UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Leonardo Peronio Bassin

2.4 GHz Wireless Modbus

Canoas, Julho de 2009

Leonardo Peronio Bassin

2.4 GHz Wireless Modbus

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

Departamento:

Engenharia Elétrica

Área de Concentração

Micro-controladores

Professor Orientador:

Eng. Eletr. Augusto Alexandre Durgante de Mattos – CREA-RS: 088003-D

Canoas

2009

FOLHA DE APROVAÇÃO

Nome do Autor: Leonardo Peronio Bassin

Matrícula: 436067171

Título: 2.4 GHz Wireless Modbus

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

Professor Orientador:

Msc. Eng. Eletr. Augusto Alexandre Durgante de Mattos

CREA-RS: 088003-D

Banca Avaliadora:

Dr. Eng. Eletr. Marilia Amaral da Silveira

CREA-RS: 050909-D

Conceito Atribuído (A-B-C-D):

Dr. Eng. Eletr. João Carlos Vernetti

CREA-RS: 045852-D

Conceito Atribuído (A-B-C-D):

Assinaturas:

Autor Leonardo Peronio Bassin	Orientador Augusto Alexandre Durgante de Mattos
Avaliador Marilia Amaral da Silveira	Avaliador João Carlos Vernetti dos Santos

Relatório Aprovado em:

DEDICATÓRIA

Dedico a aos meus pais, pelo apoio, pelas horas de paciência e pelo exemplo, a Deus pela calma e tranqüilidade conquistadas nas horas de estudo, aos colegas pelo companheirismo e opiniões, a minha namorada e sua família pela compreensão nos momentos de ausência.

AGRADECIMENTOS

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

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

Aos colegas pelas opiniões e conselhos e ao Google pela infinita fonte de conhecimento.

EPÍGRAFE

Veritas vos liberabit...

RESUMO

Bassin, Leonardo Peronio - 2.4 GHz Wireless Modbus. 119f. Trabalho de Conclusão de Curso em Engenharia Elétrica - Departamento de Engenharia Elétrica. Universidade Luterana do Brasil. Canoas, RS. 2009.

Este trabalho nasceu com a motivação em pesquisar o protocolo modbus e colocá-lo sob uma nova ênfase, a transmissão do mesmo em um ambiente sem fios obedecendo todos os pré-requisitos do mesmo.

Para este objetivo ser cumprido foram utilizados microcontroladores da família PIC programados com linguagem de alto nível C18, bem como um rádio dedicado para a execução desta função, a qual se mostrou bem sucedida. Também foi elaborado software em microcomputador para a validação e demonstração dos resultados obtidos, este desenvolvido em C++ com ambiente gráfico de programação.

Palavras chave: PIC. MODBUS. C18. Wireless.

ABSTRACT

Bassin, Leonardo 2.4 GHz Wireless Modbus. 119 p. Work of Conclusion of Course in Electrical Engineering - Electrical Engineering Department. Lutheran University of Brazil. Canoas, RS. 2009.

This work started with the research the Modbus protocol and the needed to put it under a new emphasis, the transmission in a wireless environment obeying all the prerequisites of it.

To accomplish this goal some hardware was used and software routines were developed. The PIC family of microcontrollers programmed with C18 high-level language, and a radio dedicated to the implementation of this function were chosen, this proved a successful approach. It was prepared in a microcomputer some software routines for validation and demonstration of results, this was developed in C + + programming language in a graphical environment, also third parties software were used for validation and control of the tool.

Keywords: PIC. MODBUS. C18. Wireless.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RX: Recepção de Dados.

TX: Transmissão de Dados.

CLP: Controlador Lógico Programável.

IEEE: *Institute of Electrical and Electronic Engineers*.

CRC: *Cyclic Redundancy Checksum*.

TCP: *Transmission Control Protocol*.

GFSK: *Gaussian Frequency Shift Keying*.

LCD: *Liquid Cristal Display*.

LISTA DE SÍMBOLOS

F – Farad

V – Volt

S – Segundo

Hz – Hertz

SUMÁRIO

1. INTRODUÇÃO

A indústria, como um todo, é cada dia mais dependente de sistemas com atuação remota ou independente. Assim protocolos de comunicação e sistemas de transmissão de dados são empregados em larga escala. O Modbus, um protocolo amplamente utilizado, tem limitações devido ao meio físico exigindo a existência de cabeamento específico para a utilização do mesmo.

Com o intuito de simplificar a aplicação deste protocolo surge a necessidade de implementação do mesmo com a transmissão de dados sem a necessidade de transmissão por meio de cabos. Assim, será desenvolvido um sistema o qual é totalmente compatível com o modbus e que possa executar algumas operações do set de instruções do protocolo e ao mesmo tempo usar um sistema sem fio de comunicação.

Com as considerações acima propõem-se o objetivo final deste trabalho que é desenvolver um sistema compatível com o protocolo modbus que possa enviar e receber dados através de um sistema de comunicação sem fios. Com o desenvolvimento deste sistema, aplicações antigas poderão ser adaptadas a fim de tornarem-se novamente atrativas ao mercado, visto que o custo das interfaces físicas de comunicação, cabos, pode ser eliminado do projeto.

Assim, este projeto tem por objetivo elaborar um sistema que permite a validação deste conceito, um sistema sem fios capaz de executar comandos do modbus, validando esta possibilidade e verificando suas limitações quanto à perda de dados e alcance de transmissão.

Para a solução do problema proposto o universo de alternativas é amplo podendo ser composto de algumas variáveis que são apresentadas na tabela 1 e estas interferem diretamente na qualidade da solução bem como no tempo e custos de desenvolvimento.

Tabela 1: Variáveis de Projeto.

Variável	Descrição	Como se relaciona com as outras variáveis
Protocolo de comunicação	Existem diferentes protocolos de comunicação, estes obedecem a padrões internacionais que devem ser respeitados.	Dependendo da escolha o Micro-controlador ou o software do PC aumenta em complexidade
Hardware PC	O hardware de transmissão e recepção utilizado no PC	Dependendo da escolha, este pode interferir no hardware remoto e no alcance.
Alcance	Esta variável especifica o alcance da comunicação	Dependendo desta variável o hardware do PC e do sistema remoto podem ser alterados.
Hardware remoto	Hardware utilizado na recepção dos dados.	Esta escolha afeta a escolha do hardware no PC e do software.

Tendo em vista que o principal objetivo deste trabalho é a implementação do modbus em um ambiente sem fio, o estudo detalhado de opções de comunicação foi realizado abaixo; o resumo com estas opções e suas implicações; é mostrado na tabela 2.

Tabela 2: Opções de Comunicação Sem Fio.

Solução	Descrição
Sol. 1	Utilizar um link de comunicação via rádio, com uso de protocolo e hardware desenvolvido totalmente pelo aluno.
Sol. 2	Utilizar um link de comunicação usando um protocolo do tipo 802.11b
Sol. 3	Utilizar um link de comunicação do tipo bluetooth.
Sol. 4	Utilizar um radio pronto implementando a camada de software.

Para a análise das soluções do sistema wireless, a tabela 3 foi criada, a qual pondera cada uma das possíveis soluções anteriores considerando pontos fortes e fracos da solução escolhida.

Tabela 3: Pontos Fracos e Fortes do Sistema Sem Fio.

Nome da Solução	Pontos Fortes	Pontos Fracos
Sol. 1	Hardware de baixo custo.	Link sensível a interferências, hardware de transmissão e recepção devem ser construídos.
Sol. 2	Grande alcance, protocolo robusto, hardware já encontrado em computadores.	Hardware remoto complexo, software remoto complexo, protocolo proprietário.
Sol. 3	Protocolo robusto, hardware já encontrado em computadores. Software remoto simplificado	Alcance mediano, software do Pc complexo.
Sol. 4	Hardware comercial disponível	Complexidade do software pode ser um fator complicador.

A escolha da solução baseou-se principalmente em custo computacional, capacidade de processamento necessária para a implementação da solução, e complexidade de desenvolvimento de hardware. Com isso a escolha da solução 4 foi uma escolha natural visto que a mesma apresenta um equilíbrio entre as variáveis tendo um baixo tempo de desenvolvimento da interface de comunicação entre micro-controlador e o radio e um custo mediano de desenvolvimento de software, assim a implementação fica equilibrada.

Dentre os riscos na execução desta tarefa está o custo computacional exigido pela solução, a qual deve ficar restringida a um micro-controlador simples e de baixo custo que possa ser facilmente encontrado no mercado. Outro fator importante é a validação de dados que deve ser feita de forma independente sem a intervenção do desenvolvedor utilizando uma ferramenta de terceiros.

2. REFERENCIAL TEÓRICO

2.1. MODBUS Communication Protocol

2.1.1. O Protocolo

O protocolo modbus, desenvolvido pela Modicon em 1979, é um protocolo de comunicação de dados utilizado em sistemas de automação industrial. É um dos mais antigos protocolos utilizados em redes de Controladores Lógicos Programáveis (CLP) para aquisição de sinais de instrumentos e comandar atuadores. A Modicon (atualmente parte do grupo Schneider Electric) colocou as especificações e normas que definem o modbus em domínio público. Por esta razão, é utilizado em milhares de equipamentos existentes e é uma das soluções de rede mais baratas a serem encontradas e implementadas na área da automação.

Para controle de acesso, o protocolo modbus usa um mecanismo do tipo mestre-escravo. O mestre, um CLP ou computador, envia mensagens solicitando dos escravos que enviem os dados lidos pela instrumentação ou envia sinais a serem escritos nas saídas para o controle dos atuadores. O protocolo possui comandos para envio de dados discretos (entradas e saídas digitais) ou numéricos (entradas e saídas analógicas).

A Figura 1 mostra um exemplo de um mestre (CLP) com três escravos. Em cada ciclo de comunicação o CLP seleciona um dos escravos e envia um comando a este, podendo ser um comando de leitura ou escrita. Como o protocolo utiliza um controle do tipo mestre-escravo, nenhum dos módulos escravos inicia comunicação a não ser para responder às solicitações do mestre.

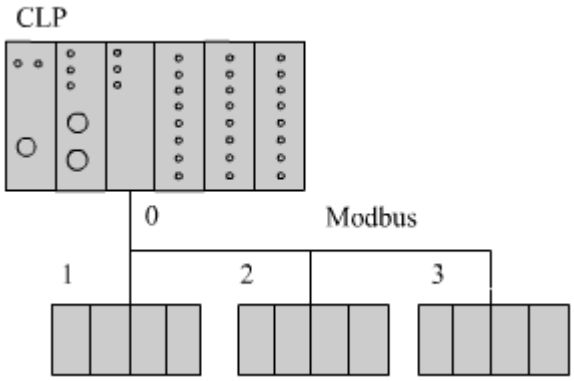

Figura 1: Estrutura Modbus tipo Mestre-Escravo .

Para transmissão de dados o protocolo utiliza como meios físicos de transmissão às interfaces RS-232, RS-485 ou Ethernet, as quais são definidas de acordo com padrões específicos e regulamentadas pelo IEEE. Estas especificações de meios de transmissão são as únicas disponíveis de acordo com o protocolo.

Para a transmissão dos dados, o protocolo modbus utiliza um encapsulamento padrão. Este pacote contém todas as informações necessárias para um escravo previamente selecionado executar uma operação. Na Figura 2 tem-se um frame completo de dados mostrando esta estrutura.

No frame de dados da figura 2, há a seguinte estrutura de dados: Endereço, que é o endereço do escravo; Função, que representa a função a ser executada; Dados, são os dados necessários para a execução da função; e CRC, usado na validação dos dados recebidos.

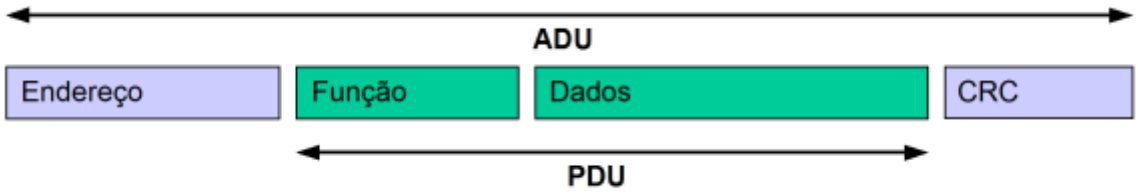

Figura 2: Estrutura de Dados.

Também na Figura 2 estão representados os campos ADU e PDU, do inglês *"application data unit"* (unidade de dados da aplicação) e "protocol data unit" (unidade de dados do protocolo). O ADU pode ser variável dependendo do protocolo selecionado para a transmissão dos dados, já o PDU é invariável e independente do tipo de meio selecionado para a transmissão.

O ADU pode ser configurado de quatro formas diferentes. Quando utilizados meios de transmissão serial, RS-232 por exemplo, o protocolo apresenta

duas variações: pode seguir os modos Modbus RTU ou Modbus ASCII. Já no modo Ethernet de comunicação o protocolo assume duas variações: pode ser Modbus/TCP ou Modbus Plus. Este ultimo é o único modo de comunicação que necessita de licenciamento para ser utilizado.

No modo RTU, os dados são transmitidos em formato binário de oito bits, permitindo a compactação dos dados em pequenos pacotes. Também no modo RTU, os endereços e valores podem ser representados em formato binário, os números inteiros, que podem variar entre -32768 e 32767, são representados por 2 bytes. Já no modo ASCII, que utiliza caracteres ASCII de sete bits, o mesmo número precisaria de quatro caracteres para ser representado (em hexadecimal), apesar do maior consumo de dados esta variação permite que o protocolo seja legível para as pessoas sem o uso de recursos computacionais extras.

Quando operando em modo TCP, os dados são encapsulados em formato binário em frames TCP para a utilização do meio físico Ethernet (IEEE 802.3). Com a escolha do Modbus/TCP, o mecanismo de controle de acesso é o CSMA-CD (Próprio da rede Ethernet) e as estações utilizam o modelo cliente-servidor. Em modo Plus, o protocolo continua utilizando o formato TCP, mas são acrescentados vários recursos adicionais de roteamento, diagnóstico, endereçamento e consistência de dados.

Para o controle da transmissão respostas de sucesso e de erro são implementadas. Nas figuras 3 e 4 existem exemplos de uma transmissão bem sucedida e uma com erros, os erros são codificados transformando o bit mais significativo do Function Code em nível lógico 1.

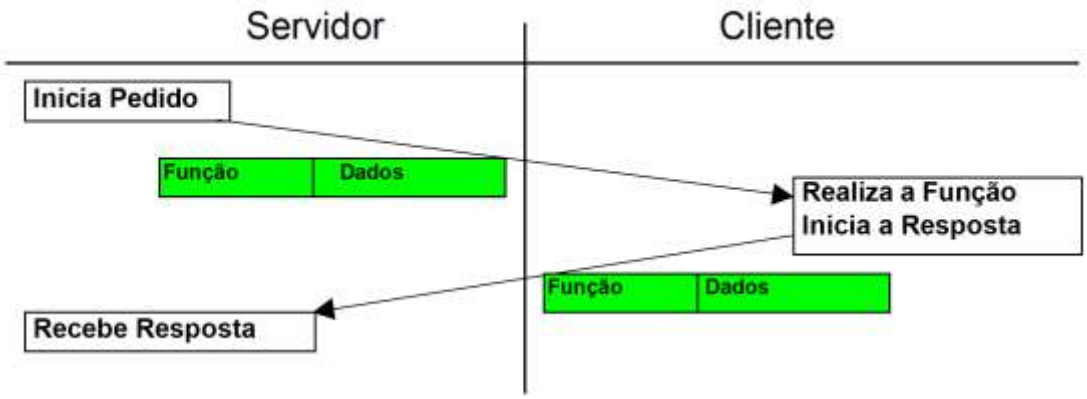

Figura 3: Transmissão de dados com sucesso.

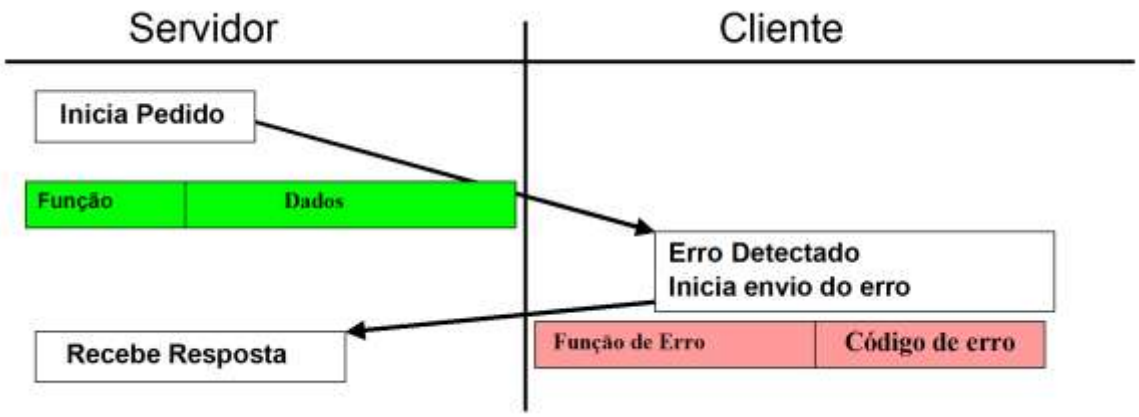

Figura 4: Transmissão de dados com falha no protocolo.

Na transmissão de frames com erros, o protocolo programa códigos de erros ajudando o servidor no tratamento de dados. Também é recomendada a utilização de um temporizador para o controle das transmissões, pois não existe garantia de resposta por parte do cliente.

O PDU do modbus é limitado pelas características do frame máximo suportado pelo protocolo RS485 no qual o ADU é de 256 bytes. Com isso descontando o byte de endereçamento e os dois bytes de CRC, o PDU fica limitado em 253 bytes, e ele poderá ser de três tipos: pedido, resposta ou erro. Estes estão definidos abaixo.

PDU de Pedido, pdu_ped:

Pdu_ped={função,dados requisitados} onde

Função: 1byte, é definido de acordo com o Modbus Function Table, figura 5.

Dados: de tamanho variável podem representar entradas e saídas digitais, entradas e saídas analógicas, ou registradores internos.

PDU de Resposta, pdu_req:

Pdu_ped={função,dados processados} onde

Função: 1byte, é definido de acordo com o Modbus Function Table, Figura 5.

Dados: de tamanho variável podem representar entradas e saídas digitais, entradas e saídas analógicas, ou registradores internos.

PDU de erro, pdu_error:

Pdu_error={ função de exceção,dados requisitados} onde

Função de exceção: 1byte, é a função original somada a 0x80h.

Dados: 1byte, é definido de acordo com o Modbus Function Error Table, Figura 6.

				Function Codes			
				code	*Sub code*	*(hex)*	*Section*
Acesso a Dados	**Acesso a BIT**	Entrada de Dados	Le de Dados Discretos	**02**		**02**	6.2
		Internal Bits Or Physical coils	Le de Saída	**01**		**01**	6.1
			Escreve uma Saída	**05**		**05**	6.5
			Escreve multiplas Saída	**15**		**0F**	6.11
	Acesso 16 bits	Physical Input Registers	Read Input Register	**04**		**04**	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	**03**		**03**	6.3
			Write Single Register	**06**		**06**	6.6
			Write Multiple Registers	**16**		**10**	6.12
			Read/Write Multiple Registers	**23**		**17**	6.17
			Mask Write Register	**22**		**16**	6.16
			Read FIFO queue	**24**		**18**	6.18
	Escrita de Arquivos		Read File record	**20**		**14**	6.14
			Write File record	**21**		**15**	6.15
	Diagnóstico		Read Exception status	**07**		**07**	6.7
			Diagnostic	**08**	**00-18,20**	**08**	6.8
			Get Com event counter	**11**		**OB**	6.9
			Get Com Event Log	**12**		**0C**	6.10
			Report Slave ID	**17**		**11**	6.13
			Read device Identification	**43**	**14**	**2B**	6.21
	Outros		Encapsulated Interface Transport	**43**	**13,14**	**2B**	6.19
			CANopen General Reference	**43**	**13**	**2B**	6.20

Figura 5: Códigos de Função.

MODBUS Exception Codes		
Code		
01	Função Ilegal	A função código recebida na consulta não é um recurso admissível para o servidor (ou escravo).
02	Endereço de Dados Ilegal	Os dados recebidos no endereço da consulta não são dados de endereço admissíveis para o servidor (ou escravo).
03	Valor de Dados Ilegal	Um valor contido na consulta de dados não é um campo de valor admissível para o servidor (ou escravo).
04	Falha no Escravo	Erro irrecuperável de processamento.
05	Conhecimento	Utilização em conjunto com a programação de comandos. Pode demorar um longo tempo para ser executada.
06	Dispositivo Remoto Ocupado	Utilização em conjunto com a programação de comandos. Pode demorar um longo tempo para ser executada.
08	Erro de Paridade de Memória	Utilização especial, em conjugação com a função códigos 20 e 21

Figura 6: Códigos de Erro.

O processo dos pedidos do mestre obedece a uma seqüência predefinida pelo protocolo modbus. Na figura 7 este fluxograma é explorado mostrando como as diferentes respostas de erros implementadas no projeto e o como o escravo processa um pedido do mestre. Esta seqüência deve ser preservada para manter-se a compatibilidade com o protocolo.

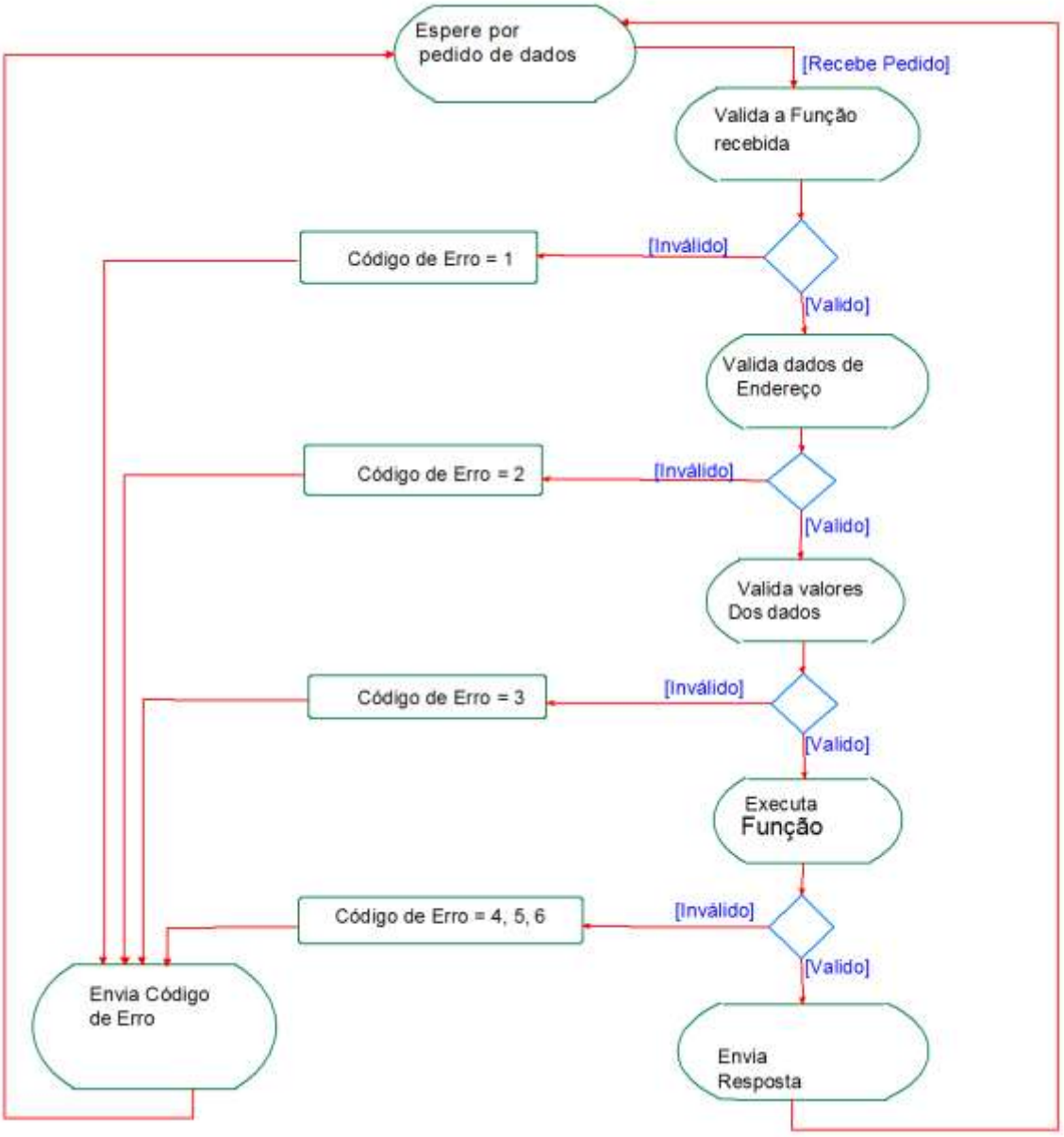

Figura 7: Estrutura de Processamento de Dados.

O protocolo modbus define quatro tipos de dados, os quais são mostrados na figura 8. Eles tem tamanho delimitado e definições fixas dentro do protocolo. Também existem restrições quanto ao tipo de operação, leitura ou escrita, que pode ser executada naqueles.

Objetos	Tipo de Objeto	Ações	Comentários
Entradas Discretas	bit	Leitura	Tipo de dados de sistemas de entrada, somente pode ser lido.
Saídas	bit	Leitura Escrita	Dado pode ser alterado pela aplicação.
Entradas Analógicas	16-bit word	Leitura	Tipo de dados de sistemas de entrada, somente pode ser lido.
Registradores	16-bit word	Leitura Escrita	Dado pode ser alterado pela aplicação.

Figura 8: Tipos de Dados.

Devidos às limitações do hardware escolhido e as múltiplas variações do protocolo e suas possíveis implementações, foram implementadas as funções de um a seis do protocolo, as quais são analisadas no capitulo 3 descrevendo o fluxo padrão de dados e um exemplo de frame de envio e recebimento para cada função a ser analisada.

2.2. O transceptor

Para a implementação do sistema sem fio utilizou-se um transceptor TRF-2.4G. Este rádio utiliza a modulação GFSK (Chave por alteração de freqüência gaussiana) e uma freqüência de transmissão de 2.4GHz. Neste modelo de modulação, os dados são codificados na forma de variações em freqüência com uma onda portadora.

O processo de transmissão de dados é efetuado da seguinte forma: antes da onda retangular, a qual contém as informações a serem transmitidas, passar pelo modulador, esta passa por um filtro gaussiano, conforme a Figura 9, para reduzir a largura espectral dos pulsos de dados. O filtro gaussiano funciona como um "formatador" de pulsos que tem a função de suavizar a transição entre os valores dos pulsos que são enviados a antena. A Figura 9 ilustra a transformação dos pulsos após passarem pelo filtro gaussiano.

Figura 9: Filtro Gaussiano e modulador

No capitulo 3, serão discutidas as particularidades do módulo escolhido e sua implementação em termos de hardware e software bem como os fluxogramas implementados.

2.3. O Micro-controlador

"Um micro-controlador é um computador programável, em um chip otimizado para controlar dispositivos eletrônicos. É uma espécie de microprocessador com memória e interfaces de E/S (I/O) integrados, enfatizando sua auto-suficiência, em contraste com um microprocessador de propósito geral, o mesmo tipo usado nos PC' s, que requer chips adicionais para prover as funções necessárias. [...] Os micro-controladores são componentes utilizados em muitos tipos de equipamentos eletrônicos, sendo a grande maioria entre os chips vendidos. Cerca de 50% são controladores "simples", outros 20% são processadores de sinais digitais (DSP's) mais especializados. Os micro-controladores podem ser encontrados em máquina de lavar, forno de microondas, telefones, etc." (1).

Para o micro-controlador, as funções e suas particularidades serão abordadas no capitulo 3, onde a implementação de hardware e software será mostrada.

3. MATERIAIS E MÉTODOS

3.1. Descrição Geral do Sistema

O sistema completo é composto de dois módulos escravos os quais estão conectados ao microcomputador através de um enlace wireless, os escravos somente executam operações quando estas são requisitadas pelo mestre, no caso desta aplicação um microcomputador e sempre retornam dados validando ou não a execução desta função.

Abaixo se encontram detalhamentos sobre os subitens de cada um dos componentes envolvidos bem como o diagrama esquemático das funções implementadas e do hardware escolhido.

3.2. Descrição dos Sistemas Eletroeletrônicos

3.2.2. Hardware

O rádio escolhido para a transmissão de dados foi o módulo Laipac TRF-2.4g, figura 10[3], o qual usa um transceptor interno, um chip VLSI modelo nRF2401 com um cristal oscilador 16MHz e uma antena tipo dipolo integrada no hardware. Os benefícios do transceptor Laipac, além de um receptor / transmissor RF padrão, é a aplicação em um único chip / dispositivo. Com operação de comunicação bidirecional, dois canais independentes de operação estão disponíveis, juntamente de um hardware com capacidade de verificação de erro; *Cyclic Redundancy Checksum* (CRC); e alta velocidade de transmissão; *ShockBurst*. As três últimas características são úteis para o micro-controlador, uma vez que o processamento necessário para programar um nível inferior de dados do protocolo é transferido para o hardware da Laipac.

Figura 10: Rádio Laipac.

O transceptor pode operar em dois modos: modo ShockBurst e modo Direto Tx / Rx[3]. Em modo ShockBurst, o dispositivo manipula o preâmbulo e o CRC, e transmite os dados em uma alta taxa de transmissão 250kpbs ou 1Mbps (definida pelo usuário). Quando um pacote válido de dados é recebido, o dispositivo irá notificar o micro-controlador através de um pino de interrupção, e os dados recebidos permanecerão em um buffer até que o micro-controlador faça a leitura dos mesmos. No modo direto, o transceptor funciona como um modulador, o qual modula a seqüência de dados recebidos para uma moduladora de 2400MHz e posterior transmissão. O micro-controlador teria que criar um preâmbulo, para acompanhar os dados recebidos para verificar um pacote válido, e lidar com verificação de erro no software. Pelos motivos apresentados, optou-se por operar o dispositivo em modo *ShockBurst*; com isso, o micro-controlador fica livre para processar outras tarefas. Na figura 11 existe um esquemático deste modo de transmissão.

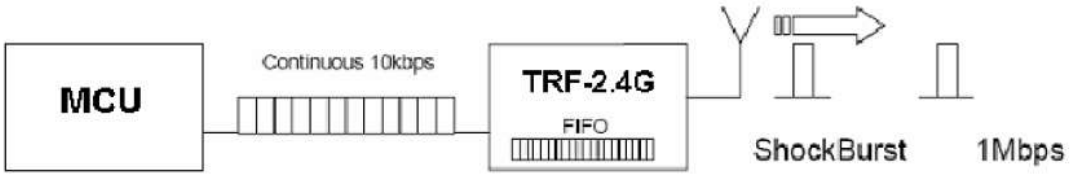

Figura 11: Transmissão de dados *ShockBurst*.

O modo de configuração bem como a rotina de transmissão e recepção de dados serão abordados no descritivo do software do micro-controlador.

Para o controle de dados, leitura de portas e leitura de dados analógicos, optou-se por um micro-controlador fabricado pela Microchip. Dentre as famílias disponíveis foi escolhido um PIC18f4620, o qual tem sua pinagem mostrada na figura 12. Este dispositivo tem uma arquitetura RISC e um tempo de execução de

instruções de 4 ciclos de clock, o qual é de 16Mhz, resultando em uma instrução a cada 0.25us.

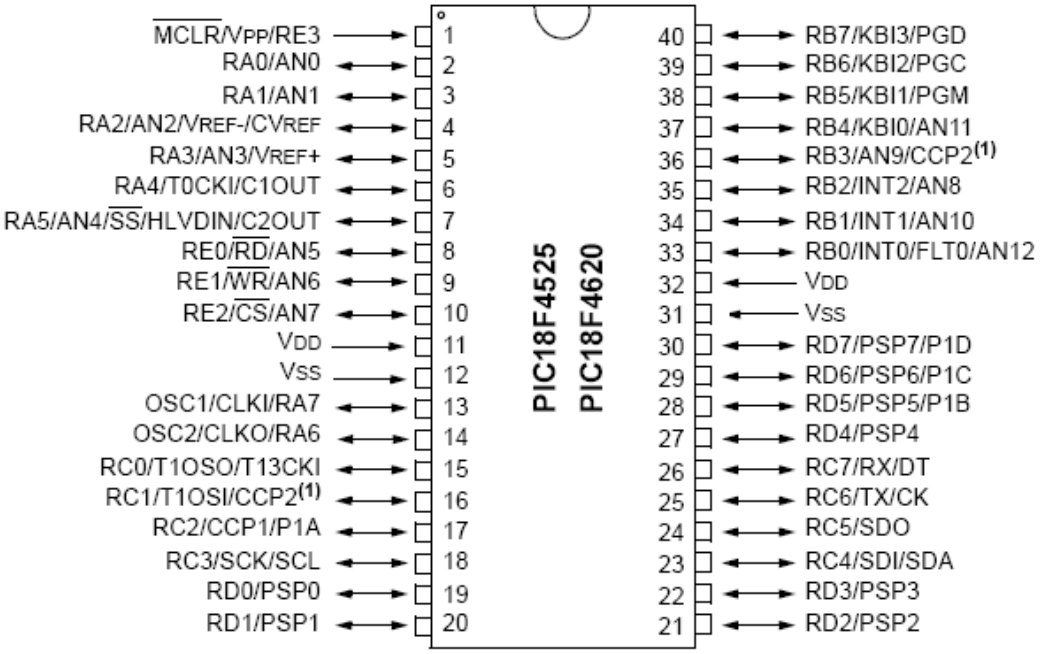

Figura 12: Diagrama de pinos do PIC.

A configuração do PIC é feita através de um hardware dedicado de programação e utilizando-se os pinos PGD e PGC; já as entradas de AN0 a AN3 são utilizadas para as leituras analógicas que podem ir de 0 a 5V. Para as entradas digitais são implementados os pinos de RB0 a RB3; já os pinos de RC0 a RC3 são os responsáveis pelas saídas digitais; os pinos de RD4 a RD7 e os pinos RD0 a RD2 são usados para o controle e endereçamento do display[2].

O display LCD, utilizado para visualização de dados nos dispositivos remotos, é do tipo 2x16 o qual tem um controlador modelo LCD HD44780, este comumente disponível no mercado local de eletrônicos. Na figura 13 um esquemático deste e na figura 14, o diagrama de pinos. Para a interface de dados, foi selecionado o modo de operação de 4 bits o qual permite menor utilização de pinos no micro-controlador.

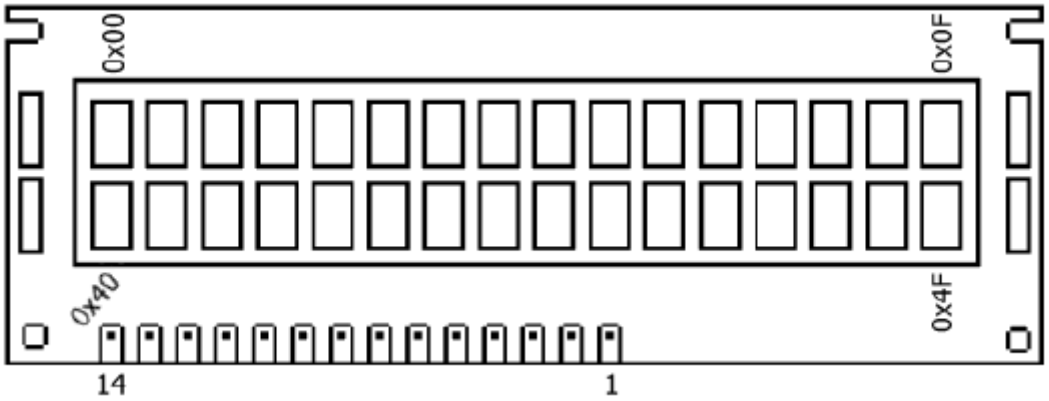

Figura 13: Display LCD

Pino	Nome	Função	Descrição
1	Vss	Alimentação -	Terra (GND)
2	Vdd	Alimentação +	+ 5 V
3	Vee	Contraste	0 até 5 V
4	RS	Comando/Dado	0 : Comando 1 : Dado
5	R/W	Escrita/Leitura	0 : Escrita 1 : Leitura
6	E	Habilitação	1 Habilita Display
7	D0	Dado	Dado (bit - signif.)
8	D1	Dado	Dado
9	D2	Dado	Dado
10	D3	Dado	Dado
11	D4	Dado	Dado
12	D5	Dado	Dado
13	D6	Dado	Dado
14	D7	Dado	Dado (bit + signif.)

Figura 14: Pinos do Display LCD.

Para a comunicação entre o microcomputador e os demais dispositivos um terceiro micro-controlador deve ser implementado, que usa as mesmas rotinas de comunicação entre ele e o transceptor. Em seu hardware, foram eliminados todos os dispositivos secundários restando somente o hardware da Laipac, para a interface entre os dispositivos remotos e um hardware RS-232 que é discutido nos parágrafos abaixo.

Para a interface entre o receptor de dados entre o microcomputador que faz o envio e o processamento de dados dos escravos é utilizado um circuito com um CI MAX232. Este é um circuito eletrônico que converte sinais de uma porta serial para sinais adequados para uso em circuitos micro processados. O MAX232

amplifica/reduz sinais RX, TX, CTX e RTS. A discrepância de voltagem entre os sistemas, micro-controlador – microcomputador, é gerada por capacitores de 10 uF. Cada receptor disponível no circuito converte entradas TIA/EIA-232-F para níveis de 5V TTL/CMOS, já o transmissor converte níveis de entrada TTL/CMOS; em níveis TIA/EIA-232-F. Na figura 15, é mostrado o esquemático elétrico usado para esta tarefa.

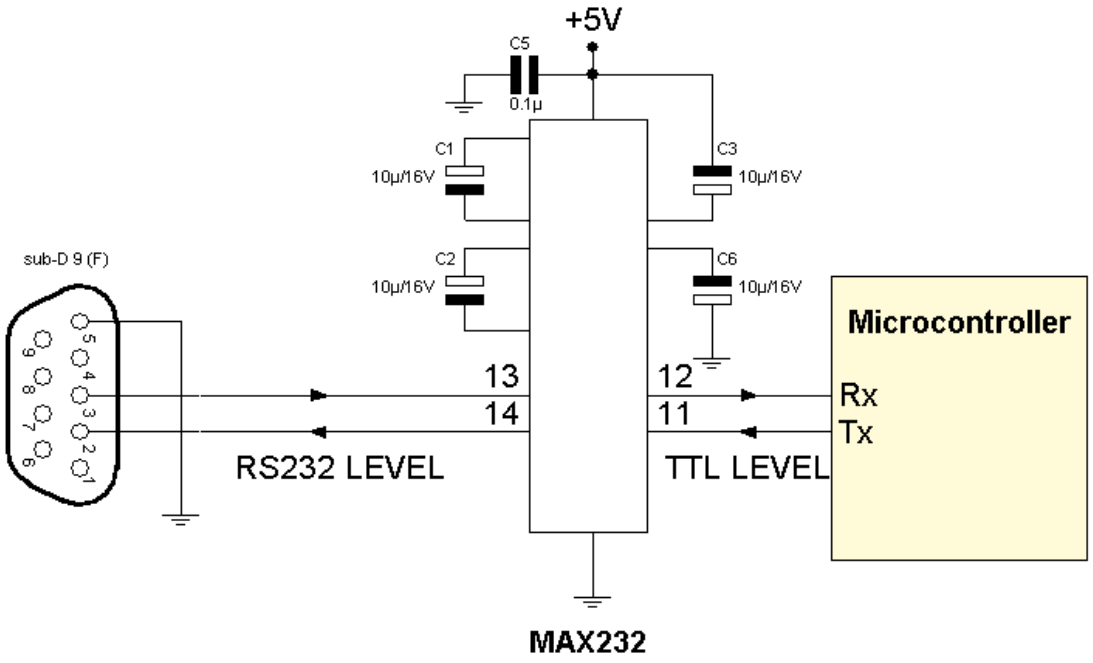

Figura 15:Hardware Max232 entre Microcomputador e Micro-Controlador.

Para a interface de comunicação entre o micro-controlador e o transceptor utilizado, um hardware dedicado teve de ser construído. Este tem por objetivo fazer a transição de nível de tensão entre os dois componentes já que o micro-controlador opera em 5V e o transceptor opera em 3V.

Esta interface está fundamentada sobre o artigo de aplicação da Philips Semicondutores AN97055[4], que descreve um "*level translator*" bidirecional para ser usado em um bus de dados I2C. O hardware foi adaptado para a solução atual e o esquemático deste encontra-se na Figura 16.

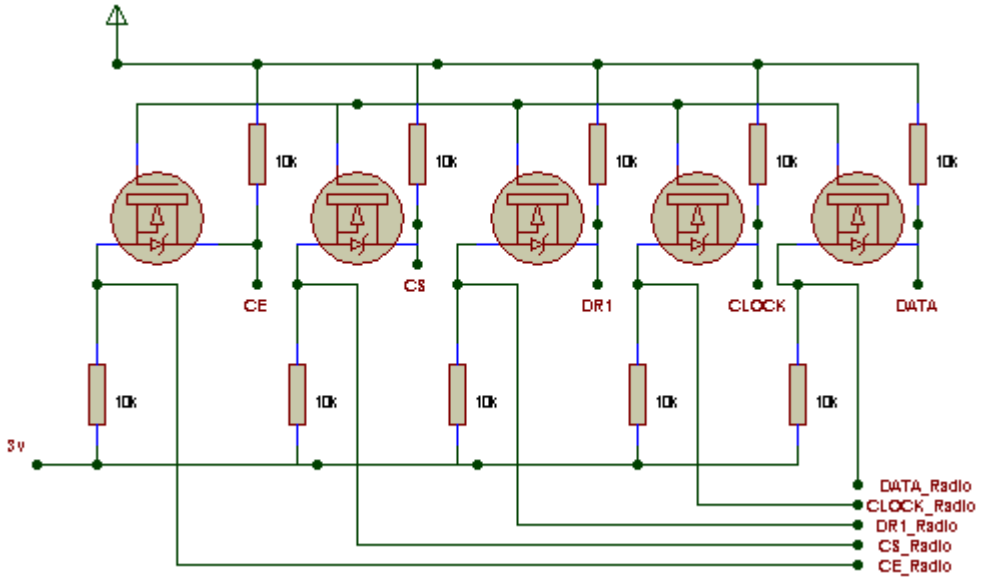

Figura 16: Conversor de Nível Bidirecional.

Os pinos CE, CS, DR1, CLOCK e DATA são ligados diretamente no micro-controlador tendo seus níveis de tensão entre 0V e 5V; já os pinos CE_Radio,CS_Radio,DR1_Radio,CLOCK_Radio e DATA_Radio estão conectados no transceptor e têm seus níveis de tensão variando entre 0V e 3.3V que estão dentro da faixa de trabalho do transceptor[4].

Nas figuras 17 e 18, encontram-se os esquemas elétricos completos do hardware remoto e do hardware ligado ao microcomputador; ambos são alimentados com 5V e tem em seu *"core" um* micro-controlador 18f4620 e um transceptor Laipac TRF-2.4G. Na figura 19, existe o esquemático ampliado do micro-controlador em um hardware remoto para melhor visualização deste.

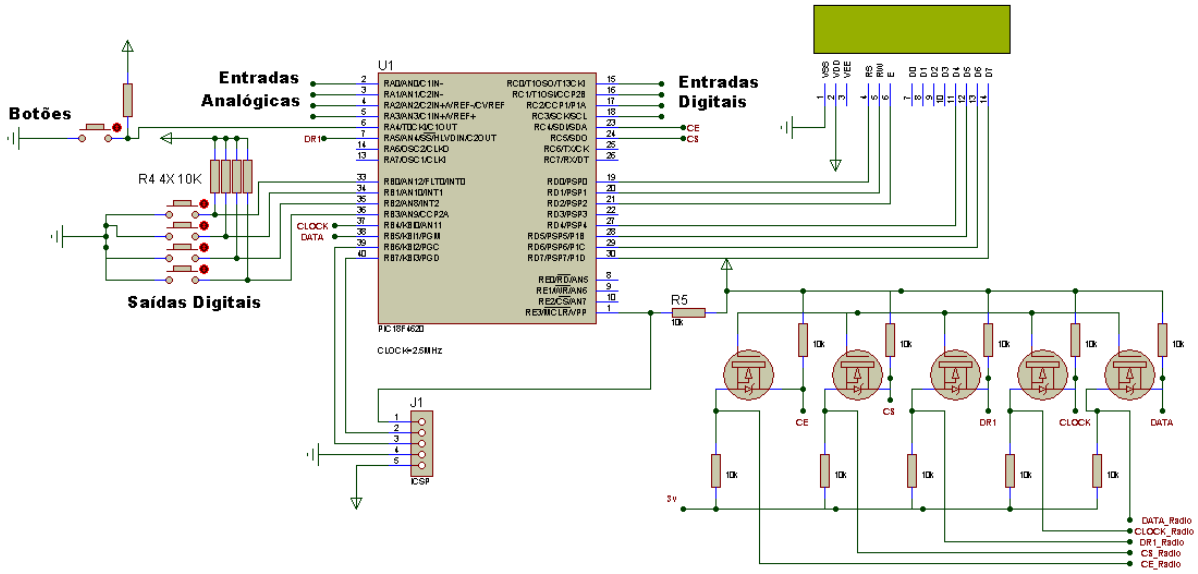

Figura 17: Esquema completo do Hardware Remoto.

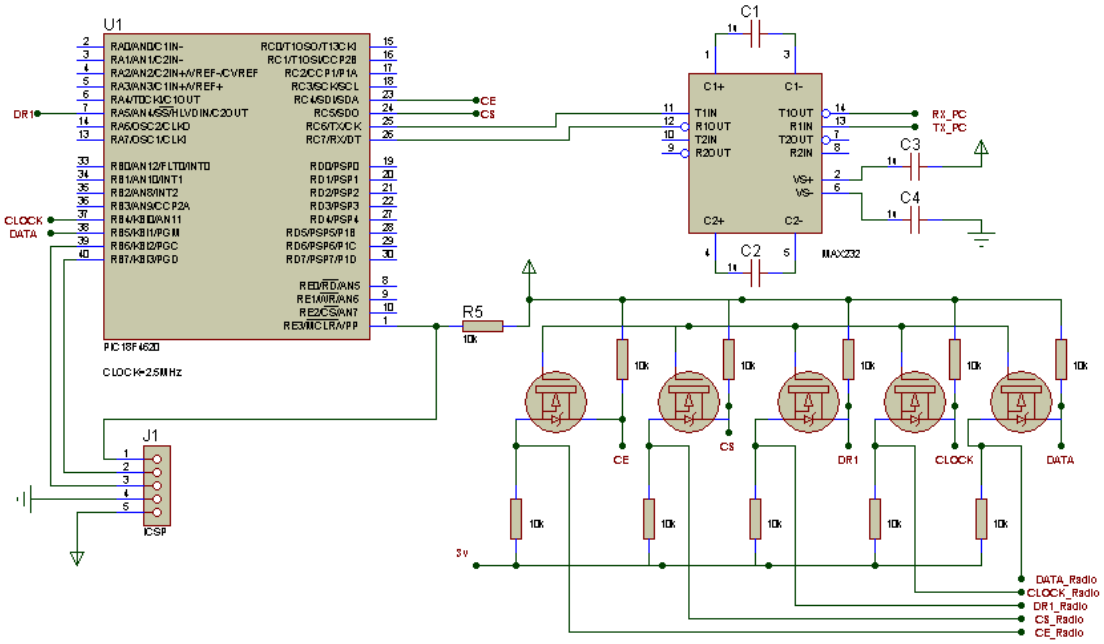

Figura 18:Esquemático do Hardware do Microcomputador.

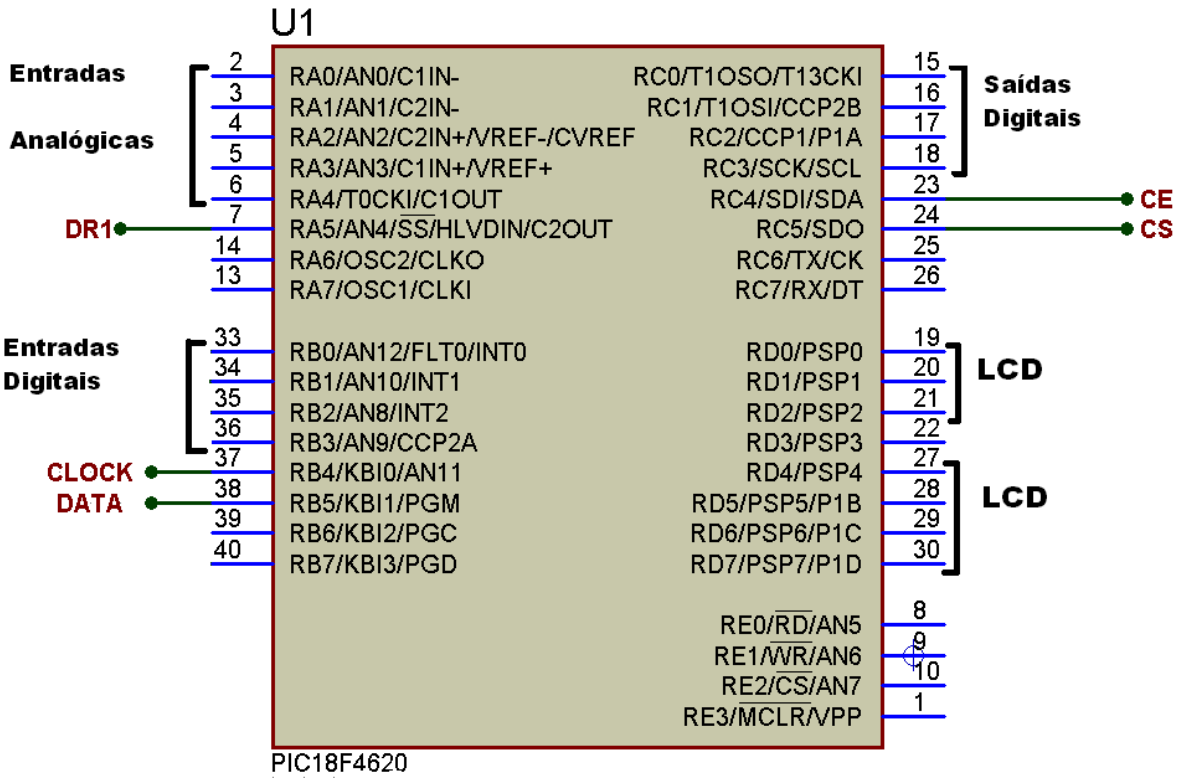

Figura 19: Micro-controlador Hardware Remoto.

3.3. *Descrição dos Sistemas Computacionais*

3.3.3. Software do Micro-controlador

Para o desenvolvimento do software do micro-controlador, foi escolhida a ferramenta de desenvolvimento MPLAB 8.3 e dentro, desta a linguagem de programação escolhida é o C18, que é uma linguagem ANSY C com bibliotecas específicas para os micro-controladores da linha 18 da Microchip.

A solução para a gravação e depuração do *"firmware"* foi o ICD2br. Este kit é licenciado pela Microchip e suporta a maioria dos microprocessadores da linha PIC e DSPIC. O dispositivo se comunica com o MPLAB via USB, tem funções de gravação e depuração dos microprocessadores em tempo real. A figura 20 apresenta o kit ICD2br.

Figura 20: Gravador, depurador ICD2br.

O software do dispositivo remoto opera da maneira especificada no protocolo ModBus, figura 21. Este contem somente um loop adicional que engloba o loop modbus, figura 22[5], e faz o monitoramento das variáveis analógicas e do estado do botão atualizando as informações no display conforme necessário.

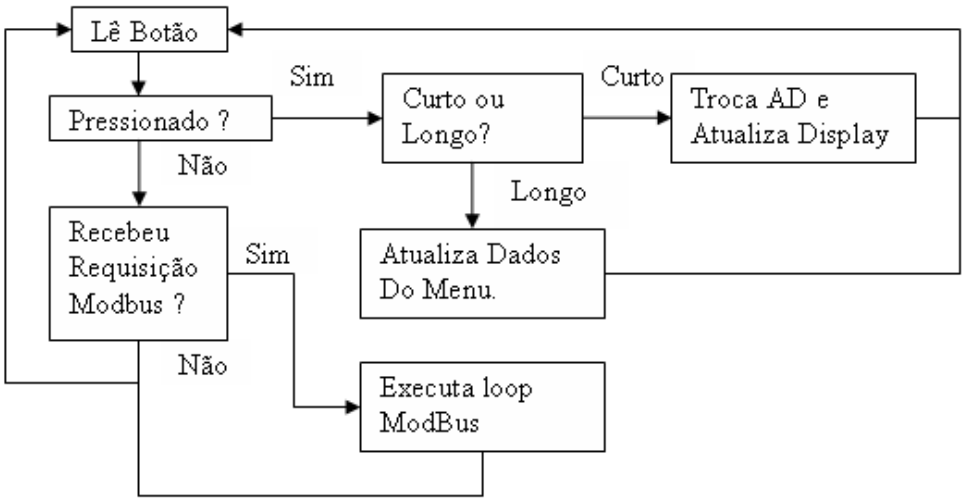

Figura 21: Loop Geral do micro-controlador.

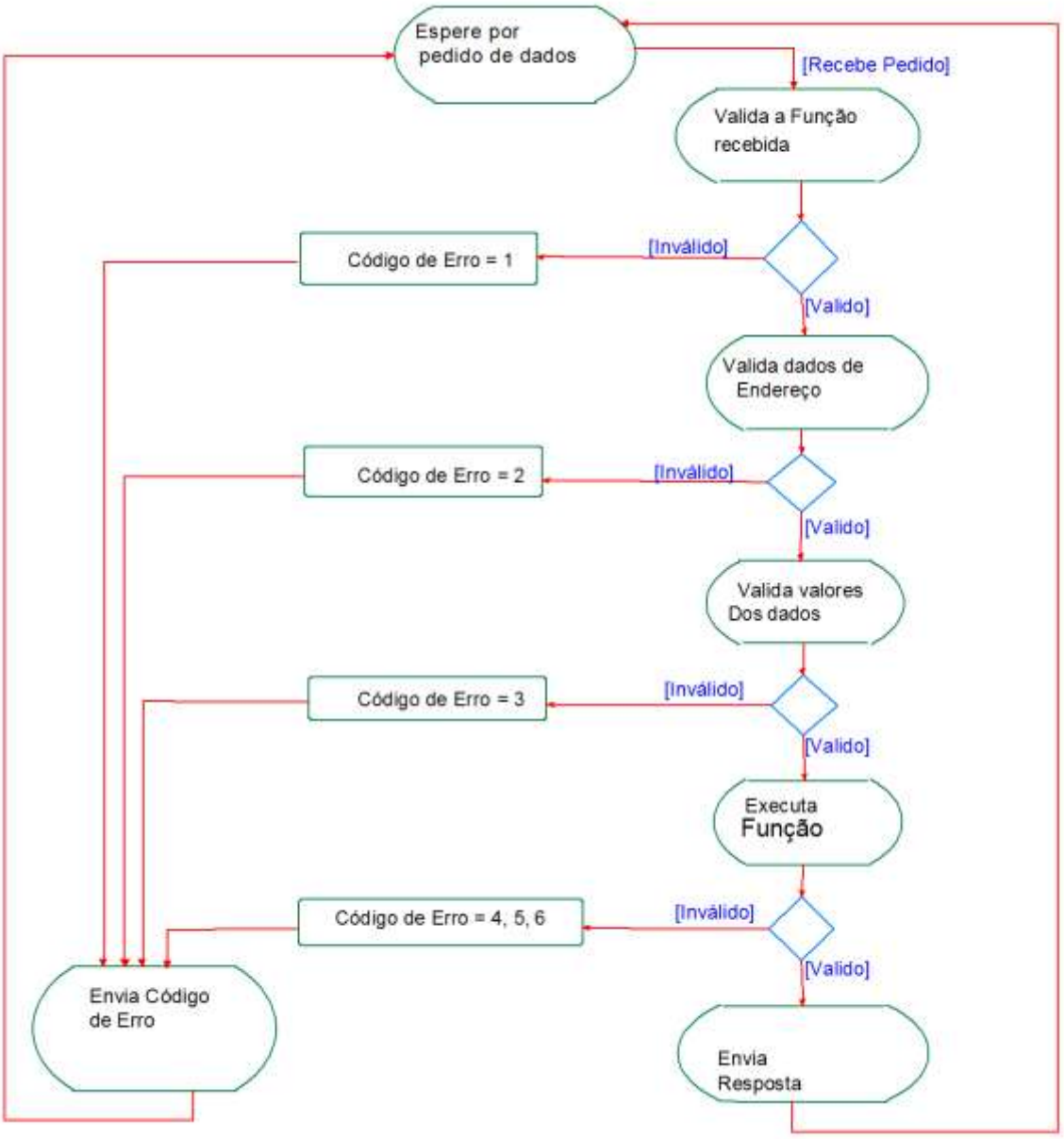

Figura 22: Loop Modbus.

O loop de testes do modbus é dividido de acordo com as funções implementadas abaixo, Em cada uma das funções é mostrado o fluxograma de cada função e um exemplo de implementação da mesma dentro do micro-controlador. O código fonte completo encontra-se no apêndice 1.

3.3.4. Funções ModBus Implementadas

O código 01, (0x01) Leitura de Saída, é o código de função usado para ler o estado das saídas de 1 a 2000 em um dispositivo remoto. O Pedido PDU especifica o endereço inicial, i.e. o endereço da primeira saída e o número de saídas. No PDU, o endereço inicial é zero. Então, as saídas numeradas de 1-16 são endereçadas como

0-15.

A resposta é acumulada por saída, bit, dentro de uma área do campo de dados. A resposta é enviada tratando o estado das saídas como 1 = Ligado e 0 = Desligado. O LSB do primeiro byte de dados contém o estado do endereço inicial. As outras saídas seguem em seqüência, compondo o byte que contém informações sobre as 8 primeiras , seguindo esta ordem nos bytes subseqüentes.

Se a quantidade de saídas requisitada não for um múltiplo de oito, serão alocados zeros (do mais alto para o mais baixo) nos bits restantes do byte. O campo de número de Bytes especifica a quantidade de bytes completos de dados.

Pedido, tabela 4.

Tabela 4: Pedido Função 1.

Código da função	1 Byte	0x01
Endereço inicial	2 Bytes	0x0000 to 0xFFFF
Quantidade de Saídas	2 Bytes	1 to 2000 (0x7D0)

Resposta, tabela 5.

Tabela 5: Resposta Função 1.

Código da função	1 Byte	0x01
Nº de bytes	1 Byte	N*
Estado das Saídas	n Byte	n = N or N+1

*N = Numero de Saídas / 8, se resto diferente de 0, N = N+1.

Erro, tabela 6.

Tabela 6: Erro Função 1.

Código da função	1 Byte	Código da função + 0x80
Código de erro	1 Byte	01 or 02 or 03 or 04

Na tabela 7, é mostrado um exemplo de um pedido de leitura das saídas de numero 20-38[5]:

Tabela 7: Exemplo Função 1.

Pedido		Resposta	
Endereço	(Hex)	Endereço	(Hex)
Código da função	01	Código da função	01
Endereço inicial Hi	00	Nº de bytes	03
Endereço inicial Lo	13	Estado das Saídas 27-20	CD
Quantidade de Saídas Hi	00	Estado das Saídas 35-28	6B
Quantidade de Saídas Lo	13	Estado das Saídas 38-36	05

É mostrado o estado das saídas 27-20 como o byte de valor CD em hexadecimal ou binário 1100 1101. A saída 27 é o MSB deste byte e a saída 20 é o LSB.

Através de convenção, é mostrado dentro de um byte o MSB à esquerda e o LSB à direita. Assim, as saídas mostradas no primeiro byte são de 27 a 20. O próximo byte tem informações sobre as saídas 35 a 28. Como são transmitidos em série os bits, eles fluem do LSB ao MSB: 20. . . 27, 28 . . . 35, e assim por diante.

É mostrado o estado das saídas 38-36 com o byte 05 hexadecimal no último byte de dados, ou 0000 0101 em binário. A saída 38 está na posição 6 da esquerda e a saída 36 é o LSB deste byte. Os cinco bits de ordem alta são preenchidos com zero.

Abaixo na figura 23[5] tem-se o fluxograma de atendimento de um pedido de leitura das entradas.

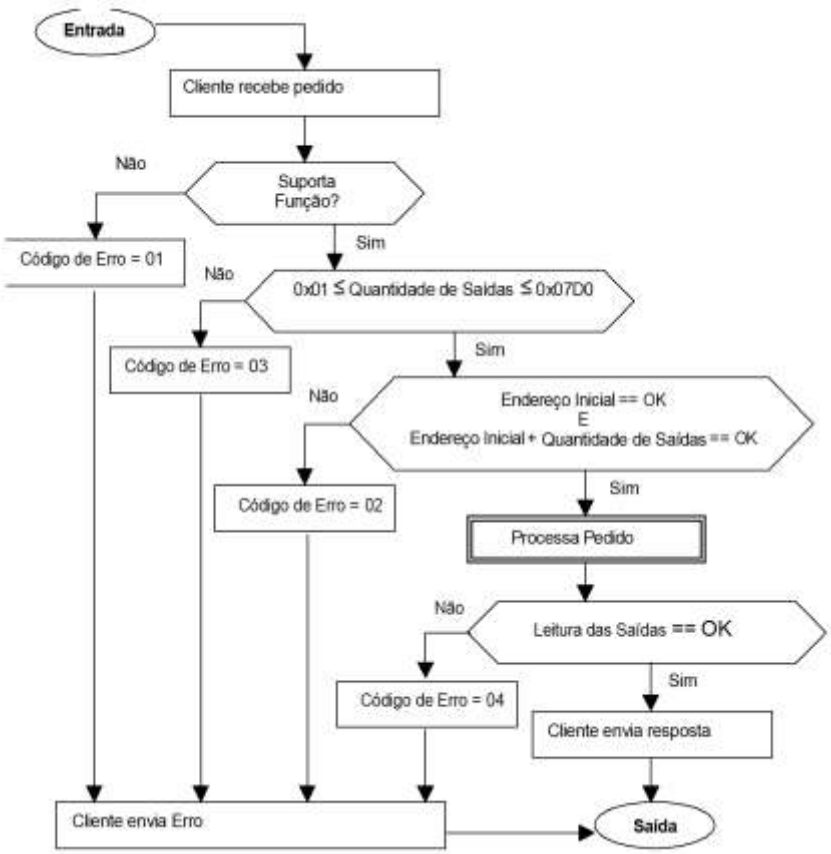

Figura 23: Fluxograma da Função 1 do ModBus.

O código 02, (0x02) Leitura das Entradas Discretas, é o código de função usado para ler de 1 para 2000 o estado das entradas discretas em um dispositivo remoto. O Pedido PDU especifica o endereço inicial, i.e. o endereço da primeira entrada e o número de entradas. No PDU, o endereço inicial é zero. Então, as entradas discretas numeradas de 1-16 são endereçadas como 0 -15.

O status das entradas é concatenado dentro de um bite sendo que cada bit representa uma entrada e o seu status é sinalizado em 1=Ligado e 0=Desligado.O LSB do primeiro byte contém o estado do endereço inicial seguido pelo estado dos próximos endereços; se existir um numero menor de endereços que bits no ultimo byte enviado, estes são preenchidos com zeros.

Pedido, tabela 8.

Tabela 8: Pedido Função 2.

Código da função	1 Byte	0x02
Endereço inicial	2 Bytes	0x0000 to 0xFFFF

Quantidade de Entradas	2 Bytes	1 to 2000 (0x7D0)

Resposta, tabela 9.

Tabela 9: Resposta Função 2.

Código da função	1 Byte	0x02
Nº de bytes	1 Byte	N*
Estado das Entradas	N* x 1 Byte	

Erro, tabela 10.

Tabela 10: Erro Função 2.

Código da função	1 Byte	0x82
Código de erro	1 Byte	01 or 02 or 03 or 04

Na tabela 11, é mostrado um exemplo de um pedido de leitura das entradas de 197 – 218[5]:

Tabela 11: Exemplo Função 2.

Pedido		Resposta	
Endereço	(Hex)	Endereço	(Hex)
Código da função	02	Código da função	02
Endereço inicial Hi	00	Nº de bytes	03
Endereço inicial Lo	C4	Estado das Entradas 204-197	AC
Quantidade de Entradas Hi	00	Estado das Entradas 212-205	DB
Quantidade de Entradas Lo	6	Estado das Entradas 218-213	35

A seqüência de leitura é a mesma utilizada no pedido de leitura de saídas mostrado anteriormente. Assim fica mostrado que o byte AC hexadecimal, 10101100 em binário, mostra no LSB o estado do endereço 197 e este está desligado.

Na figura 24[5], é mostrado o fluxograma de tratamento de um pedido de leitura de entradas.

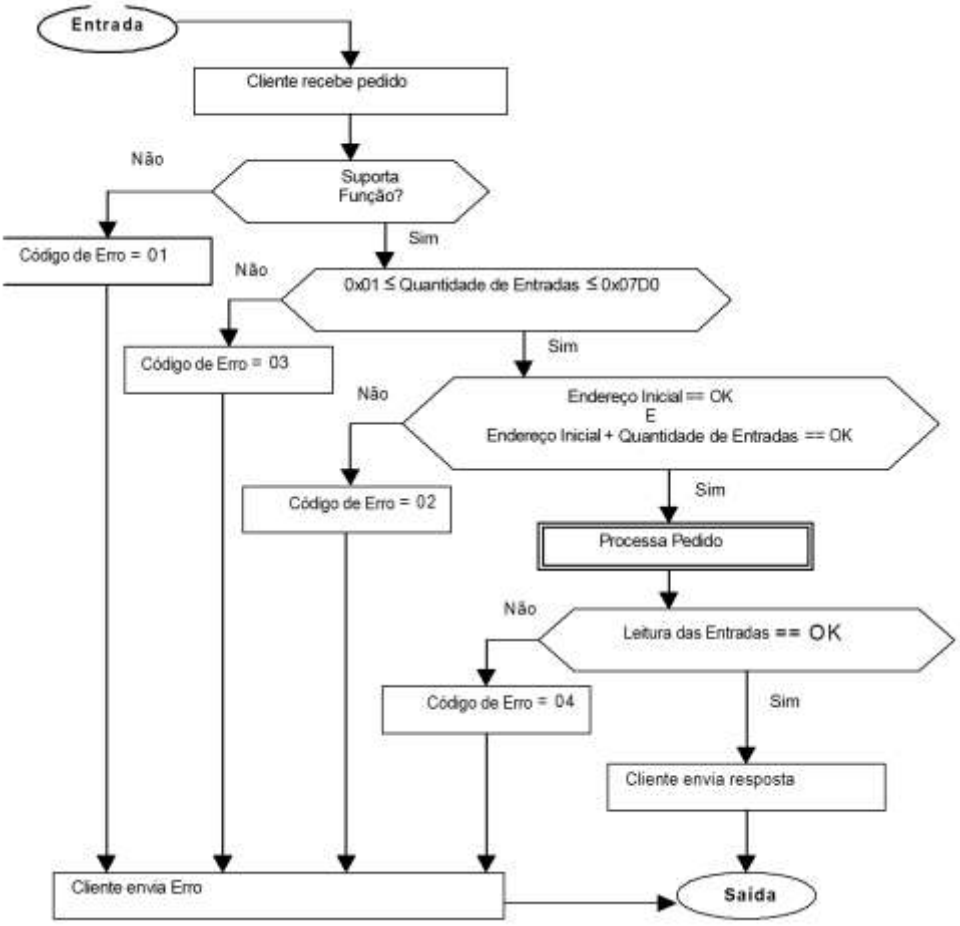

Figura 24: Fluxograma da Função 2 do ModBus.

A função 03, (0x3) Leitura de Registradores Internos, é o código de função que é usado para ler os conteúdos de um bloco contíguo de registradores em um dispositivo remoto. O Pedido PDU especifica o endereço do registrador inicial e o número de registradores que devem ser lidos. No PDU, o primeiro registrador é endereçado como zero. Então, registradores numerados de 1-16 são endereçados como 0 -15.

Os dados de um registrador na mensagem de resposta são acumulados como dois bytes, com valores justificados na direita e em binário dentro de cada byte. Para cada registrador, o primeiro byte representa o byte mais significativo da resposta e o segundo byte os bits menos significativos.

Pedido, tabela 12.

Tabela 12: Pedido Função 3.

Código da função	1 Byte	0x03
Endereço inicial	2 Bytes	0x0000 to 0xFFFF
Nº de Registradores	2 Bytes	1 to 125 (0x7D)

Resposta, tabela 13.

Tabela 13: Resposta Função 3.

Código da função	1 Byte	0x03
Nº de bytes	1 Byte	2 x N*
Valor Registradores	N* x 2 Bytes	

***N** = Nº de Registradores

Erro, tabela 14.

Tabela 14: Erro Função 3.

Código da função	1 Byte	0x83
Código de erro	1 Byte	01 or 02 or 03 or 04

Abaixo, é mostrado exemplo de pedido de leitura dos registradores de 108 – 110, tabela 15[5].

Tabela 15: Exemplo Função 3.

Pedido		Resposta	
Endereço	(Hex)	Endereço	(Hex)
Código da função	03	Código da função	03
Endereço inicial Hi	00	Nº de bytes	06
Endereço inicial Lo	6B	Valor Hi (108)	02
Registrador Hi	00	Valor Lo (108)	2B
Registrador Lo	03	Valor Hi (109)	00

	Valor Lo (109)	00
	Valor Hi (110)	00
	Valor Lo (110)	64

São mostrados os conteúdos do registrador 108 como os valores de dois byte de 02 2B hexadecimal ou 555 decimal. Os conteúdos dos registradores 109 e 110 são 00 00 e 00 64 hexadecimal ou 0 e 100 decimal, respectivamente.

Na figura 25[5], é mostrado fluxograma de analise de um pedido de leitura de registradores internos.

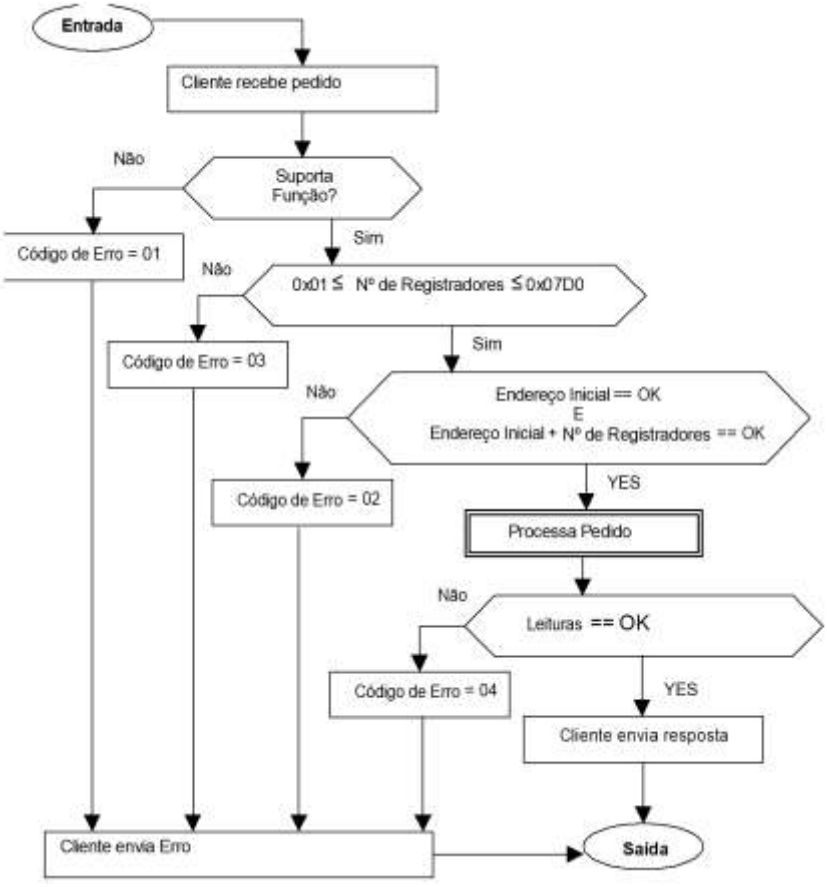

Figura 25: Fluxograma da Função 3 do ModBus.

O código de função, (0x04) Leitura de Registradores Analógicos, é usado para ler o registrador de 1 até 125, estes registradores analógicos contíguos em um dispositivo remoto. O Pedido PDU especifica o endereço de registrador inicial e o

número de registradores. No PDU, o registrador inicial tem endereço zero. Então, registradores numerados de 1-16 são endereçados como 0 -15.

Os dados de um registrador analógico na mensagem de resposta são acumulados como dois bytes, com valores justificados na direita e em binário dentro de cada byte. Para cada registrador analógico, o primeiro byte representa o byte mais significativo da resposta e o segundo byte, os bits menos significativos.

Pedido, tabela 16.

Tabela 16: Pedido Função 4.

Código da função	1 Byte	0x03
Endereço inicial	2 Bytes	0x0000 to 0xFFFF
Nº de Registradores	2 Bytes	1 to 125 (0x7D)

Resposta, tabela 17

Tabela 17: Resposta Função 4.

Código da função	1 Byte	0x03
Nº de bytes	1 Byte	2 x N*
Valor Registradores	N* x 2 Bytes	

***N** = Nº de Registradores.

Erro, tabela 18.

Tabela 18: Erro Função 4.

Código da função	1 Byte	0x83
Código de erro	1 Byte	01 or 02 or 03 or 04

Abaixo, mostra-se o pedido de leitura do registrador analógico 9, tabela 19[5]:

Tabela 19: Exemplo Função 4.

Pedido		Resposta	
Endereço	(Hex)	Endereço	(Hex)

Código da função	04	Código da função	04
Endereço inicial Hi	00	Nº de bytes	02
Endereço inicial Lo	08	Valor Analógico 9 Hi	00
Registrador Hi	00	Valor Analógico 9 Lo	0A
Registrador Lo	01		

O conteúdo do registrador analógico 9 está agrupado nos dois valores hexadecimais 00 0A, que agrupados retornam o valor decimal 10.

Na figura 26[5], mostra-se o fluxograma de dados em um pedido de leitura de dados analógicos.

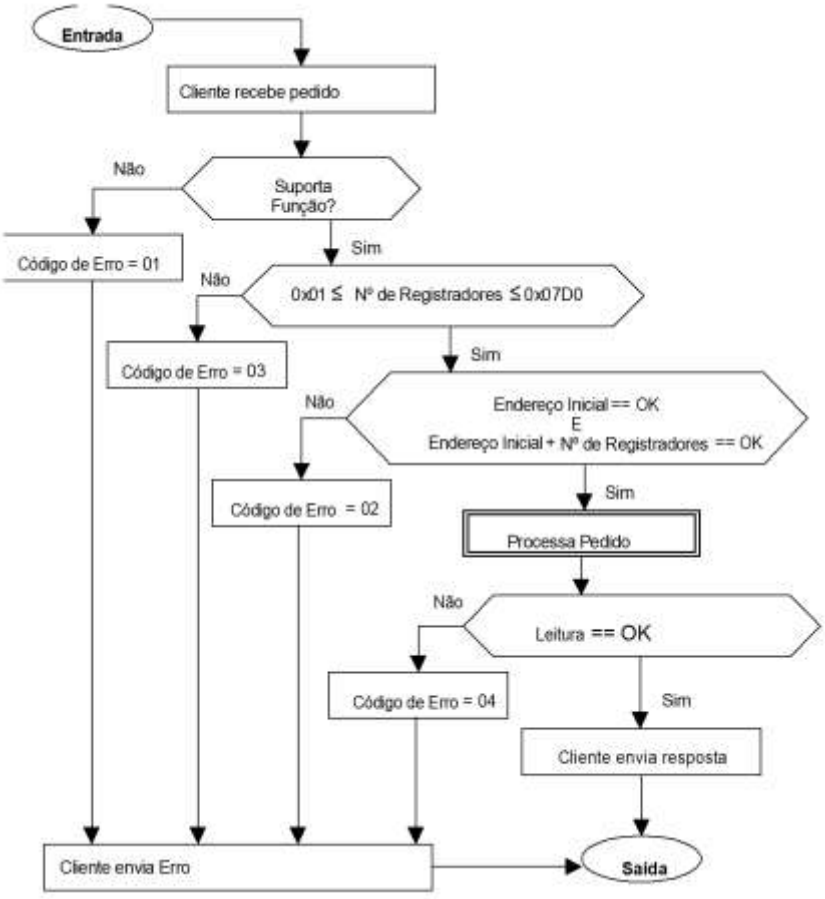

Figura 26: Fluxograma da Função 4 do ModBus.

A função 5, (0x05) Escrita de uma Saída Digital, é usada para escrever uma única saída para ativá-la ou desativá-la em um dispositivo remoto. O pedido ON / OFF é especificado por um valor constante no campo de dados . Um valor de 00 FF hexadecimal faz a saída ser ligada. Um valor de 00 00 pede que ela seja desligada. Todos os outros valores são ilegais e não afetarão a saída.

O Pedido PDU especifica o endereço da saída que deve ser forçada. Saídas são enumeradas em zero. Assim sendo, a bobina 1 é tratada como 0. O pedido ON / OFF é especificado por um dado constante no campo Valor da bobina.

Pedido, tabela 20.

Tabela 20: Pedido Função 5.

Código da função	1 Byte	0x05
Endereço	2 Bytes	0x0000 to 0xFFFF
Valor	2 Bytes	0x0000 or 0xFF00

Resposta, tabela 21.

Tabela 21: Resposta Função 5.

Código da função	1 Byte	0x05
Endereço	2 Bytes	0x0000 to 0xFFFF
Valor	2 Bytes	0x0000 or 0xFF00

Erro, tabela 22.

Tabela 22: Erro Função 5.

Código da função	1 Byte	0x85
Código de erro	1 Byte	01 or 02 or 03 or 04

Abaixo se mostra um exemplo de aplicação ligando a saída 173, tabela 23[5].

Tabela 23: Exemplo Função 5.

Pedido		Resposta	
Endereço	(Hex)	Endereço	(Hex)
Código da função	05	Código da função	05
Endereço Hi	00	Endereço Hi	00
Endereço Lo	AC	Endereço Lo	AC
Valor Hi	FF	Valor Hi	FF
Valor Lo	00	Valor Lo	00

O fluxograma de dados é mostrado na figura 27[5], abaixo.

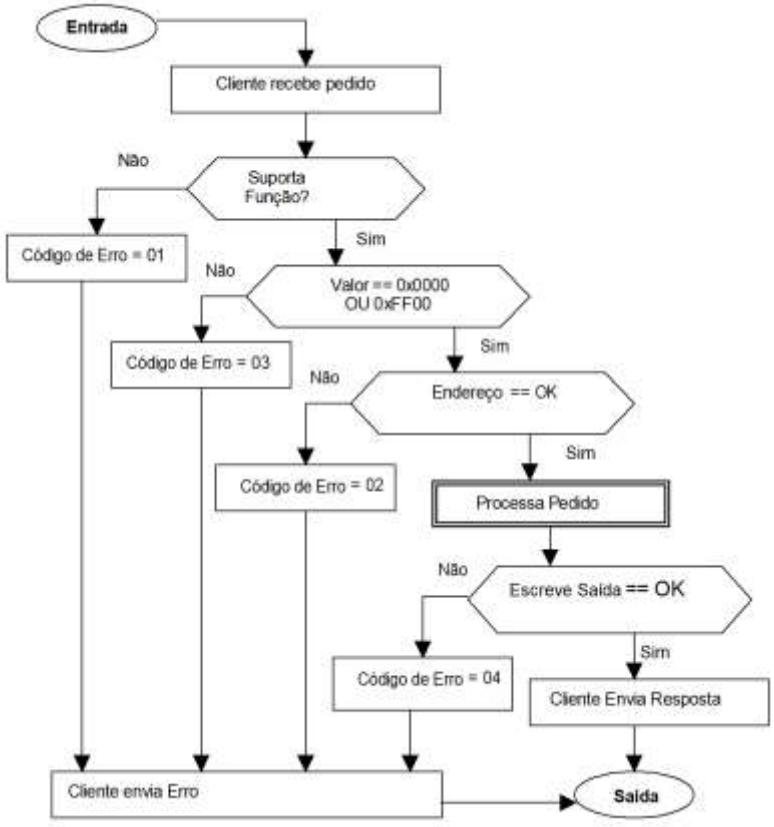

Figura 27: Fluxograma da Função 5 do ModBus.

A função 6, (0x06) Escreve um Registrador, é usada para escrever um único registrador em um dispositivo remoto. O Pedido PDU especifica o endereço do registrador a ser escrito. Os endereços tem início em zero. Portanto, o registrador 1

é tratado como 0 no PDU. A resposta normal é um eco do pedido, devolvido após que o conteúdo do registrador de destino tenha sido atualizado.

Pedido, tabela 24.

Tabela 24: Pedido Função 6.

Código da função	1 Byte	0x06
Endereço	2 Bytes	0x0000 to 0xFFFF
Valor	2 Bytes	0x0000 to 0xFFFF

Resposta, tabela 25.

Tabela 25: Resposta Função 6.

Código da função	1 Byte	0x06
Endereço	2 Bytes	0x0000 to 0xFFFF
Valor	2 Bytes	0x0000 to 0xFFFF

Erro, tabela 26.

Tabela 26: Erro Função 6.

Código da função	1 Byte	0x86
Código de erro	1 Byte	01 or 02 or 03 or 04

Mostra-se na tabela 27[5], um exemplo de escrita do registrador 2 tendo seu conteúdo alterado para 00 03 hexadecimal.

Tabela 27: Exemplo Função 6.

Pedido		Resposta	
Endereço	(Hex)	Endereço	(Hex)
Código da função	06	Código da função	06
Endereço Hi	00	Endereço Hi	00
Endereço Lo	01	Endereço Lo	01
Valor Hi	00	Valor Hi	00

Valor Lo	03	Valor Lo	03

Na figura 28[5], mostra-se o fluxograma de tratamento de dados.

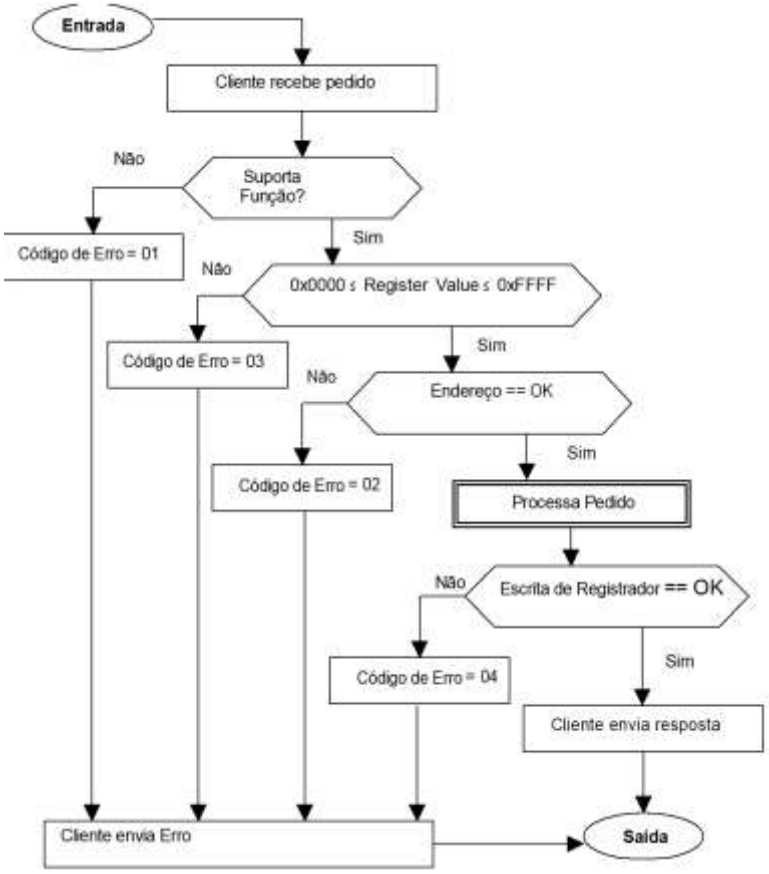

Figura 28: Fluxograma da Função 6 do ModBus.

3.3.5. Configuração do TRF-2.4G

Para o desenvolvimento do protocolo de transmissão sem fio foram seguidas rotinas especificas de controle do transceptor TRF-2.4G, as quais estão definidas no manual do TRF-2.4G[3] que está disponível para “download” em www.tato.ind.br.

O TRF-2.4g está configurado com uma palavra de configuração de 120 bits. Esta palavra define o tamanho do pacote de dados, o endereço, o tamanho da palavra de endereço, o modo de CRC, se a operação é em um ou dois canais, o modo do dispositivo, a taxa de dados, a freqüência de operação, a freqüência do cristal interno, e o modo, Rx ou Tx, de operação. Enviar dados para o buffer do transceptor é bastante simples. A palavra de configuração é enviada para o dispositivo serialmente com o bit mais significativo em primeiro lugar, os dados são disponibilizados em uma saída de I/O do micro-controlador e são lidos sucessivamente nas bordas de subidas do relógio. Para atender às exigências do

clock do dispositivo especificadas na ficha técnica do produto, é necessário manter o dado a ser lido disponível por um tempo de 500ns no bus antes de uma subida do pino de clock ser disparada. Na seqüência, o clock e os dados devem ser mantidos por pelo menos mais 500ns[3] antes de obter-se uma borda de descida.

Para entrar em modo de configuração, deve-se definir CE em nível baixo e CS em nível alto através de uma porta de saída do micro-controlador. Isso faz com que o dispositivo entre em modo de configuração e espere por uma palavra a ser escrita. Um atraso de 5µs é necessário antes de enviar os dados de configuração para o dispositivo. A palavra configuração é mostrada abaixo, figura29, usando um endereço arbitrário e configurando o dispositivo no modo de RX. Uma vez que a palavra é inteiramente enviada, a configuração do modulo é atualizada quando o transceptor tem o pino de CS colocado em nível baixo. Para a troca entre os modos de transmissão e recepção, uma operação de configuração deve ser realizada, mas somente o ultimo byte pode ser atualizado reduzindo o custo computacional desta operação. Este fluxograma é apresentado na figura 30[3].

Channel 2 Data Width							
119	118	117	116	115	114	113	112
0	0	0	0	0	0	0	0

Channel 1 Data Width							
111	110	109	108	107	106	105	104
0	0	0	1	0	0	0	0

Channel 2 Address (8 to 40 bits)																			
103	102	101	100	99	98	97	96		95	94	...	73	72		71	70	69	68	67	66	65	64
0	0	0	0	0	0	0	0		0	0	...	0	0		0	0	0	0	0	0	0	0

Channel 1 Address (8 to 40 bits)																			
63	62	61	60	59	58	57	56		55	54	...	33	32		31	30	29	28	27	26	25	24
0	0	0	0	0	0	0	0		0	0	...	0	0		1	1	0	1	1	1	0	0

Address Width						CRC Length	CRC Enable
23	22	21	20	19	18	17	16
0	0	1	0	0	0	1	1

2 Channel Enable	ShockBurst	RF Data Rate	Oscillator Freq			RF Power	
15	14	13	12	11	10	9	8
0	1	0	0	1	1	1	1

Channel Frequency							RXEN
7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	1

Figura 29: Palavra de Configuração.

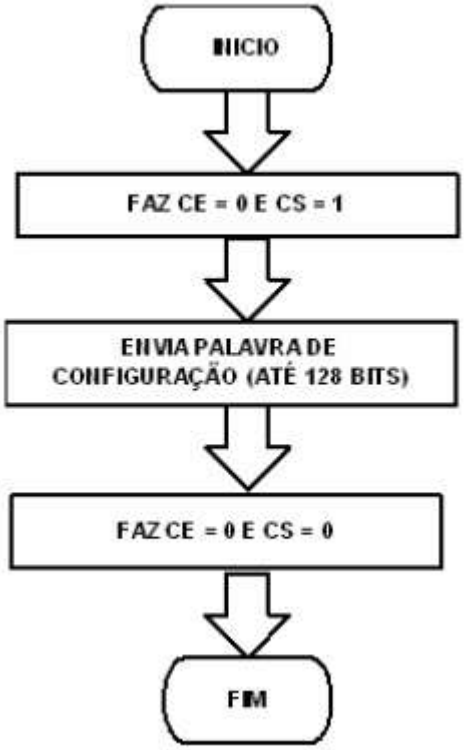

Figura 30: Fluxograma de Configuração.

Para iniciar a transmissão dos dados, o dispositivo deve ter a configuração alterada colocando este em modo TX. Para isso faz-se o registrador RXEN = 0 e o transceptor deve ser colocado em modo de operação definindo o pino CE em nível alto. Além disso, o registrador de dados deve ser selecionado através da definição do pino CS em nível baixo. Em modo *ShockBurst* de transmissão, o dispositivo deve receber um pacote de dados que inclui um endereço de destino e uma carga de dados, os quais devem ser definidos a partir do micro-controlador. O pacote é enviado seriadamente (MSB em primeiro lugar) e este será armazenado em um buffer respeitados os ciclos de tempo do mesmo modo que os ciclos de envio de uma palavra de configuração. Quando todo o pacote é enviado para o dispositivo, este fixa o pino CE em nível baixo; isso irá ativar o processamento do transceptor, que irá inclui um preâmbulo e anexar o CRC. O transceptor após a inclusão destes dados irá começar a transmissão de dados sem fio. Na figura 31, existe o fluxograma deste modo de operação.

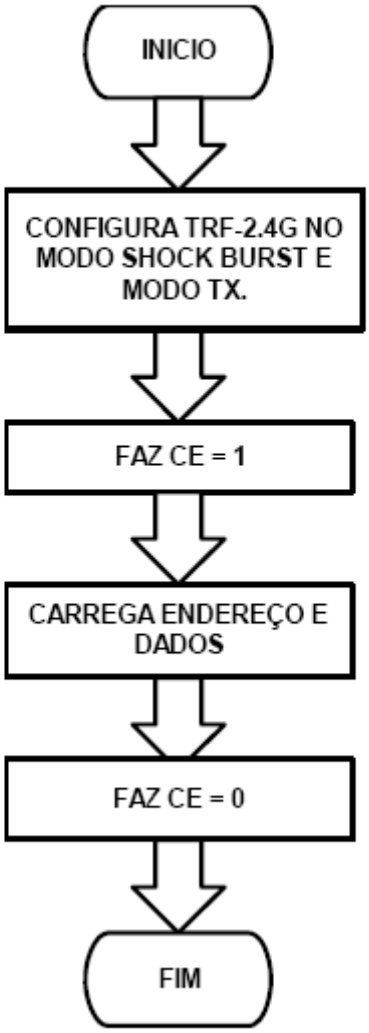

Figura 31: Transmissão de dados.

Já para a recepção de dados, o dispositivo deve ser configurado em modo RX. Para isso o registrador RXEN deve ser definido como RXEN = 1, também a sua configuração deve ser definida para o modo ativo, fixando CE em nível alto. O transceptor acompanhará os sinais disponíveis e quando receber um pacote irá analisar o preâmbulo, verificar o endereço e o CRC. Se o endereço e o CRC são válidos, são removidos do pacote os dados adicionais, preâmbulo, endereço e CRC e os dados restantes são colocados em um buffer para o micro-controlador poder realizar a leitura dos mesmos. O transceptor irá notificar o micro-controlador através de um pino sinalizando dados disponíveis, DR1 em nível alto. Após os dados serem lidos e o buffer de recepção estiver vazio, o pino DR1 é colocado em nível baixo pelo transceptor e este irá começar a monitorar outros pacotes de dados com

destino válido. Na figura 32, é mostrado fluxograma do código de recepção de dados.

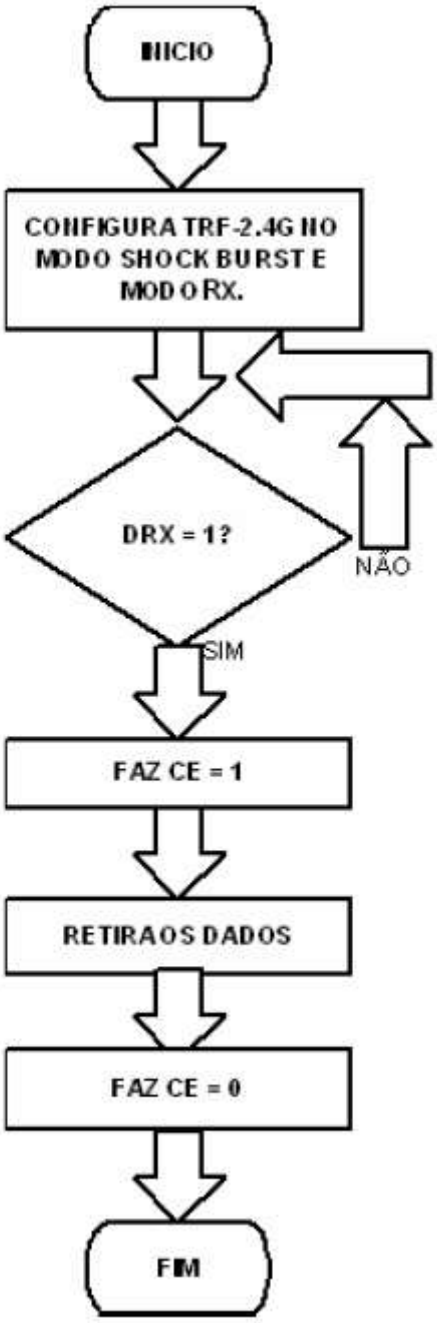

Figura 32: Recepção de dados.

3.3.6. Software do Microcomputador

Para a construção do máster, foi escolhido um microcomputador que tem poder de processamento superior para o gerenciamento de todas as tarefas do modbus, além de existirem softwares gratuitos para este protocolo que podem ser usados para a validação do software no dispositivo remoto.

Para fins de validação, foi escolhida a versão de testes do software MBTestPro da Rogue Engineering, que tem a capacidade de endereçamento desejada. Além de possuir telas de validação dos dados enviados e recebidos, com este software foi possível a verificação do software no hardware remoto por uma fonte independente retornando resultados satisfatórios.Na figura 33, é mostrada a captura da tela do software da Rogue Engineering.

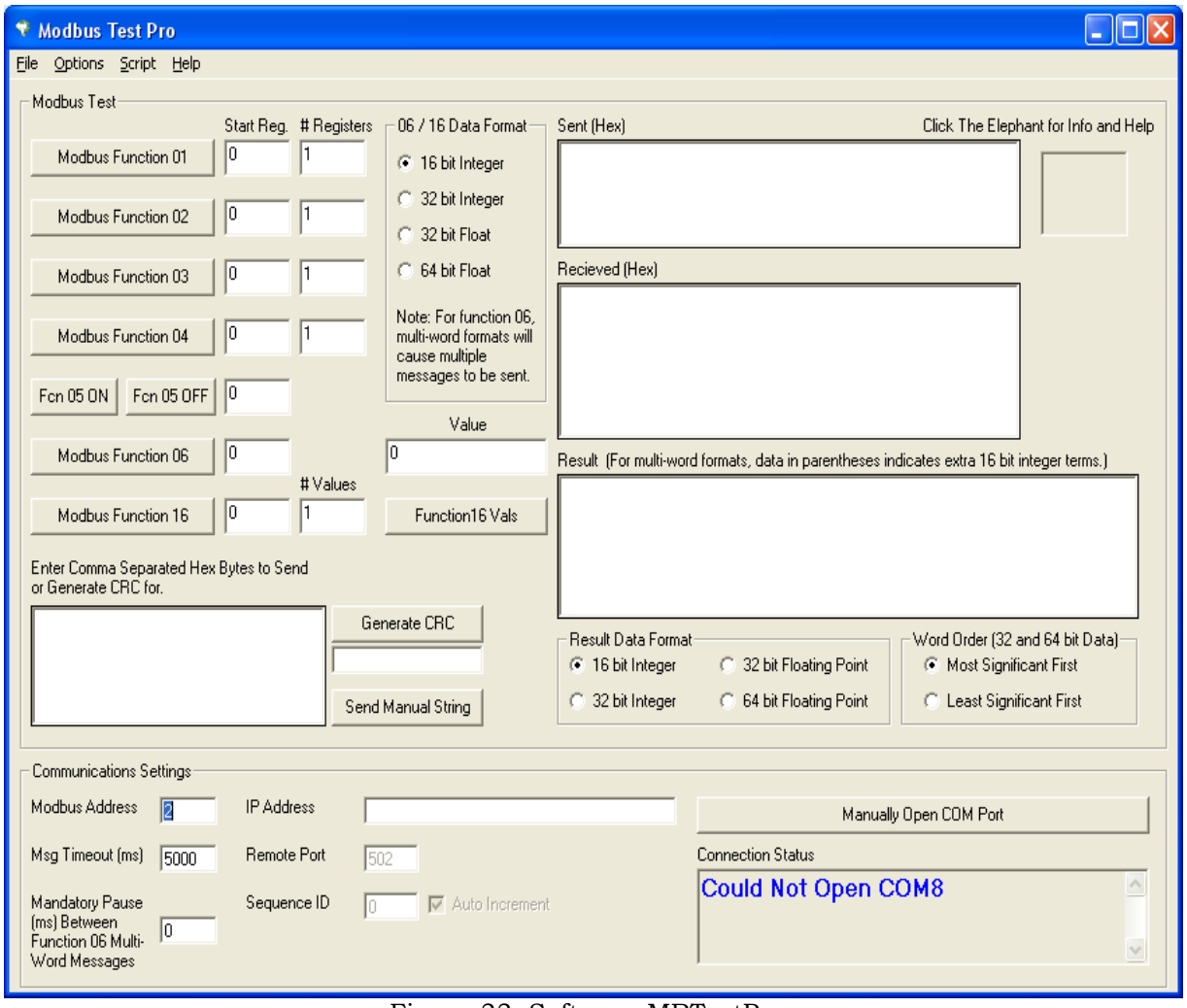

Figura 33: Software MBTestPro.

Para o software criado no microcomputador foi escolhida a ferramenta de desenvolvimento Visual C++ 2005 Express Edition, da empresa Microsoft. Esta ferramenta permite a criação de programas em um ambiente de desenvolvimento visual onde o desenvolvedor deve primeiramente desenvolver a interface com o usuário e posteriormente os códigos necessários para o a execução das tarefas necessárias. A figura 34 mostra o ambiente de desenvolvimento da Microsoft com o projeto em desenvolvimento.

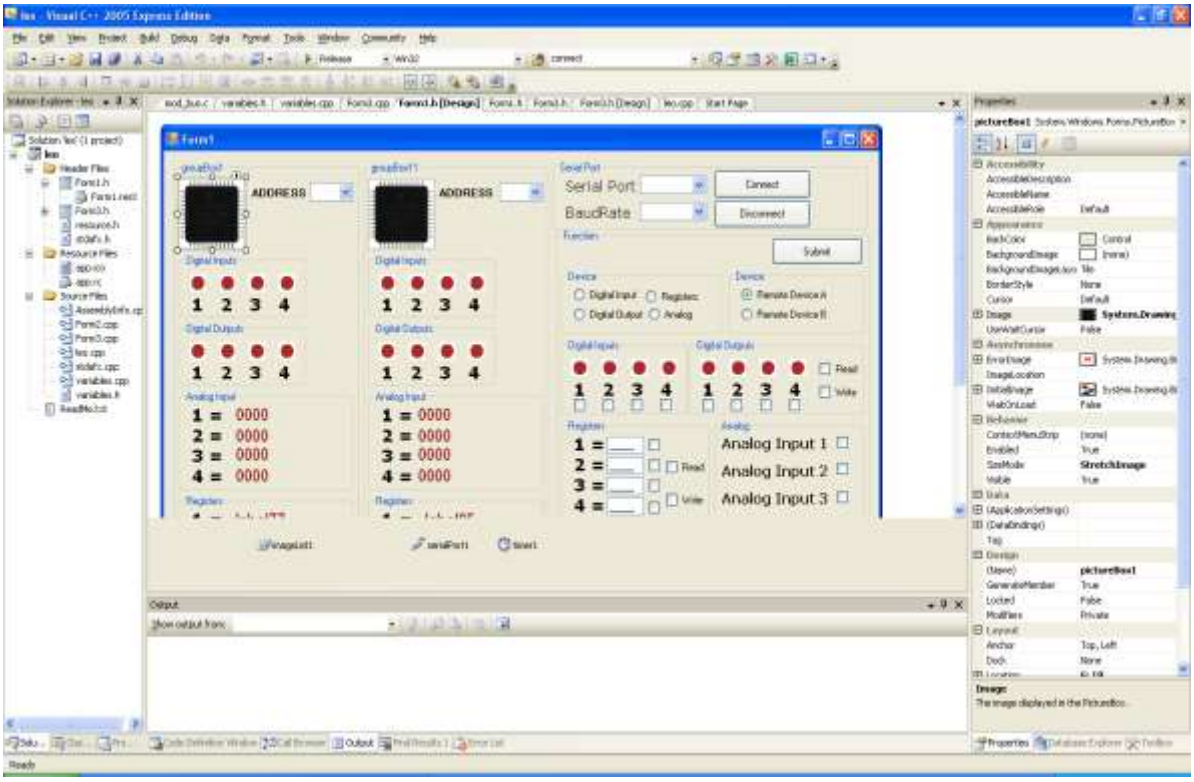

Figura 34: Visual Studio C++ 2005 Express Edition.

Com o ambiente de programação definido o desenvolvimento do software buscava um mestre capaz de endereçar dispositivos remotos independentemente, além de verificar os dados recebidos e gerenciar em uma única tela os pedidos e as respostas enviadas entre o servidor e o cliente. Na figura 35, tem-se disponível a tela final do software aplicativo.

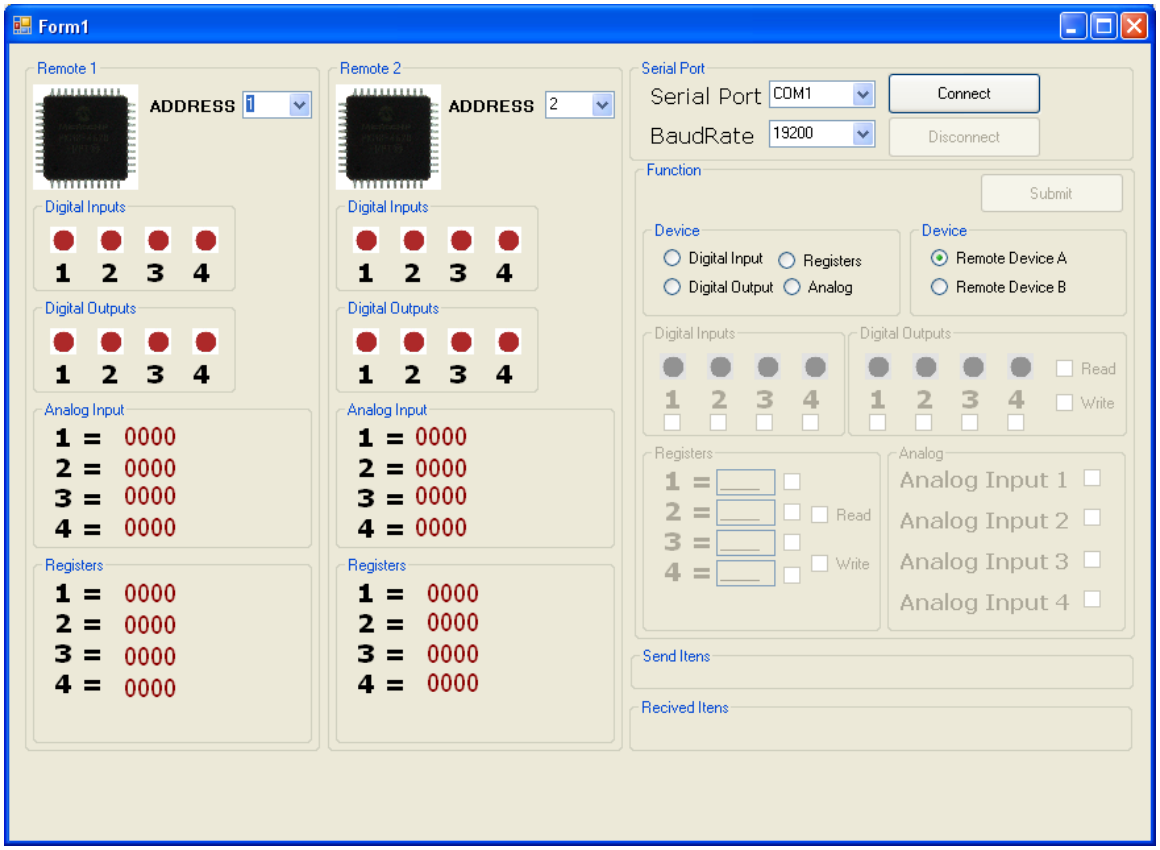

Figura 35: Software do Microcomputador.

A figura 36 exemplifica o estado de um dispositivo remoto. Pode-se notar 5 campos diferentes: Address, que é o endereço do dispositivo remoto que está sendo analisado, Digital Inputs que mostra o estado das entradas digitais, Digital Outputs que contém as informações correspondentes às saídas digitais, e os valores de registradores, Analog Input que são entradas analógicas e Registers que são registradores internos de memória.

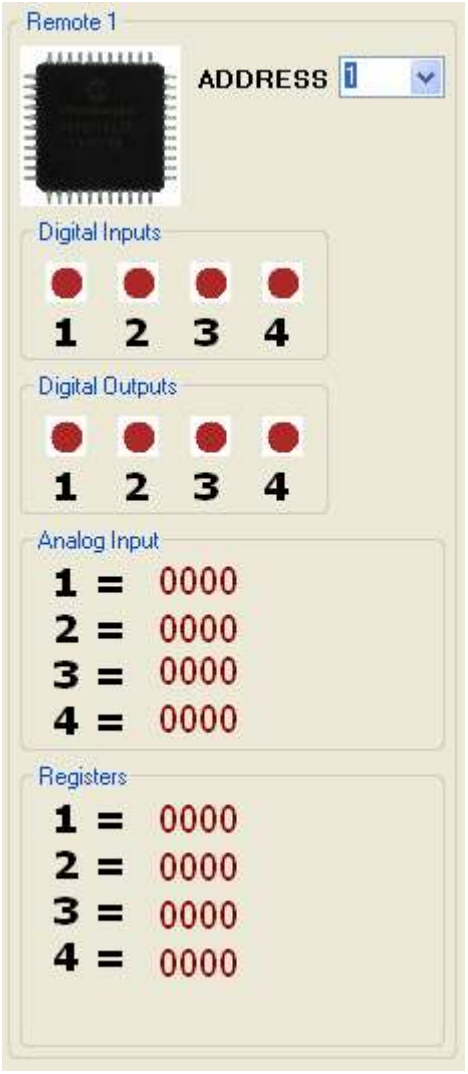

Figura 36: Estado do dispositivo remoto.

No campo de controle Serial Port estão disponíveis as configurações relativas à porta serial. Dados como número da porta e informações de *BaudRate* são configuráveis bem como um botão para a conexão, Connect, e outro para a desconexão, Disconnect, do servidor. Os botões estão disponíveis em estados alternados. Assim, quando o dispositivo está conectado o botão de desconexão torna-se habilitado, evitando assim conexões múltiplas o que acarretaria em falhas no sistema. Este grupo de configurações é mostrado na figura 37.

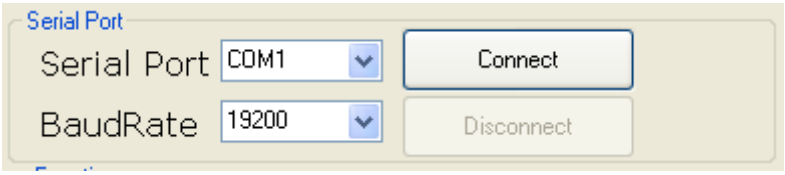

Figura 37: Funções da Porta Serial.

O grupo Functions, figura 38, engloba todas as funções implementadas dando ao usuário a opção de leitura e escrita quando disponível. Este também faz verificações das opções selecionadas pelo usuário evitando assim entradas errôneas de dado e conseqüente falha de comunicação com o dispositivo remoto. Também está implementado o botão Submit, o qual envia o pedido de dados ao dispositivo remoto selecionado, que somente é habilitado para o usuário se uma conexão serial estiver ativa.

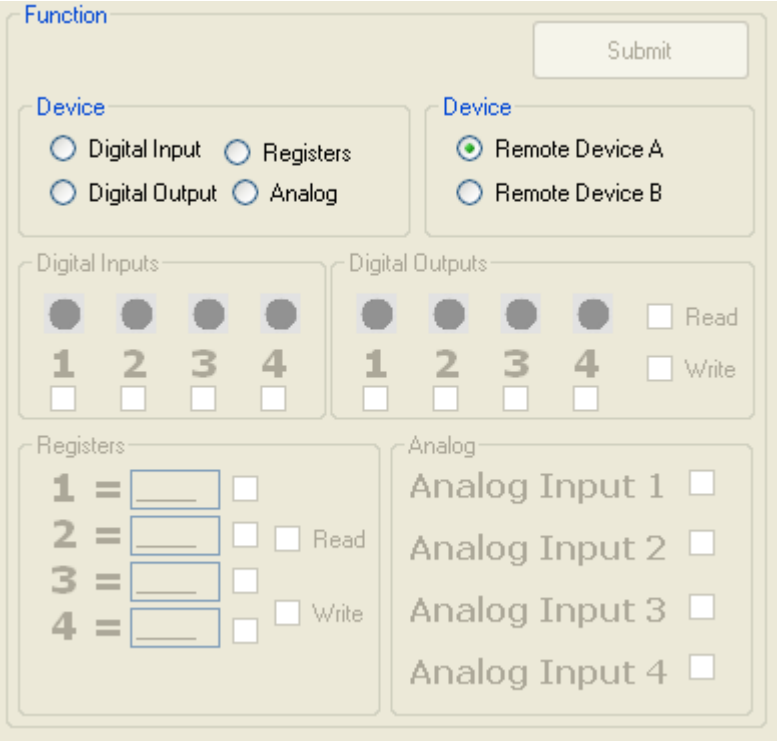

Figura 38: Funções MODBUS.

Os campos de dados Device Function e Device, figura 39, selecionam respectivamente o grupo de dados de interesse e o dispositivo remoto que irá ser analisado. Por ser um protocolo destinado a comunicação cliente servidor, somente é possível a seleção de um único dispositivo e uma única função para cada envio de dados.

Figura 39: Seleção de Dispositivo e Função.

Na aba digital inputs é possível a seleção de quais entradas digitais se deseja efetuar a leitura de dados; isso é realizado através das caixas de seleção

encontradas embaixo de cada entrada numerada de 1 a 4. Da mesma maneira, o campo Digital Outputs é atualizado quando a opção de leitura, Read, está ativa.Já na opção de escrita das saídas digitais , Write, o usuário tem a opção de mudar o estado físico da saída remota clicando sobre a figura acima do número correspondente a saída selecionada. Isto altera o estado da mesma entre ligado, Vermelho, e desligado, Verde. A figura 40 mostra esta janela.

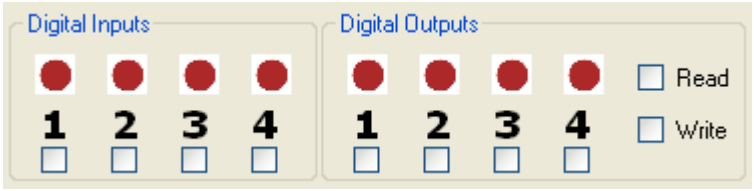

Figura 40: Entradas e Saídas Digitais.

Para a leitura das entradas analógicas foi desenvolvido o mesmo sistema utilizado para as entradas digitas. Assim, o usuário deve selecionar a entrada ou entradas nas quais deseja efetuar a leitura de dados e posteriormente enviar pedido de dados. A figura 41 mostra este painel.

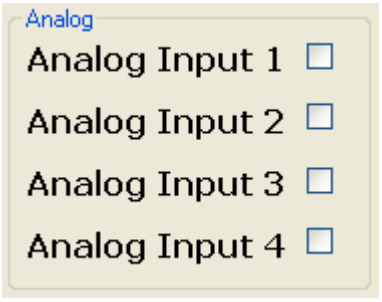

Figura 41: Portas Analógicas.

A leitura e a escrita de dados de registradores internos, figura 42, é feita através de um painel especial o qual tem a opção de escrita de valores, Write, que serão enviados para micro-controlador, que por sua vez aceitam somente valores numéricos de valor inferior a 1024 o qual corresponde a 5V no micro-controlador. Já para efeitos de leitura os campos têm seu conteúdo excluído do pacote de dados enviado para o micro-controlador.

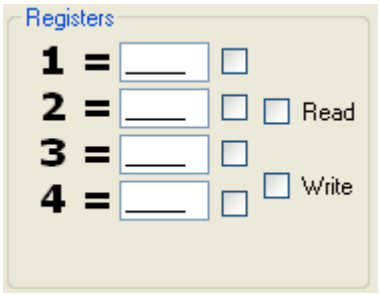

Figura 42: Registradores.

O software do microcomputador não precisa ser testado por um software externo uma vez que o dispositivo remoto foi. Assim, a comunicação somente será aceita pelo dispositivo remoto se estes dados forem dados que obedeçam à estrutura do protocolo modbus.

4. APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Todos os testes realizados no sistema de transmissão verificavam os dados que estavam trafegando no ambiente RS-232. Para estas verificações um conjunto de software foi utilizado, os quais são mostrados a seguir.

MPLAB, é um software desenvolvido pela Microchip, que tem como função a criação e depuração de códigos para a linha de microcontroladores da empresa. Na figura 43 existe um exemplo de tela de depuração. Em conjunto com este software foi utilizado um simulador de circuitos eletrônicos capaz de simular transmissões e recepção de sinais serias.

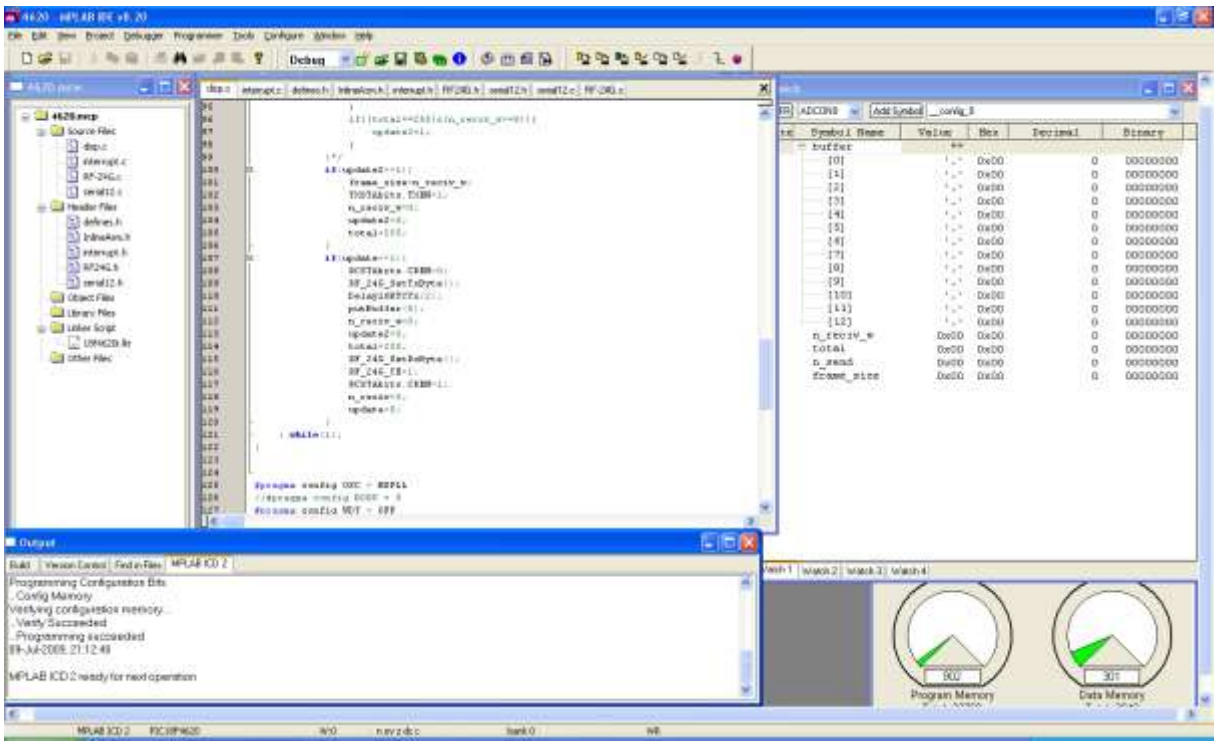

Figura 43: MPLAB em modo debug.

O Proteus, simulador elétrico, é o software utilizado para a simulação do código fonte do micro-controlador. Ele é capaz de simular o comportamento de um micro-controlador e todo o software em desenvolvimento para este dispositivo. Utilizando-se um conjunto de recursos de softwares específicos para o Modbus

pode-se avaliar o protocolo e as mensagens enviadas e recebidas. Assim todo o desenvolvimento do software foi feito em um ambiente virtual, sem a necessidade de criação do hardware em um primeiro momento. Isto aumenta a velocidade de desenvolvimento, o hardware pode ser desenvolvido após que todas as simulações estiverem concluídas, assim a validação eletrônica evita alterações posteriores. Na figura 44, é mostrada a tela do simulador Proteus e nas figuras 45 e 46 são mostradas as telas dos softwares de validação do protocolo Modbus.

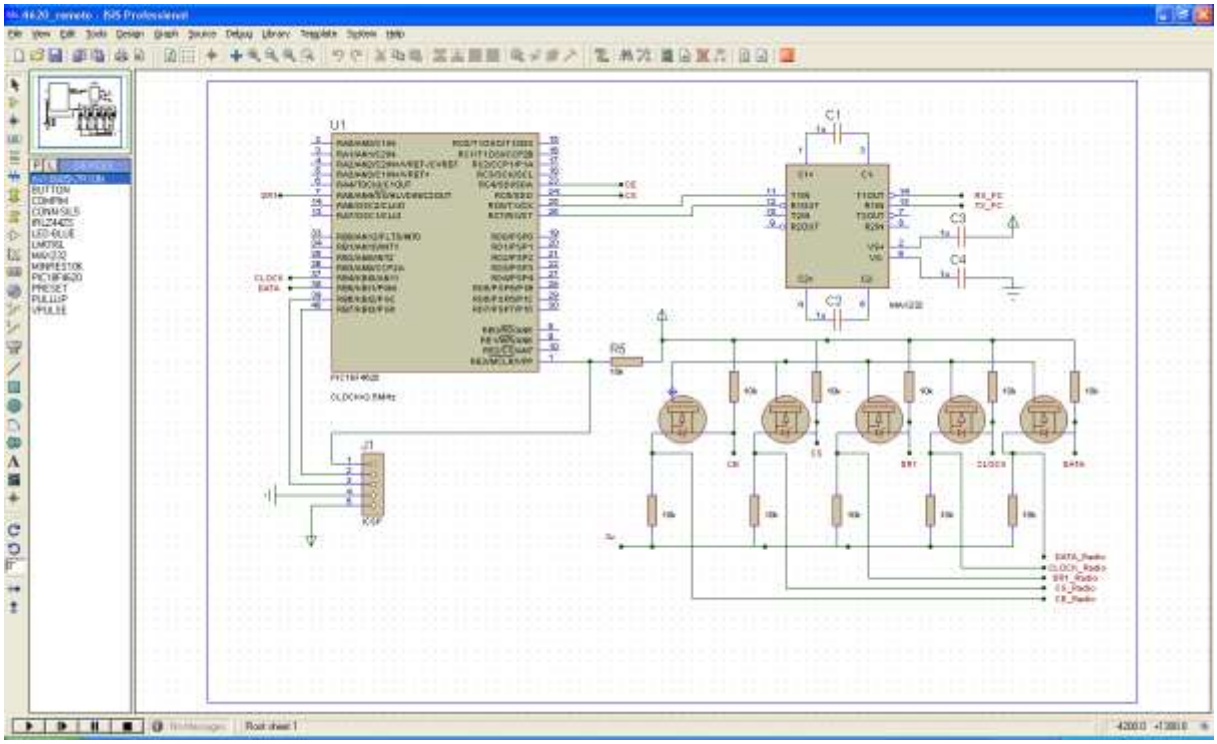

Figura 44: PROTEUS VSM.

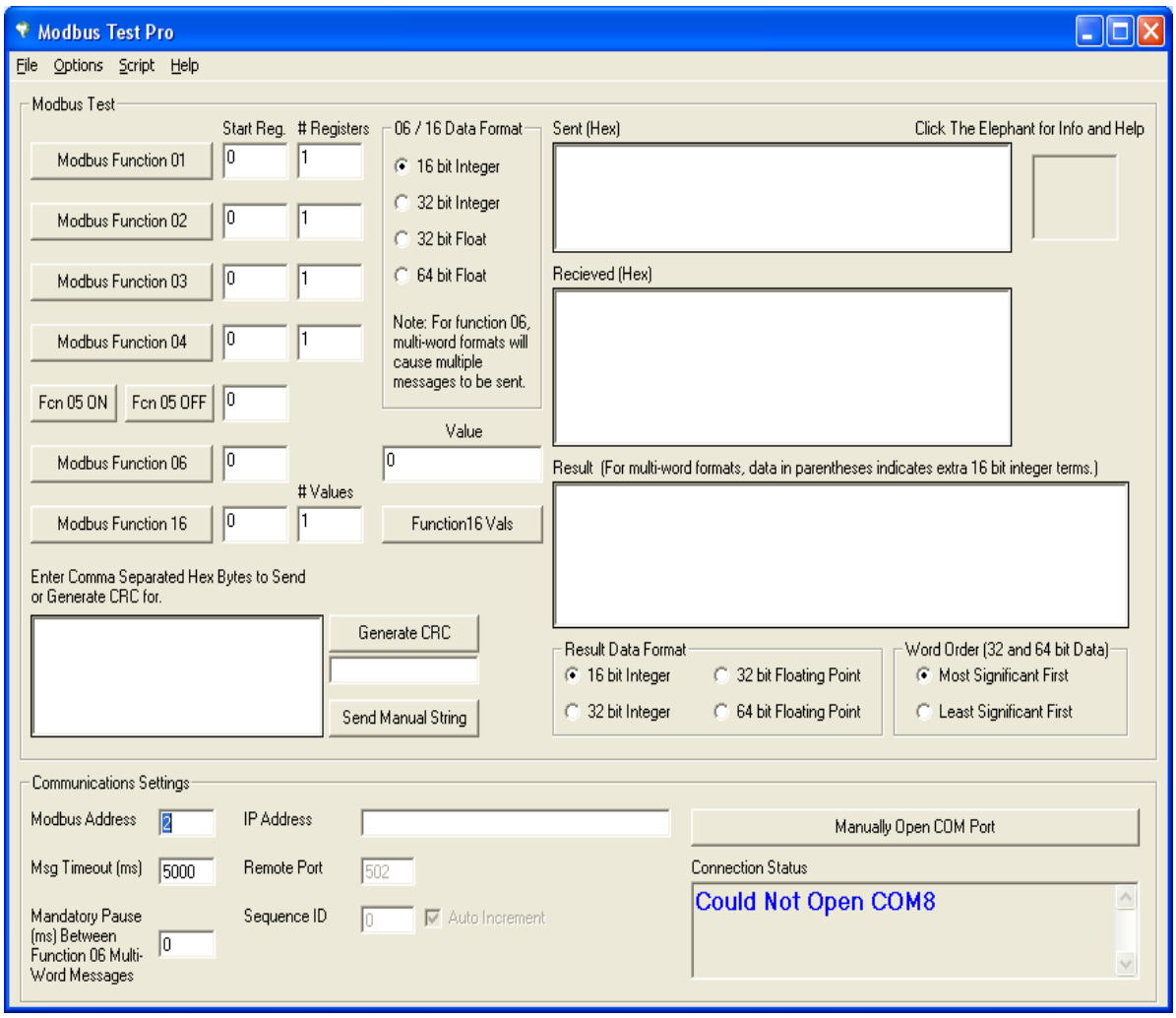

Figura 45: ModBus Test Pro.

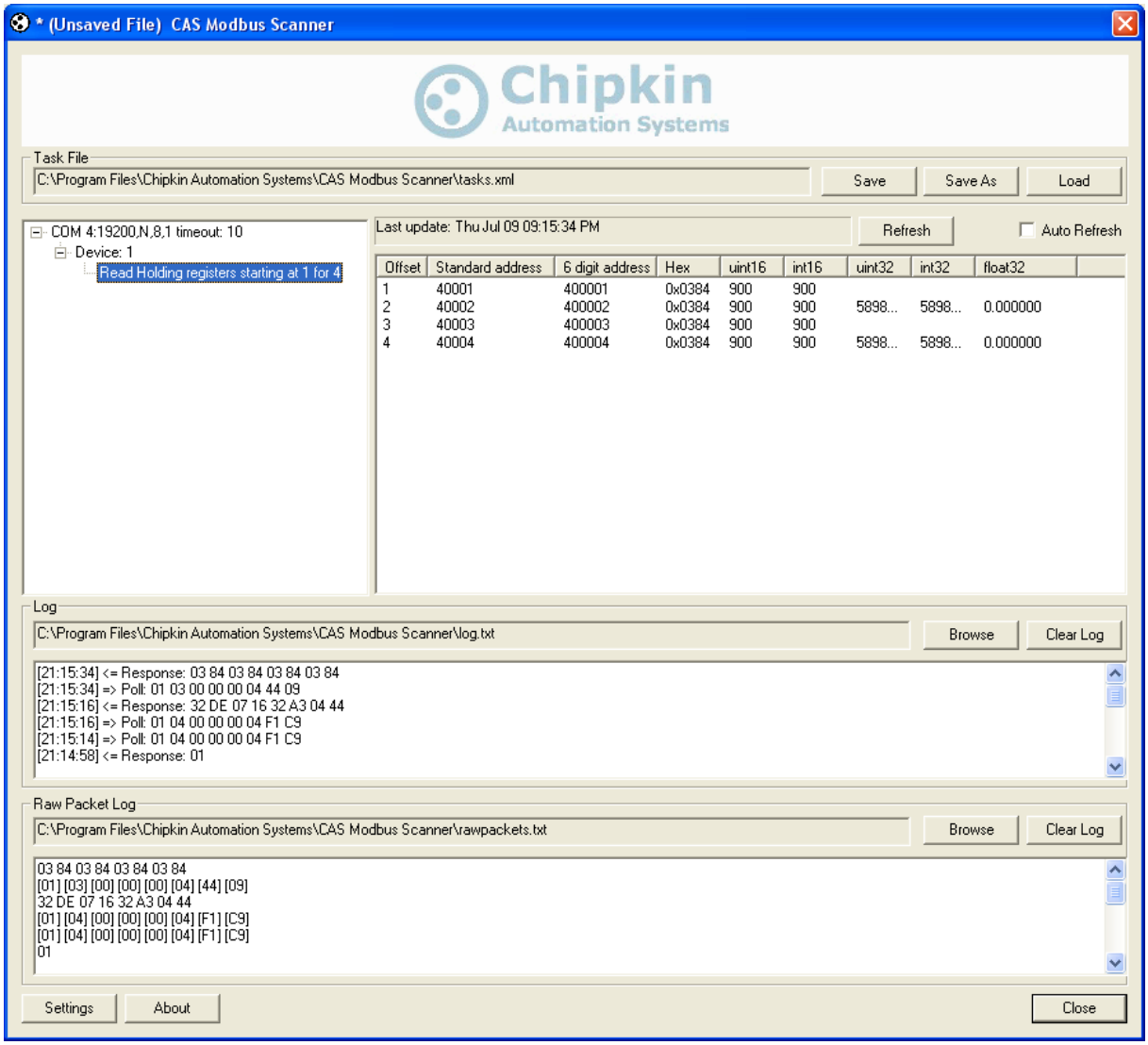

Figura 46: Cas Modbus Scanner.

Com as definições do software concluídas, a construção e posterior validação do hardware ficou simplificada. Este seguiu as mesmas rotinas de testes do software que consistem na validação dos pedidos enviados para o controlador Modbus e as respostas deste. Também foi testado o alcance do hardware. Estes resultados são apresentados na tabela 28.

Tabela 28: Alcance do Hardware.

Distancia	Porcentagem de Resposta
1 Metro	100 %
2 Metros	100 %
3 Metros	100 %
5 Metros	100 %

7 Metros	100 %
10 Metros	100 %
11 Metros	90%
11 Metros e 10 Centímetros	90%
11 Metros e 20 Centímetros	90%
11 Metros e 30 Centímetros	80%
11 Metros e 40 Centímetros	80%
11 Metros e 50 Centímetros	0%

5. CONSIDERAÇÕES FINAIS

A principal dificuldade encontrada foi transição de *level* entre as diferentes tensões do sistema: o transceptor Laipac apresenta uma especificação de difícil entendimento a qual não aceita soluções mais simples como um divisor resistivo ou o acionamento com diodos zener. A solução adotada diminui a complexidade final do sistema. Tendo em vista soluções encontradas em outros trabalhos, uma solução mais simples seria um circuito integrado dedicado a transição de *"level"*, mas estes são componentes específicos e não são comumente encontrados.

Todos os testes realizados validaram a solução. Assim, os pré-requisitos do sistema foram todos atendidos, a solução completa apresentou excelente desempenho, os requisitos de orçamento e tempo foram cumpridos. A qualidade final do projeto como um todo atendeu a todas as expectativas iniciais.

As funções aplicadas no trabalho e suas validações atenderam todos os objetivos propostos no inicio do trabalho, o projeto teve um fechamento conforme o esperado e a solução atende todos os objetivos propostos.

Pode-se sugerir futuros desenvolvimentos em cima de adaptações do hardware ligado ao microcomputador que funciona como um transmissor receptor wireless de RS-232. Algumas poucas customizações podem ser feitas para este objetivo ser concluído. Também é sugerido futuros desenvolvimentos que melhorem a capacidade de transmissão do transceptor utilizado.

Também sugere-se como melhorias a criação e validação de u novo dispositivo de rádio este com maior alcance de transmissão afim de cobrir uma área maior tornando o dispositivo mais robusto.

6. REFERÊNCIAS

[1] GIMENEZ, Salvador P. Microcontroladores 8051. Editora Pearson Education do Brasil. São Paulo, 2002.

[2] Microchip, PIC18F4620 – Datasheet. 2007.

[3] LAIPAC TECH. TRF-2.4G TRANSCEIVER – Datasheet.

[4] Schutte, Herman. Bi-directional level shifter for I²C-bus and other systems. Philips Semiconductors Systems Laboratory Eindhoven, The Netherlands.

[5] MODBUS IDA. MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b.

OBRAS CONSULTADAS

PETERS, James F. PEDRYCZ, Witold.. Engenharia de Software. Editora Campus. São Paulo, 2001.

SCHILDT, Herbert. C Total e Completo (3a. Edição). Editora Pearson Education do Brasil. São Paulo, 2002.

Souza, David José de. Desbravando o PIC (7a. Edição). Editora Érica. São Paulo, 2004.

PEREIRA, Fábio. Microcontroladores PIC – Técnicas avançadas. Editora Érica. São Paulo, 2002.

PROTEL International Limited, Design Explorer 99 SE Includes Service Pack 2. Versão 6.0.4., 1995-1999.[S.l.] : PROTEL International Limited, 1999. 1CD-ROM.

TATO, On-line. Disponível em: www.tato.ind.br. Acesso em 30 de Maio de 2009.

APÊNDICE A – CÓDIGO FONTE ESCRAVOS

C:\tcc\4620_remoto\adcycle.c

```
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include "adc.h"
#include "buttons.h"
#include "defines.h"
#include "interrupt.h"
#include "LCD.h"


//variaveis do adc;
#pragma udata adcvals = 0xA00
far unsigned int adcval[4][(AVGWIDTH+1)];
far unsigned int avgsum[(AVGWIDTH)];

#pragma udata
far unsigned int adc_res[4];
far unsigned int adc_res_max[4];
far unsigned int adc_med;
far unsigned char avgshift;
far union {
  unsigned char Whole;
  struct {
    unsigned ch1_flag:1;
    unsigned ch2_flag:1;
    unsigned ch3_flag:1;
    unsigned ch4_flag:1;
  };
} ch_flag;

far union {
  unsigned char Whole;
  struct {
    unsigned ch1_flag:1;
    unsigned ch2_flag:1;
    unsigned ch3_flag:1;
    unsigned ch4_flag:1;
  };
} ch_max_flag;


void AD_Init(void);


void AD_Init(void)
{
    unsigned char i;
    unsigned char j;
    PIE1bits.ADIE = 1;
    PIR1bits.ADIF = 0;
    ADCON2 = 0b10111110;            // AD Clock = Fosc/64, AD Aquisition Time = 0, Result right
justified
    ADCON1 = 0b00001011;            // AN0-AN4 analog, Vref+=Vdd; Vref-=Vss
    ADCON0 = 0b00000001;            // A/D Module is turned on

    for (i=0;i<5;i++){
        avgsum[i]=0;
        for(j=0;j<5;j++){
            adcval[i][j]=0;
        }
    }


    i=AVGWIDTH;
    for (avgshift = 0;i>1;avgshift++) {
        i >>= 1;
    }

}
```

C:\tcc\4620_remoto\buttons.c

```
// Buttons.c
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include "defines.h"
#include "buttons.h"

#pragma udata access unbanked

near union {
  unsigned char Whole;
  struct {

//  unsigned Button6:1;
//  unsigned Button5:1;
//  unsigned Button4:1;
//  unsigned Button3:1;
//  unsigned Button2:1;
    unsigned :4;
    unsigned Button1:1;

  };
} ButtonsState;

near union {
  unsigned char Whole;
  struct {
    unsigned FirstOk:1;
    unsigned RepeatOk:1;
    unsigned LongOk:1;
    unsigned PressDone:1;
    unsigned TimedOut:1;
    unsigned Long:1;
    unsigned Repeat:1;
    unsigned Buzzer:1;
  };
} TimeBaseFlags;

near volatile unsigned int ButtonsCounter;
near unsigned int TimeOutCounter;



#pragma code

unsigned char WaitButton()
{
  unsigned char temp;

#ifdef FORCED_TIMEOUT
  temp = TIMED_OUT;
  return temp;
#endif

Loop:
  if (TimeBaseFlags.TimedOut) {
    TimeBaseFlags.TimedOut = 0;
    return TIMED_OUT;
  };

  temp=PORTA&0x010;

  if ((temp!=ButtonsState.Whole)||(temp==(unsigned char)0x010)) {
    ButtonsCounter = 0;
    TimeBaseFlags.FirstOk = 0;
    TimeBaseFlags.RepeatOk = 0;
    TimeBaseFlags.LongOk = 0;
    TimeBaseFlags.PressDone = 0;
    if ((temp!=ButtonsState.Whole)&&(temp==(unsigned char)0x010)) {
      ButtonsState.Whole= temp;
```

```
      return KEY_RELEASE;
    } else {
      ButtonsState.Whole= temp;
      goto Loop;
    }
  };

   if (ButtonsCounter) {
    goto Loop;
  }

  if (!TimeBaseFlags.FirstOk) {
    ButtonsCounter=BUTTONFIRSTPRESS;
    TimeBaseFlags.FirstOk = 1;
    goto Loop;
  }

  if (!TimeBaseFlags.PressDone) {
    if (TimeBaseFlags.Repeat) {
      if (!TimeBaseFlags.RepeatOk) {
        TimeBaseFlags.RepeatOk = 1;
        ButtonsCounter=BUTTONREPEATSTART;
      } else {
        ButtonsCounter=BUTTONREPEATRATE;
      }
      goto Exit;
    } else if (TimeBaseFlags.Long) {
      if (!TimeBaseFlags.LongOk) {
        TimeBaseFlags.LongOk = 1;
        ButtonsCounter=BUTTONLONGPRESS;
        goto Exit;
      }
    }
    TimeBaseFlags.PressDone = 1;
    goto Exit;
  }

  goto Loop;

Exit:
  if (!ButtonsState.Button1)
    { temp=KEY1; }
  if (TimeBaseFlags.PressDone&&TimeBaseFlags.Long)
    { temp+=KEY_LONG; }
  return temp;
}

void TimeBaseInit() {
  ButtonsState.Whole =  0x0F;
  TimeBaseFlags.Whole = 0;
  ButtonsCounter = 0;
  TimeOutCounter = 0;
}

#pragma code ButtonsHi
void TimeOutDelaySet(unsigned int delay) {
  TimeOutCounter = delay;
}

#pragma code
void TimeOutDelayCancel() {
  TimeOutCounter = 0;
  TimeBaseFlags.TimedOut = 0;
}

void TimeBaseProcess ()
{
  if (TimeOutCounter) {
    if (!(--TimeOutCounter)) {
      TimeBaseFlags.TimedOut = 1;
    }
```

```
      }
      if (ButtonsCounter) {
        ButtonsCounter--;
      }
    }
```

C:\tcc\4620_remoto\buttons.h

```
// Buttons.h

#define KEY1    1
#define KEY2    2
#define KEY3    3
#define KEY4    4
#define KEY5    5
#define KEY6    6
#define KEY_RELEASE 32
#define KEY_LONG  64
#define TIMED_OUT 128

void TimeBaseProcess (void);
unsigned char WaitButton(void);
void TimeBaseInit(void);
void TimeOutDelaySet(unsigned int);
void TimeOutDelayCancel(void);

extern near union {
  unsigned char Whole;
  struct {

//  unsigned Button6:1;
//  unsigned Button5:1;
//  unsigned Button4:1;
//  unsigned Button3:1;
//  unsigned Button2:1;
    unsigned :4;
    unsigned Button1:1;
  };
} ButtonsState;

extern near union {
  unsigned char Whole;
  struct {
    unsigned FirstOk:1;
    unsigned RepeatOk:1;
    unsigned LongOk:1;
    unsigned PressDone:1;
    unsigned TimedOut:1;
    unsigned Long:1;
    unsigned Repeat:1;
    unsigned Buzzer:1;
  };
} TimeBaseFlags;
```

C:\tcc\4620_remoto\serial12.c

```
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include "interrupt.h"
#include "defines.h"
#include "mod_bus.h"



#pragma udata
far unsigned char n_send;
far unsigned char update;
far unsigned char n reciv;
far unsigned char frame_size;
far unsigned char module_address;

#pragma code
void conf serial(void){

    n_reciv=0;
    PIE1bits.RCIE=1;
    PIE1bits.TXIE=1;
    BAUDCON=0b00000000;
    TXSTA=0b00000100;
    RCSTA=0b10010000;
    IPR1=0b00000000;
    SPBRG=51;
    SPBRGH=0x00;
}


unsigned char CRC16(unsigned int dataLength,char check){ //CRC 16 for modbus checksum
    unsigned int CheckSum;
    unsigned int j;
    unsigned char lowCRC;
    unsigned char highCRC;
    unsigned char in;
    CheckSum = 0xffff;
    for (j=0; j<dataLength; j++){
        CheckSum = CheckSum^(unsigned int)buffer[j];
        for(in=8;in>0;in--)
            if((CheckSum)&0x0001)
                CheckSum = (CheckSum>>1)^0xa001;
            else
                CheckSum>>=1;
    }
    highCRC = CheckSum>>8;
    CheckSum<<=8;
    lowCRC = CheckSum>>8;
    if (check==1){
        if ( (buffer[dataLength+1] == highCRC) & (buffer[dataLength] == lowCRC ))
            return 1;
        else
            return 0;
    }
    else{
        buffer[dataLength] = lowCRC;
        buffer[dataLength+1] = highCRC;
        return 1;
    }
}
```

C:\tcc\4620_remoto\disp.c

```
#include <p18f4620.h>
#include "interrupt.h"
#include "buttons.h"
#include "lcd.h"
#include "adcycle.h"
#include "interface.h"
#include "mod_bus.h"
#include "serial12.h"
#include "rf24g.h"

#include <delays.h>
#include <timers.h>

#pragma udata
unsigned char Last_but;
unsigned char show_adc;
#pragma code
void main(void){

unsigned char aux;
unsigned char test;

unsigned int volt;

TRISA=0b00011111;
TRISB=0b11011111;
TRISC=0b11000000;
TRISD=0b00000000;
TRISE=0b00000000;

PORTC=0x00;
PORTD=0x00;
PORTB=0x00;

CCP1CON=0b00000000;
CCP2CON=0b00000000;
CMCON=0b00000111;

RCONbits.IPEN=1;
IPR2 = 0x00;
IPR1 = 0x00;

INTCONbits.INT0IF=0; //interrupt flag 0 button.
INTCONbits.INT0IE=1;
INTCON2bits.INTEDG0=1;


INTCONbits.T0IF = 0;
INTCON2bits.TMR0IP = 0;
INTCONbits.T0IE = 1;        //Enable TMR0 Overflow Interrupt

OpenTimer0(TIMER_INT_ON &   //Timer 0 is timebase for low priority interrupt
           T0_8BIT &
           T0_SOURCE_INT &
           T0_PS_1_32);

TMR0L = 0x9B;       //TIMER0FREQ is in "defines.h"

TimeBaseInit();
TMR0L = 0;
Delay10KTCYx(10);

INTCONbits.GIEH=0;
INTCONbits.GIEL=0;

AD_Init();

ADCON0bits.GO = 1; //habilita os ads
TimeBaseFlags.Long = 1;
TimeBaseFlags.Repeat=1;
```

```
T2CON=0b01111010; //Prescaler em 16 Postcaler em 2
PIE1bits.TMR2IE=1;
T2CONbits.TMR2ON=1;

LCD_Initialize();

Delay10KTCYx(10);

module_address=0x01;

n send=0;
n_reciv=0;

Delay10KTCYx(175);
Delay10KTCYx(175);
Delay10KTCYx(175);
Delay10KTCYx(175);

LCD_Clear();
LCD_PutROMStr(InterfaceMsg[0]);


TRISBbits.TRISB5=0;
Delay10KTCYx(20);

RF_24G_Config();

Delay10KTCYx(2);

RF_24G_SetRxByte();

Delay10KTCYx(2);

ADCON0bits.GO = 1; //habilita os ad

TimeBaseFlags.Whole = 0x00;
TimeBaseFlags.Long = 1;
TimeBaseFlags.Repeat=0;

INTCONbits.GIEH = 1;        //enable hi pri ints
INTCONbits.GIEL = 1;        //enable low pri ints


adc_res_max[0]=900;
adc_res_max[1]=900;
adc_res_max[2]=900;
adc_res_max[3]=900;

Coil_State.Whole=0;
show_adc=0;
do {
        TimeOutDelaySet(50);
        aux=WaitButton();
        switch (aux) {
          case 1: //CIMA
            Last_but=1;
            break;
          case 32: //cima
            if(Last_but==1){
                LCD_Clear();
                LCD_PutROMStr(InterfaceMsg[show_adc+8]);
                volt=(adc_res[show_adc]* 5000L) >> 10;
                SendValue(volt,0x40,1);
                show_adc++;
                if(show_adc==4){
                    show_adc=0;
                }
            Last_but=0;
```

```
            }
              break;
           case 65:
              ViewMenu();
           case 128:
              if(update==1){
                  RF_24G_SetTxByte();
                  process_reciv(); //process packe
                  RF 24G SetRxByte();
                  RF_24G_CE=1;
                  update=0;
              }
              break;

            default:

           break;
          }
     } while(1);
}
```

```
#pragma config OSC = HSPLL
//#pragma config BORV = 0
#pragma config WDT = OFF
//#pragma config STVR = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CPB = OFF, CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRTB = OFF, WRTC = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTRB = OFF
```

C:\tcc\4620_remoto\interface.c

```
//interface.c
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include "defines.h"
#include "LCD.h"
#include "buttons.h"
#include "adcycle.h"
#include "adc.h"
#include "interrupt.h"
#include "delays.h"
#include "serial12.h"

// FUNCTION Prototypes
void InitMsg(void);
void SendValue(unsigned int dispvalue, unsigned char initpos, unsigned char check);
unsigned char YES_NO(void);
void ViewMenu(void);
void SendMsg(unsigned char msgcode, unsigned char pos,unsigned char clear);

#pragma udata

#pragma romdata InterfaceMsgs
rom char InterfaceMsg[16][34] =
{
    "2.4 GHZ Wireless\n   MODBUS ",
    "Update Max Ch 1\n  YES( )   NO( )",
    "Update Max Ch 2\n  YES( )   NO( )",
    "Update Max Ch 3\n  YES( )   NO( )",
    "Update Max Ch 4\n  YES( )   NO( )",
    "New Max Alert:",
    "Clear Max Alert?\n  YES( )   NO( )",
    "DONE!!!",
    "CHANNEL 1:",
    "CHANNEL 2:",
    "CHANNEL 3:",
    "CHANNEL 4:",
    "",
    "Reconectado",
    "Address Setup:",
    "Save Data ?\n  YES( )   NO( )",
};

#pragma code
//rotina que escreve mensagem padrão
void InitMsg(void)
{
    LCD_Clear();
    LCD_PutROMStr(InterfaceMsg[0]);
#ifndef SIMUL
    Delay10KTCYx(100);
#endif
}

unsigned char YES_NO(void){
 unsigned char yes;
 unsigned char aux2;

  yes=0;

  LCD_GotoXY(0x4e);
  LCD_PutChar(\0x005);
  LCD_GotoXY(0x46);
  LCD_PutChar(\0x007);
  do{
    aux2=WaitButton();
    if(aux2==1){
      aux2=WaitButton();
      if(aux2==32){
        if(yes==0){
```

```
        yes=1;
        LCD_GotoXY(0x46);
        LCD_PutChar(\0x005);
        LCD_GotoXY(0x4e);
        LCD_PutChar(\0x007);
        }
        else{
        yes=0;
        LCD_GotoXY(0x4e);
        LCD_PutChar(\0x005);
        LCD_GotoXY(0x46);
        LCD_PutChar(\0x007);
        }
     }
    }
  }while(aux2!=65);
  return yes;
}

void Add_max(unsigned char channel){
  unsigned char aux2;

  SendMsg(5,0,1);
  SendValue(adc_res_max[channel],0x40,0);
  do{
    aux2=WaitButton();
    if(aux2==1){
      aux2=WaitButton();
      if(aux2==32){
        adc_res_max[channel]+=25;
        if(adc_res_max[channel]>=1023){
          adc_res_max[channel]=0;
        }
        SendValue(adc_res_max[channel],0x40,0);
     }
    }
  }while(aux2!=65);


}

void ViewMenu(void)
{
unsigned char aux, aux_address,ok;

  LCD_Clear();
  TimeOutDelayCancel();

  SendMsg(14,0,1);
  aux_address=module_address;
  SendValue(aux_address,0x40,0);

  do{
    aux=WaitButton();
    if(aux==1){
      aux=WaitButton();
      if(aux==32){
        aux_address++;
        SendValue(aux_address,0x40,0);
     }
    }
  }while(aux!=65);

  SendMsg(15,0,1);
  if(YES_NO()==1){
    module_address=aux_address;
    SendMsg(7,0,1);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
```

```
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
  }

  SendMsg(1,0,1);
  if(YES_NO()==1){
    Add_max(0);
    ch_max_flag.ch1_flag=0;
    SendMsg(7,0,1);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
  }

  SendMsg(2,0,1);
  if(YES_NO()==1){
    Add_max(1);
    ch_max_flag.ch2_flag=0;
    SendMsg(7,0,1);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
  }

  SendMsg(3,0,1);
  if(YES_NO()==1){
    Add_max(2);
    ch_max_flag.ch3_flag=0;
    SendMsg(7,0,1);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
  }

  SendMsg(4,0,1);
  if(YES_NO()==1){
    Add_max(3);
    ch_max_flag.ch4_flag=0;
    SendMsg(7,0,1);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
  }

  SendMsg(6,0,1);
  if(YES_NO()==1){
    ch_max_flag.Whole=0;
    SendMsg(7,0,1);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
    Delay1KTCYx(1000);
  }

//  EEPROMPut((ram far char*) &avisos, (rom near char*) &avisos_rom, 6 );

  LCD_Clear();
```

```
}

void SendValue(unsigned int dispvalue, unsigned char initpos, unsigned char check)
{
    unsigned int i;
    for(i=0;i<5;i++) DECValue[i]= '0';
    BinToBCD16(dispvalue);
    LCD_GotoXY(initpos);
    for(i=0;i<5;i++) {
        if (DECValue[i]==(unsigned char)'0') {
            DECValue[i]=' ';
            } else {
            break;
        }
    }

    for(i=1;i<5;i++) {
        LCD_PutChar(DECValue[i]);
    }

}

void SendMsg(unsigned char msgcode,unsigned char pos, unsigned char clear)
{
    if(clear){
        LCD_Clear();
    }
    LCD_GotoXY(pos);
    LCD_PutROMStr(InterfaceMsg[msgcode]);
}
```

C:\tcc\4620_remoto\interrupt.c

```
//interrupt.c
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include "disp.h"
#include "buttons.h"
#include "InlineAsm.h"
#include "adcycle.h"
#include "adc.h"
#include "mod_bus.h"
#include "serial12.h"
#include "rf24g.h"


#pragma udata
far unsigned char adc_avg1cnt; //n° da amostra buffer menor
far unsigned char adc_ch; //canal de ad
far unsigned char adc_amos; //n° da amostra
unsigned char write;


void InterruptHandlerLow(void);
void InterruptHandlerHigh (void);



#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh (void)
{
  _asm
   goto InterruptHandlerHigh //jump to interrupt routine
  _endasm
}

#pragma code
#pragma interrupt InterruptHandlerHigh

void InterruptHandlerHigh (void)
{

    if(INTCONbits.INT0IF==1){
          getBuffer();   // Get packet
//        SendValue(buffer[n_reciv],aux,1);
//        ++aux;
          ++n_reciv;
          if(n_reciv>=8){
            RF_24G_CE=0;
            update=1;
          }
        INTCONbits.INT0IF=0;
    }
}

//------------------------------------------------------------------------------
// Low priority interrupt vector and routine
#pragma code InterruptVectorLow = 0x18
void InterruptVectorLow (void)
{
  _asm
    goto InterruptHandlerLow //jump to interrupt routine
  _endasm
}

#pragma code
#pragma interruptlow InterruptHandlerLow //nosave=TABLAT, TBLPTR, TBLPTRU, PCLATH, PCLATU,
section("MATH_DATA")
                                         //save=PROD, section(".tmpdata")
 void InterruptHandlerLow (void)
{
    if (INTCONbits.T0IF) {
```

```
        TimeBaseProcess();
        _asm
            MOVF 0x9b,W,A
            ADDWF TMR0L,F,A
        _endasm
        INTCONbits.T0IF = 0;
    }
    if (PIR1bits.ADIF) {
        avgsum[adc_ch] += ReadADC();
      if (adc_avg1cnt==(unsigned char)(AVGWIDTH-1)) {
        adcval[adc_ch][AVGWIDTH]-= adcval[adc_ch][adc_amos];
        if(adcval[adc_ch][AVGWIDTH]>10000){
            adcval[adc_ch][AVGWIDTH]=0;
        }
        adcval[adc_ch][adc_amos]= avgsum[adc_ch]>>avgshift;
        adcval[adc_ch][AVGWIDTH]+=adcval[adc_ch][adc_amos];
        avgsum[adc_ch] = 0;
        adc_res[adc_ch]=adcval[adc_ch][AVGWIDTH]>>avgshift;
        if(adc_res[adc_ch]>adc_res_max[adc_ch]){
                ch_max_flag.Whole=(ch_max_flag.Whole|0b00000001<<adc_ch);
        }

        if (adc_ch==3){
          if (++adc_amos==(unsigned char)AVGWIDTH) {
            adc_amos=0;
          }
        }
      }
      if (++adc_ch==(unsigned char)4) {
        adc_ch=0;
        if (++adc_avg1cnt==(unsigned char)AVGWIDTH) {
          adc_avg1cnt=0;
        }
      }
      PIR1bits.ADIF = 0;
    }


    if (PIR1bits.TMR2IF) {
      ADCON0 = (adc_ch<<2) | 1;
      ADCON0bits.GO = 1;
      PIR1bits.TMR2IF = 0;
    }

    if (PIR1bits.RCIF) {
        buffer[n_reciv]=RCREG;
        n_reciv+=1;
        if(n_reciv>=8){
            RCSTAbits.CREN=0;
            update=1;
        }
        PIR1bits.RCIF=0;
    }
    if (PIR1bits.TXIF) {
        if(n_send<frame_size){
        T0CONbits.TMR0ON=0;
        T2CONbits.TMR2ON=0;
        TXREG=buffer[n_send];
        n_send++;
        }
        else{
            while(TXSTAbits.TRMT==0);
            T0CONbits.TMR0ON=1;
            T2CONbits.TMR2ON=1;
            TXSTAbits.TXEN=0;
            n_send=0;
        }
        PIR1bits.TXIF=0;
    }
}
```

C:\tcc\4620_remoto\LCD.c

```
// LCD.C
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include <delays.h>
#include "LCD.h"
#include "defines.h"
#include "InlineAsm.h"
//#include "asmmath.h"

//Pode vir a ser mapeado externamente
#define LCD_RS_LAT LATDbits.LATD0
#define LCD_RW_LAT LATDbits.LATD1
#define LCD_E_LAT  LATDbits.LATD2
#define LCD_OUT_PORT PORTD
#define LCD_OUT_LATCH LATD
#define LCD_OUT_SETUP TRISD
#define LCD_MASK    0b00001111
#define LCD_LASTBIT 0
void LCD_Put(void);
unsigned char LCD_ReadFlag(void);

typedef union {
  unsigned char Whole;
  struct {
    unsigned GotoXY:1;
  };
} LCDStatType;

#pragma udata DECValues = 0x100
far ram unsigned char DECValue[6];

#pragma udata
far ram LCDStatType LCDStat;
far ram unsigned char ch;
far ram char* tempptr;
far ROMBytePtr romtempptr;
far ram unsigned char temp, flag;
far ram unsigned char mode, nibb;

#pragma udata access unbanked

#pragma romdata LCDBitmaps
rom near char LCDBitmap[8*8] =
{0b00000,
 0b00000,
 0b01110,
 0b10000,
 0b10000,
 0b10010,
 0b01111,
 0b00100,

 0b00101,
 0b01010,
 0b01110,
 0b00001,
 0b01111,
 0b10001,
 0b01111,
 0b00000,

 0b00010,
 0b00100,
 0b01110,
 0b10001,
 0b10001,
 0b10001,
 0b01110,
 0b00000,
```

```
 0b00000,
 0b00000,
 0b00100,
 0b01000,
 0b10000,
 0b01000,
 0b00100,
 0b00000,

 0b00010,
 0b00100,
 0b01110,
 0b00001,
 0b01111,
 0b10001,
 0b01111,
 0b00000,

 0b00000,
 0b00000,
 0b10001,
 0b01010,
 0b00100,
 0b01010,
 0b10001,
 0b00000,

 0b00010,
 0b00100,
 0b01110,
 0b10001,
 0b11111,
 0b10000,
 0b01110,
 0b00000,

 0b00000,
 0b00000,
 0b00000,
 0b00000,
 0b00000,
 0b00000,
 0b00000,
 0b00000

};

#pragma code LCDSection

void NOPDelay(void)
{
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
  Nop();
}

void NOP4(void)
{
  Nop();
  Nop();
```

```
  Nop();
  Nop();
}

void LCD_PutData(unsigned char c) {
#if (LCD_MODE==LCD_8BITS)
      LCD_OUT_LATCH = c;
      LCD_RS_LAT = 1;    //Seta display para modo de tranferencia de dados
      LCD_Put();
#else
      #if (PORTNIB)      //Usa upper bits do port
          LCD_RS_LAT = 1;//Seta display para modo de tranferencia de dados
          nibb = (LCD_OUT_PORT & 0x0F) | (c & 0xF0);
          LCD_OUT_SETUP = LCD_OUT_SETUP & 0x0F;
          LCD_OUT_LATCH = nibb;
          LCD_Put();
          NOPDelay();
          nibb = (nibb&0x0F) | ((c & 0x0F)<<4);
          LCD_OUT_LATCH = nibb;
          LCD_Put();
      #else              //Usa lower bits do port
          nibb = (LCD_OUT_PORT & 0xF0) | ((c & 0xF0)>>4);
          LCD_OUT_SETUP = LCD_OUT_SETUP & 0xF0;
          LCD_OUT_LATCH = nibb;
          LCD_RS_LAT = 1;    //Seta display para modo de tranferencia de dados
          LCD_Put();
          NOPDelay();
          nibb = (nibb&0xF0) | (c & 0x0F);
          LCD_OUT_LATCH = nibb;
          LCD_Put();
      #endif
#endif
}



void LCD_PutInstr(unsigned char i) {
#if (LCD_MODE==LCD_8BITS)
      LCD_OUT_LATCH = i;
      LCD_RS_LAT = 0;    //Seta display para modo de tranferencia de instrucoes
      LCD_Put();
#else
      #if (PORTNIB)      //Usa upper bits do port
          nibb = (LCD_OUT_PORT & 0x0F) | (i & 0xF0);
          LCD_OUT_SETUP = LCD_OUT_SETUP & 0x0F;
          LCD_OUT_LATCH = nibb;
          LCD_RS_LAT = 0;    //Seta display para modo de tranferencia de instrucoes
          LCD_Put();
          if ((i != (unsigned char)LCD_SET_4BITS) && (i != (unsigned char)LCD_SET)) {
          NOP4();
          nibb = (nibb&0x0F) | ((i & 0x0F)<<4);
          LCD_OUT_LATCH = nibb;
          LCD_Put();
            }
      #else              //Usa lower bits do port
          nibb = (LCD_OUT_PORT & 0xF0) | ((i & 0xF0)>>4);
          LCD_OUT_SETUP = LCD_OUT_SETUP & 0xF0;
          LCD_OUT_LATCH = nibb;
          LCD_RS_LAT = 0;    //Seta display para modo de tranferencia de instrucoes
          LCD_Put();
          if ((i != (unsigned char)LCD_SET_4BITS) && (i != (unsigned char)LCD_SET)) {
          NOP4();
          nibb = (nibb&0xF0) | (i & 0x0F);
          LCD_OUT_LATCH = nibb;
          LCD_Put();
            }
      #endif
#endif
}
```

```
void LCD_Put(void) {
  LCD_RW_LAT = 0;
  INTCONbits.GIE = 0;
  NOP4();
  NOP4();
  LCD_E_LAT = 1;
  NOPDelay();
  NOPDelay();
  LCD_E_LAT = 0;
  INTCONbits.GIE = 1;
}

#pragma udata

void LCD_WaitBusy(void) {
  do {
    temp=LCD_ReadFlag();
    #ifdef SIMUL
        temp = 0;
    #endif
    } while (temp!=(unsigned char)0);
}

unsigned char LCD_CheckBusy(void) {
  temp=LCD_ReadFlag();
#ifdef SIMUL
  return 0;
#else
  return temp;
#endif
}

unsigned char LCD_ReadFlag(void) {
    LCD_RS_LAT = 0;
    LCD_RW_LAT = 1;
    NOP4();
#if (LCD_MODE==LCD_8BITS)
    LCD_OUT_SETUP = 255;
    LCD_E_LAT = 1;
    NOPDelay();
    flag = LCD_OUT_PORT & 0x80;
    LCD_E_LAT =0;
    LCD_OUT_SETUP = 0;
    return flag;
#else
      #if (PORTNIB)      //Usa upper bits do port
          LCD_OUT_SETUP = LCD_OUT_SETUP | 0xF0;
          LCD_E_LAT = 1;
          NOPDelay();
          flag = LCD_OUT_PORT & 0x80;
          LCD_E_LAT =0;
          NOP4();
          LCD_E_LAT = 1;
          NOPDelay();
          LCD_E_LAT =0;
          //LCD_OUT_SETUP = LCD_OUT_SETUP & 0x0F;
          return flag;
      #else              //Usa lower bits do port
          LCD_OUT_SETUP = LCD_OUT_SETUP | 0x0F;
          LCD_E_LAT = 1;
          NOPDelay();
          flag = LCD_OUT_PORT & 0x08;
          LCD_E_LAT =0;
          NOP4();
          LCD_E_LAT = 1;
          NOPDelay();
          LCD_E_LAT =0;
          LCD_OUT_SETUP = LCD_OUT_SETUP & 0xF0;
          return flag;
      #endif
#endif
```

```
}

void LCD_PutChar(char c)
{
  if (LCDStat.GotoXY) {
    LCD_PutInstr(LCD_SET_DDRAM_ADDR | c);
    LCDStat.GotoXY = 0;
  } else {
    switch (c) {
    case '\n':
      LCD_PutInstr(LCD_SET_DDRAM_ADDR | 0x40);
      break;
    case '\f':
      LCDStat.GotoXY = 1;
      break;
    default:
      LCD_PutData(c);
    }
  }
  LCD_WaitBusy();
}

void LCD_PutRAMStr(ram char* str)
{
  tempptr= str;
  while (*tempptr)
    {
    LCD_PutChar(*tempptr++);
    }
}

void LCD_PutROMStr(rom char* str)
{
  romtempptr= str;
  while (*romtempptr)
    {
    LCD_PutChar(*romtempptr++);
    }
}

void LCD_GotoXY(unsigned char XY)
{
  LCD_PutInstr(LCD_SET_DDRAM_ADDR | XY);
  LCD_WaitBusy();
}

void LCD_Clear()
{
  LCD_PutInstr(LCD_DISPLAY_CLEAR);
  LCD_WaitBusy();
}

void BINTOBCD8(unsigned char value)
{
  _asm
    MOVLB   1
    MOVLW   '0'
    MOVWF   DECValue,B
    MOVWF   DECValue+1,B
    MOVWF   DECValue+2,B
    MOVLW   0xFE
    MOVF    PLUSW2,W,A
BINTOBCD81:
    ADDLW   -100
    BTFSS   STATUS,CARRY,A
    BRA     BINTOBCD82
    INCF    DECValue,F,B
    BRA     BINTOBCD81
BINTOBCD82:
    ADDLW   100
BINTOBCD83:
```

```
    ADDLW   -10
    BTFSS   STATUS,CARRY,A
    BRA     BINTOBCD84
    INCF    DECValue+1,F,B
    BRA     BINTOBCD83
BINTOBCD84:
    ADDLW   10
    ADDWF   DECValue+2,F,B
    CLRF    DECValue+3,B
  _endasm
}


void BinToBCD16(unsigned int value)
{
  _asm
    MOVLW   0xFD
    MOVFF   PLUSW2,PRODL
    MOVLW   0xFE
    MOVFF   PLUSW2,PRODH
    MOVLB   1
    MOVLW   '0'
    MOVWF   DECValue,B
    MOVWF   DECValue+1,B
    MOVWF   DECValue+2,B
    MOVWF   DECValue+3,B
    MOVWF   DECValue+4,B
PERTOBCD160:
    MOVLW   16          //low 10000
    SUBWF   PRODL,F,A
    MOVLW   39          //high 10000
    SUBWFB  PRODH,F,A
    BTFSS   STATUS,CARRY,A
    BRA     PERTOBCD169
    INCF    DECValue,F,B
    BRA     PERTOBCD160
PERTOBCD169:
    MOVLW   16          //low 10000
    ADDWF   PRODL,F,A
    MOVLW   39          //high 10000
    ADDWFC   PRODH,F,A

PERTOBCD161:
    MOVLW   232         //low 1000
    SUBWF   PRODL,F,A
    MOVLW   3           //high 1000
    SUBWFB  PRODH,F,A
    BTFSS   STATUS,CARRY,A
    BRA     PERTOBCD162
    INCF    DECValue+1,F,B
    BRA     PERTOBCD161
PERTOBCD162:
    MOVLW   232         //low 1000
    ADDWF   PRODL,F,A
    MOVLW   3           //high 1000
    ADDWFC   PRODH,F,A
PERTOBCD163:
    MOVLW   100         //low 100
    SUBWF   PRODL,F,A
    MOVLW   0           //high 100
    SUBWFB  PRODH,F,A
    BTFSS   STATUS,CARRY,A
    BRA     PERTOBCD164
    INCF    DECValue+2,F,B
    BRA     PERTOBCD163
PERTOBCD164:
    MOVLW   100         //low 100
    ADDWF   PRODL,F,A
    MOVLW   0           //high 100
    ADDWFC   PRODH,F,A
PERTOBCD165:
```

```
    MOVLW   10          //low 10
    SUBWF   PRODL,F,A
    MOVLW   0           //high 10
    SUBWFB  PRODH,F,A
    BTFSS   STATUS,CARRY,A
    BRA     PERTOBCD166
    INCF    DECValue+3,F,B
    BRA     PERTOBCD165
PERTOBCD166:
    MOVLW   10          //low 10
    ADDWF   PRODL,W,A
PERTOBCD167:
    ADDWF   DECValue+4,F,B
  _endasm
}



#pragma udata

void LCD_PutBitMap(void)
{
  LCD_PutInstr(LCD_SET_CGRAM_ADDR);
  LCD_WaitBusy();
  for (ch=0; ch<(unsigned char)64; ch++) {
    LCD_PutData(LCDBitmap[ch]);
    LCD_WaitBusy();
  }
  LCD_PutInstr(LCD_SET_DDRAM_ADDR);
  LCD_WaitBusy();
}

void LCD_Initialize() {

  unsigned char i;

  LCDStat.Whole = 0;
  for (i=0;i<6;i++){
      DECValue[i] = 0x30;
  }
  // configure external LCD
#if (LCD_MODE==LCD_8BITS)
  Delay1KTCYx(150);          // Waits 150,000 cycles = 15 ms.
  Delay1KTCYx(150);          // Waits 150,000 cycles = 15 ms.
  LCD_PutInstr(LCD_FUNCTION_SET);
  Delay1KTCYx(50);               // Waits 50,000 cycles = 5 ms.
  Delay1KTCYx(50);               // Waits 50,000 cycles = 5 ms.
  LCD_PutInstr(LCD_FUNCTION_SET);
  Delay1KTCYx(1);                // Waits 1000 cycles = 100 us.
  LCD_PutInstr(LCD_FUNCTION_SET);
  Delay1KTCYx(50);               // Waits 50,000 cycles = 5 ms.
  Delay1KTCYx(50);               // Waits 50,000 cycles = 5 ms.
  LCD_PutInstr(LCD_FUNCTION_SET);
#else
  Delay1KTCYx(175);          // Waits 150,000 cycles = 15 ms.
  Delay10KTCYx(150);             // Waits 150,000 cycles = 15 ms.
  Delay10KTCYx(150);             // Waits 150,000 cycles = 15 ms.
  LCD_PutInstr(LCD_SET);
  Delay10KTCYx(25);              // Waits 50,000 cycles = 5 ms.
  Delay10KTCYx(50);              // Waits 50,000 cycles = 5 ms.
  LCD_PutInstr(LCD_SET);
  Delay10KTCYx(1);               // Waits 1000 cycles = 100 us.
  LCD_PutInstr(LCD_SET);
  Delay10KTCYx(25);              // Waits 50,000 cycles = 5 ms.
  Delay10KTCYx(50);              // Waits 50,000 cycles = 5 ms.
  LCD_PutInstr(LCD_SET_4BITS);
  Delay10KTCYx(25);              // Waits 50,000 cycles = 5 ms.
  Delay10KTCYx(50);              // Waits 50,000 cycles = 5 ms.
  LCD_PutInstr(LCD_FUNCTION_SET_4BITS);
#endif
  LCD_WaitBusy();
```

```
  LCD_PutInstr(LCD_DISPLAY_OFF);
  LCD_WaitBusy();
  LCD_PutInstr(LCD_DISPLAY_CLEAR);
  LCD_WaitBusy();
  LCD_PutInstr(LCD_ENTRY_MODE_SET);
  LCD_WaitBusy();
  LCD_PutInstr(LCD_DISPLAY_ON_CURSOR_OFF_BLINK_OFF);
  LCD_WaitBusy();
  LCD_PutBitMap();
}
```

C:\tcc\4620_remoto\mod_bus.c

```
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include "defines.h"
#include "serial12.h"
#include "adcycle.h"
#include "rf24g.h"


#pragma udata

unsigned char buffer[13];

unsigned char received;

#pragma udata access unbanked
near union {
  unsigned char Whole;
  struct {
    unsigned Coil_1:1;
    unsigned Coil_2:1;
    unsigned Coil_3:1;
    unsigned Coil_4:1;
    unsigned :4;
  };
} Coil_State;

near union {
  unsigned char Whole;
  struct {
    unsigned Input_1:1;
    unsigned Input_2:1;
    unsigned Input_3:1;
    unsigned Input_4:1;
    unsigned :4;
  };
} Input_State;


#pragma romdata eedata=0xF00000

rom union{
    struct{
           char on_off;
           char max_temp;
           char max_diff;
           char max_diff_up;
           char som;
           char som2;
          };
     } avisos_rom = {0, MAXTEMP_STA, MAXDIPF_STA, MAXDIPFUP_STA,1,0};
rom unsigned int holding_reg_rom[4]={0,0,0,0};
rom unsigned char Slave_Address_rom;



#pragma code
unsigned char CheckRSInBuffer(void)
{
unsigned int j = 0;
while((!received) && (j < 30000)) j++;
    return received;
}

unsigned char return_error( unsigned char error){

    buffer[1]+=0x80;
    buffer[2]=error;
    if( CRC16(3,0)){
        frame_size=5;
```

```
        putBuffer(frame_size);
        return 1; //OK
    }
    else
        return 0; //bad CRC

}


unsigned char ReadMBFrame(void){

    unsigned int counter = 0;
    unsigned char i = 0;
    counter=n_reciv;
    n_reciv=0;

    if (counter > 74) return 0; //to many bytes in frame
    if (counter < 8) return 0; //to few bytes frame
    if (buffer[0] != module_address) return 0; //wrong address
    if ((buffer[1] != 1)&(buffer[1] != 2)&(buffer[1] != 3)&(buffer[1] != 4)&(buffer[1] !=
5)&(buffer[1] != 6)){ //wrong modbus function
        i=return_error(1);
        return 0;
    }
    if( CRC16(counter - 2,1))
        return buffer[1]; //OK
    else
        return 0;    //bad CRC
}



unsigned char address_error(unsigned char check_type){


    unsigned int start_address;
    unsigned int address_limit;
    unsigned int output_value;
    unsigned int output_address;
    unsigned int n_limit;
    unsigned char i;

    n_limit=0;
    start_address=0;
    address_limit=0;
    output_value=0;
    output_address=0;
    if(check_type == 1){
        n_limit=buffer[4];
        n_limit<<=8;
        n_limit+=buffer[5];
        if((n_limit<0x01)|(n_limit>0x07d0)){
            i=return_error(3);
            return 0;
        }
        start_address=buffer[2];
        start_address<<=8;
        start_address+=buffer[3];
        address_limit=start_address+n_limit;
        if((address_limit>=0x05)|(start_address>=0x05)){
            i=return_error(2);
            return 0;
        }
        return 1;
    }
    else if(check_type == 2){
        output_value=buffer[4];
        output_value<<=8;
        output_value+=buffer[5];
        if((output_value!=0xFF00)&(output_value!=0x0000)){
            i=return_error(3);
```

```
            return 0;
        }
        output_address=buffer[2];
        output_address<<=8;
        output_address+=buffer[3];
        if((output_address>0x05)){
            i=return_error(2);
            return 0;
        }
        return 1;
    }
    else if(check_type == 3){
        output_value=buffer[4];
        output_value<<=8;
        output_value+=buffer[5];
        if((output_value==0xFF00)|(output_value==0x0000)){
            i=return_error(3);
            return 0;
        }
        output_address=buffer[2];
        output_address<<=8;
        output_address+=buffer[3];
        if((output_address>0x05)){
            i=return_error(2);
            return 0;
        }
        return 1;
    }
}


unsigned char process_rq_1(void){

    unsigned char start_address;
    unsigned char n_coil;
    unsigned char n_shift;
    unsigned char mask;

    n_coil=PORTC&0b00001111;
    n_coil=buffer[5];
    start_address=buffer[3];
    buffer[2]=0x01;
    n_shift=(start_address);
    buffer[3]=(Coil_State.Whole>>n_shift);
    n_shift=(8-n_coil);
    mask=((0xff)>>n_shift);
    buffer[3]=((buffer[3])&mask);
    CRC16(4,0);
    frame_size=6;
    putBuffer(frame_size);
    return 1;


}
unsigned char process_rq_2(void){

    unsigned char start_address;
    unsigned char n_input;
    unsigned char n_shift;
    unsigned char mask;
    unsigned char PORTAB;

    PORTAB=PORTB;
    PORTAB=PORTAB>>1;
    Input_State.Whole=PORTAB&0b00001111;
    n_input=buffer[5];
    start_address=buffer[3];
    buffer[2]=0x01;
    n_shift=(start_address);
    buffer[3]=(Input_State.Whole>>n_shift);
    n_shift=(8-n_input);
```

```
    mask=((0xff)>>n_shift);
    buffer[3]=((buffer[3])&mask);
    CRC16(4,0);
    frame_size=6;
    putBuffer(frame_size);
    return 1;
}
unsigned char process_rq_3(void){

    unsigned char start_address;
    unsigned char n_input;
    unsigned char n_bits;
    unsigned int aux_adc;
    unsigned char i;

    n_input=buffer[5];
    start_address=buffer[3];
    buffer[2]=n_input<<1;
    n_bits=buffer[2]+1;
    for(i=1;i<n_bits;i+=2){
        aux_adc=adc_res_max[start_address];
        buffer[2+i]=aux_adc>>8;
        aux_adc<<=8;
        buffer[2+i+1]=aux_adc>>8;
        start_address+=1;
    }
    n_input=(buffer[2]+3);
    CRC16(n_input,0);
    frame_size=n_input+2;
    putBuffer(frame_size);
    return 1;
}

unsigned char process_rq_4(void){

    unsigned char start_address;
    unsigned char n_input;
    unsigned char n_bits;
    unsigned int aux_adc;
    unsigned char i;

    n_input=buffer[5];
    start_address=buffer[3];
    buffer[2]=n_input<<1;
    n_bits=buffer[2]+1;
    for(i=1;i<n_bits;i+=2){
        aux_adc=adc_res[start_address];
        buffer[2+i]=aux_adc>>8;
        aux_adc<<=8;
        buffer[2+i+1]=aux_adc>>8;
        start_address+=1;
    }
    n_input=(buffer[2]+3);
    CRC16(n_input,0);
    frame_size=n_input+2;
    putBuffer(frame_size);
    return 1;
}
unsigned char process_rq_5(void){

    unsigned char coil_address;
    unsigned int coil_fstate;
    unsigned char n_bits;
    unsigned char mask;
    unsigned int check;


    coil_address=buffer[3];
    coil_fstate=buffer[4];
    coil_fstate=coil_fstate<<8;
    coil_fstate=coil_fstate+buffer[5];
```

```
    if((coil_fstate==0xFF00)|(coil_fstate==0x0000)){
        if(coil_fstate==0xFF00){
            mask=1<<coil_address;
            Coil_State.Whole=Coil_State.Whole|mask;
            PORTC=PORTC|Coil_State.Whole;
        }else{
            mask=1<<coil_address;
            mask=~mask;
            Coil_State.Whole=Coil_State.Whole&mask;
            PORTC=PORTC&Coil_State.Whole;
        }
    }else{
        mask=return_error(2);
        return 1;
    }
    check=PORTC&0b00001111;
    if(check!=Coil_State.Whole){
       return 0;
    }else{
     frame_size=8;
     putBuffer(frame_size);
    }
    return 1;
}

unsigned char process_rq_6(void){

    unsigned char reg_address;
    unsigned int n_input_hi;
    unsigned int n_input_lo;
    unsigned int check;

    n_input_hi=buffer[4];
    n_input_lo=buffer[5];
    reg_address=buffer[3];
    adc_res_max[reg_address]=n_input_hi<<8;
    check=n_input_hi<<8;
    adc_res_max[reg_address]=adc_res_max[reg_address]+n_input_lo;
    check=check+n_input_lo;
    if(check!=adc_res_max[reg_address]){
       return 0;
    }else{
     frame_size=8;
     putBuffer(frame_size);
    }
    return 1;
}

void process_reciv(void){

    unsigned int i;


    i=ReadMBFrame();
    switch (i){
        case 0:
        break;
        case 1:
            i=address_error(0x01);
            if(i==1){
                i=process_rq_1();
                if(i==0){
                    i=return_error(4);
                }
            }
        break;
        case 2:
            i=address_error(0x01);
            if(i==1){
                i=process_rq_2();
                if(i==0){
```

```
                    i=return_error(4);
                }
            }
        break;
        case 3:
            i=address_error(0x01);
            if(i==1){
                i=process_rq_3();
                if(i==0){
                    i=return_error(4);
                }
            }
        break;
        case 4:
            i=address_error(0x01);
            if(i==1){
                i=process_rq_4();
                if(i==0){
                    i=return_error(4);
                }
            }
        break;
        case 5:
            i=address_error(0x02);
            if(i==1){
                i=process_rq_5();
                if(i==0){
                    i=return_error(4);
                }
            }
        break;
        case 6:
            i=address_error(0x03);
            if(i==1){
                i=process_rq_6();
                if(i==0){
                    i=return_error(4);
                }
            }
        break;
        default:
        break;
    }

}
```

C:\tcc\4620_PC\RF-24G.c

```
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include <delays.h>
#include "RF24G.h"
#include "serial12.h"

////////////////////////////////////////////////////////////////////////////

#define numOfBytes      1


////////////////////////////////////////////////////////////////////////////
//Configuration Bytes
//
//Bytes 14-02: Shockburst Configuration
//Bytes 01-00: General Device Configuration

//Byte 14: Length of data payload section RX channel 2 in bits
//    The total number of bits in a ShockBurst RF package may not exceed 256!
//    Maximum length of payload section is hence given by:
//    DATAx_W(bits) = 256 - ADDR_W - CRC
#define DATA2_W             numOfBytes * 8

//Byte 13: Length of data payload section RX channel 1 in bits
#define DATA1_W             numOfBytes * 8

//Byte 12-08: Channel 2 Address
#define ADDR2_4             0x01
#define ADDR2_3             0x02
#define ADDR2_2             0x03
#define ADDR2_1             0x42
#define ADDR2_0             0x41

//Byte 07-03: Channel 1 Address
#define ADDR1_4             0x01
#define ADDR1_3             0x02
#define ADDR1_2             0x03
#define ADDR1_1             0x42
#define ADDR1_0             0x41

//Byte 02
//    Bit 07-02: ADDR_W     - Number of address bits (both RX channels)
//                            Maximum number of address bits is 40 (5 bytes)
//    Bit    01: CRC_L      - 8 or 16 bits CRC
//    Bit    00: CRC_EN     - Enable on-chip CRC generation/checking
//    Combine (via |) together constants from each group
//                          0b76543210
#define ADDR_W_5_BYTE       0b10100000
#define ADDR_W_4_BYTE       0b10000000
#define ADDR_W_3_BYTE       0b01100000
#define ADDR_W_2_BYTE       0b01000000
#define ADDR_W_1_BYTE       0b00100000

#define CRC_L_8_BIT         0b00000000
#define CRC_L_16_BIT        0b00000010

#define CRC_EN_DISABLE      0b00000000
#define CRC_EN_ENABLE       0b00000001

//Byte 01
//    Bit    07: RX2_EN     - Enable two channel receive mode
//    Bit    06: CM         - Communication mode ( Direct or ShockBurst)
//    Bit    05: RFDR_SB    - RF data rate (1Mbps requires 16MHz crystal)
//    Bit 04-02: XO_F       - Crystal frequency (Factory default 16MHz crystal mounted)
//    Bit 01-00: RF_PWR     - RF output power
//    Combine (via |) together constants from each group
//                          0b76543210
#define RX2_EN_DISABLE      0b00000000
```

```
#define RX2_EN_ENABLE       0b10000000

#define CM_DIRECT           0b00000000
#define CM_SHOCKBURST       0b01000000

#define RFDR_SB_250_KBPS    0b00000000
#define RFDR_SB_1_MBPS      0b00100000

#define XO_F_4MHZ           0b00000000
#define XO_F_8MHZ           0b00000100
#define XO_F_12MHZ          0b00001100
#define XO_F_16MHZ          0b00001100
#define XO_F_20MHZ          0b00010000

#define RF_PWR_N20DB        0b00000000  // -20db
#define RF_PWR_N10DB        0b00000001  // -10db
#define RF_PWR_N5DB         0b00000010  // -5db
#define RF_PWR_0DB          0b00000011  // 0db (Full Power)

//Byte 01
//    Bit 07-01: RF_CH#     - Frequency channel (2400MHz + RF_CH# * 1.0MHz)
//    Bit    00: RXEN       - RX or TX operation
//    Combine (via |) together constants from each group
//                          0b76543210
#define RF_CH               0b11100000 // 64 - 2464GHz

#define RXEN_TX             0b00000000
#define RXEN_RX             0b00000001
////////////////////////////////////////////////////////////////////////////////

#define BUF_MAX             numOfBytes
#define PWUPDELAY()         Delay1KTCYx(175);
int buf[BUF_MAX];


void CLKDELAY(void){
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
}
void CSDELAY(void){
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
```

```
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
}

void putByte( unsigned char b ) {  //MSB bit first

   int i;

     unsigned char cnt;
     unsigned char data;
     data=b;
     RF_24G_CLK1=0;
     for(cnt=8;cnt>0;cnt--)
        {
         if((data&0x80)>0)
            {RF_24G_DATA=1;}
         else
            {RF_24G_DATA=0;}
        CLKDELAY();
        RF_24G_CLK1=1;
        CLKDELAY();
        RF_24G_CLK1=0;
        data<<=1;
        }
RF_24G_DATA=0;
```

```
}

unsigned char getByte(void) {  //MSB bit first
   int i = 0;
   unsigned char cnt,data;

     data=0;
     for(cnt=8;cnt>0;cnt--)
        {
         data<<=1;
         if(RF_24G_DATA)
            {data|=0x01;}
         else
            {data&=0xfe;}
        CLKDELAY();
        RF_24G_CLK1=1;
        CLKDELAY();
        RF_24G_CLK1=0;
        }
     return(data);

}

void RF_24G_Config(void) {
   RF_24G_CE=0;
   RF_24G_CS=0;
   RF_24G_CLK1=0;
   RF_24G_DATA=0;
   PWUPDELAY();
   RF_24G_CS=1;
   CSDELAY();

   putByte(DATA2_W);
   putByte(DATA1_W);
   putByte(ADDR2_0);
   putByte(ADDR2_1);
   putByte(ADDR2_2);
   putByte(ADDR2_3);
   putByte(ADDR2_4);
   putByte(ADDR2_0);
   putByte(ADDR2_1);
   putByte(ADDR2_2);
   putByte(ADDR2_3);
   putByte(ADDR2_4);
   putByte(ADDR_W_5_BYTE | CRC_L_16_BIT | CRC_EN_ENABLE);
   putByte(RX2_EN_DISABLE | CM_SHOCKBURST | RFDR_SB_250_KBPS | XO_F_16MHZ | RF_PWR_0DB);
   putByte(RF_CH | RXEN_RX);

   RF_24G_CE=0;
   RF_24G_CS=0;
   RF_24G_CLK1=0;
}



void RF_24G_SetTxByte(void) {
   TRISBbits.TRISB5=0;
   RF_24G_CE=0;
   RF_24G_CS=1;
   CSDELAY();
   putByte(RF_CH | RXEN_TX);
   RF_24G_CS=0;
   RF_24G_CLK1=0;
}
void RF_24G_SetRxByte(void) {
   TRISBbits.TRISB5=0;
   RF_24G_CE=0;
   RF_24G_CS=1;
   CSDELAY();
   putByte(RF_CH | RXEN_RX);
   RF_24G_CS=0;
```

```
    TRISBbits.TRISB5=1;
    RF_24G_CLK1=0;
    RF_24G_CE=1;
}

void putBuffer(char frame_size) {
    int i;

    for( i=0; i<frame_size ; i++) {
        RF_24G_CE=1;
        CSDELAY();
        putByte(ADDR2_0);
        putByte(ADDR2_1);
        putByte(ADDR2_2);
        putByte(ADDR2_3);
        putByte(ADDR2_4);
        putByte(buffer[i]);
        RF_24G_CE=0;
        RF_24G_CLK1=0;
        Delay10KTCYx(2);
    }
}

void getBuffer(void) {

    buffer[n_reciv_w] = getByte();
    RF_24G_CLK1=0;
    RF_24G_CE=1;
}
```

```
//adcycle.h

#include "defines.h"

extern far unsigned int adcval[4][(AVGWIDTH+1)];
extern far unsigned int avgsum[(AVGWIDTH)];
extern far unsigned int adc_res[4];
extern far unsigned int adc_res_max[4];
extern far unsigned char avgshift;

extern far union {
  unsigned char Whole;
  struct {
    unsigned ch1_flag:1;
    unsigned ch2_flag:1;
    unsigned ch3_flag:1;
    unsigned ch4_flag:1;
  };
} ch_flag;
extern far union {
  unsigned char Whole;
  struct {
    unsigned ch1_flag:1;
    unsigned ch2_flag:1;
    unsigned ch3_flag:1;
    unsigned ch4_flag:1;
  };
} ch_max_flag;

// FUNCTION Prototypes
void AD_Init(void);


// Buttons.h

#define KEY1    1
#define KEY2    2
#define KEY3    3
#define KEY4    4
#define KEY5    5
#define KEY6    6
#define KEY_RELEASE 32
#define KEY_LONG  64
#define TIMED_OUT 128

void TimeBaseProcess (void);
unsigned char WaitButton(void);
void TimeBaseInit(void);
void TimeOutDelaySet(unsigned int);
void TimeOutDelayCancel(void);

extern near union {
  unsigned char Whole;
  struct {

//  unsigned Button6:1;
//  unsigned Button5:1;
//  unsigned Button4:1;
//  unsigned Button3:1;
//  unsigned Button2:1;
    unsigned :4;
    unsigned Button1:1;
  };
} ButtonsState;

extern near union {
  unsigned char Whole;
  struct {
    unsigned FirstOk:1;
    unsigned RepeatOk:1;
    unsigned LongOk:1;
```

```
    unsigned PressDone:1;
    unsigned TimedOut:1;
    unsigned Long:1;
    unsigned Repeat:1;
    unsigned Buzzer:1;
  };
} TimeBaseFlags;

#define TotalADChannels 4
//
#define CH1OFF -0
#define CH2OFF -0
#define CH3OFF -0
#define CH4OFF -0
//
//
#define LIMIT_OPEN 1010
#define LIMIT_HIGH 960
#define LIMIT_LOW 80
//
#define MSG_PLUGTHERM 127
#define MSG_REQACTION 126
//
#define MAXTEMP_MAX 99
#define MAXTEMP_MIN 45
#define MAXTEMP_STA 80
#define MAXDIFF_MAX 100
#define MAXDIFF_MIN 20
#define MAXDIFF_STA 50
#define MAXDIFFUP_MAX 80
#define MAXDIFFUP_MIN 20
#define MAXDIFFUP_STA 40
//
#define LCD_4BITS   0
#define LCD_8BITS   1
#define LCD_MODE    0
#define UPPER       0
#define LOWER       0
#define PORTNIB     1
//
//// AD Channels
#define ADC_TREF 0          // TREF         at channel 0
#define ADC_SENS1 1         // SENS1        at channel 1
#define ADC_SENS2 2         // SENS2        at channel 2
#define ADC_SENS3 3         // SENS3        at channel 3
#define ADC_SENS4 4         // SENS4        at channel 4

#define BUTTONFIRSTPRESS (unsigned int)5
#define BUTTONREPEATSTART (unsigned int) 250
#define BUTTONLONGPRESS (unsigned int)500
#define BUTTONREPEATRATE  (unsigned int)150
#define MENU_TIMEOUT     (unsigned int)64000
//
//#define BUZZER LATCbits.LATC7
//
#define AVGWIDTH 4  //Must be 1, 2, 4, 8 or 16 -> define number of samples for average
//#define DACOUT
//#define DAC_MODE 1
//
//
//#define FS LATCbits.LATC1
//#define LDAC LATCbits.LATC0
//
#define buffer_length 12
//
//
typedef far unsigned char* RAMBytePtr;
typedef near rom char* ROMBytePtr;
//
typedef near rom void (*ShowFuncPtr)(unsigned char);
typedef near rom unsigned char (*ROMFunc)(void);
```

```
//
typedef union {
  unsigned int D;
  struct {
    unsigned char L;
    unsigned char H;
  };
} DLH;

typedef union {
  unsigned long short int SL;
  unsigned int D;
  struct {
    unsigned char L;
    unsigned char H;
    unsigned char U;
  };
} DLHU;
//

// InlineAsm.h

#ifndef F
#define F 1
#endif

#ifndef W
#define W 0
#endif

//#ifndef A
#define A 0
//#endif

#ifndef B
#define B 1
#endif

#ifndef CARRY
#define CARRY 0
#endif

#ifndef DC
#define DC 1
#endif

#ifndef Z
#define Z 2
#endif

#ifndef OV
#define OV 3
#endif

#ifndef N
#define N 4
#endif

#ifndef FAST
#define FAST 1
#endif

#ifndef NOTFAST
#define NOTFAST 0
#endif

//interface.h
extern rom char InterfaceMsg[16][34];
// FUNCTION Prototypes
void InitMsg(void);
void SendValue(unsigned int dispvalue, unsigned char initpos, unsigned char check);
```

```
void ViewMax(void);
void ViewMenu(void);
void SendMsg(unsigned char msgcode,unsigned char pos, unsigned char clear);
// LCD.h
#include "defines.h"

#define LCD_SET                                 0b00110000
#define LCD_SET_4BITS                           0b00100000
#define LCD_FUNCTION_SET                        0b00111000
#define LCD_FUNCTION_SET_4BITS                  0b00101000
#define LCD_DISPLAY_OFF                         0b00001000
#define LCD_DISPLAY_ON_CURSOR_OFF_BLINK_OFF     0b00001100
#define LCD_DISPLAY_ON_CURSOR_ON_BLINK_OFF      0b00001110
#define LCD_DISPLAY_ON_CURSOR_ON_BLINK_ON       0b00001111
#define LCD_DISPLAY_ON_CURSOR_OFF_BLINK_ON      0b00001101
#define LCD_DISPLAY_CLEAR                       0b00000001
#define LCD_CURSOR_HOME                         0b00000010
#define LCD_ENTRY_MODE_SET                      0b00000110
#define LCD_SET_DDRAM_ADDR                      0b10000000
#define LCD_SET_CGRAM_ADDR                      0b01000000
#define LCD_CURSOR_LEFT                         0b00010000
#define LCD_CURSOR_RIGHT                        0b00010100

#pragma udata
extern far ram unsigned char DECValue[6];

#pragma udata access unbanked

void LCD_PutData(unsigned char);
void LCD_PutInstr(unsigned char);
void LCD_WaitBusy(void);
unsigned char LCD_CheckBusy(void);
void LCD_PutChar(char);
void LCD_PutRAMStr(ram char*);
void LCD_PutROMStr(rom char*);
void LCD_GotoXY(unsigned char);
void LCD_Clear(void);
void BINTOBCD8(unsigned char);
void BinToBCD16(unsigned int value);
void LCD_PutBitMap(void);
void LCD_Initialize(void);
#include "defines.h"

//Modbus.h
extern unsigned char buffer[buffer_length];
extern near union {
  unsigned char Whole;
  struct {
    unsigned Coil_1:1;
    unsigned Coil_2:1;
    unsigned Coil_3:1;
    unsigned Coil_4:1;
    unsigned :4;
  };
} Coil_State;

unsigned char process_rq_1(void);
unsigned char process_rq_2(void);
unsigned char process_rq_3(void);
unsigned char process_rq_4(void);
unsigned char process_rq_5(void);
unsigned char process_rq_6(void);
unsigned char address_error(unsigned char check_type);
unsigned char ReadMBFrame(void);
unsigned char return_error(unsigned char error);
void process_reciv(void);

//RF24G.h
#define RF_24G_DR1      PORTBbits.RB0
#define RF_24G_DATA     PORTBbits.RB5
#define RF_24G_CLK1     PORTAbits.RA5
```

```
#define RF_24G_CE       PORTCbits.RC4
#define RF_24G_CS       PORTCbits.RC5


extern unsigned char opa;
void putByte(unsigned char b);
unsigned char getByte(void);
void RF_24G_Config(void);
void RF_24G_SetTxByte(void);
void putBuffer(char frame_size);
void getBuffer(void);
void RF_24G_SetRxByte(void);

#ifndef SERIAL12_H
#define SERIAL12_H
extern unsigned char pos_ung;
extern unsigned char sending_data;
extern far unsigned char n_send;
extern far unsigned char update;
extern far unsigned char n_reciv;
extern far unsigned char frame_size;
extern far unsigned char module_address;

void conf_serial(void);
void update_data(void);
unsigned char CRC16(unsigned int dataLength,char check);
void send_ip(void);

#endif
```

APÊNDICE B – CÓDIGO FONTE HARDWARE PC

C:\tcc\4620_PC\InlineAsm.h

```
// InlineAsm.h

#ifndef F
#define F 1
#endif

#ifndef W
#define W 0
#endif

//#ifndef A
#define A 0
//#endif

#ifndef B
#define B 1
#endif

#ifndef CARRY
#define CARRY 0
#endif

#ifndef DC
#define DC 1
#endif

#ifndef Z
#define Z 2
#endif

#ifndef OV
#define OV 3
#endif

#ifndef N
#define N 4
#endif

#ifndef FAST
#define FAST 1
#endif

#ifndef NOTFAST
#define NOTFAST 0
#endif
```

C:\tcc\4620_PC\disp.c

```
#include <p18f4620.h>
#include "interrupt.h"
#include "serial12.h"
#include "rf24g.h"

#include <delays.h>
#include <timers.h>

#pragma udata
unsigned char Last_but;
unsigned char show_adc;
//unsigned char total;
#pragma code
void main(void){

unsigned char aux;
unsigned char test;

unsigned int volt;

TRISA=0b00011111;
TRISB=0b11011111;
TRISC=0b11000000;
TRISD=0b00000000;
TRISE=0b00000000;

PORTC=0x00;
PORTD=0x00;
PORTB=0x00;

CCP1CON=0b00000000;
CCP2CON=0b00000000;
CMCON=0b00000111;

RCONbits.IPEN=1;
IPR2 = 0x00;
IPR1 = 0x00;

INTCONbits.INT0IF=0; //interrupt flag 0 button.
INTCONbits.INT0IE=1;
INTCON2bits.INTEDG0=1;

Delay10KTCYx(10);

INTCONbits.GIEH=0;
INTCONbits.GIEL=0;

ADCON1 = 0b00001011;            // AN0-AN4 analog, Vref+=Vdd; Vref-=Vss

Delay10KTCYx(10);

conf_serial();
n_send=0;
n_reciv=0;
n_reciv_w=0;
Delay10KTCYx(175);
Delay10KTCYx(175);
Delay10KTCYx(175);
Delay10KTCYx(175);

TRISBbits.TRISB5=0;
Delay10KTCYx(20);

RF_24G_Config();

Delay10KTCYx(2);

RF_24G_SetTxByte();

Delay10KTCYx(2);
```

```
RCSTAbits.CREN=1; //habilita a recepção

INTCONbits.GIEH = 1;        //enable hi pri ints
INTCONbits.GIEL = 1;        //enable low pri ints
update=0;
update2=0;
n_reciv_w=0;
total=255;

do {

/*          test=RF_24G_DR1;
            if(test==1){
                getBuffer();
                ++n_reciv_w;
                if(n_reciv_w==3){
                    if(buffer[2]>0){
                        total=buffer[2]+5;
                    }
                }
                if(total==n_reciv_w){
                    update2=1;
                }
                if((total==255)&(n_reciv_w>=8)){
                    update2=1;
                }
            }*/
            if(update2==1){
                frame_size=n_reciv_w;
                TXSTAbits.TXEN=1;
                n_reciv_w=0;
                update2=0;
                total=255;
            }
            if(update==1){
                RCSTAbits.CREN=0;
                RF_24G_SetTxByte();
                Delay10KTCYx(2);
                putBuffer(8);
                n_reciv_w=0;
                update2=0;
                total=255;
                RF_24G_SetRxByte();
                RF_24G_CE=1;
                RCSTAbits.CREN=1;
                n_reciv=0;
                update=0;
            }
    } while(1);
}


#pragma config OSC = HSPLL
//#pragma config BORV = 0
#pragma config WDT = OFF
//#pragma config STVR = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CPB = OFF, CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRTB = OFF, WRTC = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTRB = OFF
```

C:\tcc\4620_PC\interrupt.c

```
//interrupt.c
#ifdef __18F4620
#include "p18f4620.h"
#endif


#include "InlineAsm.h"
#include "serial12.h"
#include "rf24g.h"


#pragma udata
unsigned char total;

void InterruptHandlerLow(void);
void InterruptHandlerHigh (void);



#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh (void)
{
  _asm
   goto InterruptHandlerHigh //jump to interrupt routine
  _endasm
}

#pragma code
#pragma interrupt InterruptHandlerHigh

void InterruptHandlerHigh (void)
{
 if(INTCONbits.INT0IF==1){
                getBuffer();
                ++n_reciv_w;
                if(n_reciv_w==3){
                    if(buffer[2]>0){
                        total=buffer[2]+5;
                    }
                }
                if(total==n_reciv_w){
                    update2=1;
                }
                if((total==255)&(n_reciv_w>=8)){
                    update2=1;
                }
        INTCONbits.INT0IF=0;
    }
}

//------------------------------------------------------------------------------
// Low priority interrupt vector and routine
#pragma code InterruptVectorLow = 0x18
void InterruptVectorLow (void)
{
  _asm
    goto InterruptHandlerLow //jump to interrupt routine
  _endasm
}

#pragma code
#pragma interruptlow InterruptHandlerLow //nosave=TABLAT, TBLPTR, TBLPTRU, PCLATH, PCLATU,
section("MATH_DATA")
                                        //save=PROD, section(".tmpdata")
 void InterruptHandlerLow (void)
{

    if (PIR1bits.RCIF) {
        buffer[n_reciv]=RCREG;
        n_reciv+=1;
        if(n_reciv>=8){
```

```
            RCSTAbits.CREN=0;
            update=1;
        }
        PIR1bits.RCIF=0;
    }
    if (PIR1bits.TXIF) {
        if(n_send<frame_size){
        TXREG=buffer[n_send];
        n_send++;
        }
        else{
            while(TXSTAbits.TRMT==0);
            TXSTAbits.TXEN=0;
            n_send=0;
        }
        PIR1bits.TXIF=0;
    }
}

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

C:\tcc\4620_PC\RF-24G.c

```
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include <delays.h>
#include "RF24G.h"
#include "serial12.h"

////////////////////////////////////////////////////////////////////////////

#define numOfBytes      1


////////////////////////////////////////////////////////////////////////////
//Configuration Bytes
//
//Bytes 14-02: Shockburst Configuration
//Bytes 01-00: General Device Configuration

//Byte 14: Length of data payload section RX channel 2 in bits
//    The total number of bits in a ShockBurst RF package may not exceed 256!
//    Maximum length of payload section is hence given by:
//    DATAx_W(bits) = 256 - ADDR_W - CRC
#define DATA2_W             numOfBytes * 8

//Byte 13: Length of data payload section RX channel 1 in bits
#define DATA1_W             numOfBytes * 8

//Byte 12-08: Channel 2 Address
#define ADDR2_4             0x01
#define ADDR2_3             0x02
#define ADDR2_2             0x03
#define ADDR2_1             0x42
#define ADDR2_0             0x41

//Byte 07-03: Channel 1 Address
#define ADDR1_4             0x01
#define ADDR1_3             0x02
#define ADDR1_2             0x03
#define ADDR1_1             0x42
#define ADDR1_0             0x41

//Byte 02
//    Bit 07-02: ADDR_W     - Number of address bits (both RX channels)
//                            Maximum number of address bits is 40 (5 bytes)
//    Bit    01: CRC_L      - 8 or 16 bits CRC
//    Bit    00: CRC_EN     - Enable on-chip CRC generation/checking
//    Combine (via |) together constants from each group
//                          0b76543210
#define ADDR_W_5_BYTE       0b10100000
#define ADDR_W_4_BYTE       0b10000000
#define ADDR_W_3_BYTE       0b01100000
#define ADDR_W_2_BYTE       0b01000000
#define ADDR_W_1_BYTE       0b00100000

#define CRC_L_8_BIT         0b00000000
#define CRC_L_16_BIT        0b00000010

#define CRC_EN_DISABLE      0b00000000
#define CRC_EN_ENABLE       0b00000001

//Byte 01
//    Bit    07: RX2_EN     - Enable two channel receive mode
//    Bit    06: CM         - Communication mode ( Direct or ShockBurst)
//    Bit    05: RFDR_SB    - RF data rate (1Mbps requires 16MHz crystal)
//    Bit 04-02: XO_F       - Crystal frequency (Factory default 16MHz crystal mounted)
//    Bit 01-00: RF_PWR     - RF output power
//    Combine (via |) together constants from each group
//                          0b76543210
#define RX2_EN_DISABLE      0b00000000
```

```
#define RX2_EN_ENABLE       0b10000000

#define CM_DIRECT           0b00000000
#define CM_SHOCKBURST       0b01000000

#define RFDR_SB_250_KBPS    0b00000000
#define RFDR_SB_1_MBPS      0b00100000

#define XO_F_4MHZ           0b00000000
#define XO_F_8MHZ           0b00000100
#define XO_F_12MHZ          0b00001100
#define XO_F_16MHZ          0b00001100
#define XO_F_20MHZ          0b00010000

#define RF_PWR_N20DB        0b00000000  // -20db
#define RF_PWR_N10DB        0b00000001  // -10db
#define RF_PWR_N5DB         0b00000010  // -5db
#define RF_PWR_0DB          0b00000011  // 0db (Full Power)

//Byte 01
//    Bit 07-01: RF_CH#     - Frequency channel (2400MHz + RF_CH# * 1.0MHz)
//    Bit    00: RXEN       - RX or TX operation
//    Combine (via |) together constants from each group
//                          0b76543210
#define RF_CH               0b11100000 // 64 - 2464GHz

#define RXEN_TX             0b00000000
#define RXEN_RX             0b00000001
////////////////////////////////////////////////////////////////////////////////

#define BUF_MAX             numOfBytes
#define PWUPDELAY()         Delay1KTCYx(175);
int buf[BUF_MAX];


void CLKDELAY(void){
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
}
void CSDELAY(void){
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
```

```
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
}

void putByte( unsigned char b ) {  //MSB bit first

   int i;

     unsigned char cnt;
     unsigned char data;
     data=b;
     RF_24G_CLK1=0;
     for(cnt=8;cnt>0;cnt--)
         {
          if((data&0x80)>0)
             {RF_24G_DATA=1;}
          else
             {RF_24G_DATA=0;}
         CLKDELAY();
         RF_24G_CLK1=1;
         CLKDELAY();
         RF_24G_CLK1=0;
         data<<=1;
         }
RF_24G_DATA=0;
```

```
}

unsigned char getByte(void) {  //MSB bit first
   int i = 0;
   unsigned char cnt,data;

     data=0;
     for(cnt=8;cnt>0;cnt--)
        {
         data<<=1;
         if(RF_24G_DATA)
            {data|=0x01;}
         else
            {data&=0xfe;}
        CLKDELAY();
        RF_24G_CLK1=1;
        CLKDELAY();
        RF_24G_CLK1=0;
        }
     return(data);

}

void RF_24G_Config(void) {
   RF_24G_CE=0;
   RF_24G_CS=0;
   RF_24G_CLK1=0;
   RF_24G_DATA=0;
   PWUPDELAY();
   RF_24G_CS=1;
   CSDELAY();

   putByte(DATA2_W);
   putByte(DATA1_W);
   putByte(ADDR2_0);
   putByte(ADDR2_1);
   putByte(ADDR2_2);
   putByte(ADDR2_3);
   putByte(ADDR2_4);
   putByte(ADDR2_0);
   putByte(ADDR2_1);
   putByte(ADDR2_2);
   putByte(ADDR2_3);
   putByte(ADDR2_4);
   putByte(ADDR_W_5_BYTE | CRC_L_16_BIT | CRC_EN_ENABLE);
   putByte(RX2_EN_DISABLE | CM_SHOCKBURST | RFDR_SB_250_KBPS | XO_F_16MHZ | RF_PWR_0DB);
   putByte(RF_CH | RXEN_RX);

   RF_24G_CE=0;
   RF_24G_CS=0;
   RF_24G_CLK1=0;
}



void RF_24G_SetTxByte(void) {
   TRISBbits.TRISB5=0;
   RF_24G_CE=0;
   RF_24G_CS=1;
   CSDELAY();
   putByte(RF_CH | RXEN_TX);
   RF_24G_CS=0;
   RF_24G_CLK1=0;
}
void RF_24G_SetRxByte(void) {
   TRISBbits.TRISB5=0;
   RF_24G_CE=0;
   RF_24G_CS=1;
   CSDELAY();
   putByte(RF_CH | RXEN_RX);
   RF_24G_CS=0;
```

```
    TRISBbits.TRISB5=1;
    RF_24G_CLK1=0;
    RF_24G_CE=1;
}

void putBuffer(char frame_size) {
    int i;

    for( i=0; i<frame_size ; i++) {
         RF_24G_CE=1;
         CSDELAY();
         putByte(ADDR2_0);
         putByte(ADDR2_1);
         putByte(ADDR2_2);
         putByte(ADDR2_3);
         putByte(ADDR2_4);
         putByte(buffer[i]);
         RF_24G_CE=0;
         RF_24G_CLK1=0;
         Delay10KTCYx(2);
    }
}

void getBuffer(void) {

    buffer[n_reciv_w] = getByte();
    RF_24G_CLK1=0;
    RF_24G_CE=1;
}
```

C:\tcc\4620_PC\serial12.c

```
#ifdef __18F4620
#include "p18f4620.h"
#endif

#include "interrupt.h"
#include "defines.h"


#pragma udata
unsigned char n_send;
unsigned char update;
unsigned char update2;
unsigned char n_reciv;
unsigned char n_reciv_w;
unsigned char frame_size;
unsigned char buffer[13];

#pragma code
void conf_serial(void){

    n_reciv=0;
    PIE1bits.RCIE=1;
    PIE1bits.TXIE=1;
    BAUDCON=0b00000000;
    TXSTA=0b00000100;
    RCSTA=0b10010000;
    IPR1=0b00000000;
    SPBRG=51;
    SPBRGH=0x00;
}
```

C:\tcc\4620_PC\interrupt.h

```
//interrupt.h
extern unsigned char total;
```

C:\tcc\4620_PC\RF24G.h

```
#define RF_24G_DR1      PORTBbits.RB0
#define RF_24G_DATA     PORTBbits.RB5
#define RF_24G_CLK1     PORTAbits.RA5
#define RF_24G_CE       PORTCbits.RC4
#define RF_24G_CS       PORTCbits.RC5

extern unsigned char buffer[13];
void putByte(unsigned char b);
unsigned char getByte(void);
void RF_24G_Config(void);
void RF_24G_SetTxByte(void);
void putBuffer(char frame_size);
void getBuffer(void);
void RF_24G_SetRxByte(void);
```

C:\tcc\4620_PC\defines.h

```
////#define SIMUL
////#define FORCED_TIMEOUT
//
//#define DEFAULT_DELAY 1500
//
//#define TIMER0FREQ 247
//#define TIMER0FREQ 31
//#define TIMER2FREQ 200
//
#define TotalADChannels 4
//
#define CH1OFF -0
#define CH2OFF -0
#define CH3OFF -0
#define CH4OFF -0
//
//
#define LIMIT_OPEN 1010
#define LIMIT_HIGH 960
#define LIMIT_LOW 80
//
#define MSG_PLUGTHERM 127
#define MSG_REQACTION 126
//
#define MAXTEMP_MAX 99
#define MAXTEMP_MIN 45
#define MAXTEMP_STA 80
#define MAXDIFF_MAX 100
#define MAXDIFF_MIN 20
#define MAXDIFF_STA 50
#define MAXDIFFUP_MAX 80
#define MAXDIFFUP_MIN 20
#define MAXDIFFUP_STA 40
//
#define LCD_4BITS   0
#define LCD_8BITS   1
#define LCD_MODE    0
#define UPPER       0
#define LOWER       0
#define PORTNIB     1
//
//// AD Channels
#define ADC_TREF 0          // TREF         at channel 0
#define ADC_SENS1 1         // SENS1        at channel 1
#define ADC_SENS2 2         // SENS2        at channel 2
#define ADC_SENS3 3         // SENS3        at channel 3
#define ADC_SENS4 4         // SENS4        at channel 4

#define BUTTONFIRSTPRESS (unsigned int)5
#define BUTTONREPEATSTART (unsigned int) 250
#define BUTTONLONGPRESS (unsigned int)500
#define BUTTONREPEATRATE  (unsigned int)150
#define MENU_TIMEOUT     (unsigned int)64000
//
//#define BUZZER LATCbits.LATC7
//
#define AVGWIDTH 4  //Must be 1, 2, 4, 8 or 16 -> define number of samples for average
//#define DACOUT
//#define DAC_MODE 1
//
//
//#define FS LATCbits.LATC1
//#define LDAC LATCbits.LATC0
//
#define buffer_length 12
//
//
typedef far unsigned char* RAMBytePtr;
typedef near rom char* ROMBytePtr;
//
typedef near rom void (*ShowFuncPtr)(unsigned char);
```

```
typedef near rom unsigned char (*ROMFunc)(void);
//
typedef union {
  unsigned int D;
  struct {
    unsigned char L;
    unsigned char H;
  };
} DLH;

typedef union {
  unsigned long short int SL;
  unsigned int D;
  struct {
    unsigned char L;
    unsigned char H;
    unsigned char U;
  };
} DLHU;
//
```