ALEXANDRE VAZ DE ALMEIDA

IMPLEMENTAÇÃO DE UM SISTEMA DE AUTOMAÇÃO RESIDENCIAL MODULAR SEM FIO: MÓDULO PERIFÉRICO

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Sistemas de Energia e Automação

ORIENTADOR: Prof. Dr. Dennis Brandão

São Carlos

2009

DEDICATÓRIA

Dedico este projeto aos meus pais, Clóvis e Olga, por toda a paciência, educação, atenção e suporte me apoiando em minhas decisões.

Dedico ao meu irmão, Renê, por exemplar conduta como pessoa e pela infinita boa vontade para me ensinar tantas coisas.

Dedico a minha namorada, Ana Eliza, muito fôfa, que sempre foi uma companheira maravilhosa, ouvindo e aconselhando em todos os momentos.

Dedico, em memória, ao Tio Carlos, por todas as coisas que me ensinou, pela imensa sabedoria e bondade, servindo como moral para minha pessoa, no melhor de mim.

"*Para conseguir a amizade de uma pessoa digna é preciso desenvolvermos em nós mesmos as qualidades que naquela admiramos.*" - Sócrates

AGRADECIMENTOS

Agradeço ao Prof. Dr. Dennis Brandão, pela ajuda e confiança em nosso trabalho.

Agradeço também ao Prof. Clóvis de Almeida, meu pai, que em muitos momentos nos ajudou com sua vasta experiência aconselhando e criticando construtivamente, sempre.

Agradeço ao amigo Alex Okajima Watanabe pelo bom humor em criticar e pelas dicas tão relevantes para facilitar meu trabalho.

Agradeço ao amigo Fábio Domingues Caetano, pelas diversas vezes que teve a paciência de nos ouvir e opinar de forma inteligente, simplificando nosso trabalho para darmos um passo de cada vez.

Agradeço ao amigo Luiz Henrique Pessôa Da Costa Trondoli pelas diversas horas em que se empenhou para me ajudar, pelas experiências compartilhadas e exigência de explicações para tudo.

Agradeço ao amigo Sérgio Zauner Gimenes por toda a chatisse sincera ao me ensinar tantas coisas de eletrônica; o que veio a despertar uma paixão pela engenharia elétrica.

RESUMO

Este projeto tem por objetivo o desenvolvimento e a construção de um sistema protótipo microcontrolado capaz de associar entradas e saídas discretas, como sensores e atuadores, utilizando-se de combinações de estado e operações lógicas simples. O sistema é composto de módulos independentes com 4 entradas e 4 saídas, cuja lógica é programada por uma central ligada a um computador. O diferencial deste projeto é a utilização de comunicação sem fio (RF) entre os módulos e a central, eliminando a necessidade de conexão física, agregando assim flexibilidade às aplicações. Outra característica importante diz respeito à autonomia que cada módulo tem para gerenciar suas tarefas, não dependendo da central para coordená-las; isto permite ao usuário programar o módulo e utilizá-lo mesmo onde a central não esteja presente.

Palavras-chave: automação residencial, módulo periférico, comunicação sem fio, microcontrolador, PIC.

ABSTRACT

The purpose of this project is the development and construction of a microcontrolled prototype system capable of associating discrete inputs with outputs, as sensors and actuators, using state combinations and simple logical operations. The system is made of independent modules with 4 inputs and 4 outputs, whose logic is programmable by a central unit which is connected to a computer. The cardinal difference of this project is the use of wireless communication between the modules and the central unit, eliminating thus the need of a hardware connection, providing flexibility to the application. Another important characteristic is the autonomy that each module has to manage its own tasks, not depending on the central unit to coordinate them. This permits the user to program the module and use it even where the central unit is not present.

Keywords: residential automation, wireless communication, microcontroller, PIC.

1. SUMÁRIO

1. LISTA DE FIGURAS

2. LISTA DE ABREVIATURAS E SIGLAS

PCI	Placa de Circuito Impresso
CI	Circuito Integrado
LSN	Less Significant Nibble
MSN	Most Significant Nibble
TCC	Trabalho de Conclusão de Curso
LED	*Light Emitting Diode*
PC	*Personal Computer*
PIC	*Peripheral Interface Controller*

3. INTRODUÇÃO

Seguindo a proposta de o TCC ser um projeto interdisciplinar e, aliado ao gosto pessoal por áreas da engenharia como programação, eletrônica e automação, surgiu a idéia de desenvolver um projeto flexível para atender algumas necessidades residenciais, a partir de um produto simples, de baixo custo e bastante versátil para automatizar ações do cotidiano em uma residência. Os benefícios desta automatização são a comodidade, por ter um sistema ajudando o usuário no seu dia-dia, economia de energia elétrica acionando lâmpadas e outros dispositivos somente quando necessário, e a integração direta com a segurança do ambiente, com sistemas de trava e identificação mais sofisticados. Estas e outras idéias são tratadas com mais detalhes na crescente área denominada domótica.

A domótica teve suas primeiras aplicações no Brasil na década de 80, na construção de prédios com recursos de suporte mais dinâmicos, integrando o controle de iluminação, temperatura, segurança e comunicação num único sistema. A partir daí este conceito difundiu-se com rapidez, apoiada pelas inovações tecnológicas em sensoriamento e telecomunicações.

Novas empresas tomaram iniciativa no Brasil em fornecer soluções para automação residencial e comercial, impulsionadas pela demanda do setor civil, em novos conjuntos habitacionais, residências de luxo e usuários com interesse de melhorar e valorizar seus lares.

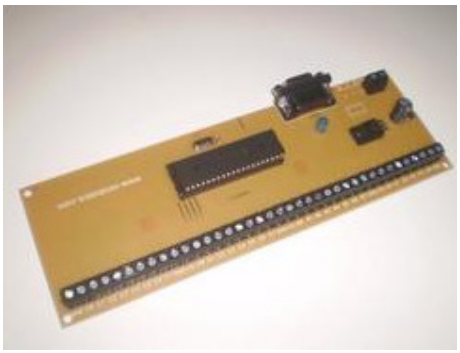
Apesar de ainda ser pouco conhecida e ter um custo ainda bastante elevado para grande parte do público, a domótica, como novo ramo da automação está crescendo e promete vir a ter muitos adeptos.

Este projeto, apesar de possuir foco na automação de residências, pode facilmente, devido a sua flexibilidade, ser implementado em automação comercial e industrial. Esta flexibilidade decorre da possibilidade de se conectar qualquer tipo de sensor e acionar qualquer tipo de saída discreta, a exemplo de uma esteira numa indústria para monitorar a presença de um produto; se o objeto estiver na posição correta para um sensor, a esteira caminha, ou mesmo um atuador descarta o produto. Ou numa casa, onde o usuário ao chegar, um sensor de presença na garagem acende uma luz externa e abre o portão da garagem.

O mercado atual oferece opções à automação que abrangem diversas faixas tanto de complexidade quanto de custo. A intenção deste projeto é oferecer uma alternativa

intermediária, permitindo ao usuário com conhecimento básico de lógica combinacional configurar e instalar os módulos – tudo a um custo acessível.

Alguns exemplos de projetos existentes no mercado podem ser observados nas figuras 1, 2 e 3.



Aplicações: Central de detecção de sensores de contato seco em geral: sensores magnéticos , infravermelho de movimento e coletor de dados de ciclos de produção de parque de máquinas .

Características da placa: Conexão com a porta serial RS232 do PC, retorna o estado de todas as 30 entradas ao mesmo tempo a cada 300ms, alimentação 12 volts, aceita sensores de comum em GND ou 5 volts.

Figura 1 - 30 entradas RS232

Aplicações: Acionamento de cargas dimmerizáveis.

Caracterísicas: 8 canais independentes, pode adicionar até 32 dimmers no mesmo barramento de controle, potência por canal 500w em 110v e ou 1000w em 220 volts, etapa de potência opto isolada, código fonte em Delphi, confirmação de comando do canal e potência por software, dimerização automática ou manual por software, tempo de dimmerização ajustável para cada canal

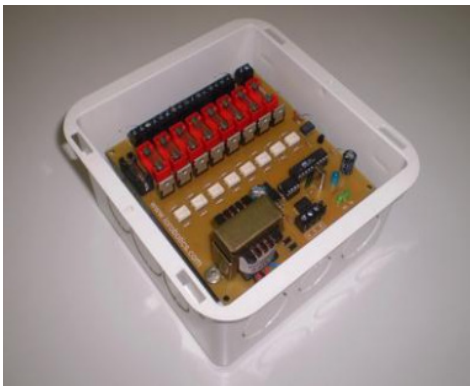


Figura 2 - 8 dimmers digitais para rede Multipoint

Aplicações: Em prédios, residência ou comércio, monitoramento e controle da temperatura ambiente, medição da temperatura de objetos: chocadeiras, câmaras, freezers, motores e estufas.

Características: Conversor digital, não precisa ajustar, resolução 16 bits, range -45 a + 125 graus Celsius, até 32 placas idênticas no mesmo cabo**,** retorno da temperatura e do estado dos relês.

Figura 3 - Termômetro digital

4. OBJETIVOS

Com respeito às motivações expostas anteriormente concentrou-se esforços para realizar o estudo e projeto de um sistema de automação residencial sem fio de baixo custo, facilmente configurável por envolver a utilização de microcontroladores e programação de linguagem de alto nível. Além disso, realizou-se também o projeto do circuito eletrônico utilizado.

Desta forma objetivou-se o desenvolvimento de sistemas simples, onde os resultados obtidos, as habilidades desenvolvidas, e as conseqüentes pesquisas serviram de suporte para a proposta final do projeto.

5. SEQUÊNCIA DAS ATIVIDADES REALIZADAS

O projeto como um todo foi realizado basicamente em nove etapas:

- Estudo das necessidades do usuário – pesquisa com pessoas ligadas a área de instalações elétricas residenciais e opiniões de pessoas próximas;
- Estudo do microcontrolador a ser adotado no módulo central e o módulo periférico considerando o número de pinos necessários, a memória de programa, interfaces de comunicação e custos para cada tipo de módulo
- Estudo da linguagem e ambiente de programação a ser utilizado para programar o *software* de interface com o usuário e o *software* para operação dos módulos;
- Projeto do algoritmo funcional para cada módulo;
- Desenvolvimento do protocolo de comunicação e configuração dos módulos;
- Montagem dos circuitos em protoboard e testes funcionais;
- Rotinas de teste operacional para cada módulo;
- Teste de integração entre o computador e os módulos;
- Aprimoramentos finais.

O projeto foi desenvolvido por um grupo de três alunos de graduação, cada qual responsável por uma parte do projeto, porém, com direta integração entre as partes. Assim, foram criadas três monografias para o mesmo projeto. Cada documento contém explicações detalhadas de uma das três partes e as outras duas partes foram tratadas resumidamente, considerando-as como conteúdo a ser detalhado nas outras monografias.

6. DESCRIÇÃO DO PROJETO

6.1. Programa de Interface com o Usuário

Programa desenvolvido em *Visual Basic* com o objetivo de realizar a interface com o usuário permitindo ao mesmo a seleção de operações desejadas e a gravação das configurações nos módulois periféricos.

6.2. Módulo Central

O papel do módulo central é traduzir para os módulos periféricos os comandos passados pelo computador.

Para isso, possui uma interface RS232 para comunicação com o computador e uma interface RF para comunicação com os módulos periféricos.

6.3. Módulo Periférico

Módulo responsável por realizar toda a automatização do ambiente a partir das configurações recebidas do programa utilizado pelo usuário. As configurações são diferenciadas em tarefas, cada uma relacionada a uma carga a ser controlada.

Os sinais lógicos dos sensores são interpretados pelo módulo, e efetuando as operações lógicas já programadas, aciona ou desaciona a carga correspondente a cada tarefa.

7. PROGRAMA DE INTERFACE COM O USUÁRIO

7.1. Considerações iniciais

Nesta seção serão exibidas algumas definições e conceitos teóricos utilizados no desenvolvimento do projeto, além de justificativas para os procedimentos e opções definidas.

7.2. Linguagem de programação adotada

O Visual Basic é uma linguagem de programação desenvolvida pela Microsoft, e é parte integrante do pacote Microsoft Visual Studio. Sua versão mais recente é parte do pacote Visual Studio .NET, voltada para aplicações .Net. Sua versão anterior era parte do Microsoft Visual Studio 6.0, ainda muito utilizado atualmente.

Como um aperfeiçoamento do BASIC, a linguagem é dirigida por eventos (*event driven*), e possui também um ambiente de desenvolvimento integrado (*IDE - Integrated Development Environment*) totalmente gráfico, facilitando enormemente a construção da interface das aplicações GUI (*Graphical User Interface*), daí o nome "*Visual*". Em suas primeiras versões, o Visual Basic não permitia acesso a bancos de dados sendo, portanto, aplicações bem simples. Mas devido ao sucesso entre as empresas, que faziam uso de componentes adicionais fabricados por terceiros para acesso a dados, a linguagem logo adotou tal recurso, permitindo fácil acesso a bases de dados.

A escolha da linguagem de programação para realizar a interface com o usuário foi baseada na característica mais pronunciada do Visual Basic que é justamente a orientação a objeto. Desta forma, proporciona uma aparência sofisticada e ao mesmo tempo uma interface fácil, intuitiva e amigável ao usuário. Assim, foi possível dispor ao usuário uma maneira muito prática para configurar os módulos periféricos.

7.3. *Software* de Interface com o usuário

A interface com o usuário foi desenvolvida objetivando a simplicidade de operação, possibilitando a um usuário doméstico com conhecimento mínimo de

lógica combinacional realizar a configuração dos módulos da melhor forma para seu uso.

A Figura 4 exibe a janela de configuração que se abre ao iniciar o software.

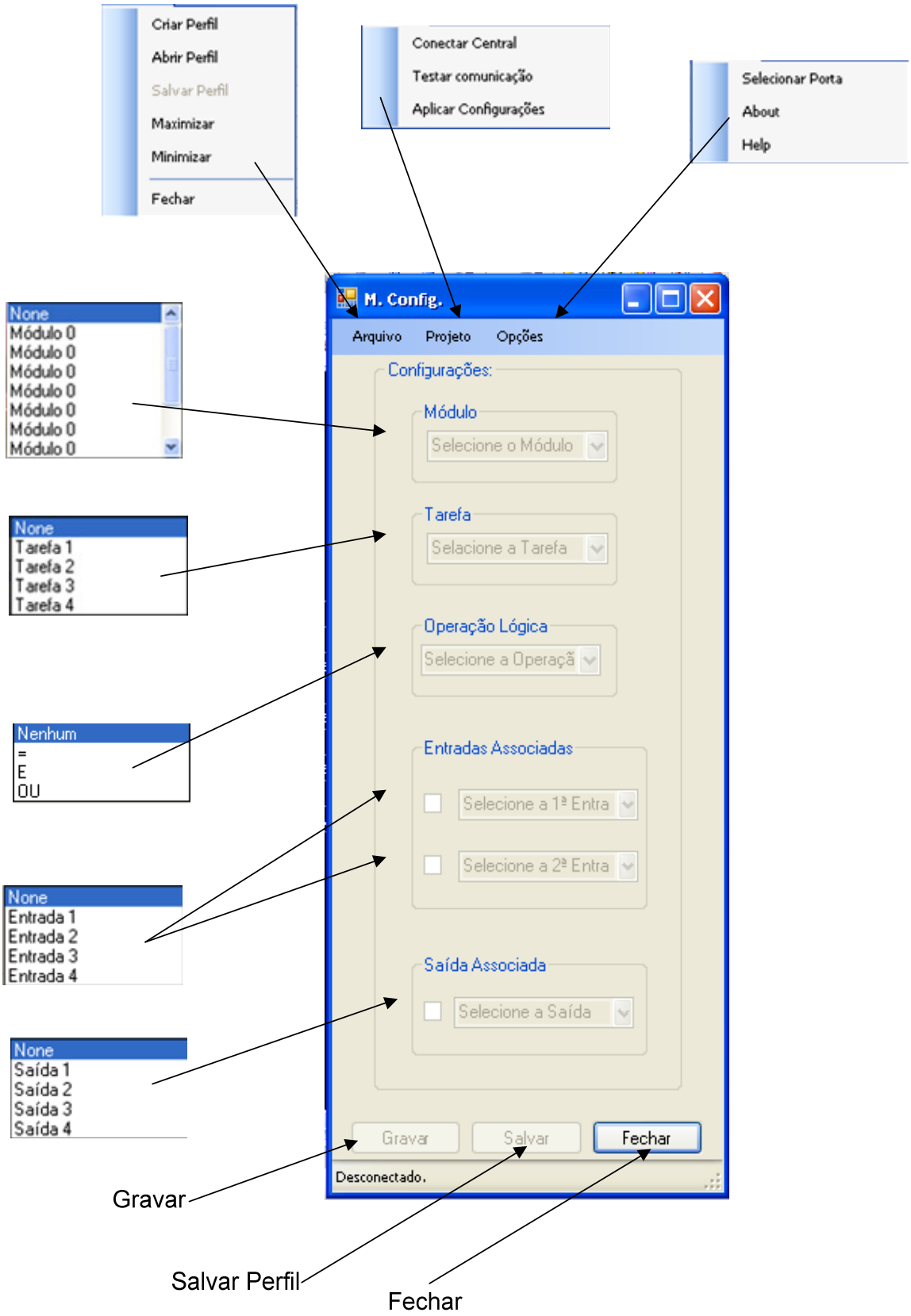


Figura 4 - Janela de configuração

O software configura módulos com quatro entradas e quatro saídas digitais. Estas entradas e saídas podem ser combinadas de forma conveniente para realizar acionamentos em geral, a partir dos sinais de entrada conectados. Cada associação lógica foi chamada de tarefa, e a associação de quatro tarefas foi chamada de perfil, assim, após configurar quatro tarefas, o usuário salva um perfil que pode ser gravado em qualquer módulo detectado.

8. COMUNICAÇÃO COM O MÓDULO CENTRAL

O meio de comunicação entre um computador com o software de configuração e cada módulo isolado é o módulo central. Este se comunica de modo serial com o computador e por rádio freqüência com cada elemento periférico. A Figura 5 ilustra a idéia de comunicação do sistema.

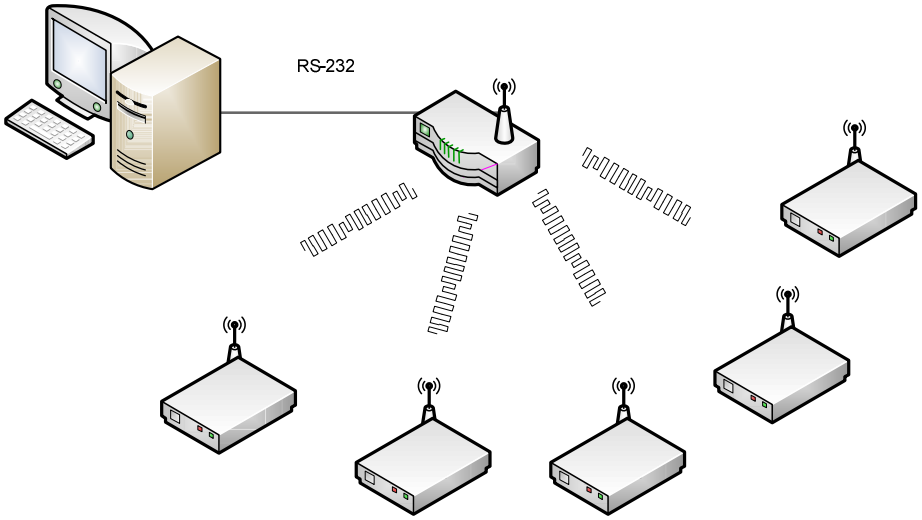


Figura 5 - Diagrama de fluxo de dados

Há basicamente dois tipos de troca de dados entre central e computador. No primeiro ocorre a contagem dos módulos ativos e a identificação dos endereços. O segundo tipo executa a programação propriamente dita com a troca dos dados de configuração.

Como descrito anteriormente é necessário conectar-se com a central antes de executar a gravação dos dados. Isto se deve ao carregamento das informações dos módulos ativos, a qual ocorre da seguinte maneira:

- Envio do caractere de abertura de comunicação (‘I’);
- Recebimento do caractere de confirmação (‘I’);
- Envio do endereço do primeiro módulo (‘1’);
- Recebimento do caractere de confirmação (‘S’ para módulo presente, e ‘N’ para módulo ausente);
- Envio do endereço do próximo módulo (2 até 10);

- Assim sucessivamente, “perguntando” à central se o endereço está ativo ou não e gravando os endereços ativos no banco de variáveis.

Nota-se que a comunicação é baseada em uma “pergunta” e uma “resposta”, ou seja, a comunicação sempre verifica o estado do canal em ambos os sentidos. Para evitar que um ou outro dispositivo “espere” uma reposta indefinidamente, foram estabelecidos temporizadores *timeout*, que a cada envio de dado espera um tempo máximo pré-determinado, encerrando a comunicação e exibindo uma tela de erro quando este tempo é excedido.

Para cada endereço ativo é incluído na caixa de seleção de módulo o número dos respectivos endereços dos módulos encontrados. Assim, ao término da primeira troca de dados, é possível escolher o módulo a ser configurado a partir de seu endereço. O endereço a ser conferido depende somente do software de configuração, dentro dos 256 endereços possíveis de um byte. O processo do ponto de vista do computador, de forma geral, pode ser entendido segundo o fluxograma da Figura 6.

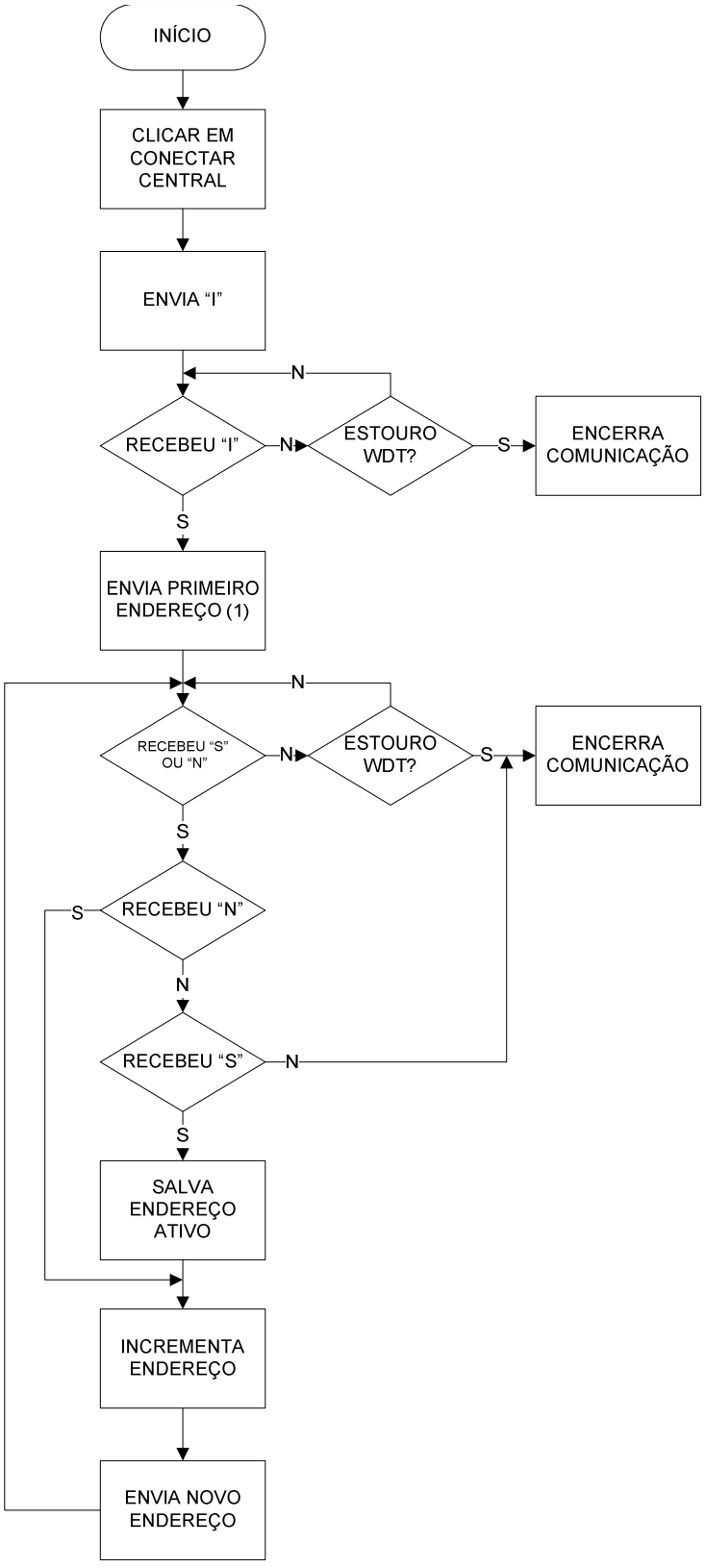


Figura 6 - Fluxograma de conexão com a central

Seguindo o fluxograma da Figura 6 nota-se que a central pode ser conectada sucessivas vezes, o que é interessante do ponto de vista da aplicação prática, uma vez que torna possível a verificação de todos os módulos conectados e a imediata detecção de um possível módulo fora do alcance da central.

O outro processo de comunicação com a central ocorre quando há a gravação dos dados nos módulos periféricos. A comunicação se dá baseada no envio e recebimento da mesma informação. Por exemplo: um byte de endereço é enviado à central, esta retransmite este byte como forma de confirmar o recebimento da informação. O computador verifica se a informação retransmitida é válida, realizando assim, uma verificação dos dados. Tal transferência de dados ocorre da seguinte forma:

- Envio do byte de início de comunicação (“C”);
- Recebimento do byte de confirmação (“C”);
- Envio do endereço do módulo a ser gravado (byte de endereço);
- Recebimento do byte de endereço;
- Envio do byte de operação;
- Recebimento do byte de operação;
- Envio do byte de I/O utilizado
- Recebimento do byte de I/O;
- Envio do byte de sinais invertidos;
- Recebimento do byte de sinais negados;
- O envio dos bytes de configuração ocorre quatro vezes completando a tarefa programada. A qualquer momento “A” pode ser recebido da central, indicando uma falha na comunicação entre a central e o módulo, provocando uma janela de alerta advertindo ao usuário que uma nova tentativa de gravação é necessária.

Um melhor entendimento da troca de dados pode ser obtida acompanhando-se o fluxograma da Figura 7, o qual compõe as informações trocadas do ponto de vista do computador.

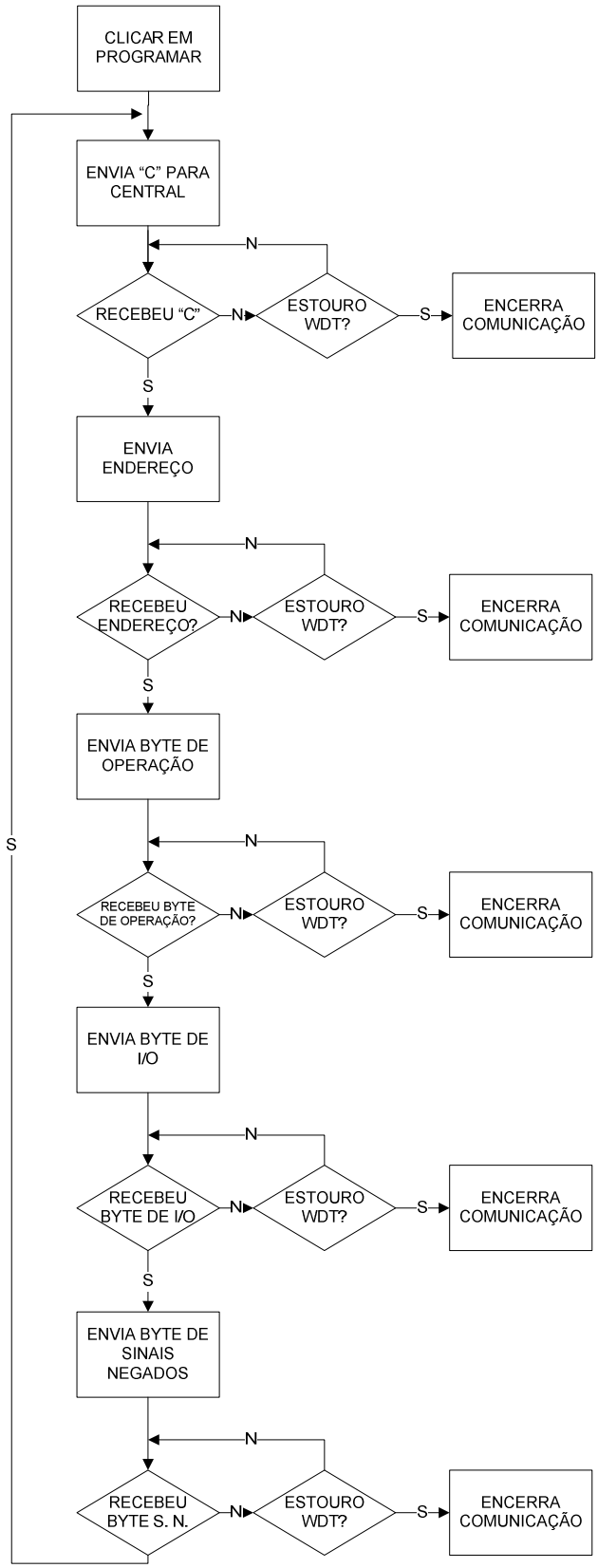


Figura 7 - Fluxograma de configuração

9. O MÓDULO CENTRAL

O módulo central, ou simplesmente chamado de central, é um circuito digital composto de um microcontrolador PIC16F877A da Microchip, um *encoder* e um *decoder* Holtek, modelos HT12E e HT12D, um MAX232 e um par de rádios emissor-receptor RT4 e RR3 da Telecontrolli, além de componentes auxiliares, tais como capacitores, resistores, conectores, fonte de tensão, chaves e cristal.

O papel da central é fazer o interfaceamento entre o computador em que está instalado o *software* M. Config e todos os módulos periféricos adquiridos pelo usuário. Este interfaceamento é feito respeitando um protocolo de comunicação muito bem definido, o que confere a este sistema confiabilidade e facilidade de detecção de erros causados por possíveis interferências externas.

A central está programada para receber pacotes de dados (*bytes*) do computador via comunicação serial e enviá-los aos módulos periféricos através de uma interface sem fio composta pelo par de rádios citados. Esta interface possibilita à central não só enviar dados, mas também recebê-los, completando a comunicação entre ela e os periféricos.

Além de traduzir para os módulos periféricos os comandos passados pelo computador, a central foi criada com o intúito de tornar o projeto mais versátil. Uma vez adquiridos e instalados os módulos periféricos, o usuário não precisará desconectá-los para que seja feita a programação, pois a central se comunica via rádio-freqüência. Isto agrega outra característica interessante à central: por não precisarem ser conectados fisicamente a ela, é possível reprogramar todos os módulos periféricos de uma só vez com um simples comando “Gravar” no *software*. Quando isso acontece, a central recebe os comandos do computador e repassa-os, um a um, para cada módulo periférico. Tal procedimento é repetido até que todos os módulos estejam gravados.

Para aumentar a segurança no envio de dados via rádio-freqüência a central usa um protocolo de verificação de palavra de dados que consiste em enviar o *byte*, recebê-lo e verificar se o *byte* recebido condiz com o enviado, deste modo ela pode tomar atitudes como reenviar o *byte* ou alertar erro de comunicação.

Devido ao fato de cada palavra de dado ter 8 bits e o par *encoder*/*decoder* trabalhar com um *nibble*, a central e os periféricos comunicam-se *nibble* a *nibble*. Para isso, a central divide o *byte* recebido pelo computador em dois *nibbles* e os envia separadamente para os periféricos. Cada periférico recompõe os *nibbles* como *byte* para poder “compreender” a instrução.

A central foi programada de modo a ser versátil, ela pode tanto dizer ao computador quantos módulos estão presentes e quais são estes módulos, como também programá-los.

Quando inicializada, a central espera por um comando do computador, que pode ser um “I” ou um “C”. Esta comunicação é feita pela porta serial do microcontrolador a uma taxa de 9600 *baud*. O “I” recebido diz para a central que o sistema está em modo de inicialização, portanto ela terá de informar ao computador quais dos módulos estão presentes. Logo após esta instrução, ela reenvia o “I” para o computador para que este faça a verificação de erro e saiba que pode continuar a comunicação. Neste momento a central espera receber o endereço do módulo que o computador deseja saber se está presente. O endereço virá no *nibble* menos significativo do *byte* recebido. Após o recebimento deste endereço, a central utiliza o *encoder* conectado ao rádio para endereçar o módulo em questão e envia a palavra “1 1 1 1” (Traduzida como: “Você está aí?”) ao mesmo. Esta palavra faz parte do protocolo de comunicação e não pode ser confundida com nenhum comando passado ao módulo periférico. Cabe, agora, frisar uma característica fundamental do projeto da central: Todo dado (*nibble*) recebido de um periférico pela central é recebido via interrupção do pino RB0, que está ligado ao VT (*Valid Transmission*) do decodificador. Deste modo, após perguntar se o módulo está presente, a central conta um tempo de 200ms, que funciona como *time out*, se o módulo responder “1 1 1 1” dentro deste intervalo o programa entra na interrupção, grava o valor para saber que o módulo está presente e retorna à contagem de tempo. Terminado o tempo de resposta a central verifica se o módulo está ou não presente, caso esteja, a central envia a letra “S” para o computador, caso contrário será enviada a letra “N”. Neste ponto o computador já sabe se o módulo está ou não presente e a central volta para esperar um novo byte de endereço, para verificar se o próximo módulo está ou não presente através do mesmo procedimento. Para esta versão do projeto ficou definido que o número máximo de módulos a serem conectados seria dez, portanto o computador enviará dez endereços de módulos para a central, que repetirá a verificação dez vezes, uma para cada módulo e retornará para esperar por um “C” ou um “I” novamente, caso seja requisitado pelo computador.

De volta ao início do programa, a central estará à espera da letra “C” ou “I”, e neste caso ela receberá a letra “C” (Configuração), pois já enviou para o computador quais módulos estão presentes e agora eles deverão ser configurados.

Uma vez recebida a letra “C” o programa a reenvia para o computador para que este faça a verificação de erro e saiba que pode continuar a comunicação. Neste momento a central espera pelo endereço do primeiro módulo a ser programado (quem escolhe isso é o computador, o que confere à central a capacidade de obedecer prioridades de programação

dos módulos). Recebido este *byte*, a central coloca o endereço do módulo no *encoder* e o reenvia através da serial para o computador, para que este faça novamente a verificação de erro e saiba que pode continuar a comunicação.

Neste ponto a central está esperando pelo primeiro *byte* de configuração que será enviado ao periférico. Recebido este *byte* ela o divide em dois *nibbles*. O primeiro *nibble* (o menos significativo) é então enviado para o módulo em questão e aguarda-se a resposta. Recebida a resposta, a central guarda o *nibble* recebido, que deve ser igual ao enviado e manda o *nibble* mais significativo, pois sabe, pela resposta do periférico, que este está pronto para receber mais um dado. O mesmo procedimento de resposta ocorre e a central monta um *byte* com os dois *nibbles* recebidos e os compara com os dois enviados. Esta é a verificação de erro de envio que confere mais segurança à comunicação e permite a tomada de decisões. Nesta versão do projeto foi implementada a verificação para que o usuário possa saber se ocorreu algum tipo de erro na comunicação, para isso, caso o *byte* recebido seja diferente do *byte* enviado, a central envia a letra “A” para o computador, para que este saiba do ocorrido e entra em *loop*, colocando em nível alto e baixo uma saída, na qual pode ser ligada a qualquer tipo de alerta sonoro ou visual. Para este projeto escolhemos o alerta visual via *LED*.

Caso não tenha ocorrido falha na comunicação, a central esperará pelo próximo *byte* de configuração do periférico. Isso será feito três vezes completando o número de *bytes* necessários para se configurar uma tarefa em um módulo, como será explicado mais adiante. Após isto, a central volta ao início do programa para esperar mais uma vez por um “C” ou “I”.

Este procedimento reforça a característica de a central ser um módulo versátil, pois não restringe ao usuário ou às novas versões de programa um número determinado de tarefas que cada módulo poderá receber, facilitando a melhoria do projeto e abrindo margem a outras aplicações.

10. O MÓDULO PERIFÉRICO AUTÔNOMO

Esta etapa do projeto conta com a descrição mais detalhada do componente responsável pela execução das tarefas: o módulo periférico. É constituído fisicamente por um sistema lógico de comando, um sistema de comunicação via rádio e um circuito de potência.

O sistema lógico de comando foi implementado com o microcontrolador da Microchip, o PIC16F873A, um CI de baixo custo, de fácil programação e suficientemente robusto. A linguagem de programação utilizada foi “C”, e os programas foram desenvolvidos no compilador MikroC PRO 2009 (v. 1.65). Trata-se de um excelente *software* que converte e compila programas escritos em linguagem C para linguagem *assembly*, gerando os arquivos necessários para gravação no PIC. Vale destacar ainda das facilidades deste *software* algumas funções muito úteis para programadores que estejam migrando de compiladores de linguagem *assembly* para compiladores C, tais como funções prontas de temporização, comunicação serial, acesso à memória, controle de *LCD*, etc. Também oferece ferramentas úteis de diagnóstico para “depurar” o código.

As características do microcontrolador utilizado são apresentados na Figura 8 e a configuração dos pinos na Figura 9. A descrição de cada pino pode ser encontrada no datasheet do fabricante (vide referências bibliográficas).

Especificações gerais	PIC16F873A-I-SP
Número de pinos	28
Corrente de operação (mA)	7
Tensão de operação (V)	4 - 5,5
Freqüência de Operação (MHz)	DC – 20
Memória Flash de Programa (palavras de 14-bits)	4K
Memória de dados - RAM (bytes)	192
Memória de dados EEPROM (bytes)	128
Interrupções	14
Portas de Entrada/Saída	Portas A, B, C
Timers	3
Módulos PWM	2
Comunicação Serial	MSSP, USART
Comunicação Paralela	—
Módulos de conversores AD (10-bits)	5 canais de entrada
Comparadores Analógicos	2
Instruções de programação	35 Instruções

Figura 8 - Características do PIC16F873A

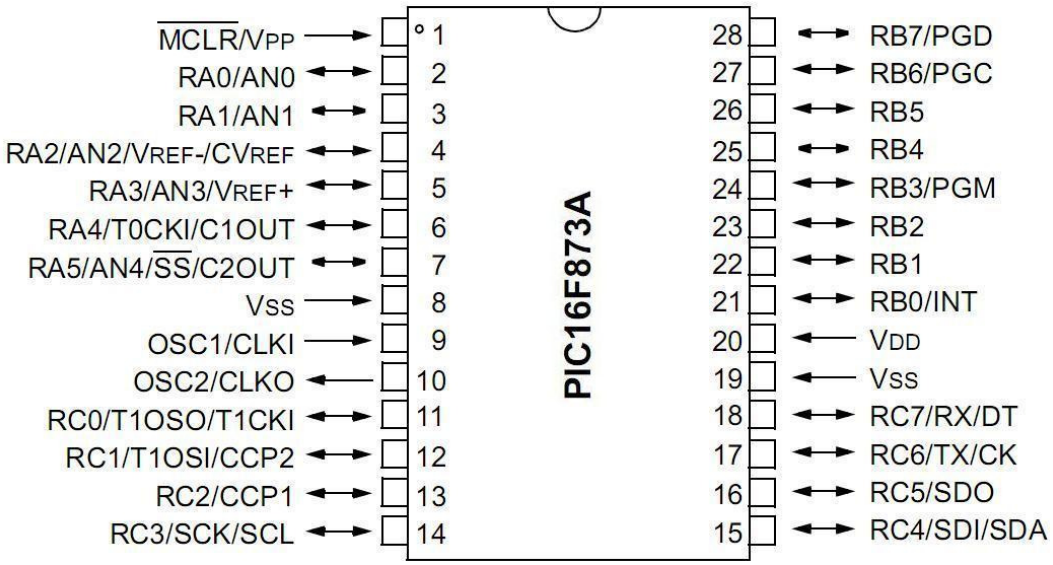


Figura 9 - Pinos do PIC16F873A

O módulo pode executar até quatro tarefas, seqüencialmente e sem prioridade entre estas. Ainda que não tenha a simultaneidade de tarefas, o processo todo ocorre de forma muito rápida, sendo que o intervalo de tempo entre uma entrada relevante e a resposta

esperada não é perceptível pelo usuário. A princípio não foram tratadas as situações onde o tempo de resposta seja realmente crítico, mas é possível mudar alguns parâmetros de *hardware* para adequar o sistema para uma resposta ainda mais rápida.

As tarefas recebidas pelo módulo periférico são armazenadas na memória EEPROM interna do microcontrolador PIC16F873A, o que previne a perda das configurações na ocorrência de uma interrupção no fornecimento de energia elétrica.

Na Figura 10 temos o fluxograma operacional do módulo periférico.

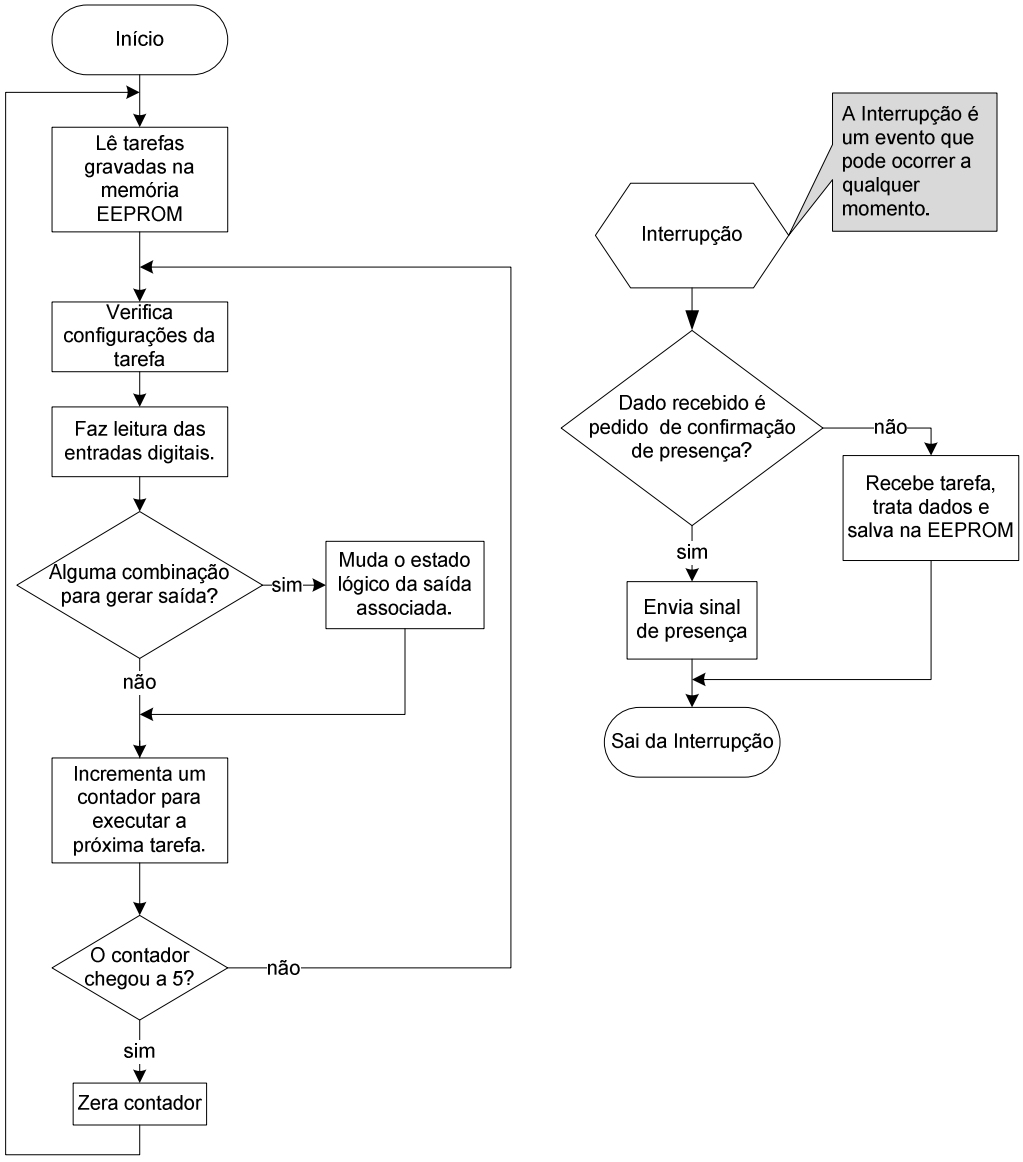


Figura 10 - Fluxograma ilustrativo do funcionamento do módulo periférico

10.1. Endereçamento e comunicação

O sistema de comunicação entre os módulos periféricos e o módulo central é feito via rádio utilizando pequenos módulos RF pré-fabricados (marca *Telecontrolli*) de somente uma portadora cada, na freqüência de 433,92 MHz e com alcance máximo de 60 metros (sem obstáculos). Para esta freqüência de portadora foi calculado o comprimento da antena para se obter um bom alcance. Para um rádio que trabalha na freqüência de 433,92MHz, pode ser usado um fio de cobre rígido como antena, de comprimento igual a 17,5 cm.

Um par de rádios composto por receptor e emissor foi instalado em cada módulo (central e periféricos), possibilitando a comunicação bidirecional. Apesar disso, a comunicação é feita no modo *half-duplex*, pois as transmissões e recepções são feitas em rotinas diferentes pelo microcontrolador (não simultâneas). Por conseqüência isto também torna a verificação de dados mais fácil. Um diagrama básico de conexão dos módulos de rádio podem ser observados nas Figuras 11 e 12, assim fotos dos modelos do fabricante Telecontrolli (Figuras13 e 14)

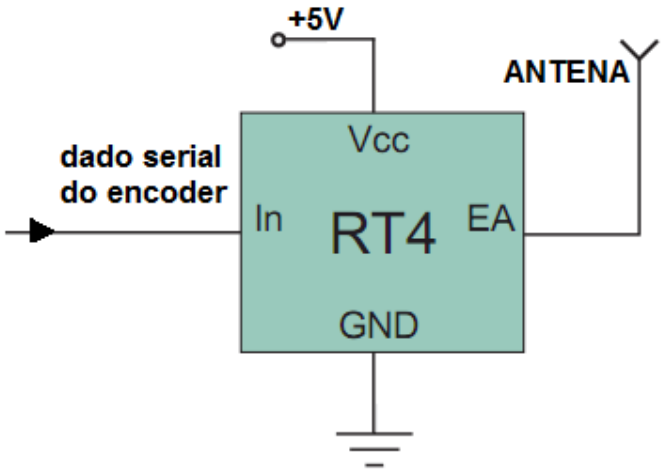


Figura 11 - Esquema elétrico simples do transmissor RF

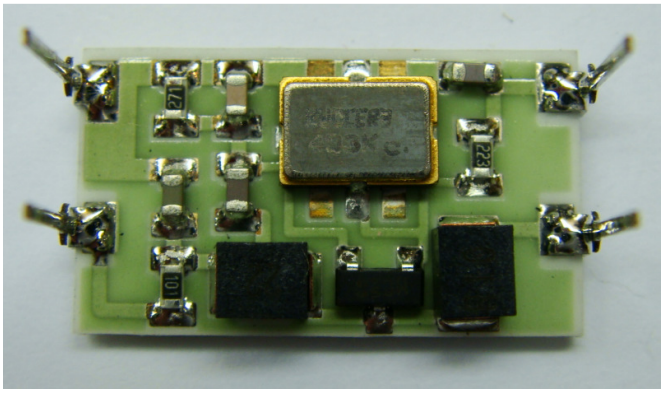


Figura 12 - Módulo RF emissor Telecontrolli – 433,92MHz

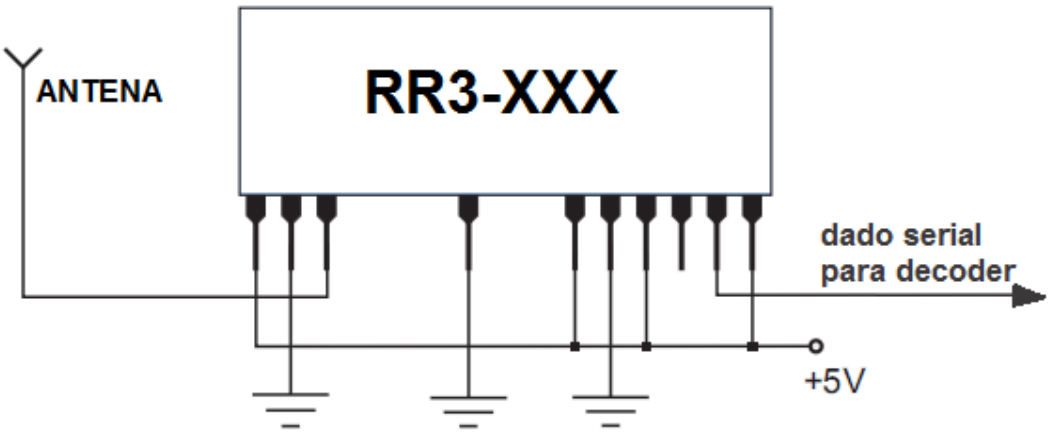


Figura 13 - Esquema elétrico simples do receptor RF

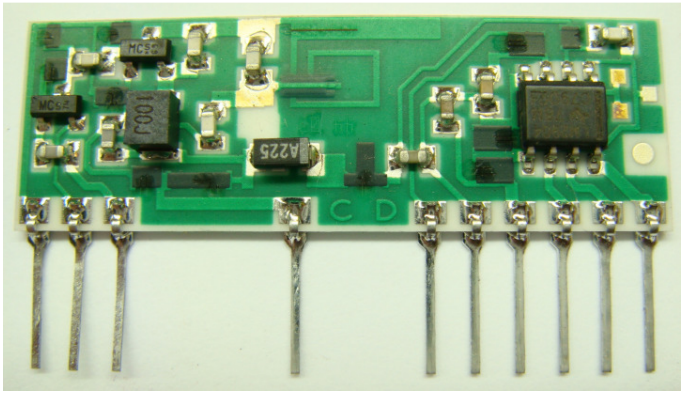


Figura 14 - Módulo RF receptor Telecontrolli – 433,92MHz

O endereço de cada módulo é uma combinação binária de quatro bits definida via *hardware* a partir de mini-chaves (DIP Switch) soldadas na placa de circuito impresso do módulo. O dispositivo responsável pelo endereço do módulo em si é um *decoder*, Holtek – modelo HT12D, responsável por repassar as informações (o *nibble* menos significativo) para o microcontrolador somente quando o endereço recebido estiver correto. Ou seja, durante a programação todos os módulos receberão informações via rádio, mas somente será programado o módulo com endereço equivalente ao atribuído pelo módulo central. O pacote de informações enviado pela central é um *byte* composto por quatro bits de endereço e quatro bits de dados (Figura 15).

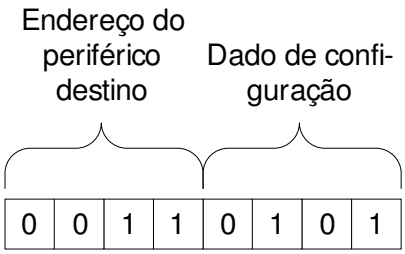
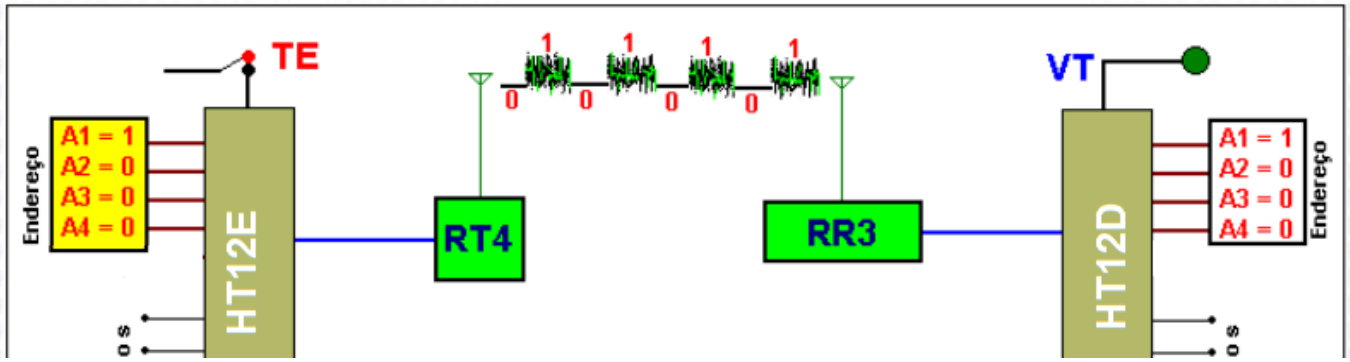


Figura 15 - Byte contendo endereço e dado

O módulo central também possui um *decoder* similar ao dos periféricos com um endereço fixo, o qual cada módulo periférico está configurado para responder confirmações de verificação de recebimento.

Para o envio de dados foi utilizado o dual do *decoder* citado, o *encoder* Holtek – modelo HT12E, com bits de endereçamento atribuídos via *software* no módulo central, vista a necessidade de se alterar as combinações de endereço para se comunicar com cada módulo periférico. Nos módulos periféricos este *encoder* possui um endereço fixo (via *hardware*), assim como é o da central, pois somente podem estabelecer troca de dados com o módulo central.

Na Figura 16 é ilustrada a comunicação e endereçamento feitos pelos módulos de rádio e *encoder*/*decoder*.



Os *encoder* e *decoder* implementados trazem várias outras vantagens para o sistema, como filtragem de ruído e interferências recebidos pelo rádio, a verificação de dados feita pelos próprios *encoders* – cada *encoder* envia 3 vezes o mesmo dado e o *decoder* deve receber 3 vezes este mesmo para então repassar a informação ao microcontrolador. Outra vantagem é a impossibilidade do recebimento de uma informação enviada pelo próprio módulo, seja este central ou periférico, pois há um endereço distinto para cada unidade. Na Figuras 17, 18 e 19 seguem a identificação de cada pino e os respectivos fluxogramas funcionais.

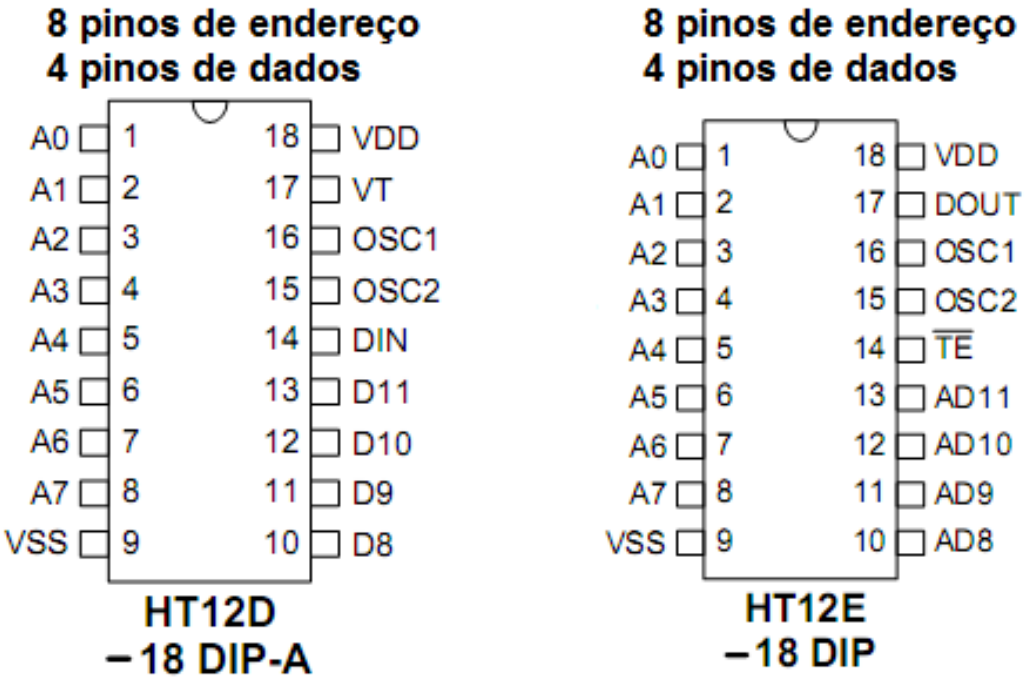


Figura 17 - Pinagem dos encoder e decoder utilizados

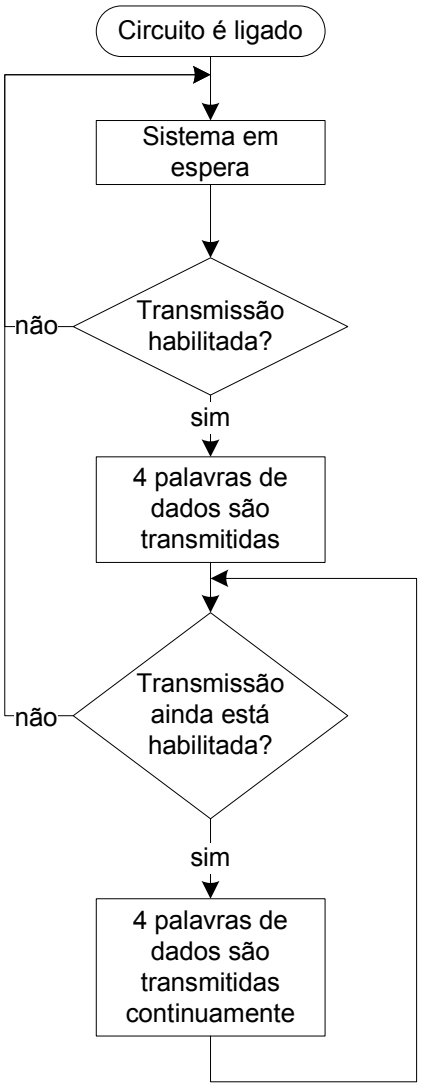


Figura 18 - Diagrama funcional do encoder HT12E

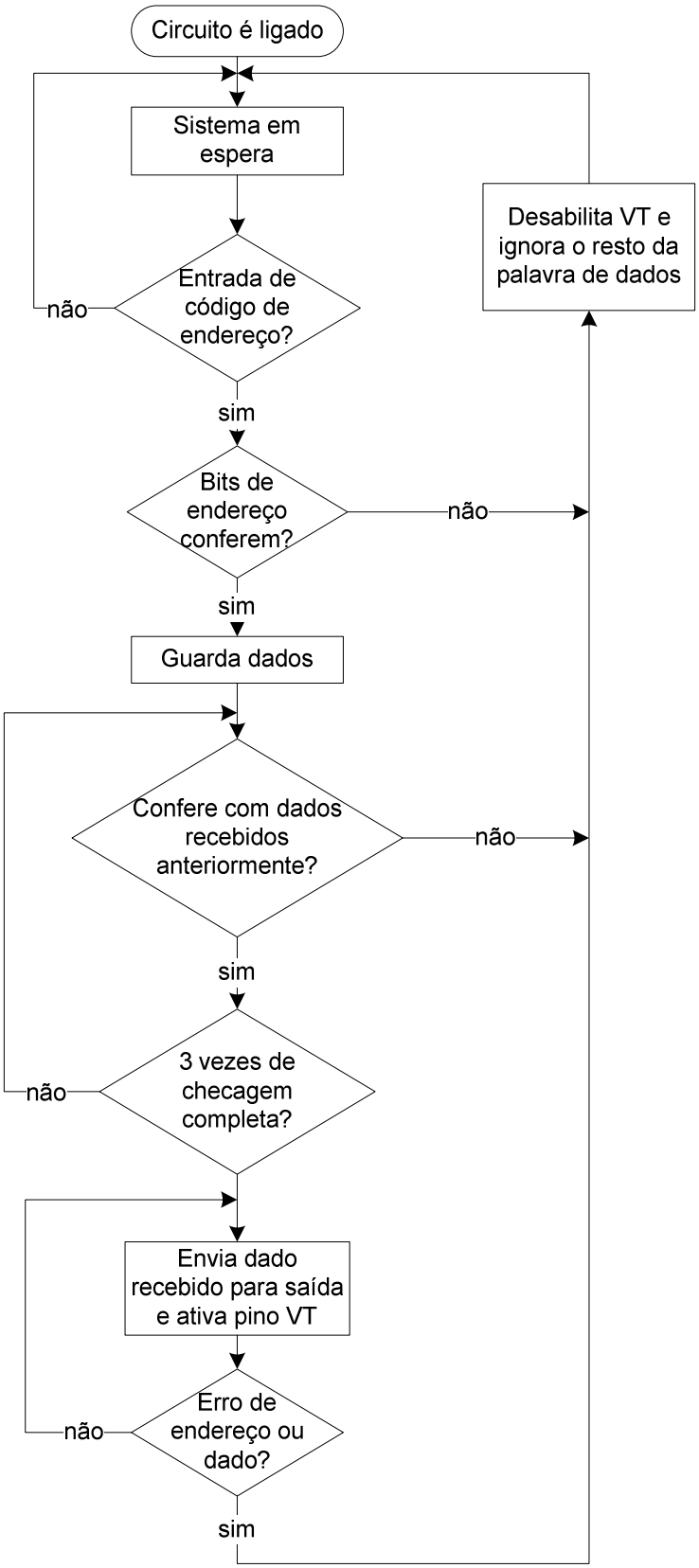


Figura 19 - Fluxograma funcional do decoder HT12D

10.2. O procedimento de configuração de um módulo

Um módulo pode ser configurado a qualquer momento, bastando apenas a central emitir o código correto para o endereço do módulo a ser configurado. Este executará as rotinas necessárias para a recepção das configurações.

Logo que o microcontrolador do módulo periférico recebe informações da central, a informação é analisada e classificada ou como um comando de verificação de presença do módulo ou uma nova configuração de tarefa a ser gravada na memória EEPROM de dados.

Para o caso de ser um comando de verificação de presença do módulo é enviada uma resposta (código) simples para a central como confirmação. Já no caso de se tratar de informações de configuração então o microcontrolador segue para rotinas da sua interrupção onde tratará os *nibbles* e salvará os dados adequadamente na memória.

Após a configuração ser completada o módulo inicia a execução das tarefas imediatamente.

Na Figura 20 temos o esquema ilustrativo do processo de configuração de um módulo.

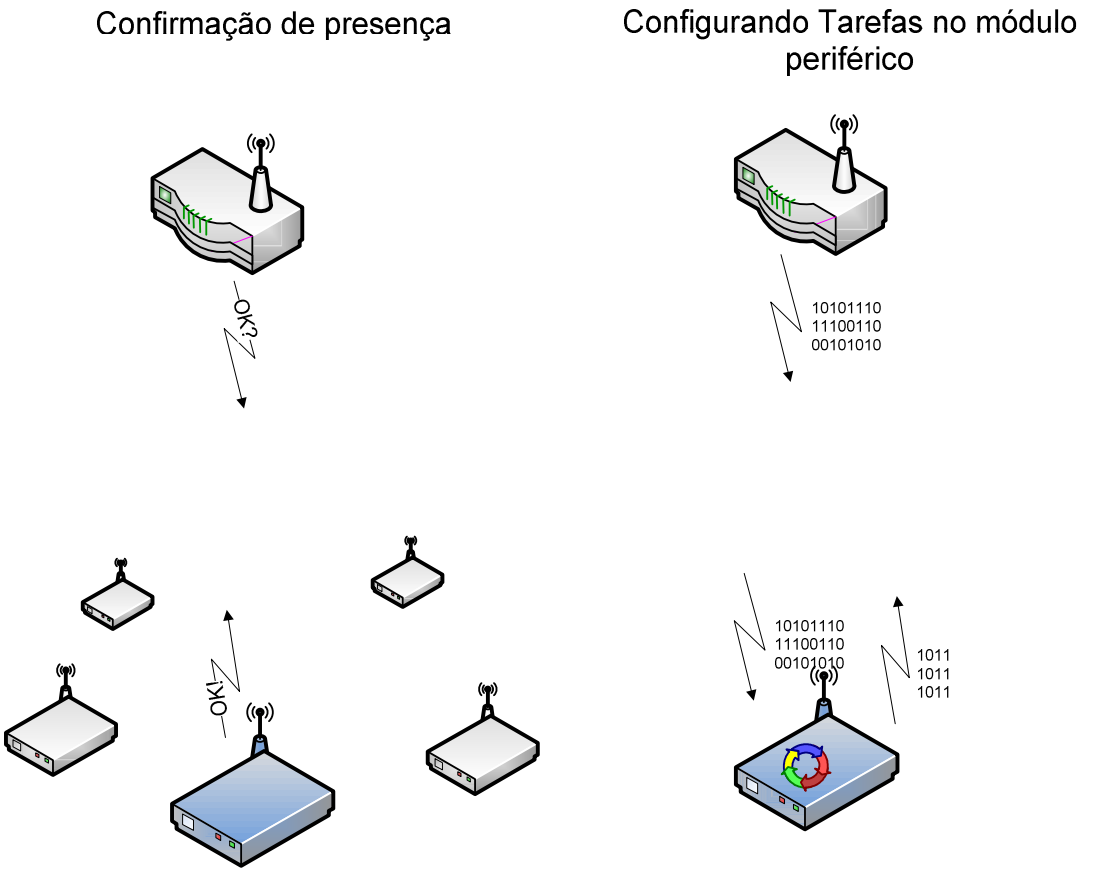


Figura 20 - Comunicação entre módulos central e periférico

10.3. Como as saídas e entradas lógicas funcionam

Após o módulo salvar as configurações na memória EEPROM o sistema começa a executar as tarefas salvas.

Cada tarefa é composta por três *bytes*: *byte* de seleção, *byte* de operação e *byte* de estados.

O byte de seleção indica as entradas e saídas que foram selecionadas para a execução da tarefa. Como cada módulo possui quatro entradas e quatro saídas, então os quatro bits mais significativos do byte de seleção indicam as entradas escolhidas (valor “1”) ou não (valor “0”) e os outros quatro bits são destinados para uma das saídas a ser utilizada (bit assume valor “1”).

Na Figura 21 temos um exemplo em que o usuário escolheu utilizar as entradas “dois” e “quatro” e a saída “dois” do módulo.

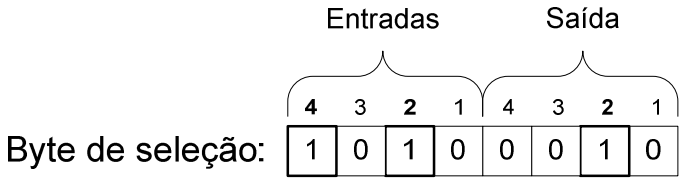


Figura 21 - Byte de seleção das entradas e saídas a serem utilizadas

O *byte* de operação utiliza apenas o *nibble* mais significativo e pode definir três tipos de operação, cada qual formada por uma seqüência de *bits* (Figura 22).

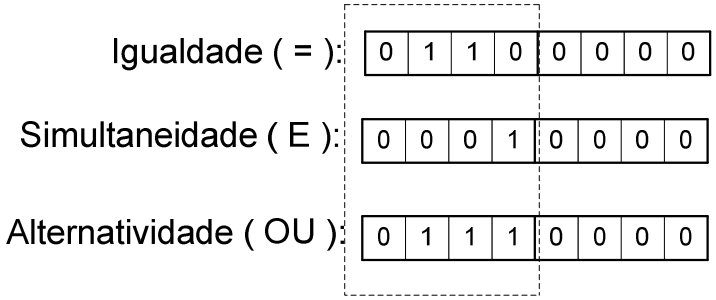


Figura 22 - Bytes utilizados para representar operações lógicas

O *byte* de estados contém valores para as entradas selecionadas e serão comparados com os valores reais vindos dos sensores ligados ao módulo. Dependendo da operação escolhida pelo *byte* de operação o módulo tomará a decisão de acionar a saída selecionada ou não.

Como exemplo temos a seguinte situação exibida na Figura 23.

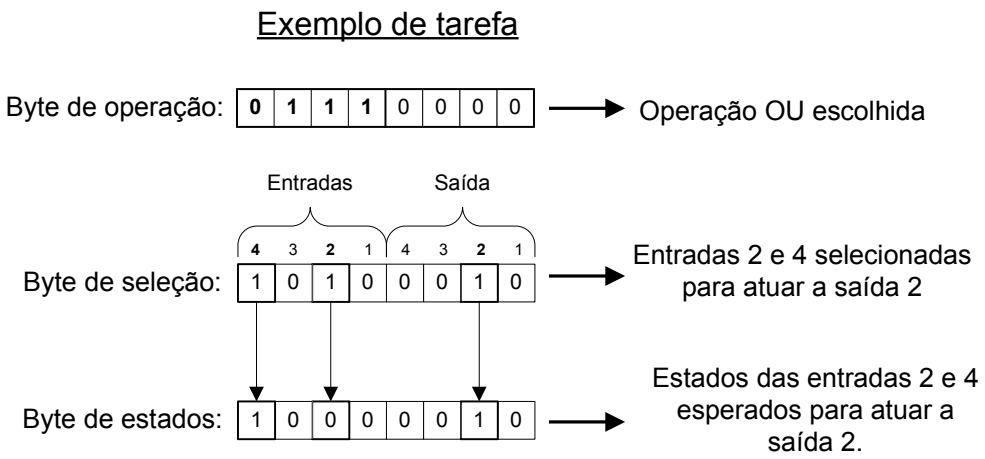


Figura 23 - Exemplo explicativo de dados relativos a uma tarefa

10.4. Limitações do projeto

- Somente uma saída lógica pode ser utilizada em cada tarefa.
- Os equipamentos acoplados ao módulo periférico somente poderão ser ligados ou desligados.
- O módulo periférico comunica-se somente com o módulo central – não há nenhuma interação com demais módulos periféricos.

10.5. Conexão das cargas elétricas

Diversos tipos de carga elétrica e sensores podem ser conectados ao módulo periférico sem necessariamente ter uma alimentação extra. O circuito do módulo fornece alimentação em corrente contínua para a grande maioria dos sensores do mercado e tem saídas lógicas de potência para equipamentos, chaveando (por meio de relés) tensões alternadas padronizadas, como 110V e 220V.

Exemplos de cargas passíveis de utilização com o módulo periférico:

- ✓ Lâmpadas (incandescentes, fluorescentes, halógenas, etc)
- ✓ Motores AC assíncronos, (abertura de portões, portas, ventiladores, etc)
- ✓ Bombas d'água (submersas ou não – poços e piscinas)

- ✓ Servo-motores (acionamento de persianas, câmeras, painéis)
- ✓ Solenóides (abertura e fechamento de válvulas, trancas)
- ✓ Alimentação de tomadas (110/220 V)
- ✓ Aquecedores e Ar Condicionados

Exemplos de sensores bastante conhecidos e de fácil implementação:

- ✓ Sensores de presença (infravermelho, ultra-som)
- ✓ Sensores de luminosidade (LDR)
- ✓ Sensores de temperatura (NTC, LM35)
- ✓ Sensores magnéticos (*reed switch*)
- ✓ Sensores indutivos
- ✓ Sensores capacitivos
- ✓ Chaves manuais (botões, interruptores)
- ✓ Chaves fim de curso

Os sensores listados podem fornecer diferentes tensões de saída, implicando para o módulo periférico a necessidade de aceitar níveis de tensão variados. Assim, como forma de se obter versatilidade em aplicações fez-se o uso de acopladores ópticos (Fairchild – modelo 4N25), drenando uma corrente elétrica mínima do sensor e proporcionando tempo de resposta pequeno, de (2µs). Acopladores óticos (Figura 24) também têm a função de bloquear eventuais picos de tensão que podem danificar circuitos sensíveis como, por exemplo, os circuitos de comando.

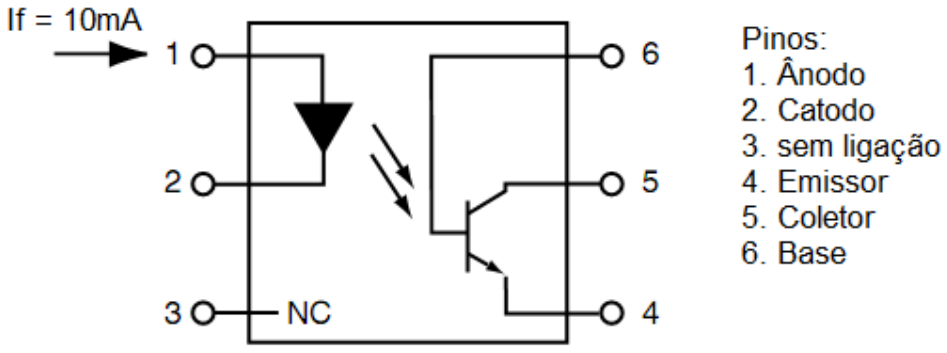


Figura 24 - Esquema elétrico interno do acoplador óptico (4N25) e pinagem

11. RESULTADOS

O software de configuração dos módulos funcionou perfeitamente. Ao longo do desenvolvimento, tanto o programa pricipal desenvolvido quanto as ferramentas propostas mostraram-se úteis e eficientes. O software também atendeu as expectetivas no que diz respeito à facilidade de uso. A Figura 25 mostra um computador com o programa de configuração aberto.

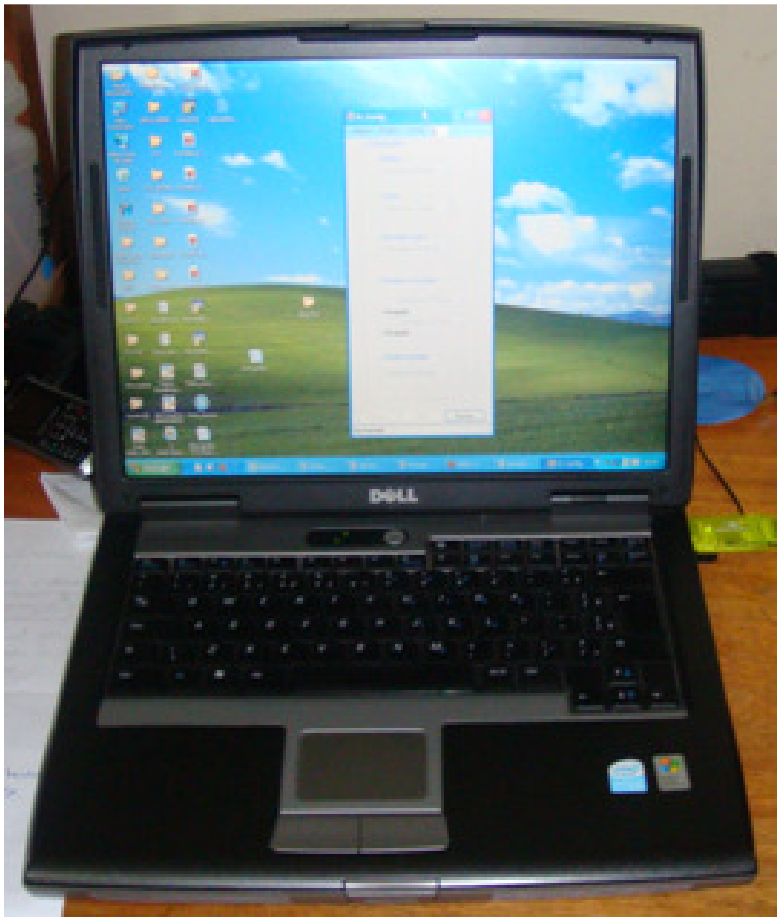


Figura 25 - Computador com o Programa M. Config. aberto

A interface serial com o módulo central, assim como as intefaces de rádio entre central e periféricos também mostraram-se confiáveis e estáveis, superando as expectativas. A Figura 26 mostra o Circuito do módulo central montado em protoboard. E a Figura 27 mostra o circuito do módulo periférico.

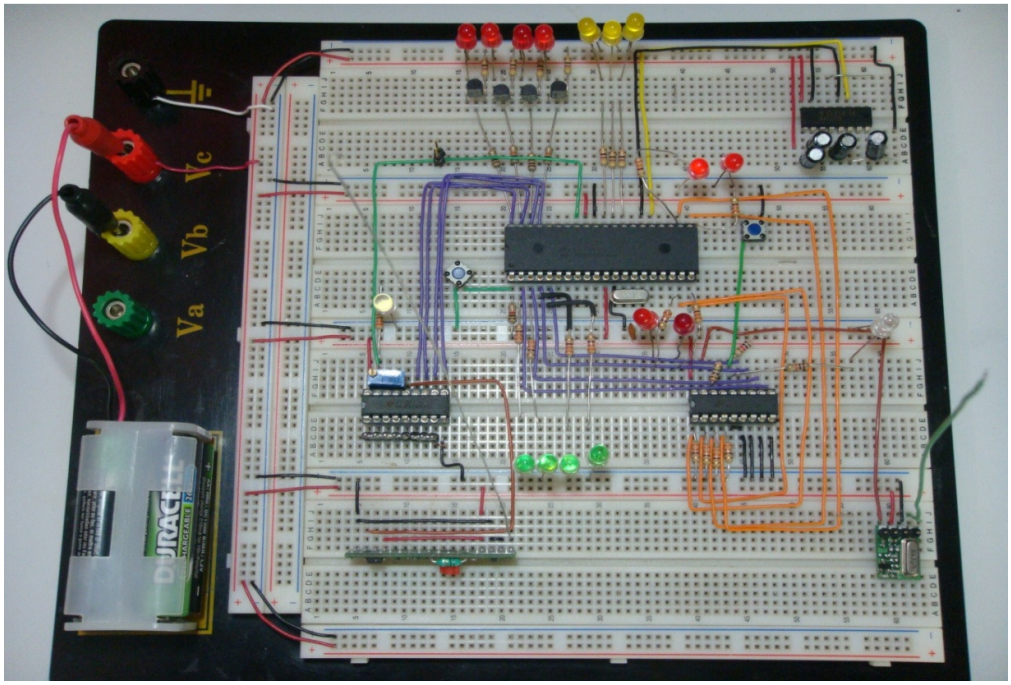


Figura 26 - Circuito do módulo central montado em protoboard

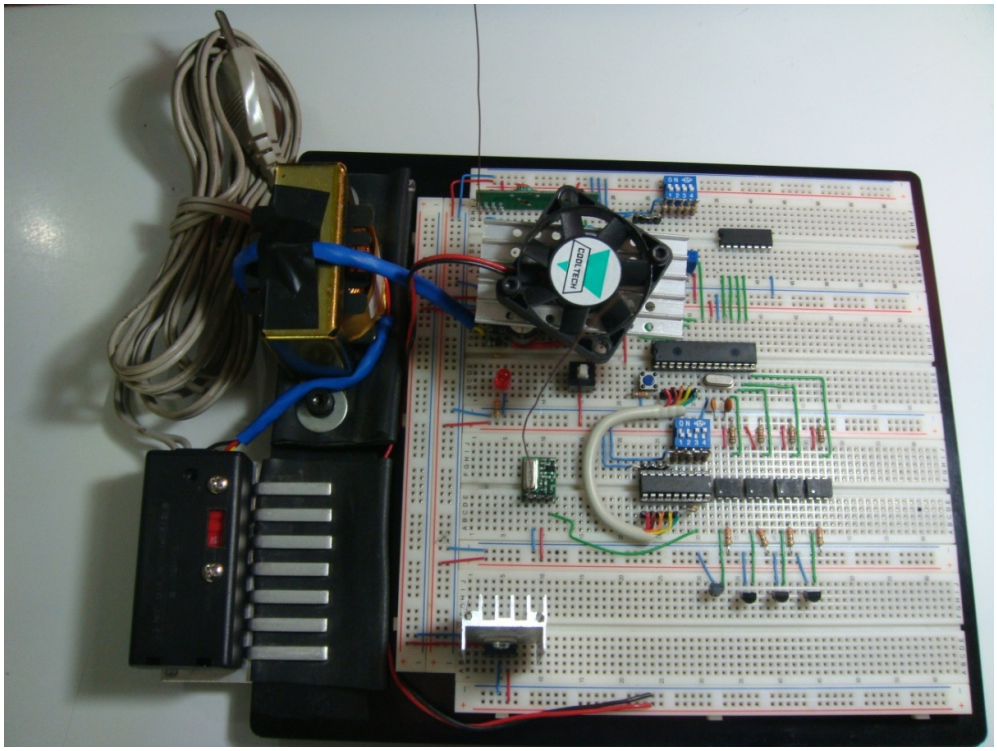


Figura 27 - Circuito do módulo periférico montado em protoboard

Após comprovado o funcionamento e a comunicação das partes, confeccionou-se duas placas de madeira com exemplos de cargas (motores e lâmpadas) instaladas, além de sensores de proximidade, movimento e fins de curso. Nesta também foi fixada uma placa de circuito impresso com relês, os quais acionam o circuito de potência (Figura 29). Tais placas foram confeccionadas com cargas alimentadas em 110V (Figura 28) e 220V (Figura 30), a fim de exemplificar a versatilidade do projeto.

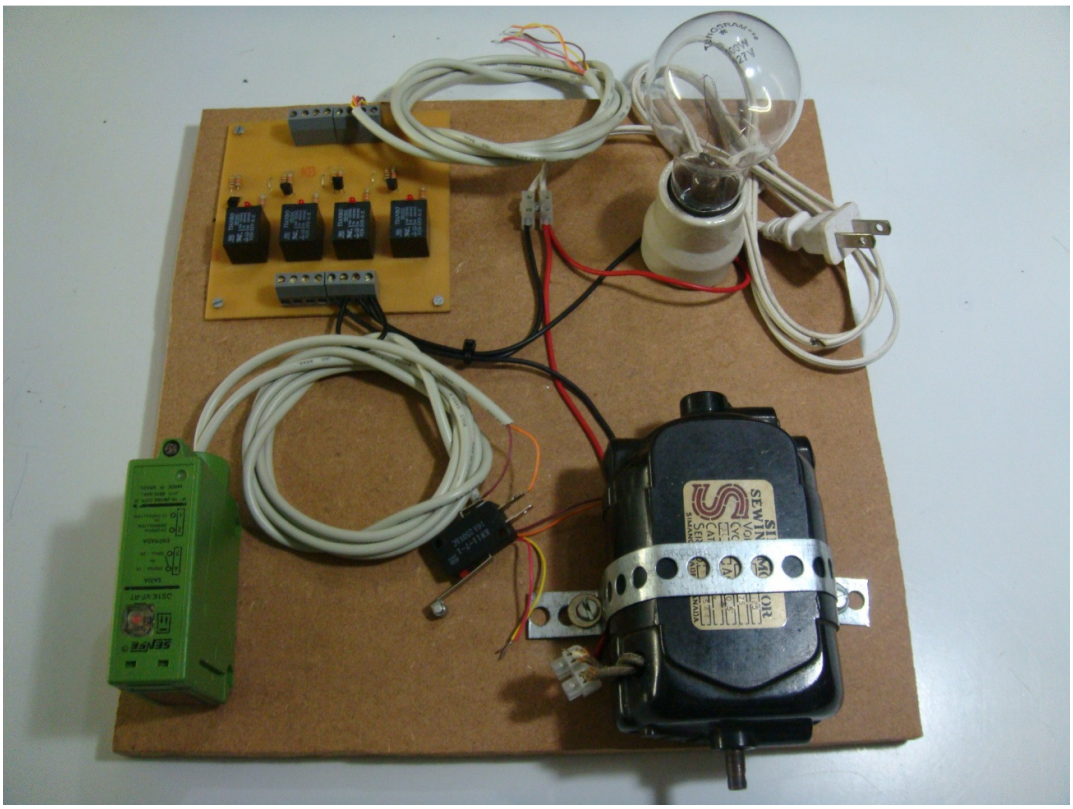


Figura 28 - Montagem das cargas e sensores

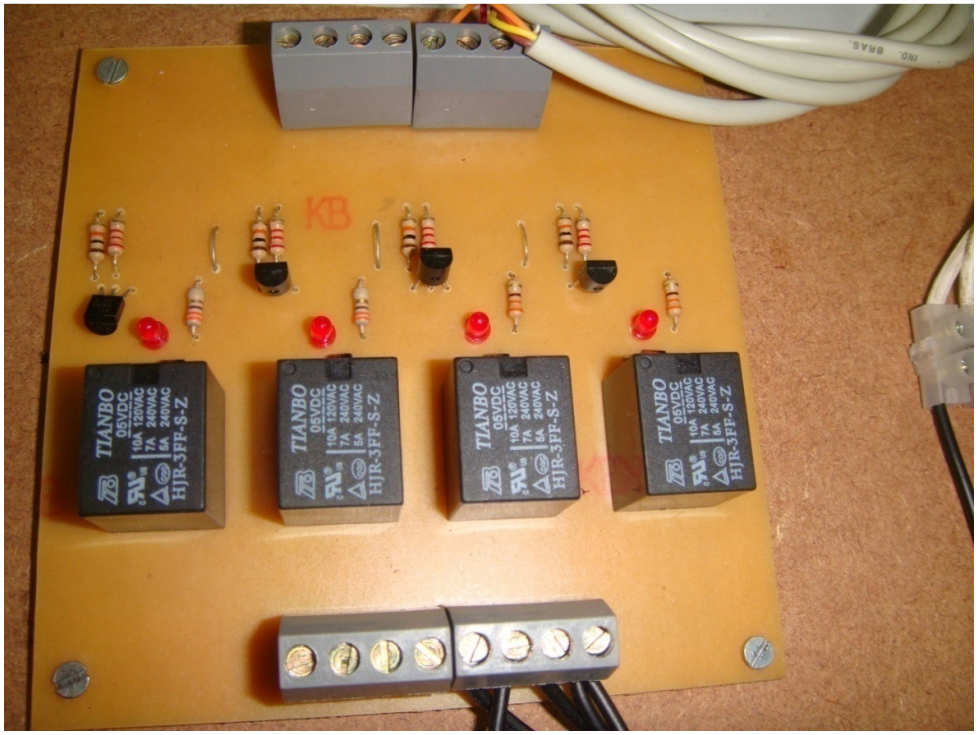


Figura 29 - Detalhe da placa de relês

Figura 30 - Montagem das cargas e sensores

12. CONCLUSÕES

A proposta deste trabalho teve, em muitas ocasiões, de ser reestudada e rediscutida a fim de se direcionar com clareza os rumos do projeto. Isto porque observou-se muitos possíveis problemas, ignorados no momento da idealização do mesmo. A execução do projeto consistiu-se basicamente de duas etapas; a primeira de estudos sobre as necessidades de um usuário comum e os produtos já existentes na área de domótica, a segunda sobre as especificações técnicas gerais do projeto. Posteriormente estudou-se em detalhes o microcontrolador a ser utilizado, junto dos softwares de desenvolvimento. Paralelamente a isto se deu início às atividades práticas (montagem de protótipos dos módulos central e periférico em *protoboard)*. Durante as montagens foi preciso realizar testes funcionais dos módulos de rádio, dos CI's *encoder* / *decoder*, entre outras singularidades, objetivando resolver passo a passo problemas de hardware limitantes e algumas vezes até rever partes do projeto por conta disto.

Como etapa final, situações diversas previstas para uma aplicação comum foram simuladas com os módulos protótipos. Nesta etapa foi possível avaliar qualitativamente todos os recursos oferecidos pelo sistema, e, apesar do desempenho satisfatório, foram criticados alguns pontos frágeis e pequenas debilidades, enquadradas para futuras melhorias. Algumas delas como aprimoramentos do *software* de interface com o usuário, maior robustez nas comunicações entre módulos e considerações prevendo uso incorreto ou mau uso valem atenção para aperfeiçoar a segurança e confiabilidade do sistema.

O intento dos integrantes deste trabalho foi de mostrar na prática os assuntos estudados durante o curso de graduação.

Dentro da idéia proposta e tempo disponível foram atingidos os objetivos esperados e obtiveram-se resultados com indicativos promissores. Os experimentos práticos criaram expectativas otimistas para a continuidade do projeto, com margem para muitos aprimoramentos, não cabíveis no escopo deste trabalho, mas perfeitamente factíveis sem a necessidade de demais recursos que os já utilizados até então.

13. REFERÊNCIAS BIBLIOGRÁFICAS

FAIRCHILD®. *Fairchild Semiconductor Webpage.* ***General Purpose 6-PIN Photo-transistor Optocoupler***. Disponível em: <http://www.fairchildsemi.com/ds/4N%2F4N25-M.pdf>. Acesso em: 22 set. 2009

MESSIAS, A. R. artigo: **LPTWIRELESS - CONTROLANDO 8 DISPOSITIVOS VIA RF (RR3 E RT4)**. Disponível em: <http://www.rogercom.com/pparalela/LptWireless.htm>. Acesso em: 15 ago. 2009

MICROCHIP®. Microchip Website. ***PIC16F87X Data Sheet - 28/40-Pin 8-Bit CMOS FLASH Microcontrollers***. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>. Acesso em: 21 mar. 2009.

ROBERTO, P. **Comunicação Serial.** Disponível em: <http://gerds.utp.br/roberto/roberto/4Bim/Serial.ppt>. Acesso em: 15 set. 2009.

SOUZA, J. D., LAVINIA, N. C. **Conectando o PIC**: Recursos Avançados. 2ª Edição. São Paulo: Érica, 2003. 379p.

Telecontrolli®. *Telecontrolli Website.* ***Radio Receiver - RR3-XXX - Super Regenerative Radio Receiver With Laser Trimmed Inductor****.* Disponível em: <http://www.telecontrolli.com/pdf/receiver/rr3.pdf>. Acesso em: 12 set. 2009.

Telecontrolli®. *Telecontrolli Website.* ***Radio Transmitter - RT4-XXX - Radio Transmitter Module with SAW Resonator and External Antenna****.* Disponível em: <http://www.telecontrolli.com/pdf/transmitter/rt4.pdf>. Acesso em: 12 set. 2009.

14. ANEXOS

14.1. Anexo 1: Código do programa do Módulo Periférico

```
/*-------------------------------------------------------------------------------------------------*/
/*------------------------------PROGRAMA PARA MÓDULOS PERIFÉRICOS-----------------------------------*/
/*-------------------------------------------------------------------------------------------------*/
/*---------------- variáveis-------------------------*/
unsigned short NIBBLE_LSB         = 0b00000000;      // Variável para separar nibble menos significativo
unsigned short NIBBLE_MSB         = 0b00000000;      // Variável para separar nibble mais significativo
unsigned short FILTRO_LSB         = 0b00001111;      // Variável para fazer "AND" com byte a ser quebrado
unsigned short FILTRO_MSB         = 0b11110000;      // Variável para fazer "AND" com byte a ser quebrado
unsigned short OPERACAO_OR        = 0b01110000;      // Definição da palavra para a função lógica OR (OU)
unsigned short OPERACAO_AND       = 0b00010000;      // Definição da palavra para a função lógica AND(E)
unsigned short OPERACAO_IGUAL     = 0b01100000;      // Definição da palavra para a função IGUAL (=)
unsigned short ENDERECO_EEPROM  [4][3];              // Endereço onde serão salvas as config na eeprom
unsigned short CONTA_NIBBLE       = 1;               // Conta nibbles recebidos pela Central (começa em 1!)
unsigned short CONTA_BYTE         = 0;               // Contador de bytes gravador por linha da matriz
unsigned short TAREFA             = 0;               // Contador para a linha onde será gravado cada série de 3 bytes recebido
unsigned short OPERACAO_DA_EEPROM  = 0;              // Operação recebida da Central e gravada na eeprom do módulo
unsigned short SELECAO_IO_EEPROM   = 0;              // Byte indicativo das entradas e saídas selecionadas para atuar
unsigned short ESTADOS_IO_EEPROM   = 0;              // Byte indicativo dos estados para que as entradas atuem para cada
saída
unsigned short BYTE               = 0;               // Variável para carregar dados gravados na EEPROM
unsigned short SELECAO_SAIDA      = 0;               // Guarda saída selecionada
unsigned short SELECAO_ENTRADA    = 0;               // Guarda entradas selecionadas
unsigned short ESTADO_SAIDA       = 0;               // Guarda estado da saída
unsigned short ESTADO_ENTRADA     = 0;               // Guarda estado(s) da(s) entrada(s)

/*-----------------------------------DECLARAÇÃO DE FUNÇÕES-------------------------------------------*/
void TESTE_SAIDA1 (void);                            // Função para executar tarefas na saída 2 do portA
void TESTE_SAIDA2 (void);                            // Função para executar tarefas na saída 2 do portA
void TESTE_SAIDA3 (void);                            // Função para executar tarefas na saída 2 do portA
void TESTE_SAIDA4 (void);                            // Função para executar tarefas na saída 2 do portA
/*-----------------------------------------INTERRUPÇÃO-----------------------------------------------*/
void interrupt (void)                                //
{                                                    //
   if (INTCON.INTF = 1)                              // Verifica se é a interrupção externa (RB0) que foi chamada
    {                                                //
      if (CONTA_NIBBLE == 1)                         // Compara contador para verificar se é o primeiro nibble a receber
        {                                            //
          if (TAREFA == 4)                           // Implica em fim do número de tarefas (4 no máx)
            {                                        //-------------------------------------------------------------\
              CONTA_NIBBLE = 1;                      // Reseta contador de nibbles recebidos                        |
              TAREFA = 0;                            // Reseta contador de linhas para gravação na EEPROM           |
              goto FIM_GRAVACAO;                     // Vai para o fim da interrupção e depois volta para o "main"  |
            }                                        //-------------------------------------------------------------/
          NIBBLE_LSB = PORTB;                        // Pega dado do PortB
          NIBBLE_LSB >> 1;                           // Rotaciona nibble de 1 posição para direita para compensar o RB0 usado p/ intrrp.
          NIBBLE_LSB = NIBBLE_LSB & FILTRO_LSB;  // Filtra dado considerando somente o nibble_lsb
          PORTA = NIBBLE_LSB;                        // Reflete o nibble recebido para a Central
          if (NIBBLE_LSB == 0b00001111)              // Comparação para avaliar se é a verificação de existência deste módulo
(00001111)
            {                                        //-----------------------------------------------------------------------\
              PORTA = NIBBLE_LSB;                    // Reflete sinal para Central indicando presença no sistema (só para um feedback)
|
              goto FIM_GRAVACAO;                     // Vai para fim da função de interrupção e de lá então voltar para a função "main"|
            }                                        //-----------------------------------------------------------------------/
           CONTA_NIBBLE = 0;                         // Este valor é necessário para mudar o contador para receber o nibble 2
        }                                            // ao final da rotina da interrupção
                                                     //
      if (CONTA_NIBBLE == 2)                         // Compara contador para chacar se é o segundo nibble a receber
        {                                            //
          NIBBLE_MSB = PORTB;                        // Pega dado do PortB
          NIBBLE_MSB << 3;                           // Rotaciona nibble recebido para a esquerda, para virar msb
          NIBBLE_MSB = NIBBLE_MSB & FILTRO_MSB;  // Separa o nibble_msb do dado recebido no PortB
          PORTA = NIBBLE_MSB;                        // Reflete nibble recebido para a Central
          /*COLOCAR AQUI UM TEMPO PARA MANDAR PELO ENCODER E DEPOIS DESABILITAR O ENVIO PELA PORTA*/
          BYTE = NIBBLE_MSB | NIBBLE_LSB;        // Combina os dois nibbles recebidos na variavel "BYTE"
          CONTA_NIBBLE = 1;                          // Rebobina contador de nibbles recebidos
          CONTA_BYTE = CONTA_BYTE ++;            // Incrementa contador de bytes
          if (CONTA_BYTE == 1)                       // Rotina para gravar Byte2 na EEPROM
```

```
          {                                         //-------------------------------------------------\
            EEPROM_Write (ENDERECO_EEPROM [TAREFA][0], BYTE);    // Escreve BYTE2 na eeprom                |
            Delay_ms (20);                                  // Tempo necessário para garantir escrita na EEPROM |
          }                                         //-------------------------------------------------/
        if (CONTA_BYTE == 2)                                // Rotina para gravar Byte3 na EEPROM
          {                                         //-------------------------------------------------\
            EEPROM_Write (ENDERECO_EEPROM [TAREFA][1], BYTE);    // Escreve BYTE3 na eeprom                |
            Delay_ms(20);                                   // Tempo necessário para garantir escrita na EEPROM |
          }                                         //-------------------------------------------------/
        if (CONTA_BYTE == 3)                                // Rotina para gravar Byte4 na EEPROM
          {                                         //-------------------------------------------------\
            EEPROM_Write (ENDERECO_EEPROM [TAREFA][2], BYTE);    // Escreve BYTE4 na eeprom                |
            Delay_ms(20);                                   // Tempo necessário para garantir escrita na EEPROM |
            TAREFA = TAREFA ++;                               // Incrementa a posição p/ próxima linha da matriz  |
            CONTA_BYTE = 0;                                //                                                  |
          }                                         //-------------------------------------------------/
       CONTA_NIBBLE = 1;                                    // Rebobina o contador de nibbles
       }                                            //
   }                                                //
FIM_GRAVACAO:                                               // Rotina para sair da interrupção quando receber
   if (CONTA_NIBBLE = 0)                                    // sinal de fim ou para carregar novo valor do conta
     {                                              // dor de nibbles.
       CONTA_NIBBLE = 2;                                    //
     }                                              //
}
/*---------------------------------------*ROTINA PRINCIPAL*-------------------------------------------*/
void main (void)                            //
{                                   //
/*------------Configurações iniciais----------*/
ADCON1 = 0x06;                              // Configura o PORTA como digital input
TRISA = 0b00111111;                             // Porta de saída para encoder, PORTA.F4 habilita envio no encoder
TRISB = 0b00011111;                             // Porta de entrada do decoder
TRISC = 0b11110000;                             // Porta de entrada e saída para periféricos
INTCON.GIE = 1;                             // Habilita interrupções globais
INTCON.PEIE = 1;                            // Habilita interrupções de periféricos
INTCON.INTE = 1;                            // Habilita a interrupção externa em RB0

PORTA = 0x00;                               // Zera PortA
PORTB = 0x00;                               // Zera PortB
PORTC = 0x00;                               // Zera PortC
/*-----------------------------------------------------------------------------------------------------*/
INICIO:                                                 //
     OPERACAO_DA_EEPROM = Eeprom_read (ENDERECO_EEPROM [TAREFA][1]);      // Lê operação da EEPROM
          Delay_ms(20);                                     // Tempo para garantir leitura da EEPROM
                                                    //
     SELECAO_IO_EEPROM  = Eeprom_read (ENDERECO_EEPROM [TAREFA][0]);      // Carrega Byte com seleções de
entradas e saída
          Delay_ms(20);                                     // Tempo para garantir leitura da EEPROM
          SELECAO_SAIDA   = SELECAO_IO_EEPROM & FILTRO_LSB;             // Carrega bits da seleção de saída - só 1
saída permitida!
          SELECAO_ENTRADA = SELECAO_IO_EEPROM & FILTRO_MSB;             // Carrega bits da seleção das entradas - 2
seleções no máximo
                                                    //
     ESTADOS_IO_EEPROM  = Eeprom_read (ENDERECO_EEPROM [TAREFA][2]);      // Carrega dado da EEPROM na
variável
          Delay_ms(20);                                     // Tempo para garantir leitura da EEPROM
          ESTADO_SAIDA   = ESTADOS_IO_EEPROM & FILTRO_LSB;              // Byte com estado da saída selecionada
          ESTADO_ENTRADA = ESTADOS_IO_EEPROM  & FILTRO_MSB;             // Byte com estado(s) da(s) saída(s)
selecionada(s)

     if (SELECAO_SAIDA == 0b00000001)       // Verifica se a saída 1 foi selecionada
       {                                    //
         TESTE_SAIDA1 ();               // Chamada para função de operações para a Tarefa relacionada a esta saída
       }                                    //
     if (SELECAO_SAIDA == 0b00000010)       // Verifica se a saída 2 foi selecionada
       {                                    //
         TESTE_SAIDA2 ();               // Chamada para função de operações para a Tarefa relacionada a esta saída
       }                                    //
     if (SELECAO_SAIDA == 0b00000100)       // Verifica se a saída 3 foi selecionada
       {                                    //
         TESTE_SAIDA3 ();               // Chamada para função de operações para a Tarefa relacionada a esta saída
       }                                    //
     if (SELECAO_SAIDA == 0b00001000)       // Verifica se a saída 4 foi selecionada
       {                                    //
         TESTE_SAIDA4 ();               // Chamada para função de operações para a Tarefa relacionada a esta saída
```

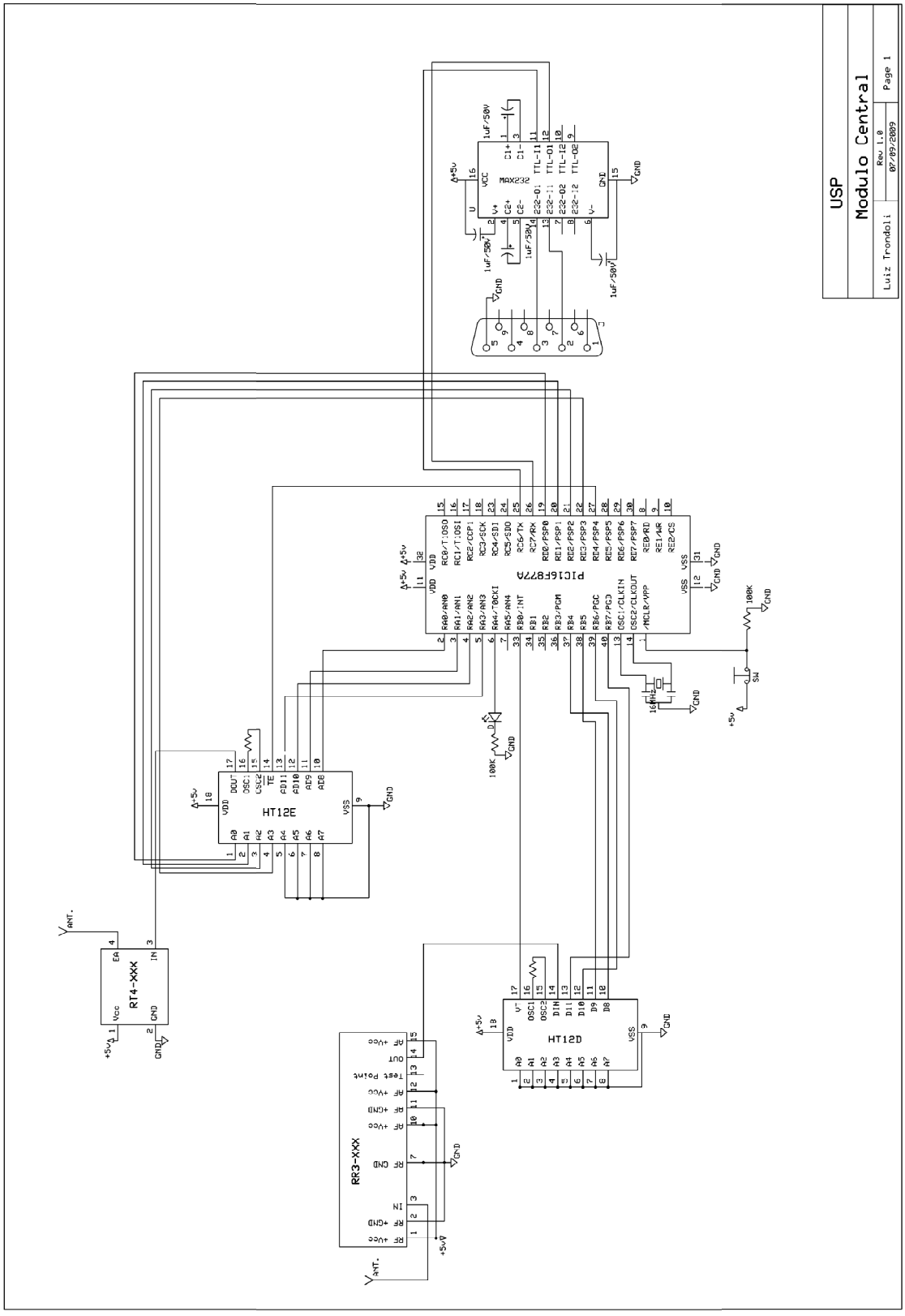
```
      }                           //
    TAREFA = TAREFA ++;                 // Incrementa posição da EEPROM a ser lida
    if (TAREFA == 4)                  // Verifica se as 4 tarefas já foram executadas
      {                           //
        TAREFA = 0;                   // Reseta posição de memória para as tarefas
      }                           //
    goto INICIO;                      // FIM DA FUNÇÃO MAIN!
}
/*-----------------------------FUNÇÕES DE OPERAÇÕES-----------------------------------------------------*/
void TESTE_SAIDA1 (void)
   {
   if (OPERACAO_DA_EEPROM == OPERACAO_IGUAL)                    // Operação de igual (=)
     {
      switch (SELECAO_ENTRADA)
           {
           case 0b00010000:
               if (PORTC.F4 == ESTADO_ENTRADA.F4)              // Comparara entrada1 do PortC com Entrada configurada
                  PORTC.F0 =  ESTADO_SAIDA.F0;                 // Seta estado escolhido da saída na saída 1
           case 0b00100000:
               if (PORTC.F5 == ESTADO_ENTRADA.F5)              // Comparara entrada2 do PortC com Entrada configurada
                  PORTC.F0 =  ESTADO_SAIDA.F0;                 // Seta estado escolhido da saída na saída 1
           case 0b01000000:
               if (PORTC.F6 == ESTADO_ENTRADA.F6)              // Comparara entrada3 do PortC com Entrada configurada
                  PORTC.F0 =  ESTADO_SAIDA.F0;                 // Seta estado escolhido da saída na saída 1
           case 0b10000000:
               if (PORTC.F7 == ESTADO_ENTRADA.F7)              // Comparara entrada4 do PortC com Entrada configurada
                  PORTC.F0 =  ESTADO_SAIDA.F0;                 // Seta estado escolhido da saída na saída 1
           }
     }
   if (OPERACAO_DA_EEPROM == OPERACAO_AND)                      // Operação "E"
     {
      switch (SELECAO_ENTRADA)
           {
           case 0b00110000:                                    //  ____________________________________
               if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F5 == ESTADO_ENTRADA.F5)   // /Compara valor das
portas selecionadas   \
                  PORTC.F0 =  ESTADO_SAIDA.F0;                 // |com valores setados nos estados das     |
           case 0b01010000:                                    // |portas. Se coincidir para o caso, setará|
               if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F6 == ESTADO_ENTRADA.F6)   // |a saída 1 do PortC
com o valor de estado|
                  PORTC.F0 =  ESTADO_SAIDA.F0;                 // |configurado para tal.                  |
           case 0b01100000:                                    // \____________________________________/
               if (PORTC.F5 == ESTADO_ENTRADA.F5 && PORTC.F6 == ESTADO_ENTRADA.F6)   //
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b10010000:
               if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F4 == ESTADO_ENTRADA.F4)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b10100000:
               if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F5 == ESTADO_ENTRADA.F5)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b11000000:
               if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           }
     }
   if (OPERACAO_DA_EEPROM == OPERACAO_OR)                       // Operação "OU"
     {
      switch (SELECAO_ENTRADA)
           {
           case 0b00110000:
               if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F5 == ESTADO_ENTRADA.F5)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b01010000:
               if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b01100000:
               if (PORTC.F5 == ESTADO_ENTRADA.F5 || PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b10010000:
               if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F4 == ESTADO_ENTRADA.F4)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b10100000:
               if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F5 == ESTADO_ENTRADA.F5)
                  PORTC.F0 =  ESTADO_SAIDA.F0;
           case 0b11000000:
```

```
                if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F0 =  ESTADO_SAIDA.F0;
            }
        }
    }
void TESTE_SAIDA2 (void)
    {
    if (OPERACAO_DA_EEPROM == OPERACAO_IGUAL)       //operacao de igual (=)
      {
       switch (SELECAO_ENTRADA)
            {
            case 0b00010000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b00100000:
                if (PORTC.F5 == ESTADO_ENTRADA.F5)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b01000000:
                if (PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b1000000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            }
      }
    if (OPERACAO_DA_EEPROM == OPERACAO_AND)
      {
       switch (SELECAO_ENTRADA)
            {
            case 0b00110000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F5 == ESTADO_ENTRADA.F5)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b01010000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b01100000:
                if (PORTC.F5 == ESTADO_ENTRADA.F5 && PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b10010000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F4 == ESTADO_ENTRADA.F4)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b10100000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F5 == ESTADO_ENTRADA.F5)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b11000000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            }
      }
    if (OPERACAO_DA_EEPROM == OPERACAO_OR)
      {
       switch (SELECAO_ENTRADA)
            {
            case 0b00110000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F5 == ESTADO_ENTRADA.F5)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b01010000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b01100000:
                if (PORTC.F5 == ESTADO_ENTRADA.F5 || PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b10010000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F4 == ESTADO_ENTRADA.F4)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b10100000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F5 == ESTADO_ENTRADA.F5)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            case 0b11000000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F1 =  ESTADO_SAIDA.F1;
            }
      }
    }
void TESTE_SAIDA3 (void)
```

```
    {
    if (OPERACAO_DA_EEPROM == OPERACAO_IGUAL)       //operacao de igual (=)
      {
       switch (SELECAO_ENTRADA)
            {
            case 0b00010000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b00100000:
                if (PORTC.F5 == ESTADO_ENTRADA.F5)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b0100000:
                if (PORTC.F6 == ESTADO_ENTRADA.F6)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b10000000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
                }
           }
    if (OPERACAO_DA_EEPROM == OPERACAO_AND)
      {
       switch (SELECAO_ENTRADA)
            {
            case 0b00110000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F5 == ESTADO_ENTRADA.F5)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b01010000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F6 == ESTADO_ENTRADA.F6)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b01100000:
                if (PORTC.F5 == ESTADO_ENTRADA.F5 && PORTC.F6 == ESTADO_ENTRADA.F6)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b10010000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F4 == ESTADO_ENTRADA.F4)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b10100000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F5 == ESTADO_ENTRADA.F5)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b11000000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F6 == ESTADO_ENTRADA.F6)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            }
      }
    if (OPERACAO_DA_EEPROM == OPERACAO_OR)
      {
       switch (SELECAO_ENTRADA)
            {
            case 0b00110000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F5 == ESTADO_ENTRADA.F5)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b01010000:
                if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F6 == ESTADO_ENTRADA.F6)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b01100000:
                if (PORTC.F5 == ESTADO_ENTRADA.F5 || PORTC.F6 == ESTADO_ENTRADA.F6)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b10010000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F4 == ESTADO_ENTRADA.F4)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b10100000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F5 == ESTADO_ENTRADA.F5)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            case 0b11000000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F6 == ESTADO_ENTRADA.F6)
                   PORTC.F2 =  ESTADO_SAIDA.F2;
            }
      }
}
void TESTE_SAIDA4(void)
{
    if (OPERACAO_DA_EEPROM == OPERACAO_IGUAL)       //operacao de igual (=)
      {
       switch (SELECAO_ENTRADA)
            {
            case 0b00010000:
```

```
                if (PORTC.F4 == ESTADO_ENTRADA.F4)
                    PORTC.F3 =  ESTADO_SAIDA.F3;
            case 0b00100000:
                if (PORTC.F5 == ESTADO_ENTRADA.F5)
                    PORTC.F3 =  ESTADO_SAIDA.F3;
            case 0b01000000:
                if (PORTC.F6 == ESTADO_ENTRADA.F6)
                    PORTC.F3 =  ESTADO_SAIDA.F3;
            case 0b10000000:
                if (PORTC.F7 == ESTADO_ENTRADA.F7)
                    PORTC.F3 =  ESTADO_SAIDA.F3;
            }
    }
  if (OPERACAO_DA_EEPROM == OPERACAO_AND)
    {
     switch (SELECAO_ENTRADA)
          {
          case 0b00110000:
              if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F5 == ESTADO_ENTRADA.F5)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b01010000:
              if (PORTC.F4 == ESTADO_ENTRADA.F4 && PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b01100000:
              if (PORTC.F5 == ESTADO_ENTRADA.F5 && PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b10010000:
              if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F4 == ESTADO_ENTRADA.F4)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b10100000:
              if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F5 == ESTADO_ENTRADA.F5)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b11000000:
              if (PORTC.F7 == ESTADO_ENTRADA.F7 && PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          }
    }
  if (OPERACAO_DA_EEPROM == OPERACAO_OR)
    {
     switch (SELECAO_ENTRADA)
          {
          case 0b00110000:
              if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F5 == ESTADO_ENTRADA.F5)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b01010000:
              if (PORTC.F4 == ESTADO_ENTRADA.F4 || PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b01100000:
              if (PORTC.F5 == ESTADO_ENTRADA.F5 || PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b10010000:
              if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F4 == ESTADO_ENTRADA.F4)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b10100000:
              if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F5 == ESTADO_ENTRADA.F5)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          case 0b11000000:
              if (PORTC.F7 == ESTADO_ENTRADA.F7 || PORTC.F6 == ESTADO_ENTRADA.F6)
                  PORTC.F3 =  ESTADO_SAIDA.F3;
          }
    }
}
```

14.2. Anexo 4: Esquema elétrico do módulo central



14.3. Anexo 5: Esquema elétrico do módulo periférico