MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
(Real Academia de Artilharia, Fortificação e Desenho, 1792)

Cap LEONARDO GHIDETTI MORAES DE ANDRADE

Ten THASSIA LOPES CORREIA DOS SANTOS

SISTEMA DE CONTROLE DE DISPOSITIVOS REMOTOS VIA REDE GSM/GPRS

Rio de Janeiro

2013

INSTITUTO MILITAR DE ENGENHARIA

Cap LEONARDO GHIDETTI MORAES DE ANDRADE

Ten THASSIA LOPES CORREIA DOS SANTOS

SISTEMA DE CONTROLE DE DISPOSITIVOS REMOTOS VIA REDE GSM/GPRS

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Instituto Militar de Engenharia.

Orientador**:** Cel R/1 Marcus Vinícius dos Santos Fernandes –D.C.

Rio de Janeiro

2013

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro - RJ CEP: 22290-270

621.38 Andrade, Leonardo Ghidetti de Moraes.
A553s Sistema de Controle de dispositivos Remotos via Rede GSM/GPRS/ Leonardo Ghidetti de Moraes Andrade, Thassia Lopes Correia dos Santos; orientados por Marcus Vinícius dos Santos Fernandes. - Rio de Janeiro: Instituto Militar de Engenharia, 2013.

142 p.: il.

Projeto de Final de Curso - Instituto Militar de Engenharia – Rio de Janeiro, 2013.

1. Engenharia eletrônica. 2. Controle remoto. 3. Modem. 4. Microcontrolador. 5. GPRS. 6. AT. 7. Domótica. I. Santos, Thassia Lopes Correia. II. Fernandes, Marcus Vinícius dos Santos. III. Título. IV. Instituto Militar de Engenharia.
CDD 621.38

INSTITUTO MILITAR DE ENGENHARIA

Cap LEONARDO GHIDETTI MORAES DE ANDRADE

Ten THASSIA LOPES CORREIA DOS SANTOS

SISTEMA DE CONTROLE DE DISPOSITIVOS REMOTOS VIA REDE GSM/GPRS

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia Eletrônica do Instituto Militar de Engenharia.
Orientador: Cel R/1 Marcus Vinícius dos Santos Fernandes- D.C.

Aprovada em 24 de junho de 2013 pela seguinte Banca Examinadora:

_____

Cel R/1 Marcus Vinícius dos Santos Fernandes - D.C., do IME

_____

Maj Mauro Cezar Rebello Cordeiro – D.C, do IME

_____

Maj Luciene da Silva Demenicis - D.C., do IME

Rio de Janeiro
2013

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Domótica é a área do conhecimento e também da engenharia voltada ao desenvolvimento de soluções de automação residencial para dispor, aos seus usuários, maior conforto e segurança. Dentro do contexto da automação residencial, a comunicação de dados entre dispositivos é um dos fatores mais importantes na área de aquisição de dados. Baseando-se na comunicação sem fio, houve a necessidade de se desenvolver um sistema de monitoração e controle que se comunique a um computador remoto, utilizando tecnologias com padrões abertos e de baixo custo, como o padrão de rede serial RS485 e o microcontrolador PIC. Para realizar a comunicação, foi utilizado o modem GSM/GPRS da Telit. Todo o processamento necessário é realizado por um microcontrolador PIC16F877A que faz a interface entre os dispositivos e o modem.

Palavras-chave: controle remoto, modem, microcontrolador, GPRS, AT, domótica.

ABSTRACT

Home automation is a field of knowledge and also of engineering dedicated to the development of solutions to provide greater comfort and safety to its users. Within the context of home automation, data communication between devices is one of the most important factors in the field of data acquisition. Based on wireless communication, there was a necessity to develop a monitoring and control system that communicates to a remote computer, using technologies with open standards and low cost, as the standard RS485 serial network and PIC microcontroller. It was used modem GSM / GPRS Telit in order to establish the communication. All the processing necessary is performed by a PIC16F877A microcontroller which performs the interface between devices and the modem.

Keywords: remote control, modem, microcontroller, GPRS, AT, home automation.

1. Introdução

Ao longo do tempo, a computação vem ajudando pessoas comuns a fazerem seus trabalhos de forma cada vez mais rápida e eficiente. Com o advento da automação, tarefas repetitivas podem ser realizadas por máquinas.

Gradativamente, a automação vem se tornando comum nas instalações residenciais, visando o conforto, entretenimento, economia e segurança, resultando no que pode ser chamado de **domótica**. Os sistemas domóticos permitem a gestão de todos os recursos habitacionais, através da utilização de computadores, controle remoto, aparelhos celulares, e outros dispositivos fixos e móveis capazes de realizar tarefas complexas e interagir com o usuário e com o meio físico. Em outras palavras, remotamente, permitem ligar ou desligar uma televisão, abrir ou fechar uma persiana, aumentar ou diminuir a temperatura de um ar-condicionado, entre outros.

Com o avanço das telecomunicações, principalmente da comunicação sem fio, os dispositivos móveis vem ganhando espaço considerável no mercado de transmissão de dados, tanto na comunicação de longa distância como, principalmente, nos ambientes locais.

Nos ambientes residenciais, muitas soluções estão sendo pesquisadas e desenvolvidas com o objetivo de possibilitar a comunicação entre dispositivos sem o uso de fio, tornando mais flexível e prático o uso de tais equipamentos. O presente trabalho visa mostrar uma solução de baixo custo para a automação residencial. Usando chips microcontroladores e uma rede RS-485 para comunicação entre eles e um modem GSM/GPRS, será desenvolvido um sistema capaz de controlar a iluminação e equipamentos elétricos de uma residência, através da rede GPRS.

1.1 Objetivo

Este trabalho tem como objetivo desenvolver um projeto de automação residencial capaz de realizar o monitoramento e controlar alguns dispositivos de uma residência através de dispositivos móveis.

1.2 Justificativa

Uma casa inteligente contém um sistema capaz de gerenciar todo o tráfego de informações, bem como o controle de equipamentos, permitindo um conforto maior com o menor gasto possível de energia. Os sistemas de segurança, os sistemas de aquecimento, ventilação e ar-condicionado, e entretenimento tendem a beneficiar as novas construções, mas os projetos de reforma também podem ser contemplados com esta nova tecnologia. Com isso, a centralização dos sistemas de controle tende a diminuir o tempo gasto com o projeto e a facilitar a manutenção. Permite também a busca de possíveis erros, de forma rápida, assim como o disparo das ações necessárias para solucioná-los. Com a aplicação certa das técnicas, pode-se conseguir atingir altos níveis de conforto e segurança a custos relativamente baixos.

Para uma boa parte da população, o conforto, comodidade e a segurança providos pela domótica, não passam de um sonho de consumo inatingível. No entanto, esta realidade pode ser modificada com soluções que proporcionem tais benefícios, aliando também, um projeto de baixo custo. Dentro deste cenário, este trabalho tem o desafio de apresentar uma solução simples de domótica, que possui baixo custo, com o objetivo de desmistificar a ideia de que a automação residencial não é uma tecnologia palpável.

1.3 Metodologia

Primeiramente, foi feito um estudo sobre o modem utilizado e os comandos AT. Depois, realizou-se a comunicação entre os modems (um deles ligado a um computador de IP fixo e outro, a um computador conectado à rede GPRS, provida por empresa de telefonia celular). Em seguida, foi proposto um modelo de casa inteligente e desenvolveram-se o hardware e o software de controle necessários à sua implementação.

1.4 Estrutura

O trabalho está estruturado da seguinte forma: No item 2, é feita uma revisão da literatura utilizada. No item 3, é descrita a implementação do projeto (hardware e software). No item 4, são apresentados os testes e a fase final do projeto. A bibliografia está disponível no item 5.

2. Revisão de Literatura

2.1 Tecnologia GSM/GPRS

A tecnologia GSM/GPRS é uma integração das redes GSM com GPRS (*General Packet Radio Service*), ou seja, é a junção de uma rede GPRS em cima da outra rede celular GSM, que possui abrangência global. Com a popularização do uso da internet e de outros serviços de dados, observou-se que as redes GSM não comportariam grandes quantidades de dados nos diferentes estágios do sistema. Por esse motivo, as redes GPRS foram desenvolvidas para aceitar serviços de dados, pois foram criadas com base em transmissão por comutação de pacotes, que utiliza a fonte de rádio para tráfego em rajadas. Para que as operadoras possam utilizar os serviços GSM e GPRS, os dois sistemas compartilham características entre si, tais como as bandas de freqüência, a estrutura de *frames* e técnicas de modulação. Mas, no entanto, a cobrança pelo uso do sistema GPRS é feita pela quantidade de dados transmitidos, enquanto que no sistema GSM, é feita por tempo de conexão. A tecnologia GSM/GPRS explora as redes celulares e internet, e esta integração com a internet é umas das grandes vantagens do sistema GPRS, porque permite a conexão com qualquer ponto do mundo em diferentes equipamentos. Essa versatilidade do sistema é algo importante em qualquer sistema de monitoramento remoto de dados. O sistema possui também recursos de segurança, chaves de autenticação, código de identificação pessoal (Código PIN – *Personal Identification Number*), etc., que podem ser usados para controlar e proteger conexões entre os dispositivos. A transmissão de dados pode ser feita através de duas maneiras: via GPRS- Internet ou via SMS (*Short Message Services*). A transmissão de dados via GPRS – Internet é feita na forma de protocolos em camadas. A rede GPRS utiliza dois tipos de protocolos para a transferência de dados, o IP e o X.25.

2.2 Módulo GSM/GPRS GM862-QUAD

O módulo de comunicação usado no projeto é o modelo GM862 (Telit), que é utilizado em larga escala no mercado. O módulo GSM/GPRS GM862 (Figura 1) é uma interface compacta que apresenta as seguintes características:

- Baixo consumo;
- Possui memória flash e RAM;
- Alimentação 12 Volt;
- Possui conector de antena GSM;
- Possui entrada para cartão SIM;
- Possui saída serial.
- Formato de comunicação de alto nível (comandos AT);
- SMS (*Short Message Service*);

- Pilha TCP/IP interna;
- Acesso à rede GSM/GPRS.

O módulo trabalha com banda Quad-Band EGSM 850/900/1800/1900 MHz e classe B de estação móvel. O multiplexador GSM 7.10 permite a aquisição de dados via porta serial, com integração à internet via protocolo TCP/IP. Oferece um desempenho de voz, SMS, dados e fax. A Figura 1 mostra o modem GSM/GPRS externamente e a antena GSM. O modem possui os seguintes conectores de interfaceamento: conector da antena, da fonte de alimentação, entrada serial e entrada para o chip SIM.

Figura 1 - Modem GSM/GPRS

2.3 Comandos AT

Os modems são módulos largamente divulgados, com ligação à rede telefônica, e cuja interface com computadores pessoais seguem as normas *standard.* A troca de informações entre um computador e um modem ligado via porta serial utiliza geralmente um protocolo, designado como **comandos AT**. O padrão AT é uma linguagem de comandos orientados por linha. Cada comando é constituído por três elementos: O prefixo, o corpo de comando, e o caractere de fim de comando ou terminação. O prefixo consiste nos caracteres “AT”. O corpo de comando é constituído por caracteres individuais. E o caractere de fim de comando é o “<CR>”.

Segundo Arthut (2007), uma linha de comando AT pode conter um ou mais comandos, usando delimitadores para separar cada comando. Esse delimitador pode ser um ponto e vírgula ou um espaço para comandos básicos.

Quando um comando é enviado, o modem responde com uma mensagem (*Result Code*), ou "Código Resultante", que avisa para o modem o resultado do comando que foi requisitado. Os comandos AT são projetados de acordo com a ITU (*International Telecommunication Union).*

2.4 A rede de comunicação

A rede de comunicação escolhida foi a rede RS485, por ser uma rede robusta e bastante utilizada, por poder operar no modo multiponto, possuir capacidade de comunicação com cabos de grandes comprimentos e facilidade de conversão do padrão RS232 para o padrão RS485, características essenciais para a rede deste protótipo. Para que os dispositivos pudessem se comunicar por esta rede, foram necessários circuitos integrados conversores, que serviram para converter os sinais do padrão RS232 do modem para o padrão TTL e os sinais TTL do microcontrolador para RS485. A seguir, pode ser visto o modo de ligação destes circuitos integrados na rede de comunicação. A Figura 2 mostra a ligação do CI MAX232 para converter os sinais da interface RS232 do modem para os sinais TTL. Já a Figura 3, mostra a ligação do CI MAX485 para converter os sinais TTL dos microcontroladores para os sinais do padrão RS485.

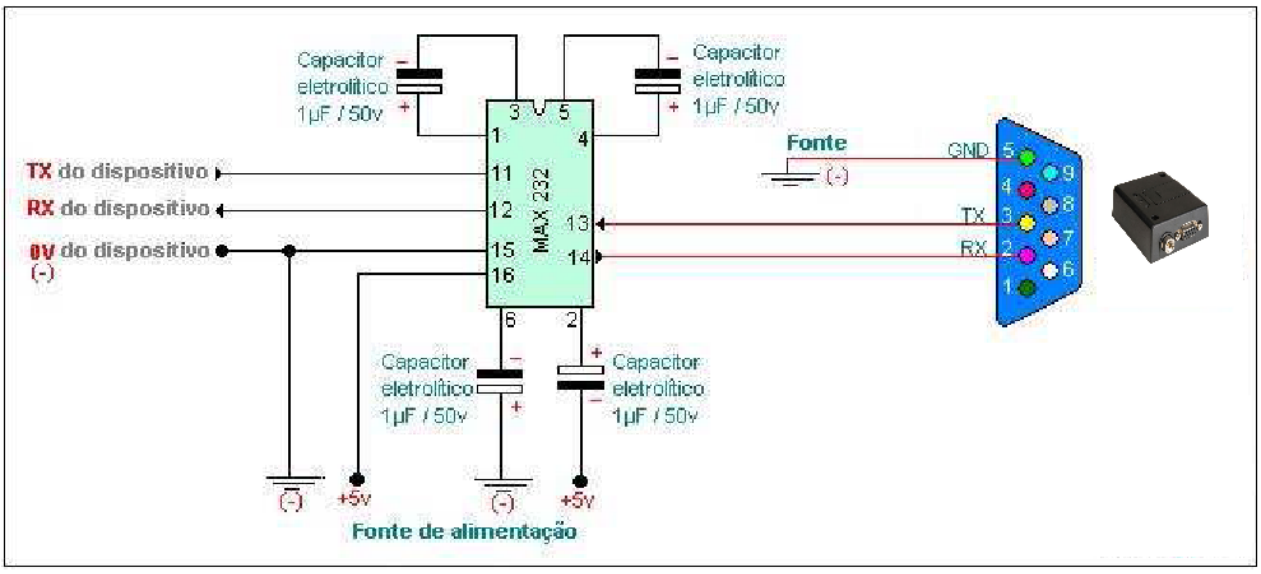


Figura 2 - Ligação do CI MAX232 na interface RS232 para conversão TTL/ RS232

Autor: Rogercom.com

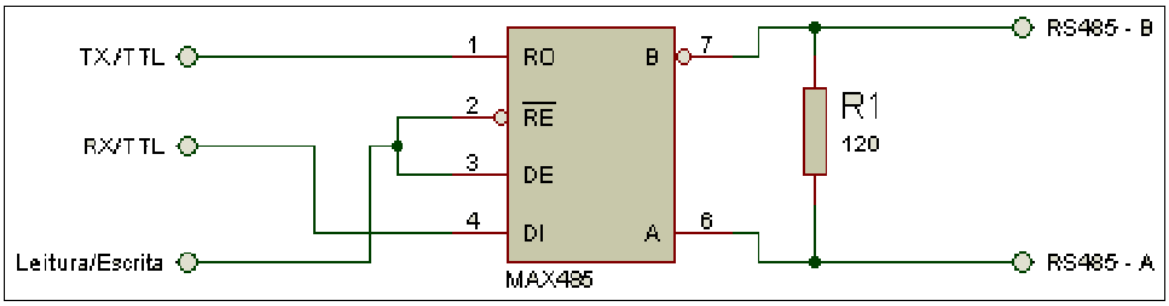


Figura 3 - Ligação do CI MAX485 para conversão de TTL/RS485

Autor: Rogercom.com

O RS-485 é um padrão elétrico utilizado para comunicação de dados de alta velocidade, em distâncias de até 1200 metros, sendo adequado para ambientes expostos a elevados níveis de distúrbios. Suas principais características são:

· Transmissão diferencial balanceada;

· Característica multiponto;

· Apenas uma fonte simples de +5V para alimentar os circuitos de transmissão e recepção;

· Transmissão de dados em modo comum com tensões de -7V até +12V;

· Suporta até 32 dispositivos.

A distância do cabo depende da taxa de transmissão utilizada. Por exemplo, uma taxa de transmissão de dados de 10Mbps pode ocorrer até uma distância máxima de 12 metros, enquanto taxas de transmissão de até 100kbps podem ocorrer em distâncias de até 1200 metros.

2.5 O reed-switch

O reed-switch é composto de uma cápsula de vidro e de duas lâminas de um material ferromagnético (ligas de níquel e ferro). As duas lâminas são colocadas muito próximas, sem que haja contato entre elas (com uma extremidade afixada no vidro) e mergulhadas num gás inerte, para não sofrerem oxidação ou deformação mecânica (para durarem mais).

- Para acionar o reed-switch, isto é, para haver contato elétrico entre as lâminas, é necessário induzir a magnetização delas, fazendo com que elas se atraiam magneticamente. Basta aproximar um pequeno ímã do reed-switch, como mostra a Figura 4.

- São usados para acionar, magneticamente, dispositivos eletro-eletrônicos como alarmes, trancas elétricas, portas, circuitos eletrônicos de partida.

Figura 4 - Funcionamento do reed-switch

2.6 O relé

Os relés são dispositivos comutadores electromecânicos. Representam-se pelo símbolo:

Relé Simples Relé duplo

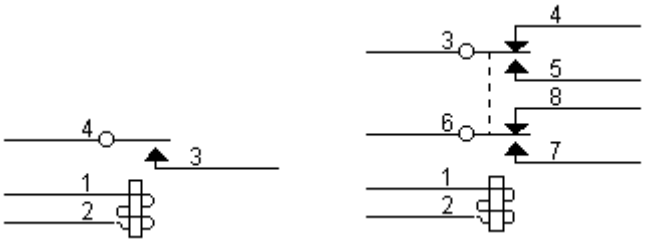


Figura 5 - Tipos de relé

Uma bobina ao ser percorrida por uma corrente, gera um campo magnético no seu núcleo que atrai um ou vários contatos elétricos, permitindo ligar, desligar ou comutar um circuito elétrico externo.

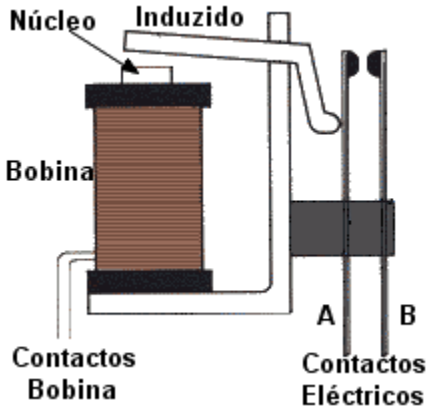


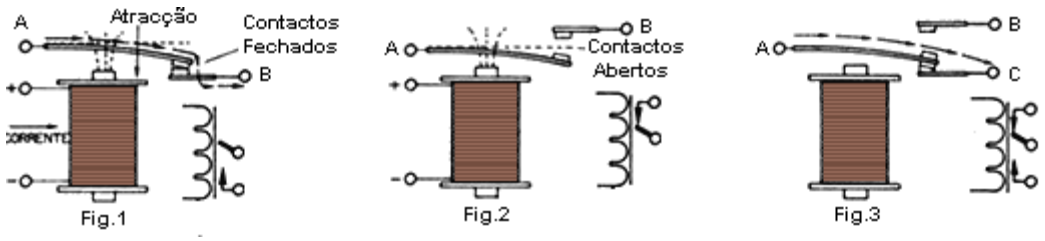
Figura 6 - Estrutura do relé

No exemplo da imagem, uma bobina, ao receber uma tensão nos seus terminais, cria um campo magnético que, através do seu núcleo, atrai o induzido, fechando os contatos entre os pontos A e B.

2.6.1 Funcionamento do relé

No funcionamento de um relé, na sua forma mais comum de aplicação permite três tipos básicos de funcionamento.

- Fig.1- O relé fecha o circuito entre os terminais A e B.
- Fig.2- O relé abre o circuito entre os terminais A e B.
- Fig.3- O relé comuta a tensão que entra no terminal A comutando entre o terminal B e C.



2.6.2 Circuito do Relé

A figura abaixo mostra o circuito de funcionamento de um relé:

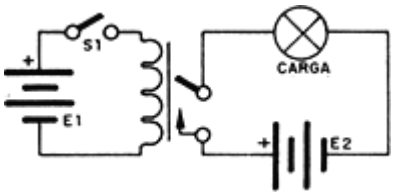


Figura 7 - Circuito do relé

A estrutura do relé permite, ao ser energizado com correntes muito pequenas, em relação à corrente do circuito controlado, o controle de circuitos de altas correntes (motores, lâmpadas, máquinas industriais...), diretamente a partir de dispositivos eletrônicos fracos, (transistores, circuitos integrados, fotorresistores, etc).

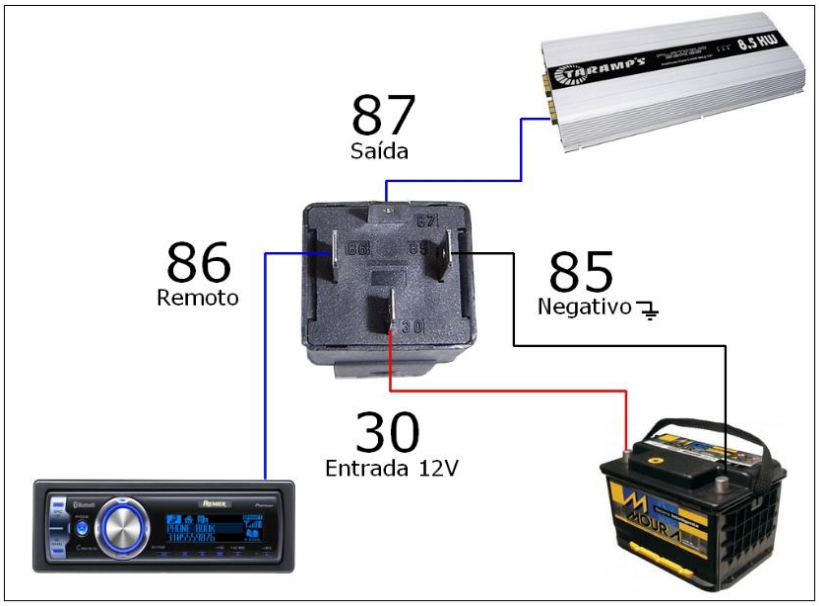


Figura 8 - Contatos do relé

2.7 O LDR (Light Dependent Resistor)

LDR é um componente eletrônico passivo do tipo resistor variável, mais especificamente, é um resistor cuja resistência varia conforme a intensidade da luz que incide sobre ele. Tipicamente, à medida que a intensidade da luz aumenta, a sua resistência diminui. O LDR é construído a partir de material semicondutor com elevada resistência elétrica. Quando a luz que incide sobre o semicondutor tem uma frequência suficiente, os fótons que incidem sobre o semicondutor libertam elétrons para a banda condutora que irão

melhorar a sua condutividade e assim diminuir a resistência. Os resultados típicos para um LDR poderão ser:

- Escuridão: resistência máxima, geralmente mega ohms.
- Luz muito brilhante: resistência mínima, geralmente dezenas de ohms.

Figura 9 – LDR

2.8 Protocolo Modbus

O protocolo Modbus foi desenvolvido em 1979 pela empresa MODICON, Inc., Industrial Automation System, que define um protocolo de aplicação do tipo cliente / servidor entre os dispositivos conectados, independentes do meio físico utilizado. A estrutura de comunicação Modbus atualmente pode ser implementada em:

• TCP/IP sobre *Ethernet*

• Mestre / Escravo em transmissão serial assíncrona sobre vários meios (EIA/TIA 232-E, EIA-422, EIA/TIA-485; fibra óptica, rádio, etc.) e

• Modbus Plus em rede *token pass* de alta velocidade.

2.8.1 Estrutura mestre / escravo

O sistema desenvolvido utiliza o protocolo Modbus na estrutura mestre / escravo, que basicamente é constituída por uma estrutura de mensagens compostas por bytes e que determina como cada dispositivo:

• Identifica seu endereço na rede;
• Reconhece uma mensagem endereçada a ele;
• Determina o tipo de ação a ser executada;
• Obtém toda a informação necessária para executar a ação.

Havendo necessidade de responder ao comando recebido, o dispositivo envia uma mensagem, mesmo ocorrendo erro na ação a ser executada. Apenas um dispositivo (mestre) pode iniciar a comunicação (chamada *query*) e os outros dispositivos (escravos) respondem enviando os dados requisitados pelo mestre ou realizando uma ação solicitada pela *query*. O mestre pode endereçar um escravo individualmente ou iniciar uma mensagem *broadcast* para todos os escravos. Apenas o escravo endereçado retorna uma mensagem (chamada *response*) a uma *query*. *Responses* nunca são retornadas para *query* do tipo *broadcast*.
O protocolo Modbus estabelece o formato para a *query* definindo:
• O endereço do escravo (ou *broadcast*);
• Código da função que define a ação a ser realizada pelo escravo;
• Parâmetros ou dados pertencentes à função;
• Checagem de erro.

As *responses* são construídas também nos mesmos moldes da *query*, obedecendo ao formato correspondente à função enviada pelo mestre, definindo:
• A confirmação da ação realizada;
• Parâmetros ou dados pertencentes à função solicitada;
• Checagem de erro para verificar a integridade da mensagem enviada.

Se o escravo for incapaz de executar a ação pedida, o escravo construirá uma mensagem de erro e irá emiti-la na resposta. Se um erro ocorrer no recebimento da mensagem, o escravo não responde à *query*, pois no erro de comunicação, dois ou mais escravos poderiam responder à *query* ao mesmo tempo.

2.8.2 Formato da mensagem

Na mensagem do protocolo Modbus mestre / escravo, há identificadores de início e fim de *framing*. Este recurso permite aos dispositivos da rede detectarem o início de uma mensagem, ler o campo de endereço e determinar qual dispositivo está sendo endereçado (ou todos, se a mensagem é *broadcast*), e reconhecer quando a mensagem é completada. As mensagens poderão ser detectadas parcialmente podendo gerar erros de acordo com o

resultado. O formato do *framing* Modbus mestre / escravo, como mostra a Tabela 1 é constituído de seis campos: início do *framing*, o endereço do escravo que deve receber a mensagem, o código da função a ser executada, os parâmetros ou dados da função, um campo para checagem de erro e, por último, indicação do fim do *framing*.

Tabela 1 - Estrutura do framing

Início da mensagem	Endereço escravo	Função	Parâmetros ou dados	Checagem de erro	Fim da mensagem

2.8.2.1 Endereço do escravo

A faixa de endereços válidos para os escravos é de 0 a 247. Individualmente os dispositivos escravos são endereçados de 1 a 247. O mestre endereça o escravo colocando o seu endereço no campo de endereço da mensagem e quando o escravo envia uma *response*, ele coloca o seu próprio endereço no campo de endereço da mensagem para que o mestre reconheça qual escravo está respondendo. O endereço 0 é utilizado para acesso tipo *broadcast*, em que todos os escravos reconhecem, mas não realizam *responses*.

2.8.2.2 Função

O código da função varia de 0 a 255, sendo o número 0 reservado para a função de *broadcast*, e os números 128 a 255 reservados para representação de erro. Quando um escravo responde a uma requisição, ele usa o campo de função para indicar execução normal ou erro. Em caso de erro, o sétimo bit do campo função estará em nível 1. Em caso de execução normal, o código da função é deixado intacto e é ecoado na resposta. Por exemplo, se ocorrer algum tipo de erro, ao invés do escravo retornar a função 05h (0000 0101), que foi enviada, o escravo retornará 85h (1000 0101) no campo de função, indicando a ocorrência de erro e juntamente com esta sinalização, o escravo coloca no campo de dados um código que indica o tipo de erro.

2.8.2.3 Parâmetros ou dados

O campo de dados é usado para transportar informações adicionais que o escravo pode usar para execução da ação especificada no campo de função pelo mestre. Este campo é constituído por conjuntos de 2 dígitos hexadecimais na faixa de 00h a FFh. Se nenhum

erro ocorrer, o campo de dados do escravo conterá as informações requisitadas pelo mestre. Se algum erro ocorrer, o campo conterá o código do tipo de erro, como mostra a Tabela 2, juntamente com o sétimo bit do campo função em nível 1.

Tabela 2 - Tipos de erro

Código	Identificação	Significado
01h	Função inválida	A função solicitada pelo mestre não está implementada no escravo.
02h	Sensor ou registrador inválido	O escravo não possui o sensor ou registrador especificado no comando enviado pelo mestre.
03h	Valor de dados inválido	Algum valor no campo de dados é inválido.
04h	Falha no dispositivo	Erro por parte do escravo durante a execução solicitada pelo mestre.
05h	Estado de espera	O escravo reconheceu o comando do mestre, mas o notifica que o mesmo será processado num período maior que o normal.
06h	Dispositivo ocupado	O escravo está atendendo outro comando.
07h	Não reconhecimento	O escravo não conseguiu executar o comando.
08h	Erro de paridade de memória	O escravo detectou erro de paridade.

2.8.2.4 Checagem de erro

Há dois tipos de métodos para checagem de erro para o protocolo Modbus mestre / escravo: o LRC (*Longitudinal Redundancy Check*) e o CRC (*Cyclical Redundancy Check*). Estes métodos dependem do tipo do modo de transmissão utilizado. O sistema desenvolvido utiliza como método para a checagem de erro o LRC, que consiste na soma dos valores dos campos da mensagem, excluindo o início de mensagem, ":" (3Ah), e fim de mensagem, CRLF (0Dh 0Ah). O dispositivo que recebe a mensagem recalcula um LRC durante o recebimento do *framing*, e compara o valor calculado ao valor real que recebeu no campo de LRC. Se os dois valores não forem iguais, há ocorrência de erro.

2.8.3 Modos de transmissão

A estrutura Modbus mestre / escravo pode trabalhar em dois modos de transmissão serial: ASCII (*American Standard Code for Information Interchange*) ou RTU (*Remote Terminal Unit*). O modo de transmissão determina a configuração binária das mensagens que trafegam na rede e como os dados serão empacotados na mensagem e decodificados. Estabelecido o modo de transmissão, devem-se definir os parâmetros da porta serial (*baud rate*, paridade, etc.). O modo e os parâmetros da serial devem ser os mesmos para todos os dispositivos da rede. O modo ASCII foi utilizado como modo de transmissão porque permite intervalos de até 1 segundo entre a transmissão de caracteres consecutivos sem que um erro de *timeout* seja gerado. Entretanto, no modo ASCII, cada byte é enviado como 2 caracteres ASCII usando os caracteres "0 - 9" e "A - F".

2.8.3.1 Modo de transmissão ASCII

Neste modo, as mensagens são iniciadas com o caractere":", valor ASCII 3Ah, e terminadas com os caracteres de retorno – avanço de linha (CR - LF), valores ASCII 0Dh e 0Ah respectivamente. Os caracteres permitidos na transmissão para os outros campos da mensagem são "0" a "9" e "A" a "F" correspondentes aos caracteres ASCII 30h a 39h e 41h a 46h, respectivamente. Quando os dispositivos são configurados para comunicar em modo ASCII, em cada byte da mensagem são enviados dois caracteres ASCII (0 a 9 ou A a F). A principal vantagem deste modo é que ele permite um grande intervalo de tempo entre caracteres sem causar erro. Porém, o tamanho da mensagem em bytes é aumentado significativamente. Como a perda de dados é um fator a se evitar, então este modo de transmissão, tolerante a tempos longos entre bytes de comunicação, foi o modo escolhido para o projeto. O formato de cada byte em modo ASCII é:

Sistema de codificação:

Caractere ASCII 0-9 (30h a 39h), A a F (41h a 46h)

Cada byte é enviado como 2 caracteres ASCII

Endereçamento (1 byte):

0: usado para "broadcast"

1 a 247: usados pelos escravos

Código da função (1 byte):

Estabelece a ação a ser efetuada.

0 a 127: funções

128 a 255: informe de erro na transmissão

Bytes de dados:

Informações adicionais necessárias;

Endereços de memória;

Quantidade de itens transmitidos;

Quantidade de bytes de campo.

Checagem de erro:

Longitudinal Redundancy Check (LRC)

Os dispositivos da rede monitoram continuamente o barramento e quando é detectado o caractere":" (3Ah), tem-se o início da decodificação do próximo campo, que indica para qual dispositivo a mensagem está sendo transmitida. Intervalos de até 1 segundo podem ocorrer entre o envio de caracteres dentro da mesma mensagem. Se ocorrer em intervalos maiores (*timeout*), o dispositivo assume ocorrência de erro.

3. Implementação

A implementação do projeto de casa inteligente consistiu do projeto do hardware, do software e da integração entre eles.

3.1 Hardware

O hardware foi implementado numa estrutura mestre – escravo, em que o microcontrolador mestre (PIC16F877) recebe os comandos do modem e os transmite aos escravos (PIC 16F628). O pinos rx e tx do conector DB9 do modem (Figura 10) são ligados ao MAX232 para que seja feita a conversão de níveis de tensão RS232/TTL do PIC. Cada um dos microcontroladores possui um MAX485 conectado a si e o MAX485 do mestre se comunica com cada MAX485 ligado a um escravo. Cada escravo está ligado a um dispositivo a ser controlado. As atividades remotamente executadas são: acender e apagar uma lâmpada, controlar a luminosidade de uma lâmpada e acionar um alarme. O projeto do hardware começou com um esboço do circuito do escravo, conforme mostra a Figura 11.

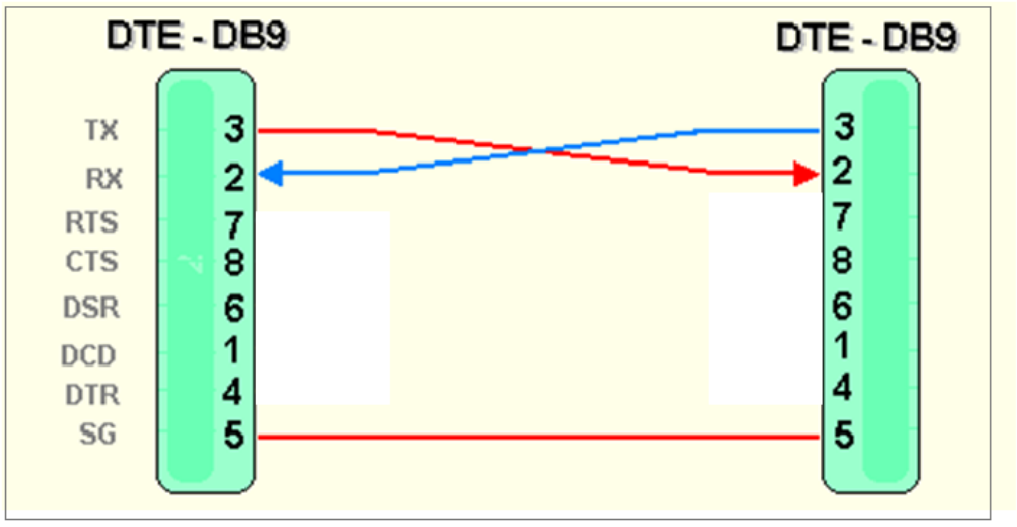


Figura 10 - Pinos do conector DB9

Fonte : (Messias, 2006)

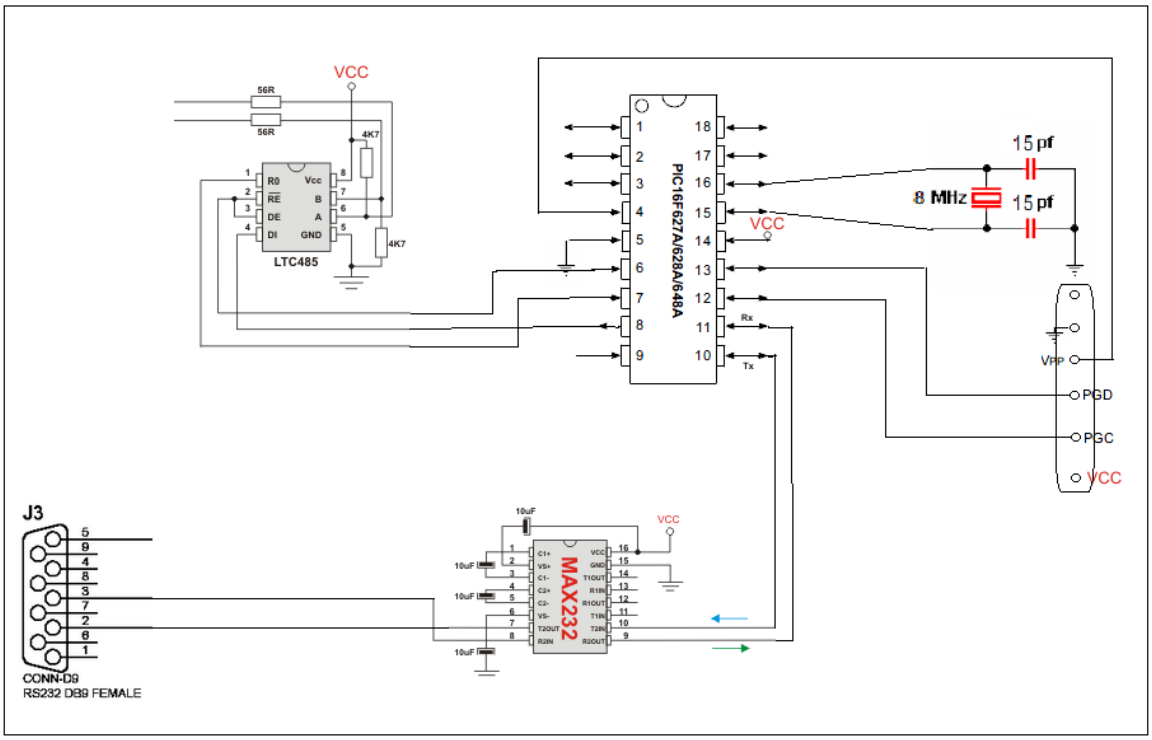


Figura 11 - Projeto do circuito do PIC16F877 (mestre)

Inicialmente, o circuito foi montado em protoboard (Figura 12). Depois, foram confeccionadas placas de circuito impresso, para garantir maior mobilidade e menor interferência de ruídos ao projeto. Para a confecção das placas de circuito impresso na fresadora para circuitos Lpkf (Figura 13), foram feitos os esquemáticos dos circuitos do mestre e dos escravos e, posteriormente, o layout, no software EAGLE 6.4.0.

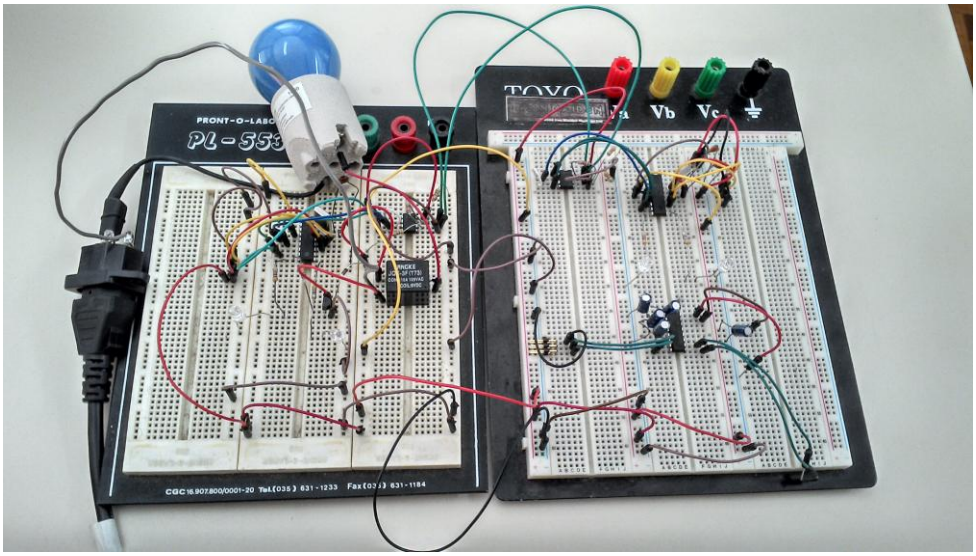


Figura 12 - Montagem do circuito em protoboard

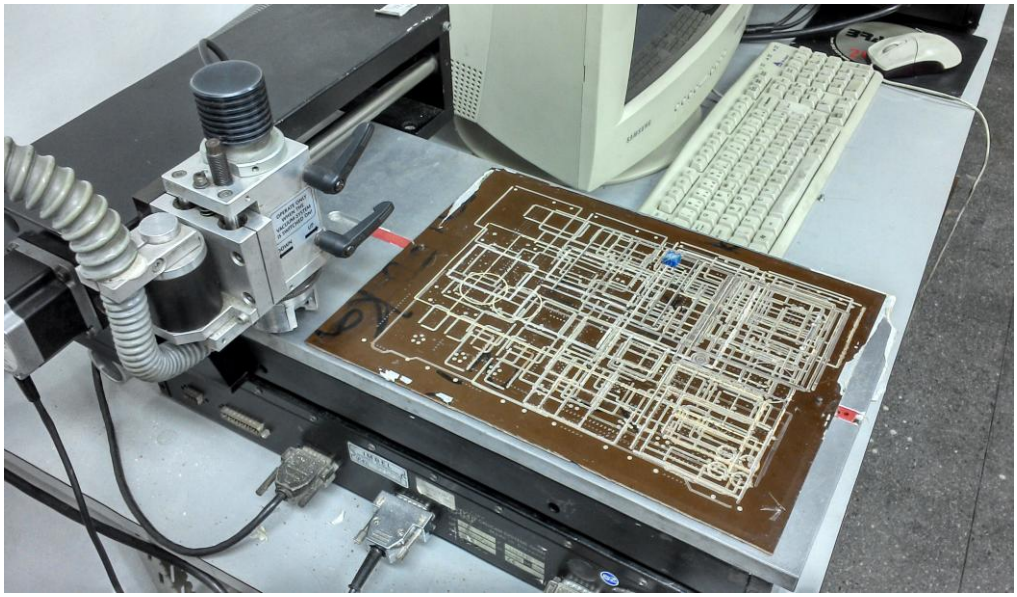


Figura 13 - Fresadora Lpkf

A figura abaixo mostra o esquemático do circuito do microcontrolador mestre, conforme o esboço da Figura 11.

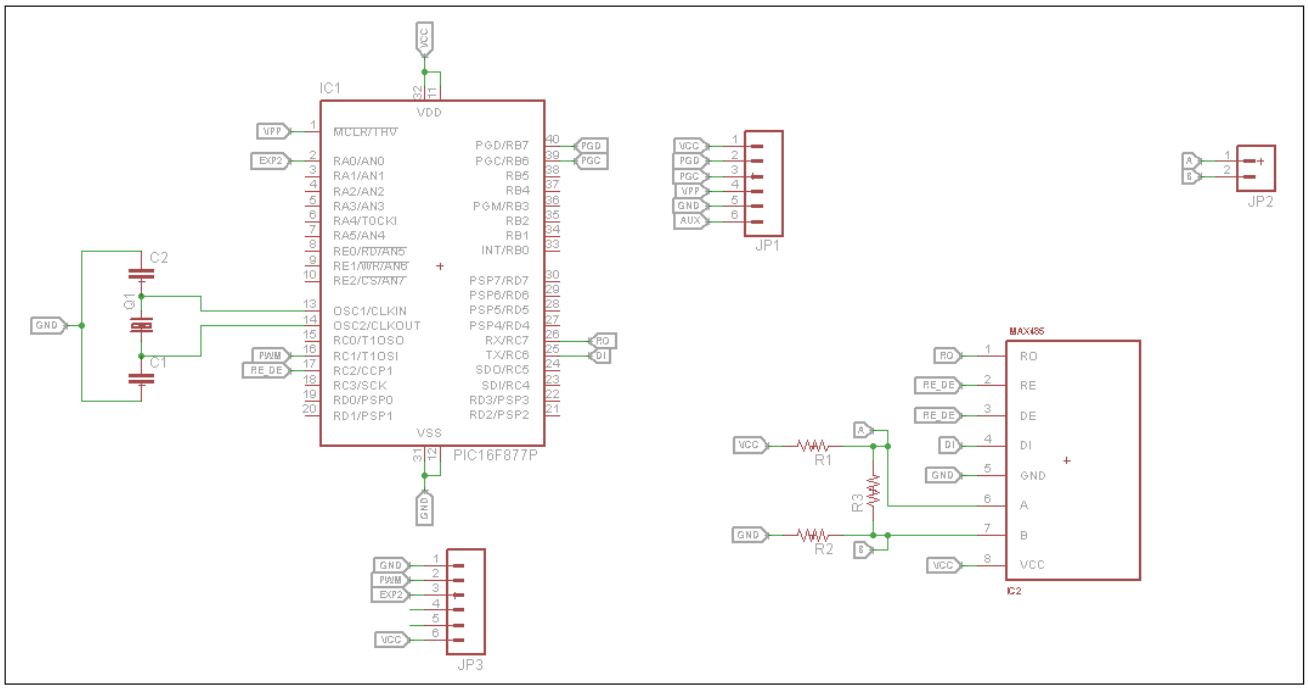


Figura 14 - Esquemático do circuito do PIC 16F877 (mestre)

O circuito da Figura 15 realiza a conversão de níveis de tensões para que diferentes dispositivos possam se comunicar entre si. Converte +12V em zero (GND) e -12V em +5V. O modem que estará conectado à entrada DB9 entende -12V como nível lógico 1 e +12V como nível lógico zero. O microcontrolador entende +5V como nível lógico 1, e zero como nível lógico zero.

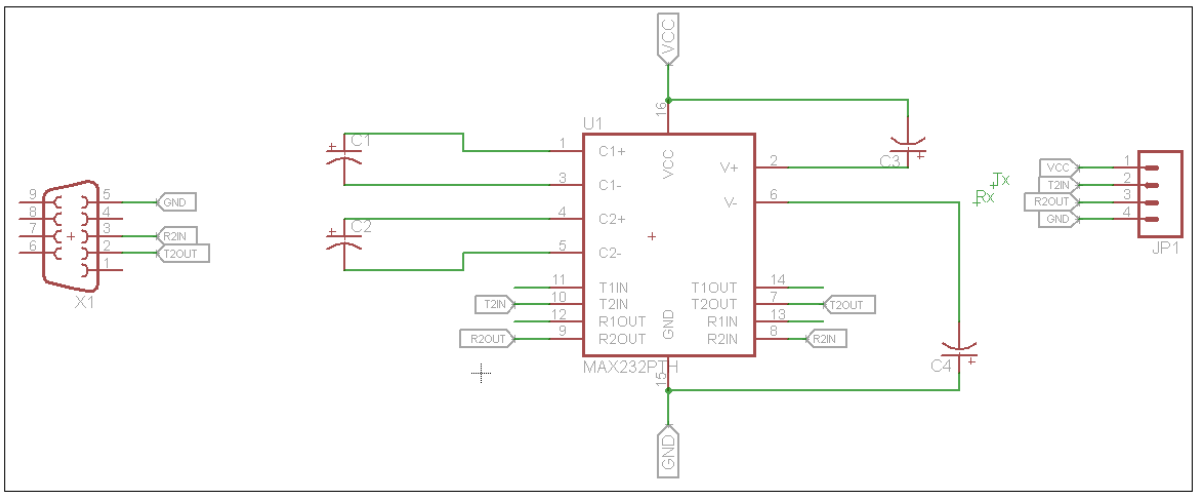


Figura 15 - Esquemático do circuito do MAX232

3.1.1 Dispositivos controlados

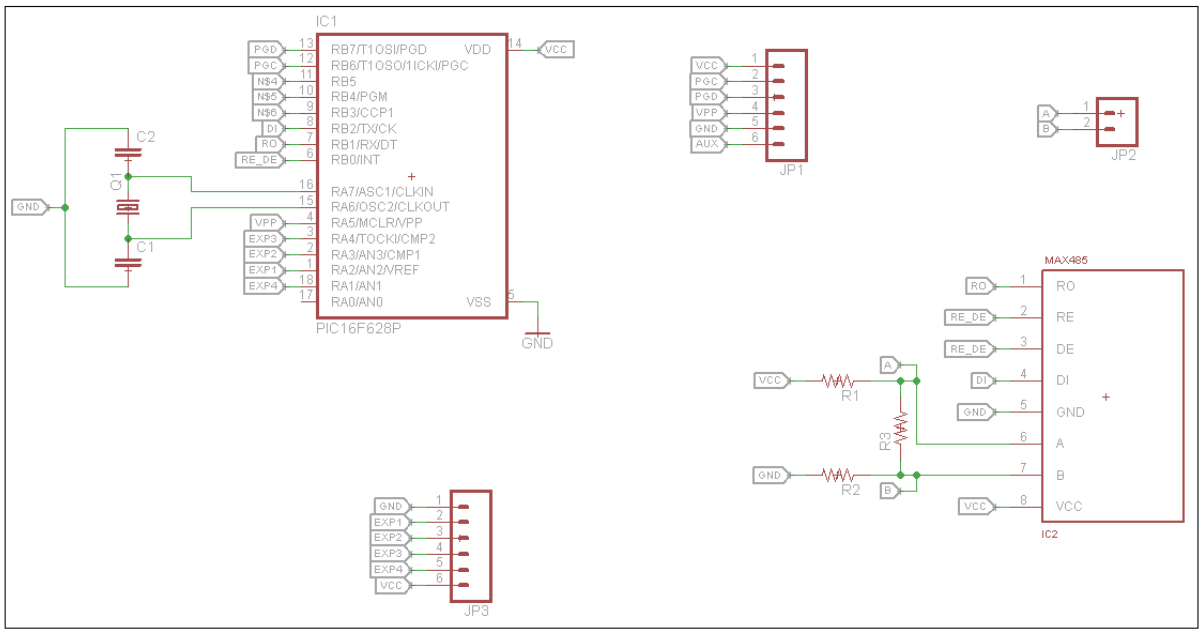


Figura 16 - Esquemático do circuito do PIC 16F628

A Figura 16 mostra o esquemático do microcontrolador escravo, o qual será ligado ao mestre através da rede RS485 e também aos dispositivos controlados. Foram necessárias três unidades desta placa, uma para cada dispositivo.

3.1.1.1 Acionamento do alarme

O circuito funciona da seguinte forma: quando a porta da residência é aberta, o *reed switch* aproxima-se de um campo magnético produzido por um imã fazendo com que suas lâminas entrem em contato, fechando o circuito. O acionamento é feito por uma chave de *pull-up.* O microcontrolador monitora (realiza um *polling*) a tensão na porta em que está conectado o *reed switch.* Enquanto o *reed switch* permanece aberto, a tensão verificada é aproximadamente 5V (nível alto) e quando ocorre o fechamento do dispositivo, esta tensão cai a zero (nível baixo). Nesse momento, o alarme é acionado, ou seja, o microcontrolador disponibiliza nível alto em sua saída, a qual se encontra conectada a um transistor. A corrente de base aumenta, assim como a de coletor, e a tensão entre coletor e base torna-se negativa, o que indica a saturação do transistor. Consequentemente, o alto falante é acionado.

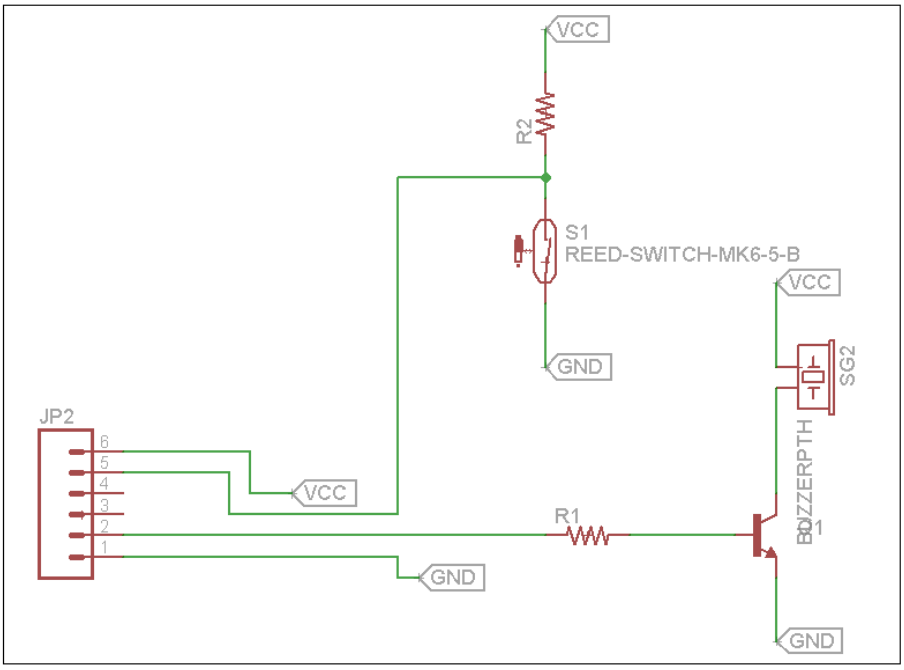


Figura 17 - Esquemático do circuito do alarme

3.1.1.2 Acende / apaga lâmpada

O circuito funciona da seguinte forma: quando o microcontrolador disponibiliza nível alto na porta em que o transistor está conectado, este é saturado, assim o relé fecha devido à indução de um campo magnético que provoca a atração de sua chave. Existe a necessidade do diodo (diodo de roda livre) para que haja um caminho de condução da corrente do relé quando o transistor parar de conduzir, pois o relé tem caráter indutivo nos terminais da bobina.

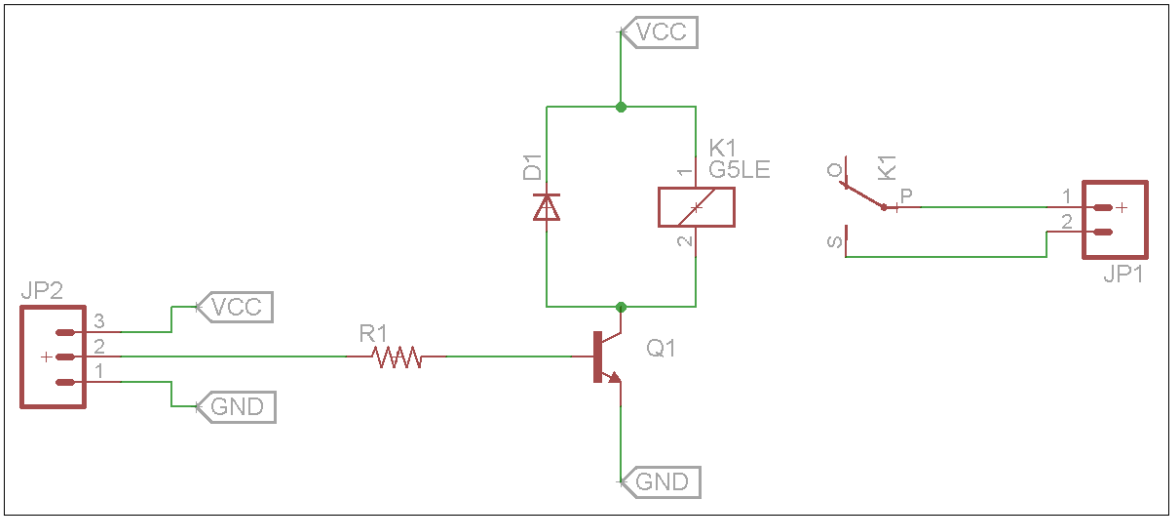


Figura 18 - Esquemático do circuito da lâmpada

3.1.1.3 Controle de luminosidade

No circuito da Figura 19, o microcontrolador, através da expansão, lê o nível de tensão em uma de suas portas analógicas e, de acordo com esse nível, muda o *Duty Cicle* da onda quadrada gerada em sua saída PWM. A mudança no nível de tensão é proporcionada por um simples divisor de tensão, pois o LDR altera o valor de sua resistência de acordo com a luminosidade incidente sobre ele. Assim, pode-se realizar a dimerização de uma lâmpada, por exemplo.

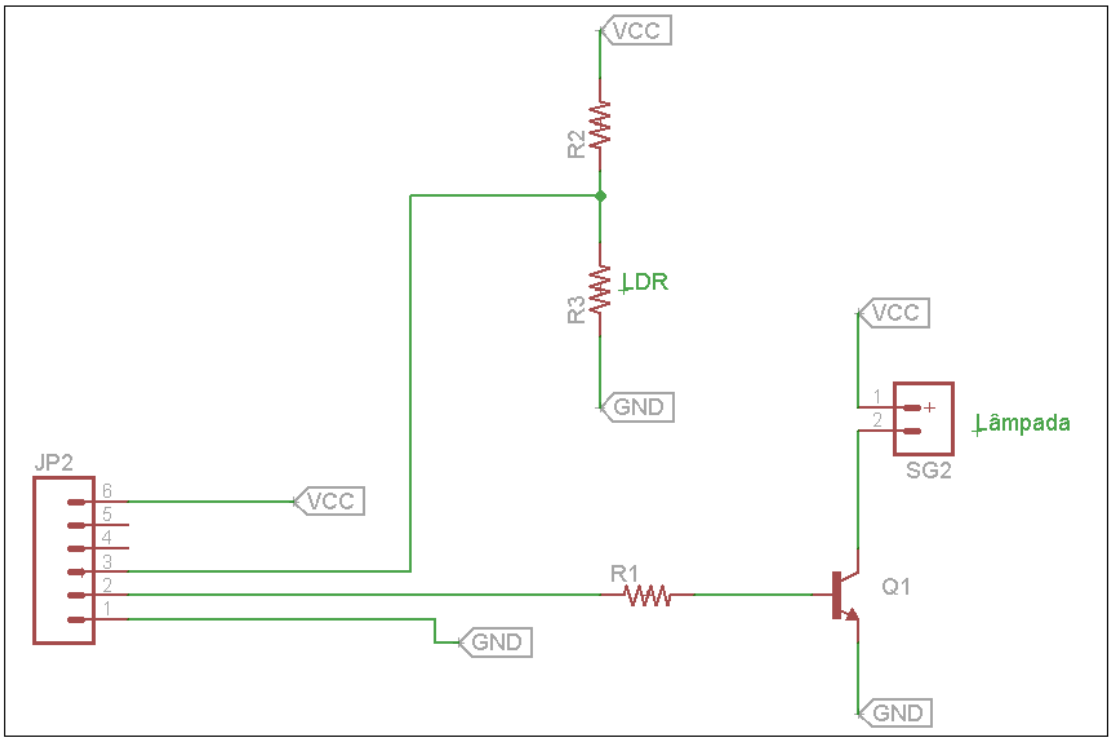


Figura 19 - Esquemático do circuito do LDR

3.2 Software

3.2.1 Comunicação cliente/servidor

O primeiro teste de comunicação (software HDC Terminal) ocorreu entre dois modems, cada um ligado a um computador. O servidor (aquele que fica na casa remotamente controlada), tem um IP fixo e o cliente, em qualquer lugar do mundo, se conecta à rede GPRS, provida por empresa de telefonia celular através do chip inserido no modem. A figura abaixo mostra a configuração da conexão no modo cliente, em que são configurados o tipo de conexão (serial), a porta COM e a velocidade de transferência de bits.

Figura 20 - Configuração da conexão cliente

Após a conexão, são preenchidos os campos “APN”, “User Id” e “Senha”, de acordo com o chip da operadora inserido no modem. É informado também o IP do servidor (fixo), conforme mostra a figura abaixo.

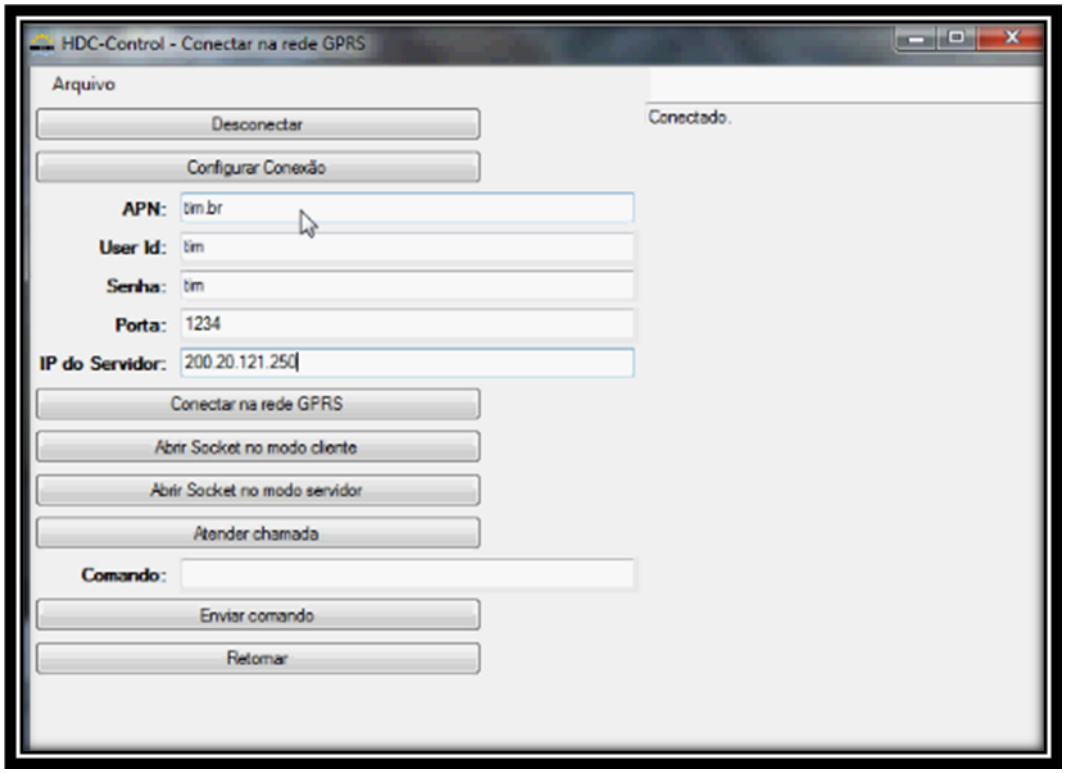


Figura 21 - Conexão cliente

A conexão na rede GPRS (Figura 22) ocorreu através dos seguintes comandos:

AT#SGACT: ativação do contexto PDP (*Packet Data Protocol*)

AT&K0: sem controle de fluxo no RS232

AT+CGDCONT: configurações do contexto PDP

AT#SCFG: configurações do socket

A seguir, o programa fornece o IP que o dispositivo possui na rede GPRS.

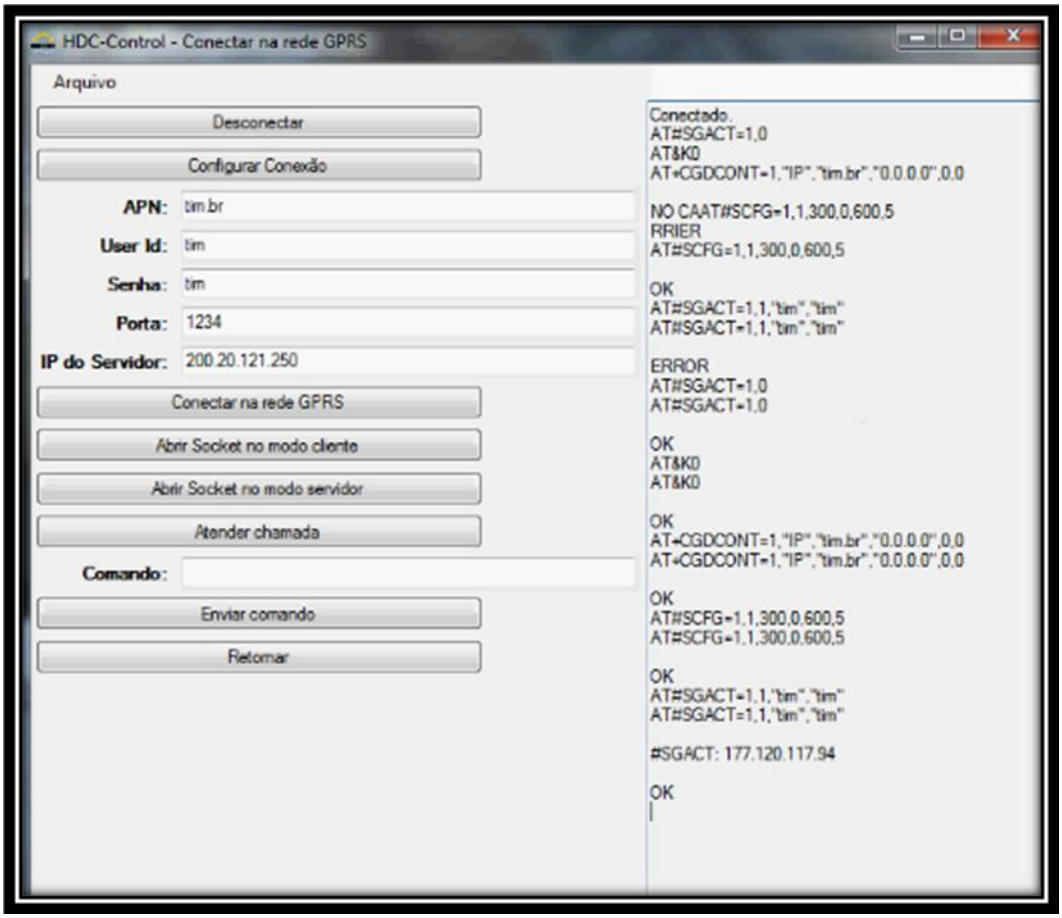


Figura 22 - Conexão na rede GPRS

Após a conexão na rede GPRS, são feitas as configurações do modo servidor. O tipo de conexão é TCP e a porta escolhida foi “1234”. Em seguida, conecta-se o servidor.

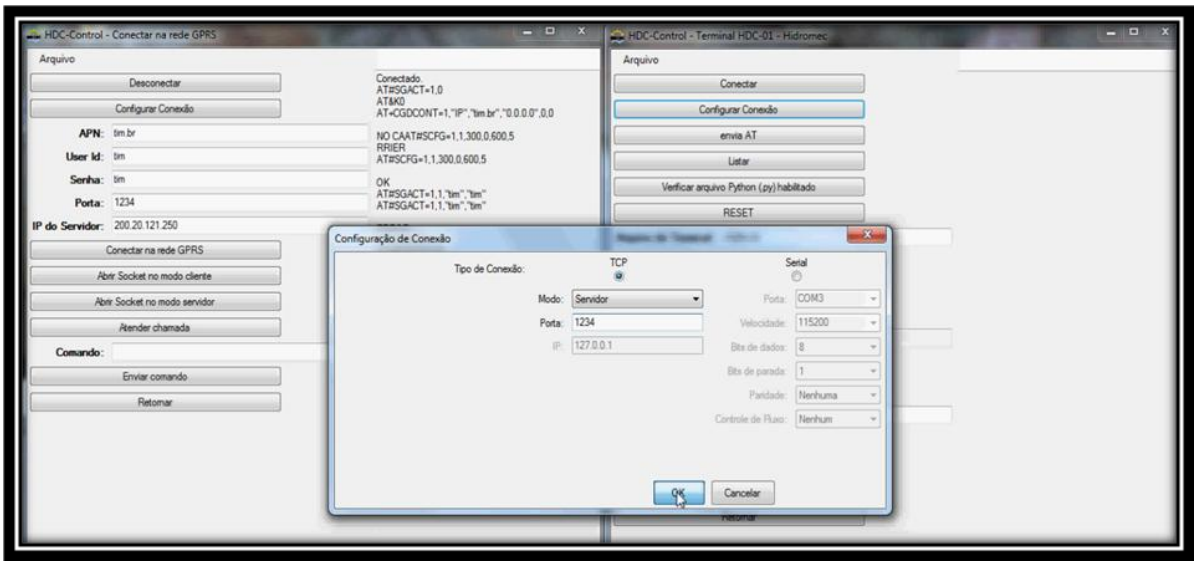


Figura 23 - Configuração da conexão servidor

O próximo passo é a abertura do socket no modo cliente através do seguinte comando:

AT#SD: abre uma conexão remota via socket

Quando o socket é efetivamente aberto, o programa imprime “CONNECT” na tela do cliente. Com isso, o que for escrito no campo “Comando” do cliente, aparece na tela do servidor e vice-versa, pois a comunicação é bidirecional. Assim, quando o cliente enviar um comando de acender uma lâmpada, por exemplo, o servidor poderá responder com uma mensagem de confirmação ou erro.

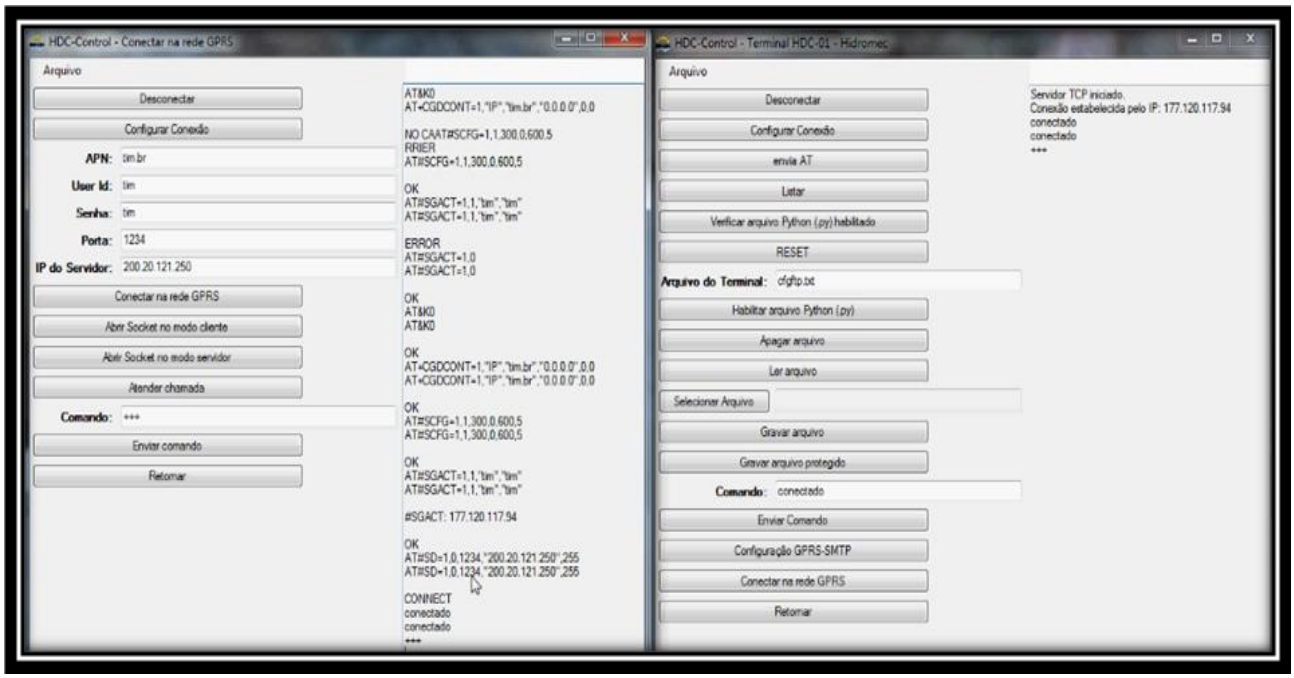


Figura 24 - Socket full duplex

3.2.2 Comunicação modem/ PIC

Após o teste de funcionamento do Modem, prosseguiu-se para a conexão do mesmo ao PIC, de forma que o envio desses comandos para estabelecer a comunicação seja realizada pelo microcontrolador, e não mais por um usuário. O código foi desenvolvido no software "MikroC for PIC". Como as portas de comunicação serial RX/TX do PIC já estavam sendo utilizadas para a aplicação do protocolo Modbus, foi necessário utilizar duas outras portas I/O para transmissão e recepção de dados via RS232. O software em questão possui uma biblioteca "SoftUART" que contém comandos para tornar isso viável. Desses comandos, foi utilizado apenas o "Soft_UART_Init", que inicializa as portas I/O como TX/RX a uma dada baud rate, e o " Soft_UART_Write ", que transmite o caracter desejado. Para o projeto, foi escolhidas as portas RA2 e RA3 como RX e TX, respectivamente. Existe também uma função "Soft_UART_Read ", porém a mesma não estava realizando a recepção dos caracteres de maneira devida, acusando erro de stop bit. Diante disso, foi criada uma rotina própria de recepção (Figura 25). Inicialmente, é feito um while de forma que o programa só dê prosseguimento caso haja o start bit, que deixa a porta RA2 em baixa. Detectado o start bit, é aguardado um tempo de meio bit para que, a partir de então, os bits relativos ao dado fossem lidos no meio do pulso, e não nas bordas, a fim de evitar erro de processamento. A

identificação dos bits ocorre da seguinte maneira: para guardar o valor do bit[i], onde i varia de 0 a 7, é feito, primeiramente, uma máscara para o bit 3 da porta A, relativo à saída RA2. O bit resultante é deslocado dentro do byte que recebeu o valor da máscara em RA2, de forma que ocupe a posição "i". Esse valor fica retido em "BYTE", que, após guardar o valor dos bits de 0 a 7, possui o dado relativo ao caracter como um todo. Após a recepção dos bits, é enviado o stop bit, devendo o mesmo estar em nível alto. O valor do período do bit é calculado a partir da taxa de transmissão. No caso, a baud rate é de 9600 bits/s, o que implica 104 µs por bit.

```
void RX (void)
{
        while(RA2_BIT); //aguarda a resposta do MODEM

                //start bit
        BYTE = 0;
        Delay_us(52); // testar os bits no meio e não nas bordas
        for(i=0;i<8;i++)
        {
                        Delay_us(80);
                        BYTE1 = PORTA&(0b00000100);
                        BYTE1 >>= 2;
                        BYTE1 <<= i;
                        BYTE = BYTE | BYTE1;
        }
        Delay_us(104);
        STOP_BIT = RA2_BIT;

}
```

Figura 25 - Rotina de recepção de caracteres

A Figura 26 mostra um exemplo de caracter recebido pelo PIC na comunicação serial. O gráfico foi medido pelo osciloscópio na porta RA2 do PIC. A escala temporal é de 100 µs, portanto há um bit por quadro. Percebe-se, no início, a mudança do nível alto (5.40V) para nível baixo (1V), caracterizando o start bit. A partir de então, tem-se a sequência 11010000, seguida por um 1 (stop bit), pois nessa transmissão optou-se por não trabalhar com bit de paridade. Os bits são transmitidos do menos significativo para o mais signigicativo, ou seja, esse caracter é, em binário, 00001011. Consultando-se a tabela

ASCII, tem-se que esse byte se refere ao caracter 'VT', responsável pelo salto para a próxima linha.

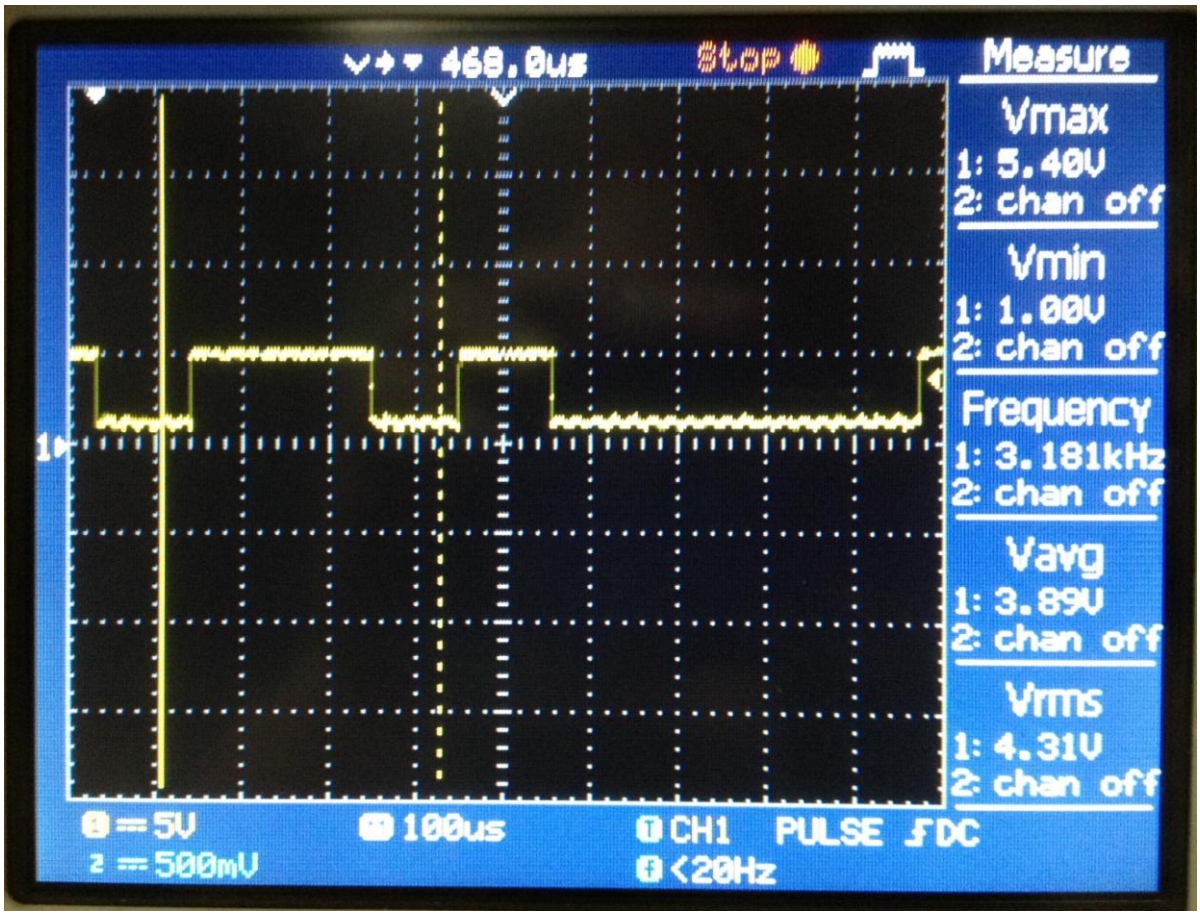


Figura 26 - Caracter 'VT'

A Figura 27 mostra a parte do código relativa ao envio de comandos ao Modem através da função Soft_UART_Write. Essa função só permite enviar caracter por caracter. Os caracteres '\n' e '\r' são necessários para que o Modem reconheça como término do envio de um comando. O "for" realiza a leitura dos bytes recebidos pelo PIC, sendo necessário ler apenas os três primeiros caracteres, que são '\n','\r' e o primeiro caracter do tipo de resposta que o modem envia ao PIC. Por exemplo, caso tenha dado erro, o Modem responde: '\n', \r' e 'ERROR'. Caso não haja erro, o Modem responde '\n', \r' e 'OK'. Como o desejado é 'OK', é feito um loop em que, caso o terceiro bit lido não seja 'O', o PIC envia o comando novamente ao Modem, até responder como resposta 'OK', para, então, dar prosseguimento nos demais comandos. É importante ressaltar que inicialmente deve ser desabilitada a função de Eco do Modem, responsável por repetir o comando recebido e apenas após isso retornar a resposta. O eco dificulta a leitura do caracter, em especial porque o Modem inicia o retorno

do comando antes do término do envio desse por parte do PIC. para tal, deve ser enviado o comando "ATE".

```
//PRIMEIRA PARTE: ENVIO DO COMANDO AT (CONFRIRMA SE O MODEM ESTÁ 'ESCUTANDO')

do
{
    Delay_ms(500);
    Soft_UART_Write('A');
    Soft_UART_Write('T');
    Soft_UART_Write('\r');
    Soft_UART_Write('\n');


    for(j=0;j<3;j++)
    {
     RX();
     recebido[j] = BYTE;
    }


}while(!(recebido[2]=='O'));
```

Figura 27 - Envio de comandos ao Modem

O código utiliza os mesmos comandos do teste feito com o Modem descrito inicialmente para estabelecimento da conexão. Portanto, foi dividido em oito partes, das quais sete são para o envio de cada comando especificado, e a oitava é para a comunicação bilateral (Figura 28). No código, foi feito um teste para verificar se a comunicação ocorre em ambos os sentidos. Caso o servidor envie '1', o PIC deveria retornar 'LED ACESO', e caso envie '2' retorna 'LED APAGADO'. Esse teste de confirmação pode facilmente se tornar um comando. Por exemplo, caso o servidor envie '1', o circuito pode ativar, através do ModBus, um dos dispositivos, além de retornar a confirmação.

Após realização de testes, percebeu-se que a comunicação ocorreu com sucesso. O código completo encontra-se no Anexo.

```
// OITAVA PARTE: COMUNICAÇÃO BILATERAL
   while(1)//aqui deve ser colocada uma flag que testa a comunicação
   {
    RX();
    if(BYTE=='1')
    {
            Soft_UART_Write('L');// apenas confirma a ação... futuramente, haverá algo como DADO = BYTE;
            Soft_UART_Write('E');
            Soft_UART_Write('D');
            Soft_UART_Write(' ');
            Soft_UART_Write('A');
            Soft_UART_Write('C');
            Soft_UART_Write('E');
            Soft_UART_Write('S');
            Soft_UART_Write('O');
            Soft_UART_Write('\r');
            Soft_UART_Write('\n');
     }
     if(BYTE=='2')
    {
            Soft_UART_Write('L');// apenas confirma a ação... futuramente, haverá algo como DADO = BYTE;
            Soft_UART_Write('E');
            Soft_UART_Write('D');
            Soft_UART_Write(' ');
            Soft_UART_Write('A');
            Soft_UART_Write('P');
            Soft_UART_Write('A');
            Soft_UART_Write('G');
            Soft_UART_Write('A');
            Soft_UART_Write('D');
            Soft_UART_Write('O');
            Soft_UART_Write('\r');
            Soft_UART_Write('\n');
     }     //deve ser feito um "if(porta do sensor==1) Soft_UART_Write('alerta')";
   }
}
```

Figura 28 - Comunicação bidirecional

4. Funcionamento do projeto

A Figura 29 mostra a montagem da comunicação do modem com o PIC 16F628, composta pela placa do MAX232, modem, placa do PIC 16F628 e fonte de 12V para o modem. O comportamento do circuito foi descrito no item anterior.

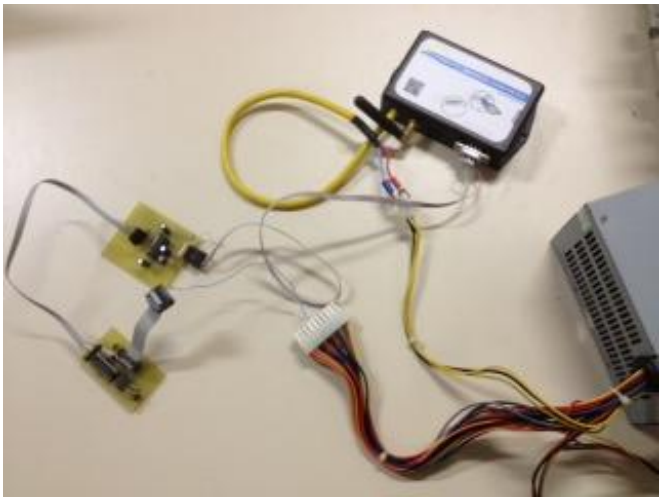


Figura 29 - Comunicação modem/ PIC

A Figura 30 mostra a placa do PIC 16F628, ligada à do reed switch pela rede RS485. O buzzer está conectado à placa do reed switch.

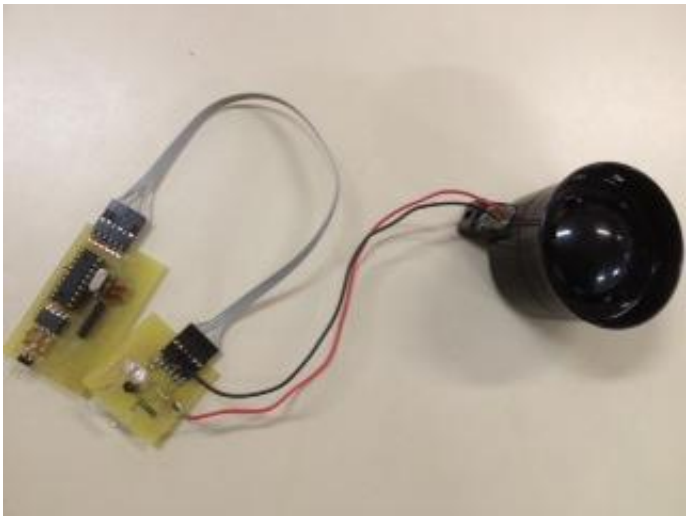


Figura 30 - Montagem do circuito do alarme

A Figura 31 mostra a placa do PIC 16F628, conectada à do reed switch pela rede RS485. A lâmpada está ligada à placa do relé.

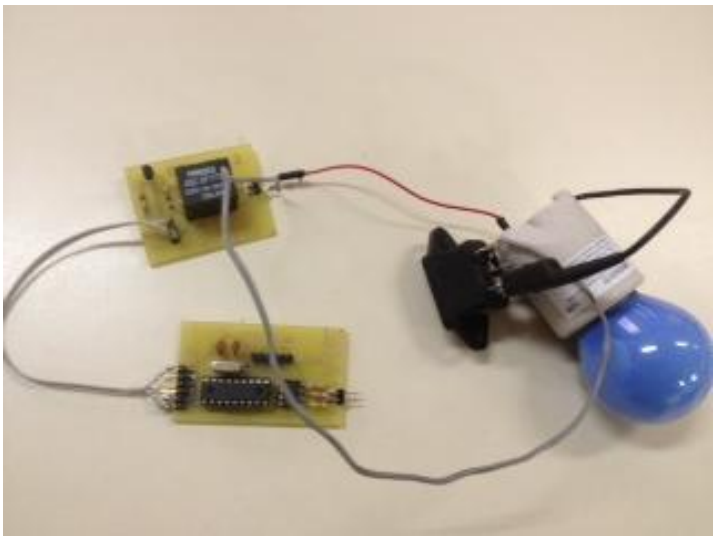


Figura 31 - Circuito de acionamento da lâmpada

Os circuitos apresentados acima funcionaram separadamente. Não foi possível a integração entre o circuito de comunicação e o de acionamento de cada um dos dispositivos.

5. Dificuldades e limitações do projeto

- Por uma questão de tempo, não foi possível a implementação, na prática, do circuito de controle de luminosidade.
- O programador do PIC nem sempre atualiza o programa após a compilação, o que gera a desconfiança de que o programa continua apresentando problemas.
- A alimentação do circuito do relé, do reed switch e da comunicação serial (entre modem e PIC), enquanto fornecida pelo computador (corrente pequena), através do programador do PIC, gerava um reset no programa. Quando a alimentação passou a ser feita por uma fonte externa (corrente maior), o problema cessou.
- O pino MCLR/ do PIC deve ser desativado no mikroC Pro para que o programa não sofra reset.
- O reed switch, por ser de vidro, é muito frágil.
- O modem, quando se desconecta da rede GPRS, só envia essa informação após um delay, o que impede que o servidor envie dados ou mesmo comandos.
- O programa Hyperterminal somente funciona como cliente.
- A operadora Claro não permite que o modem opere como cliente.
- Como o projeto não foi, de fato, implementado em uma casa, não foram encontradas algumas dificuldades reais, como instalações elétricas (altas potências que o circuito poderia não suportar) e uma possível interferência de ruídos.
- A versão do mikroc Pro para estudantes tem capacidade de 2Kbytes, insuficiente para a aplicação proposta no projeto.
- Não foi desenvolvida a interface do usuário.
- Não foi realizada a integração entre o circuito de comunicação e o de cada um dos dispositivos.

6. Conclusão

A domótica propõe uma quebra de paradigma nos costumes da sociedade atual, trazendo mais flexibilidade, conforto e segurança para as residências, de modo que os usuários que utilizam a automação possam controlar remotamente luzes e equipamentos de sua casa, através de um computador ou telefone celular.

Neste trabalho, foi apresentada a tecnologia GPRS para automação residencial. Foi realizado também, um estudo sobre o modem GSM/GPRS e suas funcionalidades. O auge do trabalho foi o desenvolvimento do *software* de comunicação entre o modem e o microcontrolador PIC.

De acordo com o projeto, o modem GSM/GPRS tem a função de um telefone celular, onde ele tem a opção de ser configurado e programado, de acordo com a especificação do usuário. Assim, qualquer pessoa pode realizar a monitoração e controle de sua residência, em qualquer parte do mundo a um custo relativamente baixo, desde que o usuário possua o IP com o qual quer se conectar.

Segundo a arquitetura proposta no projeto, um usuário pode enviar comandos remotamente, através de um computador, a um modem GSM/GPRS instalado em sua residência, o qual se comunica a um microcontrolador mestre, repassando comandos a um escravo, dependendo da aplicação desejada.

Em relação ao sistema desenvolvido, o software de comunicação entre o modem e o microcontrolador PIC16F628 apresentou bom desempenho, bem como cada uma das aplicações.

Para finalizar, o projeto agrega bastante conhecimento teórico e prático, além da possibilidade de se aplicar em diferentes áreas já citadas no trabalho. O ponto principal do projeto se refere à aquisição remota de dados, que é uma proposta de aplicação de baixo custo e acessível, independente da aplicação.

7. Referências Bibliográficas

DA SILVA , Ivan Vieira Ferreira. DE CARVALHO, Sérgio Silva. **Automação residencial de baixo custo: um protótipo com acesso web.** Disponível em: <http://semanaacademica.org.br/system/files/artigos/revistasemanaacademicadomoticaii.pdf >. Acesso em: 19 fevereiro 2013.

FERNANDES, Bruno Coutinho. **Construção de um sistema eletrônico de monitoramento de consumo de água residencial.** Vitória, 2007. Disponível em: <http://www2.ele.ufes.br/~projgrad/documentos/PG2006_1/brunocoutinhofernandes.pdf>. Acesso em: 14 março 2013.

JUCÁ, Sandro. **Aplicações práticas de Eletrônica e microcontroladores em sistemas computacionais.** Disponível em: < http://www.maracanau.ifce.edu.br/~sandrojuca/Microcontroladores1.pdf>. Acesso em: 19 fevereiro 2013.

DE OLIVEIRA, Victor Hugo Freitas. **Desenvolvimento de um sistema de telemetria remota.** Natal, 2009. Disponível em: < http://www.engcomp.ufrn.br/publicacoes/TCC-2009-1-1.pdf>. Acesso em: 5 março 2013.

DOS SANTOS, Ricardo Antônio Silva. **Domótica via dispositivos móveis.** Ouro Preto, 2010. Disponível em: < http://www.em.ufop.br/cecau/monografias/2010/Ricardo%20A.%20S.%20Santos.pdf>. Acesso em: 3 março 2013.

Telit, AT Commands Reference Guide.

Telit, Easy GPRS User Guide.

Microship, 2003. Data Sheet PIC16F87XA.

Microship, 2009. Data Sheet PIC16F627A/628A/648.

MAXIM. +5V-Powered, Multichannel RS-232 Drivers/Receivers.

MAXIM. Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers.

LDR. Disponível em: <http://pt.wikipedia.org/wiki/LDR>. Acesso em: 16 junho 2013.

Reed-switch. Disponível em: <http://www.if.ufrgs.br/mpef/mef004/20061/Cesar/SENSORES-Reed-switch.html>. Acesso em: 16 junho 2013.

Relé. Disponível em: <http://www.electronica-pt.com/index.php/content/view/179/37/>. Acesso em: 16 junho 2013.

ANEXO

1. Código do programa da conexão do Modem com a rede GRPS/GSM por meio do PIC. O compilador utilizado foi o Mikroc Pro e algumas funções são intrínsecas à sua biblioteca:

```
unsigned char i, j, error, BYTE, BYTE1, STOP_BIT=1;        // Auxiliary variables
char recebido [20];
long int cont;


void Rx();

void main()
{
//desabilitar interrupções ou outras funções das portas I/O usadas na comunicação
  CCP1IE_BIT=0;
  GIE_BIT=0;
  VROE_BIT=0;
  CMCON = 15;  // Desliga os comparadores das portas RA0 a RA3, configurando-as como I/O comuns

  error = Soft_UART_Init(&PORTA, 2, 3, 9600, 0); // Inicializa Soft UART na baud rate 9600 bps, 2 rx, 3tx
  TRISA2_BIT = 1;//RX:input
  TRISA3_BIT = 0;//TX: output

  Delay_ms(2000);//aguarda a inicialização do MODEM

 //PARTE ZERO: DESATIVAR O ECO
  //do
   //{
     //Delay_ms(50);
       Soft_UART_Write('A');
       Soft_UART_Write('T');
       Soft_UART_Write('E');
      // Soft_UART_Write('=');
       //Soft_UART_Write('0');
       Soft_UART_Write('\r');
       Soft_UART_Write('\n');
       Delay_ms(200);


   //PRIMEIRA PARTE: ENVIO DO COMANDO AT (CONFRIRMA SE O MODEM ESTÁ 'ESCUTANDO')

   do
   {
      Delay_ms(500);
```

```
    Soft_UART_Write('A');
    Soft_UART_Write('T');
    Soft_UART_Write('\r');
    Soft_UART_Write('\n');


    for(j=0;j<3;j++)
    {
     RX();
     recebido[j] = BYTE;
    }


  }while(!(recebido[2]=='O')); // se forem diferentes, retorna um valor != de zero... ou seja, não recebeu OK do modem.. deve mandar novamente o comando "#AT"

  //SEGUNDA PARTE: ENVIO DO COMANDO AT#SGACT=1,0\r\n


  do
  {
   Delay_ms(500); //aguarda um tempo antes de passar para a próxima etapa
   Soft_UART_Write('A');
   Soft_UART_Write('T');
   Soft_UART_Write('#');
   Soft_UART_Write('S');
   Soft_UART_Write('G');
   Soft_UART_Write('A');
   Soft_UART_Write('C');
   Soft_UART_Write('T');
   Soft_UART_Write('=');
   Soft_UART_Write('1');
   Soft_UART_Write(',');
   Soft_UART_Write('0');
   Soft_UART_Write('\r');
   Soft_UART_Write('\n');

    for(j=0;j<3;j++)
    {
     RX();
     recebido[j] = BYTE;
     }

  }while(!(recebido[2]=='O')); // se forem diferentes, retorna um valor != de zero... ou seja, não recebeu OK do modem.. deve mandar novamente o comando "#AT"



  //TERCEIRA PARTE: ENVIO DO COMANDO AT#KO\r\n
```

```
  do
  {
   Delay_ms(500);
   Soft_UART_Write('A');
   Soft_UART_Write('T');
   Soft_UART_Write('&');
   Soft_UART_Write('K');
   Soft_UART_Write('0');
   Soft_UART_Write('\r');
   Soft_UART_Write('\n');
   for(j=0;j<3;j++)
      {
       RX();
       recebido[j] = BYTE;
       }

  }while(!(recebido[2]=='O')); // se forem diferentes, retorna um valor != de zero... ou seja, não recebeu OK do modem.. deve mandar novamente o comando "#AT"

//QUARTA PARTE: ENVIO DO COMANDO AT+CGDCONT=1,"IP","%APNDIR","0.0.0.0",0,0\r\n

  do
  {
   Delay_ms(500);
   Soft_UART_Write('A');  Soft_UART_Write('T');  Soft_UART_Write('+');
Soft_UART_Write('C');  Soft_UART_Write('G');  Soft_UART_Write('D');
Soft_UART_Write('C');  Soft_UART_Write('O');  Soft_UART_Write('N');
Soft_UART_Write('T');
   Soft_UART_Write('=');
   Soft_UART_Write('1');
   Soft_UART_Write(',');
   Soft_UART_Write('"');
   Soft_UART_Write('I');
   Soft_UART_Write('P');
   Soft_UART_Write('"');
   Soft_UART_Write(',');
   Soft_UART_Write('"');
   Soft_UART_Write('t');
   Soft_UART_Write('i');
   Soft_UART_Write('m');
   Soft_UART_Write('.');
   Soft_UART_Write('b');
   Soft_UART_Write('r');
   Soft_UART_Write('"');
   Soft_UART_Write(',');
   Soft_UART_Write('"');
   Soft_UART_Write('0');
   Soft_UART_Write('.');
```

```
    Soft_UART_Write('0');
    Soft_UART_Write('.');
    Soft_UART_Write('0');
    Soft_UART_Write('.');
    Soft_UART_Write('0');
    Soft_UART_Write('"');
    Soft_UART_Write(',');
    Soft_UART_Write('0');
    Soft_UART_Write(',');
    Soft_UART_Write('0');
    Soft_UART_Write('\r');
    Soft_UART_Write('\n');

   for(j=0;j<3;j++)
       {
        RX();
        recebido[j] = BYTE;
        }

   }while(!(recebido[2]=='O')); // se forem diferentes, retorna um valor != de zero... ou seja, não recebeu OK do modem.. deve mandar novamente o comando "#AT"

//QUINTA PARTE: ENVIO DO COMANDO AT#SCFG=1,1,300,0,600,5\r\n

   do
   {
    Delay_ms(500);
    Soft_UART_Write('A');
    Soft_UART_Write('T');
    Soft_UART_Write('#');
    Soft_UART_Write('S');
    Soft_UART_Write('C');
    Soft_UART_Write('F');
    Soft_UART_Write('G');
    Soft_UART_Write('=');
    Soft_UART_Write('1');
    Soft_UART_Write(',');
    Soft_UART_Write('1');
    Soft_UART_Write(',');
    Soft_UART_Write('3');
    Soft_UART_Write('0');
    Soft_UART_Write('0');
    Soft_UART_Write(',');
    Soft_UART_Write('0');
    Soft_UART_Write(',');
    Soft_UART_Write('6');
    Soft_UART_Write('0');
    Soft_UART_Write('0');
    Soft_UART_Write(',');
```

```
  Soft_UART_Write('5');
  Soft_UART_Write('\r');
  Soft_UART_Write('\n');

  for(j=0;j<3;j++)
     {
      RX();
      recebido[j] = BYTE;
      }

  }while(!(recebido[2]=='O')); // se forem diferentes, retorna um valor != de zero... ou seja, não recebeu OK do modem.. deve mandar novamente o comando "#AT"

//SEXTA PARTE: ENVIO DO COMANDO AT#SGACT=1,1,"tim","tim"\r\n

  do
  {
   Delay_ms(1000);
   Soft_UART_Write('A');
   Soft_UART_Write('T');
   Soft_UART_Write('#');
   Soft_UART_Write('S');
   Soft_UART_Write('G');
   Soft_UART_Write('A');
   Soft_UART_Write('C');
   Soft_UART_Write('T');
   Soft_UART_Write('=');
   Soft_UART_Write('1');
   Soft_UART_Write(',');
   Soft_UART_Write('1');
   Soft_UART_Write(',');
   Soft_UART_Write('"');
   Soft_UART_Write('t');
   Soft_UART_Write('i');
   Soft_UART_Write('m');
   Soft_UART_Write('"');
   Soft_UART_Write(',');
   Soft_UART_Write('"');
   Soft_UART_Write('t');
   Soft_UART_Write('i');
   Soft_UART_Write('m');
   Soft_UART_Write('"');
   Soft_UART_Write('\r');
   Soft_UART_Write('\n');


   for(j=0;j<3;j++)
     {
      RX();
```

```
        recebido[j] = BYTE;
        }


   }while(!(recebido[2]=='#')); // se forem diferentes, retorna um valor != de zero... ou seja, não recebeu #SCAGCT + IP + OK do modem.. deve mandar novamente o comando "#AT"

// SÉTIMA PARTE: ENVIO DO COMANDO AT#SD=1,0,%PORTADIR,"%IPDEST",255\r\n ... põe no modo cliente

   do
   {
    Delay_ms(1000);
    Soft_UART_Write('A');
    Soft_UART_Write('T');
    Soft_UART_Write('#');
    Soft_UART_Write('S');
    Soft_UART_Write('D');
    Soft_UART_Write('=');
    Soft_UART_Write('1');
    Soft_UART_Write(',');
    Soft_UART_Write('0');
    Soft_UART_Write(',');
    Soft_UART_Write('1');
    Soft_UART_Write('2');
    Soft_UART_Write('3');
    Soft_UART_Write('4');
    Soft_UART_Write(',');
    Soft_UART_Write('"');
    Soft_UART_Write('2');  //ip: 200.20.121.250, msk:   255
    Soft_UART_Write('0');
    Soft_UART_Write('0');
    Soft_UART_Write('.');
    Soft_UART_Write('2');
    Soft_UART_Write('0');
    Soft_UART_Write('.');
    Soft_UART_Write('1');
    Soft_UART_Write('2');
    Soft_UART_Write('1');
    Soft_UART_Write('.');
    Soft_UART_Write('2');
    Soft_UART_Write('5');
    Soft_UART_Write('0');
    Soft_UART_Write('"');
    Soft_UART_Write(',');
    Soft_UART_Write('2');
    Soft_UART_Write('5');
    Soft_UART_Write('5');
    Soft_UART_Write('\r');
```

```
   Soft_UART_Write('\n');


   for(j=0;j<3;j++)
      {
       RX();
       recebido[j] = BYTE;
       }


  }while(!(recebido[2]=='C')); // se forem diferentes, retorna um valor != de zero... ou seja, não recebeu CONNECTED do modem.. deve mandar novamente o comando "#AT"

          Soft_UART_Write('C');// confirma a conexão
          Soft_UART_Write('O');
          Soft_UART_Write('N');
          Soft_UART_Write('N');
          Soft_UART_Write('E');
          Soft_UART_Write('C');
          Soft_UART_Write('T');
          Soft_UART_Write('E');
          Soft_UART_Write('D');
          Soft_UART_Write('\r');
          Soft_UART_Write('\n');

// OITAVA PARTE: COMUNICAÇÃO BILATERAL
  while(1)//aqui deve ser colocada uma flag que testa a comunicação.... se cair, faz-se um salto para o início do processo (etapa 1) para que se possa obret um novo IP e nova conexão
  {
   RX();
   if(BYTE=='1')
   {
          Soft_UART_Write('L');// apenas confirma a ação... futuramente, haverá algo como DADO = BYTE; e esse dado será enviado ao respectivo escravo para executar a ação... pode-se chamar uma função que o fará
          Soft_UART_Write('E');
          Soft_UART_Write('D');
          Soft_UART_Write(' ');
          Soft_UART_Write('A');
          Soft_UART_Write('C');
          Soft_UART_Write('E');
          Soft_UART_Write('S');
          Soft_UART_Write('O');
          Soft_UART_Write('\r');
          Soft_UART_Write('\n');
    }
    if(BYTE=='2')
   {
```

```
        Soft_UART_Write('L');// apenas confirma a ação... futuramente, haverá algo como DADO = BYTE; e esse dado será enviado ao respectivo escravo para executar a ação... pode-se chamar uma função que o fará
        Soft_UART_Write('E');
        Soft_UART_Write('D');
        Soft_UART_Write(' ');
        Soft_UART_Write('A');
        Soft_UART_Write('P');
        Soft_UART_Write('A');
        Soft_UART_Write('G');
        Soft_UART_Write('A');
        Soft_UART_Write('D');
        Soft_UART_Write('O');
        Soft_UART_Write('\r');
        Soft_UART_Write('\n');
    }   //deve ser feito um "if(porta do sensor==1) Soft_UART_Write('alerta')";
  }
}

void RX (void)
{
      while(RA2_BIT); //aguarda a resposta do MODEM

            //start bit
      BYTE = 0;
      Delay_us(52); // testar os bits no meio e não nas bordas
      for(i=0;i<8;i++)
      {
            Delay_us(80);
            BYTE1 = PORTA&(0b00000100);
            BYTE1 >>= 2;
            BYTE1 <<= i;
            BYTE = BYTE | BYTE1;
      }
      Delay_us(104);
      STOP_BIT = RA2_BIT;

}
```

2. Código do programa que utiliza o protocolo ModBus para o envio de mensagens do mestre para o escravo e vice-versa, via rede RS485.

Mestre

```
char dat[10];
char ordem=0;

sbit  rs485_rxtx_pin  at RB0_bit;
sbit  rs485_rxtx_pin_direction at TRISB0_bit;

void interrupt() {
  RS485Master_Receive(dat);
}

void main()
{
  UART1_Init(9600);
  Delay_ms(100);

  RS485Master_Init();
  dat[0] = 0xFF;
  dat[1] = 0x00;
  dat[2] = 0x00;
  dat[4] = 0;
  dat[5] = 0;
  dat[6] = 0;
```

```
RCIE_bit = 1;
TXIE_bit = 0;
PEIE_bit = 1;
GIE_bit = 1;

while (1)
{
 if(ordem==1)
  {
    dat[0]= 0x01;
    RS485Master_Send(dat,1,100);  // envia ordem para acender a lâmpada
    delay_ms(100);
    dat[0]= 0xFF;
   }
  if(ordem==2)
  {
    dat[0]= 0x02;
    RS485Master_Send(dat,1,100);  // envia ordem para apagar a lâmpada
    delay_ms(100);
    dat[0]= 0xFF;
   }
  if(ordem==3)
  {
    dat[0]= 0x01;
    RS485Master_Send(dat,1,101);  // envia ordem para desligar o alarme
```

```
    delay_ms(100);
    dat[0]= 0xFF;
   }
   if(ordem==4)
  {
    dat[0]= 0x01;
    RS485Master_Send(dat,1,102);  // envia ordem para ligar a lâmpada
    delay_ms(100);
    dat[0]= 0xFF;
   }
   if(ordem==5)
  {
    dat[0]= 0x02;
    RS485Master_Send(dat,1,102);  // envia ordem para apagar a lâmpada
    delay_ms(100);
    dat[0]= 0xFF;
   }
if (dat[5])
  {
  dat[5]=0;
   }
if (dat[4])
  {
  dat[4] = 0;
  if (dat[0]==0x00) Soft_UART_Write('O');
  if (dat[0]==0x00) Soft_UART_Write('K'); // tarefa executada
```

```
    }
   }
}
```

ESCRAVO

```
char dat[9];
char i,j;

sbit  rs485_rxtx_pin at RB0_bit;
sbit  rs485_rxtx_pin_direction at TRISB0_bit;

// Interrupt routine
void interrupt() {
 RS485Slave_Receive(dat);
}

void main()
{

  UART1_Init(9600);
  Delay_ms(100);
  RS485Slave_Init(100);

  dat[4] = 0;
  dat[5] = 0;
  dat[6] = 0;
```

```
 RCIE_bit = 1;
 TXIE_bit = 0;
 PEIE_bit = 1;
 GIE_bit = 1;
 TRISA2_BIT=0;
 RA2_BIT = 0;

RS485Slave_Init(100);

 while (1)
 {
   if (dat[5])  {  dat[5] = 0;  }

   if (dat[4])
   {
     dat[4] = 0;
     if (dat[0]==0x01)  RA2_BIT = 1;   //acende a lâmpada
     if (dat[0]==0x02)  RA2_BIT = 0;   //apaga a lâmpada
     dat[0]=0x00;
     RS485Slave_Send(dat,1)
     }
   }
}
```