCPDiscard(EMAILSocket);
                              // Limpa o Buffer de recepção
                        }
                  break;

                  //ROM char EMAIL_LINHA4[] = "DATA\r\n";
                  case SM_MAIL_DATA:
                              if (TCPIsPutReady (EMAILSocket) )  {
```

```
                              smStateEMAIL = SM_MAIL_DATA1_A;
                                    //Envia o dado
                                    str = (ROM char* ) EMAIL_LINHA4;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);
                              }
                              TCPFlush(EMAILSocket);
                  break;

                  // Deve esperar algo: 354 OK Send data ending with
<CRLF>,<CRLF>
                  case SM_MAIL_DATA1_A:
                        if ( TCPIsGetReady(EMAILSocket) ){
                  // Se o socket contem dados recebidos
                              smStateEMAIL = SM_MAIL_DATA2;
                              TCPGetArray(EMAILSocket, buffer_rec, 28);
            // Busca dos dados
                              TCPDiscard(EMAILSocket);
                        // Limpa o Buffer de recepção
                        }
                  break;

                  // TRANSFERENCIA DOS DADOS
                  //ROM char EMAIL_LINHA6[] = "From: \" nbk1 server\"
<nbk1@nbk1>\r\n";
                  //ROM char EMAIL_LINHA7[] = "To:
<a@guardian.ind.br\r\n";
                  //ROM char EMAIL_LINHA8[] = "Subject: Evento no-break
\r\n";
                  //ROM char EMAIL_LINHA9[] = "Content-type:
text/plain\r\n\r\n";
                  //ROM char EMAIL_LINHA10[] = "Evento ocorrido no no-
break:\r\n";
                  //ROM char EMAIL_LINHA10a[] = "Atendimento ao cliente:
(21)2501-6458\r\n";
                  //ROM char EMAIL_LINHA10b[] =
"cliente@guardian.ind.br\r\n";
                  //ROM char EMAIL_LINHA11[] = ".\r\n";

                  case SM_MAIL_DATA2:
                        if (TCPIsPutReady (EMAILSocket) )  {
                              smStateEMAIL = SM_MAIL_RECEIVED;

                              //
---------------------------------------------------------------------
                              //ROM char EMAIL_LINHA6[] = "From: \"
nbk1 server\" <nbk1@nbk1>\r\n";
                              str = (ROM char* ) EMAIL_LINHA6;
                              while( vv = *str++ )
TCPPut(EMAILSocket, vv);

                              //
---------------------------------------------------------------------
                              //ROM char EMAIL_LINHA7[] = "To:
<a@guardian.ind.br\r\n";
                              str = (ROM char* ) EMAIL_LINHA7;
                              while( vv = *str++ )
TCPPut(EMAILSocket, vv);

                              //
```

```
-----------------------------------------------------------------------
                                    //ROM char EMAIL_LINHA8[] = "Subject:
Evento no-break \r\n";
                                    str = (ROM char* ) EMAIL_LINHA8;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);

                                    //
-----------------------------------------------------------------------
                                    //ROM char EMAIL_LINHA9[] = "Content-
type: text/plain\r\n\r\n";
                                    str = (ROM char* ) EMAIL_LINHA9;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);

                                    //
-----------------------------------------------------------------------
                                    //ROM char EMAIL_LINHA10[] = "Evento
ocorrido no no-break:\r\n";
                                    str = (ROM char* ) EMAIL_LINHA10;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);
                                    //ROM char EMAIL_LINHA10a[] =
"Atendimento ao cliente: (21)2501-6458\r\n";
                                    str = (ROM char* ) EMAIL_LINHA10a;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);
                                    //ROM char EMAIL_LINHA10b[] =
"cliente@guardian.ind.br\r\n";
                                    str = (ROM char* ) EMAIL_LINHA10b;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);


                                    //
-----------------------------------------------------------------------
                                    // .\r\n
                                    str = (ROM char* ) EMAIL_LINHA11;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);
                                    //
-----------------------------------------------------------------------

                                    // Agora transmite
                                    TCPFlush(EMAILSocket);
                              }
                  break;

                  // Deve esperar algo: 250 Message received:
20020107184344, PUJL327,mta07-svc.abc.com@chips
                  case SM_MAIL_RECEIVED:
                        if ( TCPIsGetReady(EMAILSocket) ){  // Se o socket
contem dados recebidos
                              smStateEMAIL = SM_MAIL_QUIT;
                              TCPGetArray(EMAILSocket, buffer_rec, 28);//
Busca dos dados
                              TCPDiscard(EMAILSocket);           // Limpa
o Buffer de recepção
                        }
                  break;
```

```
                  //ROM char EMAIL_LINHA5[] = "QUIT\r\n";
                  case SM_MAIL_QUIT:
                              if (TCPIsPutReady (EMAILSocket) )   {
                                    smStateEMAIL = SM_MAIL_DESCONECT_A;
                                    //Envia o dado
                                    str = (ROM char* ) EMAIL_LINHA5;
                                    while( vv = *str++ )
TCPPut(EMAILSocket, vv);
                              }
                              TCPFlush(EMAILSocket);
                  break;

                  // Deve esperar algo: 221 mta07-svc.abc.com ESMTP
server closing connection
                  case SM_MAIL_DESCONECT_A:
                        if ( TCPIsGetReady(EMAILSocket) ){
      // Se o socket contem dados recebidos
                              smStateEMAIL = SM_MAIL_DESCONECT;
                              TCPGetArray(EMAILSocket, buffer_rec, 8); //
Busca dos dados
                              TCPDiscard(EMAILSocket);
            // Limpa o Buffer de recepção
                        }
                  break;

                  case SM_MAIL_DESCONECT:
                        EMAIL_Time_Out = TickGet();
                        TCPDisconnect(EMAILSocket);
                        smStateEMAIL = SM_CONNECT;
                        SMTP_enviando = 0;
                  // Status de envio de email
                  break;

            }

}
#endif


#if defined (EMAIL)
// Esta função fica verificando o envio de emails
void SMTP_Verificar(void){

      // Ideia: enquanto não estiver transmitindo o time-out do correio é
sempre resetado
      // Enquanto estiver transmitindo o time-out do correio não é
resetado
      // Se o correio não chegar a desconectar estoura o time-out e volta
a transmitir
      // Se estourar o time-out, desconecta, indica voltar a conectar e
reseta o time-out
      if ( ( TickGetDiff(TickGet(),EMAIL_Time_Out ) >= 10*TICK_SECOND) &&
(SMTP_enviando==1) ){
            TCPDisconnect(EMAILSocket);
            smStateEMAIL = SM_CONNECT;
            EMAIL_Time_Out = TickGet();
      // Resetar time-out do correio eletronico
    }
```

```
      if  ( (SMTP_Evento == 1) && (SMTP_enviando == 0) )  {// É para
enviar correio sobre este evento
            SMTP_Evento = 0;
      // indica já providenciado envio deste email
            SMTP_enviando = 1;
            // indica que está enviando e-mail
      }


      if (SMTP_enviando == 1) SMTP_Envia();

      if (SMTP_enviando == 0) EMAIL_Time_Out = TickGet();        // Zera
time-out do correio eletronico

}
#endif


#if defined (EMAIL)
void SMTP_Init(void){

      smStateEMAIL = SM_CONNECT;
      // para servidor de e-mail

      // Endereço IP do Servidor de E-mail
      RemoteNode.IPAddr.v[0]=10;
      RemoteNode.IPAddr.v[1]=0;
      RemoteNode.IPAddr.v[2]=0;
      RemoteNode.IPAddr.v[3]=102;

      // Endereço MAC do servidor de E-mail
      RemoteNode.MACAddr.v[0]= 0x00;
      RemoteNode.MACAddr.v[1]= 0x11;
      RemoteNode.MACAddr.v[2]= 0x09;
      RemoteNode.MACAddr.v[3]= 0x06;
      RemoteNode.MACAddr.v[4]= 0x80;
      RemoteNode.MACAddr.v[5]= 0xa1;

      SMTP_Evento = 0;
      // sem evento de correio eletronico
      SMTP_enviando = 0;
            // indica que não deve enviar email
      EMAIL_Time_Out = TickGet();

}
#endif


#if defined(TCPIP)
void TCPIP_Init(void){
      // Indica que não deve transmitir dados para estação central via
conexão TCP
      Evento_tcp_ip = 0;

      // Estado inicial da máquina de estado conexão TCP
      smState = SM_LISTEN;
}
#endif
```

```
/*********************************************************************
**
 * Minhas Funções auxiliares
 *********************************************************************
**/

/*********************************************************************
*****************************
 * Função:                    void SerialISR(void)
 *
 * Pré-Condição:        Verificar habilitação de interrupções
 *
 * Entrada:                   Nenhuma
 *
 * Saída:                     Nenhuma
 *
 * Outros Efeitos:      Nenhum
 *
 * Função:                    Carrega em um buffer o que a USART recebeu.
 *
 * Notas:                     Vetor de interrupção de baixa prioridade,
no Hyperterminal deve-se teclar ENTER para finalizar o buffer
static unsigned char rec[20];                                          //
buffer de mensagem recebida serial - interrupção
static unsigned char USARTString_rec[20];                        //
buffer de mensagem recebida serial para mostrar no navegador
static unsigned char *p;
      // ponteiro para mensagem
*********************************************************************
***************************/
#pragma interruptlow SerialISR
void SerialISR(void){

      unsigned char wordSerial;

      // RCIF: EUSART Receive Interrupt Flag bit
      // 1: The EUSART receive buffer, RCREG is full (cleared when RCREG
is read)
      // 0: The EUSART receive buffer is empty
//    if(PIR1_RCIF == 1)
            // checa flag de interrupção recepção serial
    if (PIR1bits.RCIF == 1)
    {
//          wordSerial = USARTGet();
            // Ler RCREG fazendo RCIF=0
        wordSerial = RCREG; //le o caracter recebido da serial
//        PIR1bits.RCIF = 0;  //limpa o flag de interrupcao serial
            *p = wordSerial;
            // Armazenar no buffer serial temporário
            p++;
                  // prepara próxima posição de armazenamento de
caracteres do comando
            if(wordSerial == 0x0d)
            // Recebendo <CR> (\n), indica fim do comando recebido
        {
                        *p = '\0';
```

```
            // Finalizar comando recebido
                ReadCommand(&rec[0]);                                   //
Interpreta o comando recebido
                        p=&rec[0];
                  // Volta para posição inicial do buffer
            }
      }
}// retorno através de instrução RETFIE

/*********************************************************************
*******
 * Verifica um comando recebido do PIC monitorado
 *********************************************************************
*******/
void ReadCommand(char* command)
{
    char oper[SIZE_OPER], peripheral[SIZE_PERIPHERAL],
value1[SIZE_VALUE1], value2[SIZE_VALUE2], CONNECT[10], DISCONNECT[13];
    if (strcmp(command, "CONNECT\r\n") == 0)
    {
        monitorOn = TRUE;
    }else if (strcmp(command, "DISCONNECT\r\n") == 0)
    {
        monitorOn = FALSE;
    }else if (monitorOn)
    {
        CommandSplit(&command[0], &oper[0], &peripheral[0], &value1[0],
&value2[0]);   //separa as partes do comando
        InterpretCommand(&oper[0], &peripheral[0], &value1[0],
&value2[0]);   //interpreta comando
    }
}

/*********************************************************************
*******
 * Interpreta um comando recebido, atualizando os estados dos periféricos
fornecidos
 *********************************************************************
*******/
void InterpretCommand(char* oper, char* peripheral, char* value1, char*
value2)
{
    if(strcmp(oper, "INF:S") == 0)
    {
        //INF:S Rp xxx\r\n
        if((peripheral[0] == 'R') && (peripheral[2] == '\0'))
        {
            unsigned char data = atoi(value1);
            UpdPin(peripheral[1], data);
        //INF:S Tp xxx\r\n
        }else if ((peripheral[0] == 'T') && (peripheral[2] == '\0'))
        {
            unsigned char data = atoi(value1);
            UpdTris(peripheral[1], data);
        //INF:S EE xxx xxx\r\n
        }else if ((peripheral[0] == 'E') && (peripheral[1] == 'E') &&
(peripheral[2] == '\0'))
        {
            unsigned char address, data;
            address = atoi(value1);
```

```
            data = atoi(value2);
            UpdEEPROM(address, data);
        //INF:S RAM xxxx xxx\r\n
        }else if ((peripheral[0] == 'R') && (peripheral[1] == 'A') &&
(peripheral[2] == 'M') && (peripheral[3] == '\0'))
        {
            unsigned char address, data;
            address = atoi(value1);
            data = atoi(value2);
            UpdRAM(address, data);
        //INF:S ADCx xxx\r\n
        }else if ((peripheral[0] == 'A') && (peripheral[1] == 'D') &&
(peripheral[2] == 'C') && (peripheral[4] == '\0'))
        {
            unsigned char data;
            data = atoi(value1);
            UpdADC(peripheral[3], data);
        //INF:S TMx xxx\r\n
        }else if ((peripheral[0] == 'T') && (peripheral[1] == 'M') &&
(peripheral[3] == '\0'))
        {
            unsigned char data;
            data = atoi(value1);
            UpdTimer(peripheral[2], data);
        //INF:S PWM xxxx\r\n
        }else if ((peripheral[0] == 'P') && (peripheral[1] == 'W') &&
(peripheral[2] == 'M') && (peripheral[3] == '\0'))
        {
            unsigned char data;
            data = atoi(value1);
            UpdPWM(data);
        }
    }else if (strcmp(oper, "INF:A") == 0)
    {
        //INF:A ADCx x\r\n
        if((peripheral[0] == 'A') && (peripheral[1] == 'D') &&
(peripheral[2] == 'C') && (peripheral[4] == '\0'))
        {
            unsigned char channel, data;
            channel = peripheral[3] - 48;
            data = value1[0] - 48;
            if(data == 1)
                statusADC = statusADC | Exp(2, channel);  //Exp(2,
channel) == 2^channel
            else
                statusADC = statusADC & (255 - Exp(2, channel));
        //INF:A EXTx x\r\n
        }else if ((peripheral[0] == 'E') && (peripheral[1] == 'X') &&
(peripheral[2] == 'T') && (peripheral[4] == '\0'))
        {
            unsigned char channel, data;
            channel = peripheral[3] - 48;
            data = value1[0] - 48;
            if(data == 1)
                statusIntExt = statusIntExt | Exp(2, channel);  //Exp(2,
channel) == 2^channel
            else
                statusIntExt = statusIntExt & (255 - Exp(2, channel));
        //INF:A TMx x\r\n
        }else if ((peripheral[0] == 'T') && (peripheral[1] == 'M') &&
```

```
(peripheral[3] == '\0'))
        {
            unsigned char channel, data;
            channel = peripheral[2] - 48;
            data = value1[0] - 48;
            if(data == 1)
                statusTimer = statusTimer | Exp(2, channel);    //Exp(2,
channel) == 2^channel
            else
                statusTimer = statusTimer & (255 - Exp(2, channel));
        //INF:A PWM x\r\n
        }else if ((peripheral[0] == 'P') && (peripheral[1] == 'W') &&
(peripheral[2] == 'M') && (peripheral[3] == '\0'))
        {
            unsigned char data = value1[0] - 48;
            if(data == 1)
                statusPWM = statusPWM | 1;
            else
                statusPWM = statusPWM & 254;
        //INF:A CAN x\r\n
        }else if ((peripheral[0] == 'C') && (peripheral[1] == 'A') &&
(peripheral[2] == 'N') && (peripheral[3] == '\0'))
        {
            statusCAN = value1[0] - 48;
        //INF:A I2C x\r\n
        }else if ((peripheral[0] == 'I') && (peripheral[1] == '2') &&
(peripheral[2] == 'C') && (peripheral[3] == '\0'))
        {
            statusI2C = value1[0] - 48;
        //INF:A SPI x\r\n
        }else if ((peripheral[0] == 'S') && (peripheral[1] == 'P') &&
(peripheral[2] == 'I') && (peripheral[3] == '\0'))
        {
            statusSPI = value1[0] - 48;
        //INF:A 232 x\r\n
        }else if ((peripheral[0] == '2') && (peripheral[1] == '3') &&
(peripheral[2] == '2') && (peripheral[3] == '\0'))
        {
            statusRS232 = value1[0] - 48;
        }
    }
}

/**************************************************************************
*******
 * Atualiza a informação de valor de um pino
 *************************************************************************
*******/
void UpdPin(char porta, unsigned char data)
{
    switch(porta)
    {
        case 'A':
            RA = data;
        break;
        case 'B':
            RB = data;
        break;
        case 'C':
            RC = data;
```

```
        break;
        case 'D':
            RD = data;
        break;
        case 'E':
            RE = data;
        break;
    }
}

/*************************************************************************
*******
 * Atualiza a informação de tris de um pino
 ************************************************************************
*******/
void UpdTris(char porta, unsigned char data)
{
    switch(porta)
    {
        case 'A':
            TA = data;
        break;
        case 'B':
            TB = data;
        break;
        case 'C':
            TC = data;
        break;
        case 'D':
            TD = data;
        break;
        case 'E':
            TE = data;
        break;
    }
}

/*************************************************************************
*******
 * Atualiza a informação de valores da memória EEPROM
 ************************************************************************
*******/
void UpdEEPROM(unsigned char address, unsigned char data)
{
    unsigned char pos;
    //Retira um valor das filas de endereço e valor
    for(pos = SIZE_FIFO_EEPROM - 1; pos >= 1; pos--)
    {
        addressEE[pos] = addressEE[pos-1];
        valueEE[pos] = valueEE[pos-1];
    }
    addressEE[0] = address;
    valueEE[0] = data;
}

/*************************************************************************
*******
 * Atualiza a informação de valores da memória RAM
 ************************************************************************
*******/
```

```
void UpdRAM(unsigned char address, unsigned char data)
{
    unsigned char pos;
    //Retira um valor das filas de endereço e valor
    for(pos = SIZE_FIFO_RAM - 1; pos >= 1; pos--)
    {
        addressRAM[pos] = addressRAM[pos-1];
        valueRAM[pos] = valueRAM[pos-1];
    }
    addressRAM[0] = address;
    valueRAM[0] = data;
}

/*************************************************************************
*******
 * Atualiza a informação de valores obtidos por um dos conversores a/d
 *************************************************************************
*******/
void UpdADC(unsigned char channel, unsigned char data)
{
    switch(channel)
    {
        case '0':
            valueADC0 = data;
        break;
        case '1':
            valueADC1 = data;
        break;
        case '2':
            valueADC2 = data;
        break;
        case '3':
            valueADC3 = data;
        break;
        case '4':
            valueADC4 = data;
        break;
        case '5':
            valueADC5 = data;
        break;
        case '6':
            valueADC6 = data;
        break;
        case '7':
            valueADC7 = data;
        break;
    }
}

/*************************************************************************
*******
 * Atualiza a informação de valor inicial de um dos contadores Timer
 *************************************************************************
*******/
void UpdTimer(unsigned char channel, unsigned char data)
{
    switch(channel)
    {
        case '0':
            valueTimer0 = data;
```

```
        break;
        case '1':
            valueTimer1 = data;
        break;
    }
}

/*************************************************************************
*******
 * Atualiza a informação de porção do sinal PWM em nível lógico 1
 *************************************************************************
*******/
void UpdPWM(unsigned char data)
{
    valuePWM = data;
}

/*************************************************************************
*******
 * Separa um comando recebido em partes menores
 *************************************************************************
*******/
void CommandSplit(char* word, char* oper, char* peripheral, char* value1,
char* value2)
{
    ////////////comando
    unsigned char counterWord, counter, data;
    oper[0] = '\0';
    peripheral[0] = '\0';
    value1[0] = '\0';
    value2[0] = '\0';
    counter = 0;
    counterWord = 0;
    while((counter<SIZE_OPER-
1)&&(word[counterWord])&&(word[counterWord]!=' '))
    {
        oper[counter] = word[counterWord];
        counter++;
        counterWord++;
    }
    oper[counter] = '\0';
    ////////////periférico do PIC
    if (word[counterWord] == ' ')
    {
        counterWord++;
        counter = 0;
        while((counter<SIZE_PERIPHERAL-1) && (word[counterWord]) &&
(word[counterWord]!=' ') && (word[counterWord]!='\r'))
        {
            peripheral[counter] = word[counterWord];
            counter++;
            counterWord++;
        }
        peripheral[counter] = '\0';
        ////////////valor ou endereço de uma porta ou registrador
        if (word[counterWord] == ' ')
        {
            counterWord++;
            counter = 0;
            while((counter<SIZE_VALUE1)&&(word[counterWord])&&(word[count
```

```
erWord]!=' ')&&(word[counterWord]!='\r'))
            {
                value1[counter] = word[counterWord];
                counter++;
                counterWord++;
            }
            value1[counter] = '\0';
            //////////valor de uma porta ou registrador
            if (word[counterWord] == ' ')
            {
                counterWord++;
                counter = 0;
                while((counter<SIZE_VALUE2)&&(word[counterWord])&&(word[c
ounterWord]!=' ')&&(word[counterWord]!='\r'))
                {
                    value2[counter] = word[counterWord];
                    counter++;
                    counterWord++;
                }
                value2[counter] = '\0';
            }
        }
    }
}

/*************************************************************************
*******
 * Junta strings para obter um comando a ser enviado ao PIC monitorado
 *************************************************************************
*******/
void Concat(char* command, char* typeCommand, char* peripheral, char*
value1, char* value2)
{
    unsigned char counter, pos = 0;
    for(counter = 0; typeCommand[counter]; counter++)
    {
        command[pos] = typeCommand[counter];
        if (command[pos] == '%')
        {
            command[pos] = ':';
            counter = counter + 2;
        }
        pos++;
    }
    if(peripheral[0]!='\0')
    {
        command[pos] = ' ';
        pos++;
    }
    for(counter = 0; peripheral[counter]; counter++)
    {
        command[pos] = peripheral[counter];
        pos++;
    }
    if(value1[0]!='\0')
    {
        command[pos] = ' ';
        pos++;
    }
    for(counter = 0; value1[counter]; counter++)
```

```
    {
        command[pos] = value1[counter];
        pos++;
    }
    if(value2[0]!='\0')
    {
        command[pos] = ' ';
        pos++;
    }
    for(counter = 0; value2[counter]; counter++)
    {
        command[pos] = value2[counter];
        pos++;
    }
    command[pos] = '\r';
    command[pos+1] = '\n';
    command[pos+2] = '\0';
}

/*************************************************************************
*******
 * Converte um valor inteiro de 8 bits para uma string de numDigits
dígitos
 *************************************************************************
*******/
void IntToString(unsigned char data, char* dataChar, unsigned char
numDigits)
{
    unsigned char i;
    for (i = 0; i < numDigits; i++)
        dataChar[i] = 48;
    dataChar[numDigits] = '\0';
    if (data > 0)
    {
        unsigned char counter, counterBuffer;
        char buffer[MAX_SIZE_VALUES];
        counter = 0;
        //vai armazenando a string em ordem inversa dentro de buffer
        while (data > 0)
        {
            buffer[counter] = (data % 10)+48;
            data = data / 10;
            counter++;
        }
        buffer[counter] = '\0';
        counterBuffer = numDigits - 1;
        //transfere string de buffer para dataChar com ordem correta
        for (counter = 0; buffer[counter]; counter++)
        {
            dataChar[counterBuffer] = buffer[counter];
            counterBuffer--;
        }
    }
}

/*************************************************************************
*******
 * Efetua uma exponenciação (base ^ exp)
 *************************************************************************
*******/
```

```
unsigned char Exp(unsigned char base, unsigned char exp)
{
    unsigned char result = 1;
    while(exp > 0)
    {
        result = result * base;
        exp--;
    }
    return result;
}

/*************************************************************************
**
 * Fim de minhas Funções auxiliares
 ************************************************************************
**/
```

ANEXO III – PÁGINAS WEB ARMAZENADAS NO SERVIDOR

Arquivo: “index.htm”

```
<html>

<head>
    <title>SGM-W</title>
</head>

<frameset rows="65,25,*,25" border=0 frameSpacing=0 frameBorder=0>
    <frame name="titulo" src="titulo.htm" marginheight=10 marginwidth=0
scrolling="no">
    <frame name="menu" src="menu.htm" marginheight=2 marginwidth=0
scrolling="no">
    <frame name="conteudo" src="inicial.htm" marginheight=10
marginwidth=0>
    <frame name="ativacao" src="monSt.cgi" marginheight=2 marginwidth=0
scrolling="no">
</frameset>

</html>
```

Arquivo: “menu.htm”

```
<html>

<head>
    <style type="text/css"> {Define padrões das ancoras}
        A:link {COLOR: 000000; TEXT-DECORATION: none} {cor da ancora não
visitada}
        A:visited {COLOR: 000000; TEXT-DECORATION: none} {cor da ancora
visitada}
        A:active {COLOR: 0000FF; TEXT-DECORATION: none} {cor da ancora
ativada}
        A:hover {COLOR: 0000FF; TEXT-DECORATION: none} {cor da ancora ao
passar o mouse em cima}
    </style>
</head>

<body bgcolor="FAF0A6">
    <table border="0" cellpadding="0" cellspacing="0" width="700"
align="center">
        <tr align="center" valign="center">
            <td width="140">
                <a href="inicial.htm" target="conteudo"><font
size="4">Início</font></a>
            </td>
            <td width="140">
                <a href="pinos.htm" target="conteudo"><font
size="4">Pinos</font></a>
            </td>
            <td width="140">
                <a href="servicos.htm" target="conteudo"><font
size="4">Serviços</font></a>
            </td>
            <td width="140">
                <a href="EEPROM.htm" target="conteudo"><font
size="4">EEPROM</font></a>
            </td>
            <td width="140">
                <a href="RAM.htm" target="conteudo"><font
size="4">RAM</font></a>
            </td>
        </tr>
    </table>
</body>

</html>
```

Arquivo: “inicial.htm”

```
<html>

<body>
    <p align="center">
    <font face="arial" color="0000FF" size="6"><br><b>SGM-
W</b><br><br>Sistema de Gerenciamento<br> de Microcontroladores via
Interface Web<br><br></font>

    <font face="arial" color="000000" size="4">
    <b>Desenvolvedor</b><br>
    Thiago Alves de Araújo<br>
    thiagoalves@terra.com.br<br><br>

    <b>Orientador</b><br>
    Wilian Soares Lacerda<br>
    lacerda@ufla.br<br><br>

    <b>DCC</b><br>
    Departamento de Ciência da Computação<br><br>

    <b>UFLA</b><br>
    Universidade Federal de Lavras - MG<br>
    </font>
    </p>

</body>

</html>
```

Arquivo: “monSt.cgi”

```
<html>

<meta http-equiv="refresh" content="3">

<body bgcolor="FAF0A6">
    <table align="center" border="0" cellpadding="0" cellspacing="0">
        <tr>
            <td>
                <font color=FF0000><b>Gerenciador %00tivado</b></font>
            </td>
        </tr>
    </table>
</body>

</html>
```

Arquivo: “ctrl.cgi”

```
<html>

<body>
    <form METHOD=GET action=0>
    <table align="center" border="0" cellpadding="0" cellspacing="0">
        <tr>
            <td>
                <font color=FF0000><b>Enviar comando:</b></font>
            </td>
            <td width="40">
            </td>
            <td>
                <input type=text name=0>
            </td>
            <td width="40">
            </td>
            <td>
                <input type=submit name=1 value="Enviar">

            </td>
        </tr>
    </table>
    </form>
</body>

</html>
```

Arquivo: “pinsMon.cgi”

```
<html>

<meta http-equiv="refresh" content="3">

<body>
    <p align="center">
        <font face="arial" color="FF0000" size="4">Estados dos
pinos</font><br>
    </p>
    <table border="1" cellspacing="0" align="center">
        <tr align="center" valign="center">
            <td>
            </td>
            <td>
                <b>7</b>
            </td>
            <td>
                <b>6</b>
            </td>
            <td>
                <b>5</b>
            </td>
            <td>
                <b>4</b>
            </td>
            <td>
                <b>3</b>
            </td>
            <td>
                <b>2</b>
            </td>
            <td>
                <b>1</b>
            </td>
            <td>
                <b>0</b>
            </td>
            <td>
                <b>Tris</b>
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>PortA</b>
            </td>
            <td>
                --
            </td>
            <td>
                --
            </td>
            <td>
                <img src="led_%06.jpg">
            </td>
            <td>
                <img src="led_%05.jpg">
            </td>
```

```
    <td>
        <img src="led_%04.jpg">
    </td>
    <td>
        <img src="led_%03.jpg">
    </td>
    <td>
        <img src="led_%02.jpg">
    </td>
    <td>
        <img src="led_%01.jpg">
    </td>
    <td>
        %34
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>PortB</b>
    </td>
    <td>
        <img src="led_%14.jpg">
    </td>
    <td>
        <img src="led_%13.jpg">
    </td>
    <td>
        <img src="led_%12.jpg">
    </td>
    <td>
        <img src="led_%11.jpg">
    </td>
    <td>
        <img src="led_%10.jpg">
    </td>
    <td>
        <img src="led_%09.jpg">
    </td>
    <td>
        <img src="led_%08.jpg">
    </td>
    <td>
        <img src="led_%07.jpg">
    </td>
    <td>
        %35
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>PortC</b>
    </td>
    <td>
        <img src="led_%22.jpg">
    </td>
    <td>
        <img src="led_%21.jpg">
    </td>
    <td>
        <img src="led_%20.jpg">
```

```
    </td>
    <td>
        <img src="led_%19.jpg">
    </td>
    <td>
        <img src="led_%18.jpg">
    </td>
    <td>
        <img src="led_%17.jpg">
    </td>
    <td>
        <img src="led_%16.jpg">
    </td>
    <td>
        <img src="led_%15.jpg">
    </td>
    <td>
        %36
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>PortD</b>
    </td>
    <td>
        <img src="led_%30.jpg">
    </td>
    <td>
        <img src="led_%29.jpg">
    </td>
    <td>
        <img src="led_%28.jpg">
    </td>
    <td>
        <img src="led_%27.jpg">
    </td>
    <td>
        <img src="led_%26.jpg">
    </td>
    <td>
        <img src="led_%25.jpg">
    </td>
    <td>
        <img src="led_%24.jpg">
    </td>
    <td>
        <img src="led_%23.jpg">
    </td>
    <td>
        %37
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>PortE</b>
    </td>
    <td>
        --
    </td>
    <td>
```

```
                --
            </td>
            <td>
                --
            </td>
            <td>
                --
            </td>
            <td>
                --
            </td>
            <td>
                <img src="led_%33.jpg">
            </td>
            <td>
                <img src="led_%32.jpg">
            </td>
            <td>
                <img src="led_%31.jpg">
            </td>
            <td>
                %38
            </td>
        </tr>
    </table>
</body>

</html>
```

Arquivo: “servMon.cgi”

```
<html>

<meta http-equiv="refresh" content="3">

<body>
    <p align="center">
        <font face="arial" color="FF0000" size="4">Estados dos
serviços</font><br>
    </p>
    <table border="1" cellspacing="0" align="center">
        <tr align="center" valign="center">
            <td>
                <b></b>
            </td>
            <td>
                <b>Status</b>
            </td>
            <td>
                <b>Valor</b>
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>ADC0</b>
            </td>
            <td>
                %39
            </td>
            <td>
                %40
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>ADC1</b>
            </td>
            <td>
                %41
            </td>
            <td>
                %42
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>ADC2</b>
            </td>
            <td>
                %43
            </td>
            <td>
                %44
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>ADC3</b>
```

```
    </td>
    <td>
        %45
    </td>
    <td>
        %46
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>ADC4</b>
    </td>
    <td>
        %47
    </td>
    <td>
        %48
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>ADC5</b>
    </td>
    <td>
        %49
    </td>
    <td>
        %50
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>ADC6</b>
    </td>
    <td>
        %51
    </td>
    <td>
        %52
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>ADC7</b>
    </td>
    <td>
        %53
    </td>
    <td>
        %54
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>Timer0</b>
    </td>
    <td>
        %55
    </td>
    <td>
```

```
        %56
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>Timer1</b>
    </td>
    <td>
        %57
    </td>
    <td>
        %58
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>PWM</b>
    </td>
    <td>
        %59
    </td>
    <td>
        %60
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>IntExt0</b>
    </td>
    <td>
        %61
    </td>
    <td>
        --
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>IntExt1</b>
    </td>
    <td>
        %62
    </td>
    <td>
        --
    </td>
</tr>
<tr align="center" valign="center">
    <td>
        <b>IntExt2</b>
    </td>
    <td>
        %63
    </td>
    <td>
        --
    </td>
</tr>
<tr align="center" valign="center">
    <td>
```

```
                <b>CAN</b>
            </td>
            <td>
                %64
            </td>
            <td>
                --
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>I2C</b>
            </td>
            <td>
                %65
            </td>
            <td>
                --
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>SPI</b>
            </td>
            <td>
                %66
            </td>
            <td>
                --
            </td>
        </tr>
        <tr align="center" valign="center">
            <td>
                <b>RS232</b>
            </td>
            <td>
                %67
            </td>
            <td>
                --
            </td>
        </tr>
    </table>
</body>

</html>
```

Arquivo: “EEMon.cgi”

```
<html>

<meta http-equiv="refresh" content="3">

<body>
    <p align="center">
        <font face="arial" color="FF0000" size="4">Valores da Memória
EEPROM</font><br>
    </p>
    <table border="1" cellspacing="0" align="center">
        <tr align="center" valign="center">
            <td><b>Posição</b></td>
            <td><b>Valor</b></td>
        </tr>
        <tr align="center" valign="center">
            <td>%70</td>
            <td>%71</td>
        </tr>
        <tr align="center" valign="center">
            <td>%72</td>
            <td>%73</td>
        </tr>
        <tr align="center" valign="center">
            <td>%74</td>
            <td>%75</td>
        </tr>
        <tr align="center" valign="center">
            <td>%76</td>
            <td>%77</td>
        </tr>
        <tr align="center" valign="center">
            <td>%78</td>
            <td>%79</td>
        </tr>
    </table>
</body>
</html>
```

Arquivo: “RAMMon.cgi”

```
<html>

<meta http-equiv="refresh" content="3">

<body>
    <p align="center">
        <font face="arial" color="FF0000" size="4">Valores da Memória
RAM</font><br>
    </p>
    <table border="1" cellspacing="0" align="center">
        <tr align="center" valign="center">
            <td><b>Posição</b></td>
            <td><b>Valor</b></td>
        </tr>
        <tr align="center" valign="center">
            <td>%80</td>
            <td>%81</td>
        </tr>
        <tr align="center" valign="center">
            <td>%82</td>
            <td>%83</td>
        </tr>
        <tr align="center" valign="center">
            <td>%84</td>
            <td>%85</td>
        </tr>
        <tr align="center" valign="center">
            <td>%86</td>
            <td>%87</td>
        </tr>
        <tr align="center" valign="center">
            <td>%88</td>
            <td>%89</td>
        </tr>
    </table>
</body>
</html>
```