UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

SISTEMA DE AUTOMAÇÃO RESIDENCIAL COM CENTRAL DE CONTROLE MICROCONTROLADA E INDEPENDENTE DE PC

RENÉ ALFONSO REITER JÚNIOR

BLUMENAU
2006

2006/2-24

RENÉ ALFONSO REITER JÚNIOR

SISTEMA DE AUTOMAÇÃO RESIDENCIAL COM CENTRAL DE CONTROLE MICROCONTROLADA E INDEPENDENTE DE PC

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Antônio Carlos Tavares , Orientador

BLUMENAU
2006

2006/2-24

SISTEMA DE AUTOMAÇÃO RESIDENCIAL COM CENTRAL DE CONTROLE MICROCONTROLADA E INDEPENDENTE DE PC

Por

RENÉ ALFONSO REITER JÚNIOR

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Antônio Carlos Tavares, Orientador, FURB
Membro:	Prof. Miguel A. Wisintainer – FURB
Membro:	Prof. Francisco Adell Péricas – FURB

Blumenau, 14 de novembro de 2006

Dedico este trabalho aos meus pais pelos seus incentivos à realização deste

AGRADECIMENTOS

À minha mãe pelo incentivo e cobranças.

Ao meu pai pela ajuda na eletrônica e com o Data Term (IHM).

Ao meu orientador, Antônio Carlos Tavares, por todas as sugestões.

A mais alta das torres começa no solo.

Provérbio chinês

RESUMO

Automação residencial (domótica) é cada dia mais vista em discussão na mídia. Este trabalho apresenta uma proposta de solução para automatizar uma residência sem a necessidade de projetar a casa para isso ou fazer mudanças estruturais. A automação é feita ligando ou desligando dispositivos elétricos através de placas acionadoras microcontroladas. Uma central gerencia esses acionadores, armazenando um histórico de eventos e, se o software supervisório PC for conectado, transmite para ele. É possível controlar os acionadores através da central e do PC.

Palavras-chave: Domótica. Microcontroladores. Controle supervisório.

ABSTRACT

Home automation (domotics) is more and more seen in discussion at midia. This work presents a proposal of solution to automate a home without projecting the house for this or making structural changes. The automation is made by turning on and off electric devices by microcontroller actionators circuits. A central manages these actionators, storaging a detailed report of events and, if the supervisory software is connected, then transmits to it. It's possible to control all actionators by the central and by the PC.

Key-Words: Domotics. Microcontrollers. Supervisory control.

LISTA DE ILUSTRAÇÕES

Quadro 1 - Comparativo entre tecnologias de transmissão de dados 21
Figura 1 – Esquema de mudança de estados numa transmissão em RS-485 22
Figura 2 – Evolução dinâmica de um SED 26
Figura 3 – Exemplo de autômato 27
Figura 4 – Exemplo de SCADA 29
Figura 5 – Diferenças entre microcontrolador e microprocessador 30
Figura 6 – Arquitetura interna de um MCS51 32
Figura 7 – Disposição física do sistema 36
Quadro 2 – Formato do pacote de comunicação central-acionadores 37
Figura 8 – Exemplificando *polling* 39
Quadro 3 – Comandos passados em CMD1 39
Figura 9 – Autômato de reconhecimento de pacotes no acionador 40
Figura 10 – Autômato de reconhecimento de pacotes na central 41
Quadro 4 – Formato do pacote de comunicação PC → central 41
Quadro 5 – Pacote com o comando de ajuste de data e hora 42
Quadro 6 – Pacote com o novo nome do acionador 42
Quadro 7 – Comandos passados em CMD1 pelo PC 42
Quadro 8 – Formato do pacote de envio de status ao PC 43
Quadro 9 – Formato do pacote de envio da hora da central ao PC 43
Quadro 10 – Formato do pacote de envio de registro histórico ao PC 43
Figura 10 – Autômato de reconhecimento de pacotes no PC 44
Figura 11 – Exemplo de tomada e interruptor 44
Figura 12 – Esquema elétrico do acionador 45
Figura 13 – Esquema elétrico do regulador de tensão do acionador 46
Figura 14 – Parte superior da placa acionadora 47
Figura 15 – Lado inferior da placa acionadora – lado da solda 47
Figura 16 – Fluxograma – rotina principal do acionador 48
Figura 17 – Fluxograma – rotina verificação de eventos do acionador 49
Figura 18 – Fluxograma – rotina verificação de transmissão de dados do acionador 50
Figura 19 – Foto do Data Term 52
Figura 20 – Telas da Central 53

Figura 21 – Fluxograma – rotina principal da central 54
Figura 22 – Fluxograma – rotina verificar transmissão de dados da central 55
Figura 23 – Fluxograma – rotina processa dado recebido da central 56
Figura 24 – Fluxograma – rotina processa fim do *polling* da central 57
Figura 25 – Fluxograma – função reação 58
Figura 26 – Fluxograma – rotina atualizar *display* na central 59
Figura 27 – Fluxograma – rotina comunicação com PC na central 60
Figura 28 – Diagrama de casos de uso 61
Figura 29 – Diagrama de classes 62
Quadro 11– Exemplo de arquivo com valores rotulados 63
Figura 30 – Interface `SupervisorPrincipal` 63
Figura 31 – Interface `frmAcionador` 64
Figura 32 – Interface `ConfigSerial` 64
Figura 33 – Interface `CadastroAcionador` 64
Figura 34 – Diagrama de seqüência 1 65
Figura 35 – Diagrama de seqüência 2 65
Figura 36 – Visão superior e inferior da placa acionadora 67
Quadro 12 – Principais rotinas do acionador 67
Quadro 13 – Função `verificar_comunicacao()` 68
Quadro 14 – Função `verificarOcorrenciaEventos()` 69
Figura 37 – Acionadores interligados 70
Quadro 15 – Funções `main()` e `loop()` da central 71
Quadro 16 – Função `processaFimPolling()` na central 72
Quadro 17 – Função `respondeStatusAcionadoresCOM2()` na central 73
Quadro 18 – Exemplo do conteúdo do arquivo da classe `ArquivoConfig` 74
Quadro 19 – Método `Executa` da classe `TComunicador` 75
Quadro 20 – Listagem do arquivo `LogHistoricoCentral.dat` 76
Quadro 21 – Métodos `retiraFila` e `incluiFila` 77
Figura 38 – Placa acionadora dentro da caixa elétrica 78
Figura 39 – Placa acionadora dentro da caixa elétrica com interruptor 78
Figura 40 – Esquema de ligação do acionador 79
Figura 41 – Tela de entrada 79
Figura 42 – Tela de apresentação 79

Figura 43 – Tela do menu principal.....79
Figura 44 – Tela monitoração de acionadores.....80
Figura 45 – Tela cadastro de acionadores 80
Figura 46 – Tela configura data e hora.....81
Figura 47 – Software PC – tela principal 81
Figura 48 – Software PC – configuração da porta serial.....82
Figura 49 – Software PC – cadastro dos acionadores 82
Figura 50 – Software PC – conectado 83
Figura 51 – Software PC – perda de conexão com o acionador.....83
Figura 52 – Software PC – verificando hora da central 83
Quadro 22 – função `verificar_transmissao_dados` 90

LISTA DE SIGLAS

ASCII – *American Standard Code for Information Interchange*

I²C – *Inter-Integrated Circuit*

IHM – Interface Homem-Máquina

IrDA – *Infrared Data Association*

MIDI – *Musical Instrument Digital Interface*

OSI – *Open Systems Interconect*

PC – *Personal Computer*

PID – Proporcional Integral Derivativo

RETMA - *Radio Electronics Television Manufacturers Association*

RS – RETMA *Standard*

SCADA – *Supervisory Control And Data Acquisition*

SPI – *Serial Peripheral Interface*

USB – *Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 OBJETIVOS DO TRABALHO 14
1.2 ESTRUTURA DO TRABALHO 15
2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 TRANSMISSÃO DE DADOS 16
2.1.1 Características da Transmissão 16
2.1.1.1 Canal 17
2.1.1.2 Serial e Paralela 17
2.1.2 Topologia 17
2.1.3 Detecção de Erros 18
2.1.4 Protocolos 18
2.1.4.1 Controles de acesso ao meio 19
2.1.4.1.1 Acesso ordenado em contenção 19
2.1.4.1.2 Acesso ordenado sem contenção 20
2.1.5 *Hardware* de comunicação 20
2.1.5.1 RS-232 21
2.1.5.2 RS-485 22
2.2 AUTOMAÇÃO RESIDENCIAL 23
2.2.1 Objetivos da automação residencial 23
2.2.2 Interligação entre equipamentos 24
2.3 EVENTOS DISCRETOS E CONTROLE SUPERVISÓRIO 25
2.3.1 Sistema a Eventos Discretos 25
2.3.1.1 Modelagem e Especificação de SEDs 26
2.3.2 Controle supervisório 28
2.3.3 SCADA 28
2.4 MICROCONTROLADORES 29
2.4.1 Família MCS51 30
2.4.1.1 Microcontrolador Atmel AT89S52 32
2.5 TRABALHOS CORRELATOS 33
2.6 ESTADO DA ARTE 33
3 DESENVOLVIMENTO DO TRABALHO 34

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 34
3.2 ESPECIFICAÇÃO 36
3.2.1 Protocolos de Comunicação 37
3.2.1.1 Protocolo Central-Acionadores 37
3.2.1.2 Protocolo PC-Central 41
3.2.2 Acionadores 44
3.2.2.1 Especificação Física 45
3.2.2.2 Especificação do Software 47
3.2.3 Central Controladora 51
3.2.3.1 Data Term 51
3.2.3.2 Software da Central 52
3.2.3.2.1 Sub-rotina Verificar Transmissão de Dados 54
3.2.3.2.2 Sub-rotina Atualizar Display 58
3.2.3.2.3 Sub-rotina Verificar Comunicação com PC 59
3.2.3.2.4 Sub-rotina Verificar Entradas Teclado 60
3.2.4 Supervisório PC 60
3.3 IMPLEMENTAÇÃO 66
3.3.1 Técnicas e ferramentas utilizadas 66
3.3.1.1 Acionador 66
3.3.1.2 Central 70
3.3.1.3 Software PC 73
3.3.2 Operacionalidade da implementação 77
3.3.2.1 Instalando os acionadores 78
3.3.2.2 Usando a central 79
3.3.2.3 Monitorando através do PC 81
3.4 RESULTADOS E DISCUSSÃO 84
4 CONCLUSÕES 86
4.1 EXTENSÕES 87
REFERÊNCIAS BIBLIOGRÁFICAS 88
APÊNDICE A – Função verificar transmissão de dados 90

1 INTRODUÇÃO

Cada dia que passa, a automação está mais presente, tanto nas residências quanto nas indústrias, shoppings e afins. Isso se deve ao fato do povo estar cada vez com menos tempo disponível, preocupado com a segurança e procurando sempre economizar os recursos fornecidos. Uma proposta de estudar e construir uma solução para boa parte dessas necessidades é apresentada nesse trabalho.

A domótica (automação residencial) vem evoluindo dia a dia. Atualmente o máximo que se tem acesso a este tipo de automação são pequenos confortos, como por exemplo portões eletrônicos, alarmes, luzes com *timer*, entre outros. Mas o grande objetivo é a integração de todas essas tecnologias fazendo com que estejam conectadas entre si, e que os moradores de uma residência automatizada tenham o máximo de conforto e segurança que a tecnologia possibilitar.

Neste trabalho apresenta-se um sistema de controle/monitoração de uma residência. Neste sistema, dentro de cada tomada e interruptor localiza-se um pequeno circuito eletrônico, que faz o controle do dispositivo conectado a ele. Esses dispositivos são desde eletrodomésticos até simples lâmpadas. São utilizados microcontroladores da família MCS51, todos interligados e respondendo a uma central controladora. O meio de comunicação entre estes é o RS-485, que é bastante utilizado na indústria.

A central e os circuitos nas tomadas são suficientes para a automação, e totalmente independentes de um computador, mas para mostrar uma interface mais agradável ao usuário, é possível conectar a central em um microcomputador PC, onde estará rodando o sistema operacional Linux. Esta interface é configurável e expansível, permitindo ao sistema adaptar-se em qualquer residência.

A central controladora tem um hardware bastante complexo, ao passo que os circuitos que controlarão cada dispositivo elétrico são muito mais simples.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é implementar uma central de controle microcontrolada que consiga estabelecer comunicação com um PC rodando Linux e, ao mesmo tempo, monitorar e

passar comandos para vários microcontroladores menores.

Os objetivos específicos do trabalho são:

a) criar uma interface amigável na central de controle;
b) possibilitar comunicação serial entre o Linux e a central controladora;
c) criar uma central controladora para enviar comandos aos circuitos menores que controlarão diretamente os elementos elétricos da residência;
d) montar uma placa compacta que caiba em uma simples caixa de tomada.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 mostra a fundamentação teórica do projeto, incluindo automação residencial, transmissão de dados, controle supervisório e microcontroladores. Ao fim do capítulo são descritos alguns trabalhos correlatos e comentado sobre o estado da arte da domótica.

O capítulo 3 descreve todo o processo de implementação do sistema, com fluxogramas, esquemas elétricos, diagramas de classes, diagramas de casos de uso e diagramas de seqüência. São mostradas algumas rotinas em detalhes para melhor compreensão da especificação. Por fim é dado um exemplo prático de uso do sistema seguido das conclusões e comentários sobre a implementação.

O capítulo 4 apresenta as considerações finais sobre o projeto e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados assuntos para melhor compreensão da especificação e implementação do projeto. Esses assuntos são:

a) transmissão de dados;
b) automação residencial;
c) controle supervisório;
d) microcontroladores;
e) trabalhos correlatos;
f) estado da arte.

2.1 TRANSMISSÃO DE DADOS

"A transmissão consiste, basicamente, em se fazer chegar uma informação a um ponto distante do local em que foi gerada" (ROKENBACH, 1979, p. 4).

O dispositivo que gerou a informação é o transmissor, e o que recebeu é o receptor. Dependendo da topologia de rede local, mais de um receptor pode ouvir a informação ao mesmo tempo.

Toda e qualquer informação é passada através de sinais em meio físico. Existem basicamente dois tipos de sinais: digital e analógico. Na transmissão analógica, os sinais elétricos variam continuamente entre todos os valores possíveis enquanto na transmissão digital, os sinais possuem apenas dois estados elétricos: ligado e desligado.

Esses e outros assuntos serão abordados nesse capítulo.

2.1.1 Características da Transmissão

Transmissão de dados possui algumas características como canal e transmissão serial e paralela.

2.1.1.1 Canal

Um canal é o meio de transmissão que pode ser de entrada, de saída ou de entrada e saída. Suas definições clássicas são:

e) *simplex:* pode levar a informação apenas para uma direção;
f) *half-duplex:* pode levar a informação para ambas as direções, mas não simultaneamente;
g) *full-duplex:* pode levar as informações para ambas as direções, ao mesmo tempo.

2.1.1.2 Serial e Paralela

Num equipamento eletrônico digital a informação é organizada em *bytes* que por sua vez estão organizados em *bits*. Existem duas maneiras de transmitir esses dados, serial e paralela.

Na transmissão paralela, vários bits são transmitidos simultaneamente e cada barramento representa um bit, o que acaba deixando a transmissão mais rápida porém limitada a curtas distâncias, a um maior número de canais de transmissão e a maiores interferências do meio.

Na transmissão serial os bits são enviados um de cada vez através do mesmo barramento, sendo necessário algum mecanismo para saber quando termina um bit e começa outro. Este controle é feito através de contagem de tempo de acordo com a velocidade de transmissão que o equipamento está preparado para funcionar.

Na transmissão assíncrona existem dois bits extras que sempre são enviados: o *start bit* e o *stop bit*, que tem a função de indicar quando um *byte* começa a ser transmitido e quando termina de transmiti-lo.

2.1.2 Topologia

A topologia está relacionada com a distribuição geográfica dos nodos e os elos da rede. Nodo pode ser considerado um computador da rede. Elo é a linha de transmissão de dados entre dois nodos.

Existem dois tipos básicos de topologias de rede:

a) *ponto-a-ponto*: é o tipo mais simples de rede, onde um computador na rede se conecta somente a outro computador através de um único canal, sem haver necessidade de endereçamento dos pacotes;
b) *multiponto*: é quando dois ou mais computadores se conectam através do mesmo canal, sendo necessário o endereçamento dos pacotes transmitidos. Neste tipo geralmente existe uma central, que controla o tráfego de dados. Uma técnica bastante utilizada no multiponto é o *polling*, que é explicado mais adiante.

Além desses dois tipos, existem variações usando estes, como por exemplo a rede tipo estrela e a rede em anel. Esses são chamados de estruturas mistas.

2.1.3 Detecção de Erros

Na transmissão de dados muitos são os fatores que interferem na qualidade da mesma. Ruídos elétricos podem alterar os *bits* no meio do processo, gerando uma informação incorreta. Para isso existem meios de detectar erros, e os mais simples de se implementar são: Paridade Vertical e Paridade Horizontal. Ambos consistem em adicionar informação aos dados sendo transmitidos.

> Paridade Vertical (VRC) Consiste em somar o número de bits '1' de um *byte*, que totalizará um número par ou ímpar. Desta forma pode-se optar pela paridade tipo par ou ímpar. Definido o tipo de paridade, adiciona-se um *bit* à seqüência dos 8 *bits*, de maneira que a soma desses 8 *bits* seja sempre par ou ímpar, conforme o tipo de paridade escolhida. (TAFNER; LOESCH; STRINGARI, 1996, p. 24).

> Paridade Horizontal (LRC) Consiste em checar uma mensagem em blocos. A verificação não se dará apenas em um caracter, mas sim em uma série de caracteres que compõem a mensagem. Sendo a mensagem transmitida em blocos, a técnica propõe uma averiguação redundante, ou seja, a cada 8 *bytes* será gerado um *bit* de paridade para cada *byte* que compõe este bloco (na horizonal), formando mais um *byte* que será transmitido juntamente com os 8 *bytes*.(TAFNER; LOESCH; STRINGARI, 1996, p. 26).

2.1.4 Protocolos

Para que a informação chegue ao receptor e possa ser corretamente interpretada entre

transmissor e receptor, é necessário que existam algumas regras. De acordo com AXELSON (2000, p. 4), "protocolo é um conjunto de regras que definem como os computadores vão gerenciar a sua comunicação.". Esses protocolos podem ser orientados a caractere ou a bit.

a) orientado a caractere: Os controles são feitos usando caracteres especiais da tabela ASCII. Existe um caracter que indica início de transmissão, outro que indica início de texto, outro fim da transmissão, e assim por diante. Num pacote desses, a informação é dividida em caracteres;
b) orientado a bit: "São protocolos que não utilizam caracteres especiais para delimitar blocos de mensagem. Todo o controle é tratado a nível de *bit*...". (TAFNER; LOESCH; STRINGARI, 1996, p. 36).

2.1.4.1 Controles de acesso ao meio

Existe um barramento comum por onde os diversos computadores se comunicam para transmitir os dados, com e sem contenção.

2.1.4.1.1 Acesso ordenado em contenção

Numa rede em contenção não existe ordem de acesso a rede e todos podem transmitir simultaneamente, resultando em colisões, o que acarretará, geralmente em perda das mensagens. Existem dois métodos clássicos em contenção:

a) ALOHA: cada nodo pode transmitir independente da rede estar sendo ocupada ou não. Os algoritmos no ALOHA devem estar preparados para detectar uma colisão, e quando detectada, devem retransmitir depois de um tempo aleatório;
b) CSMA: antes de transmitir, cada nodo deve "escutar" a rede para saber se existe alguma transmissão sendo feita. Caso existir, ele espera até que acabe para poder transmitir. Mesmo com essa segurança, existe a possibilidade de ocorrer colisão, quando o tratamento será feito de forma bastante semelhante ao método ALOHA.

2.1.4.1.2 Acesso ordenado sem contenção

Vários métodos para evitar colisão foram desenvolvidos sendo sua principal técnica o acesso ordenado à rede.

a) *slot*: método comumente utilizado em redes em anel. O funcionamento é semelhante a uma estrada circular com um número fixo de caminhões circulando nela. Em cada caminhão existe um *flag* indicando se ele esta cheio. Os caminhões dessa associação, são pacotes com dados. Sempre que um nodo quiser transmitir, precisa esperar passar um pacote que esteja vazio e só então colocar no pacote os dados que ele quer enviar e o endereço do destinatário. Como o número de pacotes na rede é fixo, o nodo originador saberá que o pacote foi entregue quando o pacote estiver de volta e sem os seus dados nele. Este método foi utilizado pela primeira vez por Pierce (1972) e por isso é também conhecido como anel de Pierce.
b) *polling*: este método é mais comumente usado em redes de barra comum. Deve existir um nodo central que controla o acesso dos outros nodos a rede. As estações só podem transmitir quando forem interrogadas pelo nodo central. A central questionará todas as estações, uma de cada vez. As estações, por sua vez, responderão com alguma mensagem ou, caso não tenha mensagem, enviará apenas uma mensagem de *status*.
c) passagem de permissão: “Neste tipo [...] é passada seqüencialmente de estação a estação. Somente a interface que possui a permissão em um determinado instante de tempo pode transmitir mensagens de qualquer comprimento. A ordem lógica de transmissão não é necessariamente a ordem física, embora em topologias de anel geralmente o seja.”(GOMES, 1986, p. 115).

2.1.5 *Hardware* de comunicação

A conexão física entre os dispositivos de uma rede é feita por cabos, ondas de rádio, infravermelho e outros meios. Cada uma dessas tecnologias possui suas características próprias que vão torná-las mais adequadas a cada tipo de aplicação. Essas características incluem a interface de conversão do computador para o meio, a distância entre nodos e a topologia suportada.

No quadro 1 é feita uma comparação entre as várias tecnologias de transmissão de dados existentes:

Interface	**Formato**	**Número de Dispositivos (máximo)**	**Distância máxima (m)**	**Velocidade máxima (bits/s)**
RS-232	serial assíncrona	2	15-30	20k
RS-485	serial assíncrona	32	1200	10M
IrDA	infravermelho serial assíncrona	2	2	115k
Microwire	serial síncrona	8	3	2M
SPI	serial síncrona	8	3	2,1M
I²C	serial síncrona	40	5	400k
USB	serial assíncrona	127	4,5	12M
Firewire	serial	64	4,5	400M
IEEE-488	paralela	15	18	1M
Ethernet	serial	1024	490	10M
MIDI	serial	2	4,5	31,5k
Porta Paralela	paralela	2	3-9	1M

Fonte: Axelson, 2000, p. 6.

Quadro 1 - Comparativo entre tecnologias de transmissão de dados

Existem muitos *hardwares* para comunicação entre redes mas neste trabalho são estudados mais a fundo apenas o RS-232 e o RS-485.

2.1.5.1 RS-232

Todos os computadores possuem uma interface RS-232. Esta interface possui uma arquitetura muito simples e é amplamente utilizada em diversos projetos de comunicação serial pois, ao contrário da USB, esta é facilmente adaptada a diversas eletrônicas, não necessitando de componentes complexos.

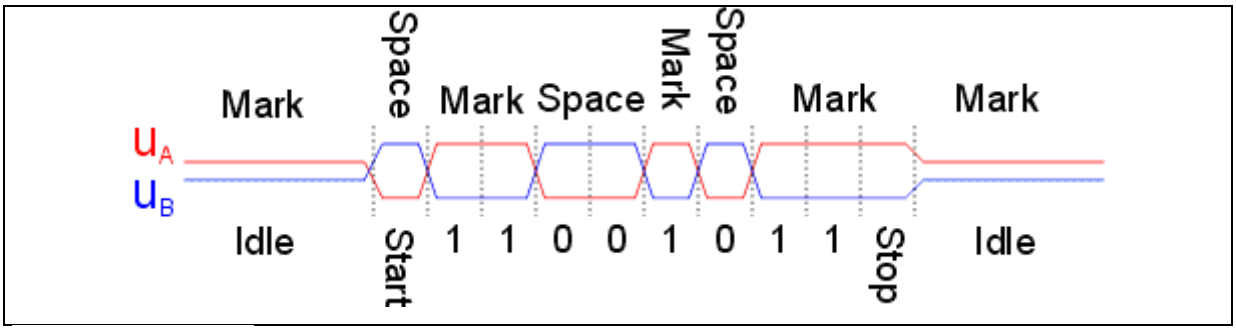
> RS-232 é a interface padrão de comunicação serial definida pela Electronic Industries Association (EIA). RS-232 existe em três tipos: A, B e C. Cada um desses tipos define uma tensão diferente para os estados ligado e desligado. O tipo mais utilizado é a especificação RS-232c, que define o bit ligado em uma tensão de -3V até -12V, e o bit desligado em uma tensão entre +3V e +12V. A especificação do RS-232 diz que o sinal pode chegar a 8m antes que ele se torne inutilizado. (SWEET, 2005)

2.1.5.2 RS-485

Esta interface é bastante utilizada na indústria pois é pouco suscetível a ruídos elétricos, permite uma distância entre o transmissor e o receptor máxima de mais de mil metros e é multiponto. Alem disso é facilmente convertida de RS-232 para RS-485 e de RS-485 para RS-232. De acordo com AXELSON (2000, p. 185) as principais características da RS-485 são:

a) multiponto, permitindo até trinta e dois nodos;
b) distância de mil e duzentos metros entre os nodos mais afastados;
c) baixo custo, pois as interfaces são bastante simples e precisam apenas de uma fonte de +5V;
d) velocidade de até 10Megabits por segundo;
e) existe pouca interferência elétrica em linhas balanceadas.

Uma das características mais marcantes do RS-485 é o sistema de linhas balanceadas. Este sistema é o principal responsável pela grande distância de transmissão entre nodos. Na figura 1 é mostrado um esquema de como as tensões se comportam numa transmissão de dados.



Fonte: RS-485 (2006).

Figura 1 – Esquema de mudança de estados numa transmissão em RS-485

No esquema, UA e UB são um par de fios que compõe uma linha de transmissão. Ambos fios funcionam em conjunto para a transmissão ser bem sucedida. Enquanto o sinal elétrico em um dos fios está com a tensão positiva, ou outro vai a neutro. Se este outro fio for para positivo, o primeiro vai a neutro e assim por diante. Desta forma o receptor pode fazer uma comparação entre os fios e aquele que estiver positivo, é o bit 1.

Este sistema evita que ruídos atrapalhem a transmissão. Quando existe um ruído, geralmente este vai ocorrer nos dois fios, em mesma intensidade. Então o receptor vai normalmente comparar os sinais dos dois fios. Aquele que estiver maior será o bit 1, mesmo que eles tenham ruídos.

2.2 AUTOMAÇÃO RESIDENCIAL

A automação residencial é geralmente confundida com televisores a controle remoto e portões eletrônicos, mas a verdade é que é um tema muito mais abrangente do que essas automações largamente conhecidas. A domótica, como é chamada a automação residencial, tem como principal objetivo a integração de todas as tecnologias de uma residência. De acordo com Daamen (2005, p. 9) um dos grandes problemas parece ser a troca de informação entre equipamentos de diferentes marcas.

Independente dos problemas, a domótica visa atender às necessidades de usuários domésticos em termos de conforto e segurança, e também apresenta algumas soluções em termos de comunicação de dados.

2.2.1 Objetivos da automação residencial

A domótica visa solucionar uma série de problemas usando eletrônica e conceitos muitas vezes vindos diretamente da ficção científica. “O objetivo da automação residencial é integrar iluminação, entretenimento, segurança, telecomunicações, aquecimento, ar condicionado e muito mais através de um sistema inteligente programável e centralizado. Como conseqüência fornece praticidade, segurança, conforto e economia para o dia a dia dos usuários” (ABREU, 2003).

De acordo com Abreu (2003), a automação residencial se divide em três tipos:

a) sistemas autômatos(*stand alone*): são divididos módulos entre os cômodos, onde cada um pode ser transmissor ou receptor e os módulos (que podem estar controlando luzes) podem ser controladas através de uma central na cabeceira da cama. O sistema pode ser montado aos poucos;
b) sistemas integrados com controle centralizado: a central é inteligente, as configurações podem ser simples ou complexas, inclui ações ativadas por eventos, equipamentos controlados via infravermelho podem ser programados pela central;
c) sistemas de automação complexos: integração total dos sistemas domésticos, a residência precisa ser projetada para essa automação, cabeamento estruturado.

Segundo Abreu (2003), os equipamentos comumente utilizados na automação residencial são:

a) telefonia e transmissão de dados;
b) aquecimento e ar condicionado;
c) iluminação;
d) home theater e som ambiente;
e) vigilância, alarme, iluminação de segurança, e circuito interno de TV;
f) aparelhos eletrodomésticos;
g) cortinas e portas automáticas;
h) *home offices*.

2.2.2 Interligação entre equipamentos

Foram criados vários padrões para os equipamentos de domótica, sempre visando a comunicação entre eles. Daamen (2005, p. 10) divide esses padrões em três categorias do modelo de referência OSI: camada de rede; camada de comunicação e camada de aplicação.

A camada de rede define as normas que fazem a comunicação no meio físico. De acordo com Daamen (2005, p. 10) as principais são:

a) RS-485;
b) ethernet;
c) fireWire;
d) homePNA: utiliza os cabos de telefonia para transmissão a velocidades de até 240Mbps
e) MoCA: utiliza cabos coaxiais de sinais de TV;
f) homePlug: a transmissão é feita diretamente na rede elétrica da residência em 240V;
g) X10: semelhante do HomePlug, mas com a rede em 110V.

“Para a Camada de Comunicação existem normas como BACnet, BatiBUS, EIB, EIBA, LonWorks e KNX. Estas normas descrevem os pacotes de dados que viajam de um lado para outro na rede, entre máquinas (terminais), sem que antes tenha sido estabelecida uma ligação” (DAAMEN, 2005, p. 10).

A camada de aplicação controla a transmissão independente da rede. É esta que controla a informação trocada entre as máquinas. Segundo Daamen (2005, p. 10) as normas comumente citadas nessa camada são a CEBus, ModBus, PROFIBUS, HAVi, EHS, OSGi e UPnP. Entre estas, destaca-se a última, UPnP, amplamente utilizada nos E.U.A. e que possui

recursos de *plug-and-play*.

2.3 EVENTOS DISCRETOS E CONTROLE SUPERVISÓRIO

Neste capítulo é explicado como funcionam os sistemas a eventos discretos, seguido de alguns conceitos de controle supervisório e concluindo com a apresentação do *Supervisory Control And Data Acquisition* (SCADA).

2.3.1 Sistema a Eventos Discretos

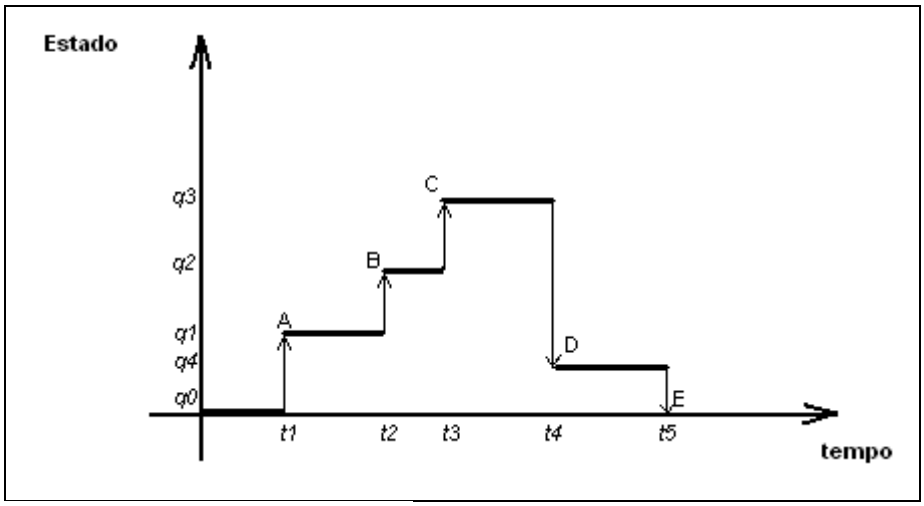
De acordo com Montgomery (2004, p. 1), “Sistemas são conjuntos de elementos, materiais ou imateriais, entre os quais se pode definir uma relação e que operam como uma estrutura organizada”. Um exemplo de sistema são os Sistemas Operacionais (SO) dos computadores, cujos elementos representados pelos processos, trocam informações com o sistema e fazem com que a máquina funcione de forma organizada.

Eventos são ações que ocorrem no sistema. Essas ações podem ser propositais, de ocorrência espontânea ou com a verificação de uma condição, e geralmente produzem mudanças de estado em intervalos de tempo aleatórios. Alguns exemplos de:

a) propositais: ligar um interruptor, abrir uma porta;
b) ocorrência espontânea: perda de conexão com o provedor de internet;
c) verificação de uma condição: temperatura de reator excede o limite de segurança.

Segundo Montgomery (2004, p. 3), “Um Sistema a Evento Discreto (SED) é definido como um sistema cuja evolução dinâmica depende da ocorrência de eventos”. Para que um SED possa existir, é necessário que existam ações ocorrendo, e que estas por sua vez gerem eventos.

Um sistema só mudará de estado quando ocorrer um evento. Se não ocorrer nenhum evento, o sistema permanecerá no mesmo estado. Isto é mostrado na Figura 2 onde de *q0* até *q4* estão representados os estados do sistema, de A até E são os eventos e de *t1* até *t5* são tempos aleatórios. Observa-se como o sistema só muda de estado quando ocorre um evento.



Fonte: MONTGOMERY (2004, p. 4).

Figura 2 – Evolução dinâmica de um SED

Montgomery (2004, p. 4) afirma que um SED é definido por três características básicas:

a) ciclo de funcionamento descrito através do encadeamento de eventos que determinam as tarefas realizadas ou em realização;
b) ocorrência de eventos simultâneos, isto é, vários eventos podem ocorrer ao mesmo tempo para alterar o estado do sistema;
c) necessidade de sincronização, pois, para que a evolução dinâmica seja correta, o início de certas atividades requer o término de outras.

2.3.1.1 Modelagem e Especificação de SEDs

Um modelo é uma representação simplificada de um problema. Essa representação geralmente é matemática ou gráfica. Tendo um modelo correto, vários estudos e análises são possíveis de serem realizados sem a necessidade de implementar o sistema.

Os modelos podem ser bastante detalhados ou muito simples. Essa complexidade vai determinar quantos sistemas diferentes o modelo poderá representar. Quanto mais simples o modelo, maior o número de sistemas representáveis.

Existem vários paradigmas usados em modelagem de SEDs. De acordo com Montgomery (2004, p. 8) os mais usuais são:

a) cadeias de Markov;
b) teoria de filas;
c) processos semi-Markovianos generalizados;

d) álgebra de processos;

e) álgebra de dióides;

f) redes de petri;

g) teoria de linguagens formais e autômatos.

Uma vez que o paradigma esteja definido, é possível fazer a especificação do sistema. Segundo Montgomery (2004, p. 9), “é necessário especificar qual o comportamento desejado, ou seja, qual a tarefa que o SED deve realizar”. O comportamento pode ser escrito em linguagem natural, porém devido às ambigüidades desta é mais adequado usar alguma linguagem formal.

Usando o paradigma da teoria de linguagens formais e autômatos é preciso fazer um conjunto de estados do SED. O estado inicial deve ser marcado e os estados finais também. Além dos estados, existem as transições do sistema. Estas devem ser definidas em um conjunto juntamente com as ações que levam essas transições.

Uma vez que o conjunto de estados e de transições esteja completo, pode-se montar um autômato finito.

Um exemplo de sistema é um modelo para abrir as comportas de uma represa onde existem quatro estados: comporta aberta; comporta fechada; comporta abrindo; comporta fechando. E existem somente duas ações: abrir comporta e fechar comporta. Tendo essas informações sobre o modelo, pode-se montar o autômato mostrado na Figura 3.

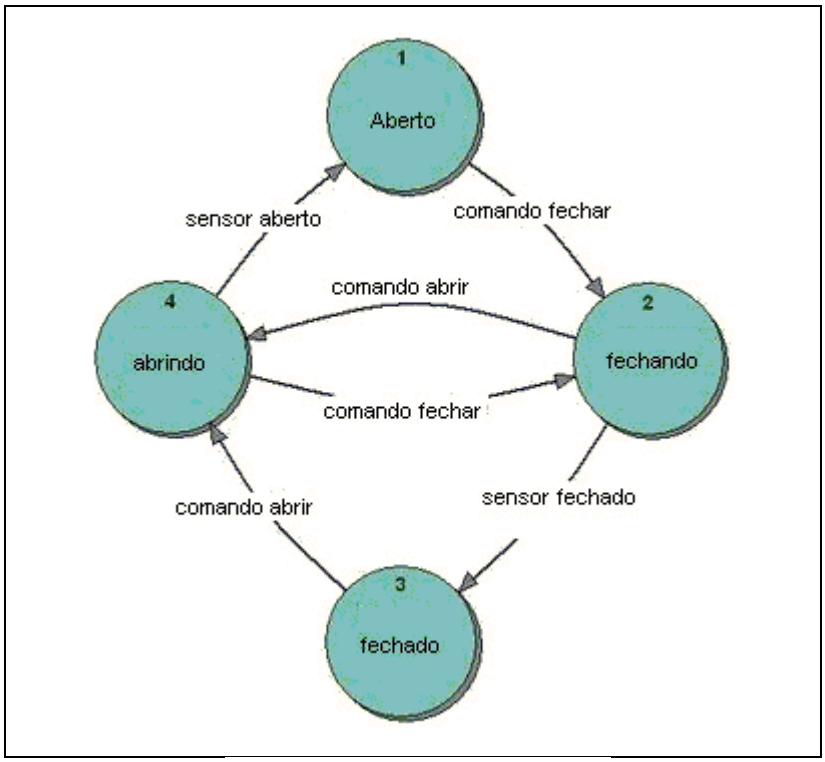


Figura 3 – Exemplo de autômato

2.3.2 Controle supervisório

Para controlar o sistema precisa-se de um supervisor. Este supervisor é geralmente uma central automática. De acordo com Montgomery (2004, p.9) “a partir de eventos gerados pelo SED, o supervisor define quais ações de controle devem ser implementadas para que seqüências específicas de eventos sejam observadas”.

No controle supervisório, os eventos são classificados em dois tipos:

a) eventos controláveis: são os eventos que o supervisor pode interferir. Exemplo: acionar uma esteira;
b) eventos não-controláveis: eventos que o supervisor pode apenas detectar o acontecimento, sem interferir. Um exemplo é a quebra de uma máquina.

Um evento não controlável pode gerar uma situação de bloqueio do sistema. É de responsabilidade do supervisor fazer o controle para evitar esse tipo de situação, seguindo seqüências preferencialmente com eventos controláveis.

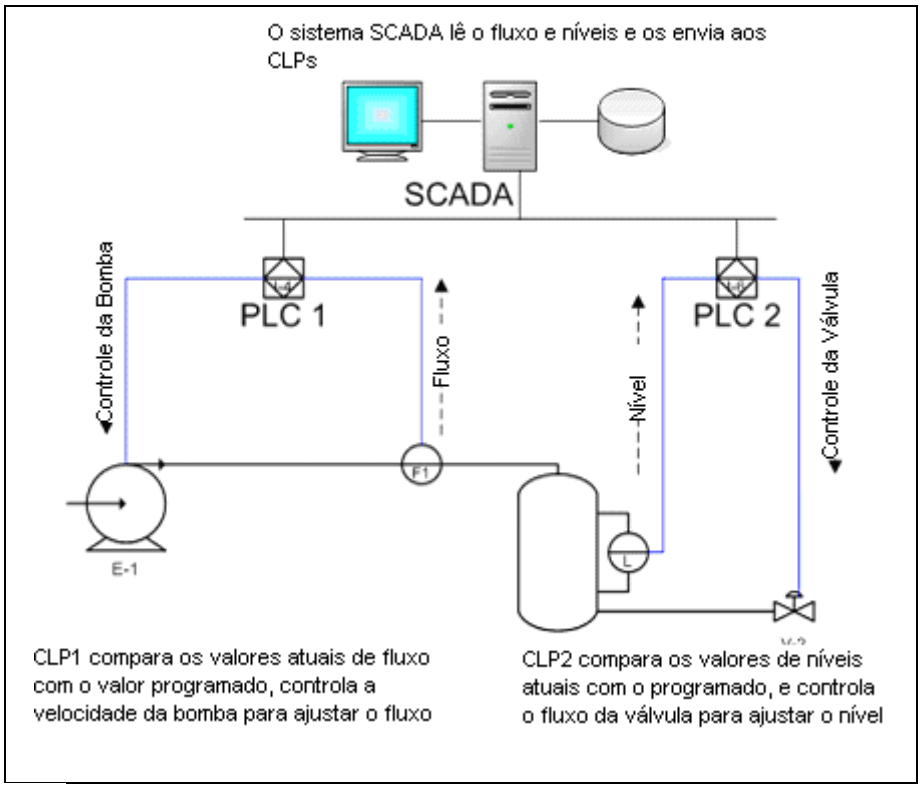
2.3.3 SCADA

Wiese (1997) define o SCADA como um “sistema de controle que consiste de um mestre, um ou mais transmissores de dados e unidades remotas de controle, e *softwares* usados para monitorar esse controle”.

Seguindo a definição de Wiese (1997), a figura 4 exemplifica um SCADA. Observa-se um computador junto de um banco de dados como centrais controladoras, duas unidades de controle remotas e um meio de comunicação entre eles. Na figura as unidades remotas controlam válvulas hidráulicas de acordo com parâmetros de controle passados pela central em tempo real. A central por sua vez armazena os dados colhidos para futura análise.

As unidades remotas coletam vários tipos de dados, entre os quais podem ser: estado do dispositivo (ligado ou desligado); temperatura; pressão; fluxo; corrente. Esses dados são formatados num protocolo conhecido pela central e então enviados.

A central controladora deve possuir uma Interface Homem Máquina (IHM) para que se possa monitorar as unidades remotas e também pode possuir algum processamento automático de eventos ocorridos nas unidades remotas. O usuário da IHM pode controlar as unidades remotas se for necessário.



Fonte: Scada (2006).

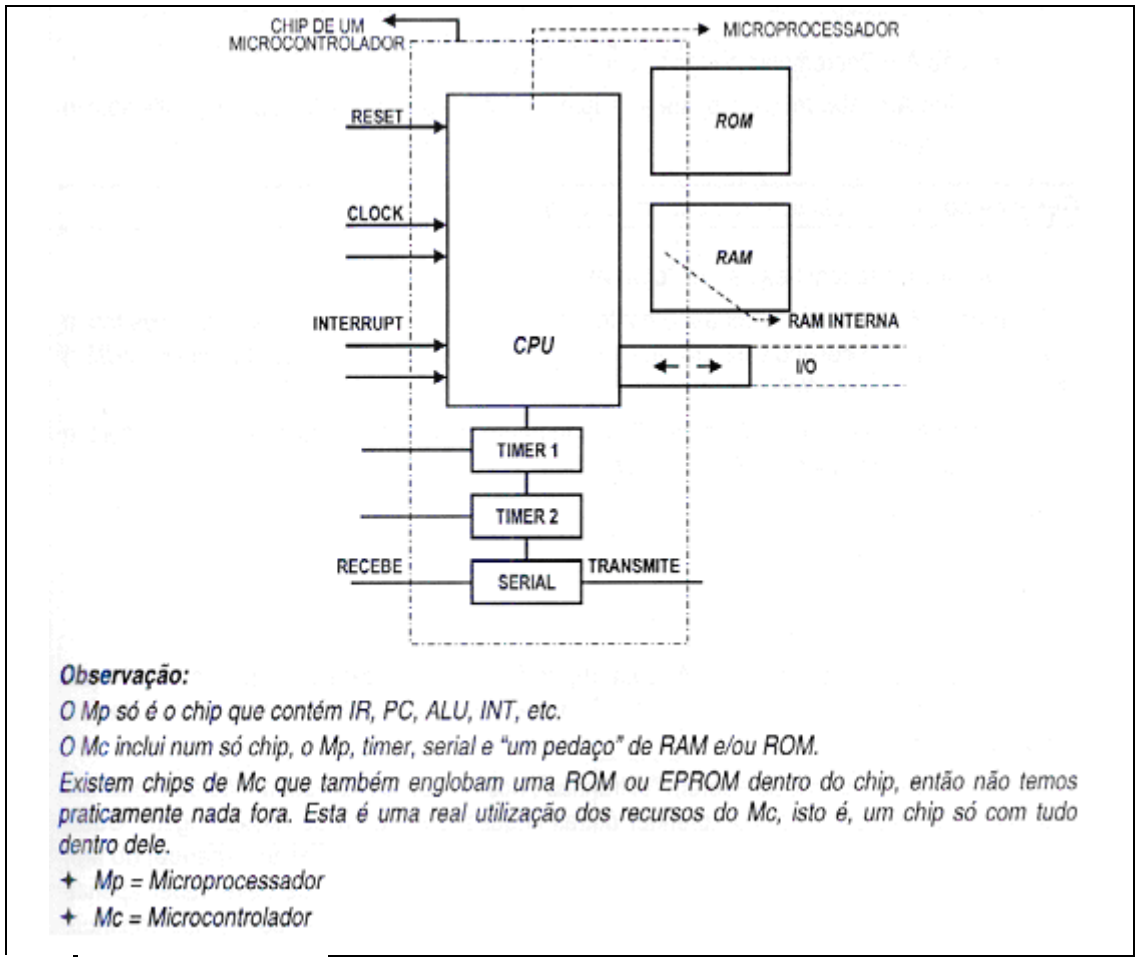
Figura 4 – Exemplo de SCADA

2.4 MICROCONTROLADORES

Os computadores atuais possuem um microprocessador que é um conjunto de circuitos integrados que processam operações lógicas e matemáticas. Esses microprocessadores precisam de memória, barramento, e toda uma série de periféricos para que funcionem adequadamente. Isso faz com que o espaço ocupado pela placa seja grande demais para pequenas aplicações eletrônicas.

É com a necessidade de miniaturização das placas que surge o microcontrolador. Ele é um processador com memória RAM, barramento, comunicação serial tudo no mesmo *microchip*. Para equipamentos eletrônicos é muito mais interessante um único *microchip* que supre as necessidades do que vários chips, cada um com sua funcionalidade.

A figura 5 mostra as diferenças básicas entre um microprocessador e um microcontrolador.



Fonte : Nicolosi (2000, p.65).

Figura 5 – Diferenças entre microcontrolador e microprocessador

Existem várias arquiteturas de microcontroladores fabricados atualmente, desde o PIC da Microchip (MICROCHIP TECHNOLOGY, 2006) e o AVR da Atmel (ATMEL CORPORATION, 2006) que possuem uma arquitetura *Reduced Instruction Set Computer* (RISC) até os MCS51 da Intel (INTEL CORPORATION, 2006) que possuem uma arquitetura *Complex Instruction Set Computer* (CISC). Todos possuem suas próprias características e funcionam melhor em aplicações diferentes.

Neste trabalho foi escolhido o MCS51 por ser fabricado por várias empresas diferentes, onde cada uma adiciona suas próprias funcionalidades, mas mantendo a compatibilidade com a especificação original da arquitetura.

2.4.1 Família MCS51

De acordo com Intel 8051(2006), o MCS51 foi desenvolvido pela Intel em 1980 e era

chamado de família 8051. Foi bastante popular entre os anos 1980 e 1990, mas então outras empresas começaram a fabricar o microcontroladores usando a mesma arquitetura (von Neumann), o que acabou fortalecendo ainda mais essa família. Empresas como a Atmel e a Dallas tem uma vasta linha de microcontroladores derivados do MCS51.

As características básicas do MCS51, de acordo com Nicolosi (2000, p. 70) são:

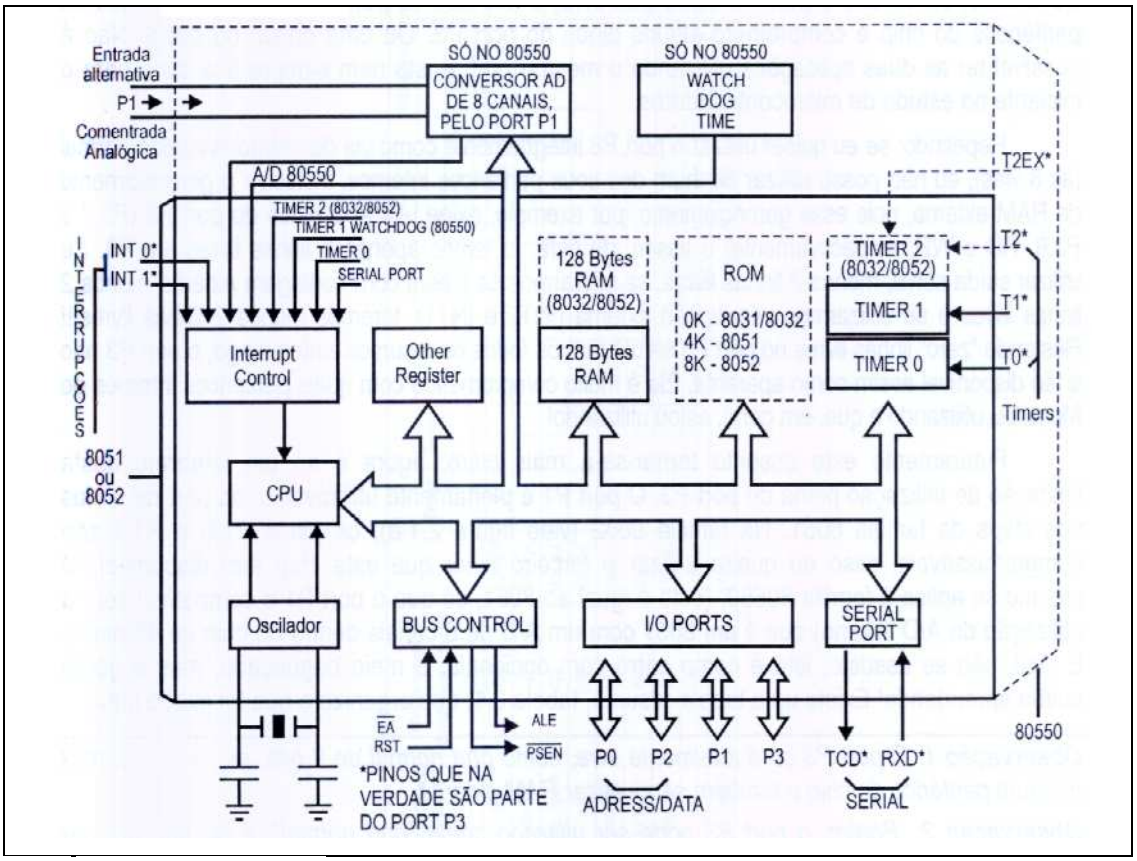
a) processador de 8 bits;
b) memória *Random Access Memory* (RAM) para uso geral de 128 bytes e mais 128 bytes reservados a registradores especiais;
h) quatro portas de entrada e saída;
i) interrupção externa;
j) dois *timers*;
k) *Read-Only Memory* (ROM) interna de 4 Kbytes;
l) capacidade de 64 Kbytes de endereçamento externo de ROM ou RAM;
m) possui arquitetura *Complex Instruction Set Computer* (CISC);
n) tem porta serial configurável em vários modos de transmissão;
o) instrução direta de divisão e multiplicação;
p) processador lógico que funciona em bits.

As portas de entrada e saída, que são conhecidas como P0, P1, P2 e P3, possuem oito bits cada uma. Algumas dessas portas apresentam funcionalidades especiais como a P3 que serve também para comunicação com periféricos internos através de transmissão serial. Os dois primeiros bits servem para transmissão de dados e a nomenclatura para se referir a estes bits é P3.0 e P3.1 (bit zero e bit um).

Apesar de algumas portas poderem ser usadas com funcionalidades especiais, elas podem, perfeitamente, ser usadas como vias de comunicação com o mundo externo, fazendo a interface entre o microcontrolador e outros componentes.

A transmissão serial pode ser configurada em dois modos básicos: síncrona e assíncrona. Além dos modos é preciso determinar o tamanho dos bytes na transmissão que podem ser de oito ou nove bits. Essas configurações são feitas no registrador especial chamado SCON. Outro registrador importante é o SBUF, que é o *buffer* de recepção e transmissão de bytes do MCS51.

A figura 6 mostra um esquema de como é a arquitetura interna de um MCS51.



Fonte : Nicolosi (2000, p.71).

Figura 6 – Arquitetura interna de um MCS51

2.4.1.1 Microcontrolador Atmel AT89S52

Para fazer uma comparação foi escolhido um MCS51 fabricado atualmente pela Atmel, o microcontrolador AT89S52, utilizado para o desenvolvimento deste trabalho em função de apresentar características como:

a) 8 Kbytes de memória flash interna para guardar o programa;

b) 3 *timers* internos;

c) *watchdog timer*.

Estas são as características diferentes do MCS51 básico.

2.5 TRABALHOS CORRELATOS

Besen (1996) implementa o controle de um cômodo residencial. Usando apenas um microcontrolador, ele faz o controle de duas lâmpadas, de um ar condicionado e ao mesmo tempo monitora as janelas e temperatura do cômodo. Para monitorar foi criado um software para PC com uma interface bastante simples e o microcontrolador comunicando-se com este *software*. Existe a dependência do PC estar ligado para poder monitorar. Outra característica do sistema é que foi feito para Windows 3.11, que era o disponível na época.

No trabalho de Censi (2001) existem comandos que são passados por *e-mail*, os quais um microcontrolador lê e processa. Alguns exemplos desses comandos podem ser ativar ou desativar tomadas e luzes. A central possui um *display* para poder acompanhar os estados dos dispositivos.

2.6 ESTADO DA ARTE

Vários projetos de automação residencial estão em desenvolvimento mas em sua maioria eles têm seguido as normas existentes apenas aperfeiçoando-as. Segundo Daamen (2005, p. 13) por enquanto, além de sistemas de controle remoto de iluminação, pouco mais existe e este panorama será pouco interessante até que outros sistemas sejam integrados.

Portanto, o grande desafio da domótica atualmente é a integração dos diversos eletrodomésticos existentes em uma residência, desde a iluminação até a temperatura do forno elétrico.

Diante desses projetos apenas uma afirmação pode ser feita, a informática embarcada esta cada vez mais presente não apenas na domótica, mas em todas as automações.A cada dia novos modelos de microcontroladores são lançados, cada um com suas funções específicas, é apenas uma questão de escolher o mais adequado ao projeto.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo é apresentado o processo de desenvolvimento do projeto, abordando os seguintes temas:

a) análise e especificação dos requisitos do problema a ser trabalhado;
b) especificação dos protocolos de comunicação PC-central e central-acionadores através de autômatos finitos;
c) especificação das placas dos acionadores através de esquemas elétricos;
d) especificação do software da central e dos acionadores através de fluxogramas;
e) especificação do software PC através de diagramas de casos de uso, classes e diagramas de seqüência;
f) implementação do software dos acionadores e da central usando linguagem C;
g) implementação do software PC usando Delphi/Kylix;
h) estudos de caso do sistema;
i) resultados obtidos e discussão.

Tanto a especificação quanto a implementação são divididos em três partes: acionadores, central e software PC.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos do sistema estão divididos em Requisitos Funcionais (RF) e Requisitos Não-Funcionais (RNF).

a) RF - ligar e desligar elementos elétricos pelo computador e pela central: o sistema deve permitir que seja ligado ou desligado qualquer elemento elétrico através do software supervisório no PC ou do software na central de controle. Exemplos de elementos elétricos: rádio; bomba para irrigação; motor da piscina; portão eletrônico. Para que seja possível ligar ou desligar, é preciso que o software do PC ou da central tenha algum tipo de indicação para saber qual acionador estará sendo afetado pelos comandos que por sua vez são passados serialmente aos acionadores.
b) RF - monitorar continuamente, independente do microcomputador PC estar ligado: a central de controle deverá operar independente do PC estar ligado e

comunicando-se com a mesma. Para isso a central deverá ter algum poder de processamento e armazenamento dos eventos gerados pelos acionadores. O PC, sempre que for ligado, perguntará a central pelos eventos gerados e os armazenará em disco. A central por sua vez limpará a fila de eventos sempre que forem descarregados ao PC.

c) RNF - elementos elétricos somente em dois estados: ligado ou desligado: os elementos elétricos podem estar somente em dois estados, sem fazer controle de intensidade da luminosidade ou velocidade de motores. Para controlar isso, são usados relés. Como os relés possuem uma limitação em corrente, não é qualquer dispositivo elétrico que pode ser ligado. Quem controlará os estados dos dispositivos é a placa acionadora, armazenando o estado em uma variável de controle interno. A central só poderá perguntar aos acionadores sobre os seus estados.
d) RF - permitir o controle de luminosidade: este é um requisito complementar ao anterior, pois o sistema deve permitir que sejam ligadas e desligadas lâmpadas de iluminação.
e) RNF - permitir interruptores convencionais nas placas acionadoras: para evitar que o local onde o sistema será instalado tenha que se adaptar visualmente a ele, é necessário que as placas acionadoras sejam acionadas por interruptores convencionais. Para isso o acionador deve processar os estados do interruptor, que poderá ser diferente do estado do dispositivo elétrico.
f) RNF - possuir uma interface fácil de ser usada: a interface deverá tornar o uso do sistema intuitivo para qualquer usuário que conheça o básico de informática. Deverá ter algum tipo de suporte a fim de saber qual acionador está selecionado, sem obrigar o usuário a decorar os endereços dos acionadores.
g) RNF - possuir uma arquitetura expansível: o sistema deverá permitir que se possa ligar e, conseqüentemente, monitorar e controlar quantos elementos elétricos forem necessários. A especificação do RS-485 permite apenas trinta e dois transmissores, portanto cada central estará limitada a trinta e um dispositivos elétricos.
h) RNF - transmitir dados de forma assíncrona: a transmissão serial deverá ser assíncrona. Como o número de acionadores será variável e deverá possuir uma autonomia para ser detectado na central sem ter sido programado, os pacotes seriais não obedecerão rígidos tempos na transmissão.
i) RNF - possuir proteção dos circuitos contra defeitos: as placas acionadoras devem

possuir alguns dispositivos de segurança para evitar conseqüências maiores em caso de falha.

j) RNF - ter a característica de não interferir esteticamente na residência: o sistema não pode aparecer onde for instalado e para isso deverá ser utilizada a estrutura do local para fazer a comunicação e alimentação dos acionadores. Este requisito que torna o trabalho diferente por permitir que o sistema seja instalado em qualquer local sem planejamento de uma automação. É um requisito especialmente importante para implementações futuras quando a central e os acionadores terão uma função de alarme na residência também.

3.2 ESPECIFICAÇÃO

A especificação é dividida em quatro partes. São demonstradas as especificações dos protocolos de comunicação central-acionadores e PC-central, seguido pela especificação dos acionadores, da central de controle e finalizando no software supervisório do PC.

A figura 7 mostra a arquitetura do sistema. Pode-se ver onde cada dispositivo está localizado. Os acionadores são interligados com a central controladora através de RS-485 e a central controladora por sua vez esta ligada ao software supervisório através de RS-232.

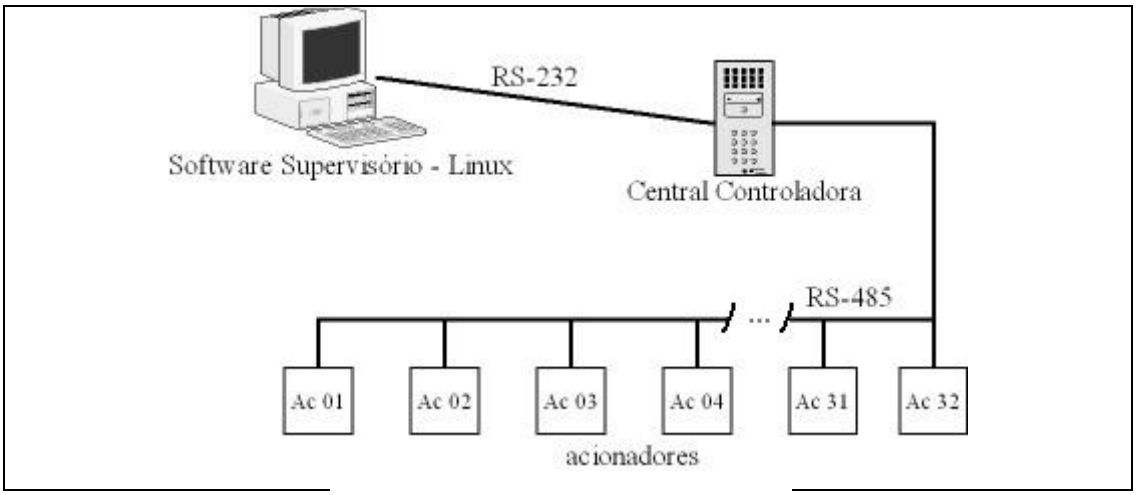


Figura 7 – Disposição física do sistema

3.2.1 Protocolos de Comunicação

São necessários dois protocolos diferentes para o sistema. Um permitindo a comunicação entre o PC e a Central e outro permitindo a comunicação entre a Central e os acionadores. A adoção de um protocolo específico visa otimizar ao máximo o tamanho dos pacotes trocados entre os elementos, coisa que não seria possível com protocolos padrões.

3.2.1.1 Protocolo Central-Acionadores

Uma das grandes preocupações da comunicação entre central e acionadores era a velocidade com que eventos seriam comunicados a central. Fazer simplesmente um *polling* parecia não atender às necessidades do sistema e por isso foi decidido fazer a comunicação com uma mistura de *polling* com algum método de acesso ordenado em contenção.

Depois de analisar o protocolo, viu-se que além de ocorrer muitos problemas nessa mistura de métodos de acesso a rede o *polling* simples seria adequado ao que o sistema se propõe, então a especificação teve de ser refeita e é esta que é apresentada a seguir.

O pacote de comunicação, tanto da central quanto dos acionadores, está exemplificado no quadro 2.

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**
STX	ADD1	ADD2	CMD1	CMD2	CMD3	ETX	BCC

Quadro 2 – Formato do pacote de comunicação central-acionadores

STX é o byte 02 da tabela ASCII, que indica início do cabeçalho. ADD1 e ADD2 são dois bytes que indicam o endereço do dispositivo ao qual o pacote esta sendo transmitido ou que foi transmitido. CMD1, CMD2 e CMD3 são três bytes de comando, onde se passam requisições ou envios de *status* e comandos de ligar ou desligar. ETX é o byte 03 da tabela ASCII e indica fim do pacote e finalmente BCC é o cálculo de paridade vertical do pacote para validar os outros bytes do mesmo.

Apesar do pacote ter o mesmo formato na central e nos acionadores, as respostas deles são ligeiramente diferentes. Os acionadores só devem processar pacotes que estejam endereçados ao seu endereço ou que sejam pacotes de *broadcast.*

Os pacotes de *broadcast* têm um tratamento especial nos acionadores com relação ao tempo de resposta, pois estes sempre são comandos de *polling.* O acionador, ao receber uma

requisição de *broadcast*, deverá responder depois de passados 20ms multiplicados pelo seu endereço. Essa estratégia torna esse *polling* ligeiramente diferente do convencional. No sistema, ao invés da central requisitar a cada acionador seu *status*, ela manda uma requisição para todos ao mesmo tempo (*broadcast*) e estes responderão cada um no seu tempo reservado.

O cálculo do tempo reservado a cada acionador leva em consideração o tamanho dos pacotes, a velocidade da transmissão e um tempo estipulado em 1s para varredura de todos os 32 acionadores previstos no projeto. A velocidade de transmissão é de 9600 Bits Por Segundo (bps). Sabendo que cada byte consumirá 10 bits (*start bit* e *stop bit*), e que em conseqüência disso cada pacote consome 80 bits, temos um tempo de 8,3ms por pacote (80 dividido 9600).

O tempo total de transmissão da central junto com a transmissão de todos os acionadores seria de 274ms (8,3ms multiplicado por 32 acionadores e mais 8,3ms da central) que é menos da metade do tempo de 1s. Então adicionou-se um tempo de tolerância para evitar problemas de relógios não sincronizados. O novo tempo reservado a cada pacote é de 20ms, que soma 660ms para o comando de *polling* e ainda deixa 340ms livres para requisições de ligar e desligar da central.

Exemplificando como acontece a transmissão:

a) central envia um comando de requisição de *status* em *broadcast*, o que caracteriza um *polling*;
b) cada acionador que receber a requisição em *broadcast* deverá esperar um período de tempo equivalente ao seu endereço, que é de 01 até 32, multiplicado por 20ms. Quando o tempo acabar, deverá então responder com seu *status*;
c) dessa forma cada acionador responderá ordenadamente e a central apenas coletará as respostas, processando-as. Erros na transmissão não são retransmitidos;
d) o *polling* termina em menos de 660ms, deixando ainda 340ms para que a central mande algum comando apenas para um acionador.

A figura 8 demonstra de maneira mais clara como acontece a transmissão.

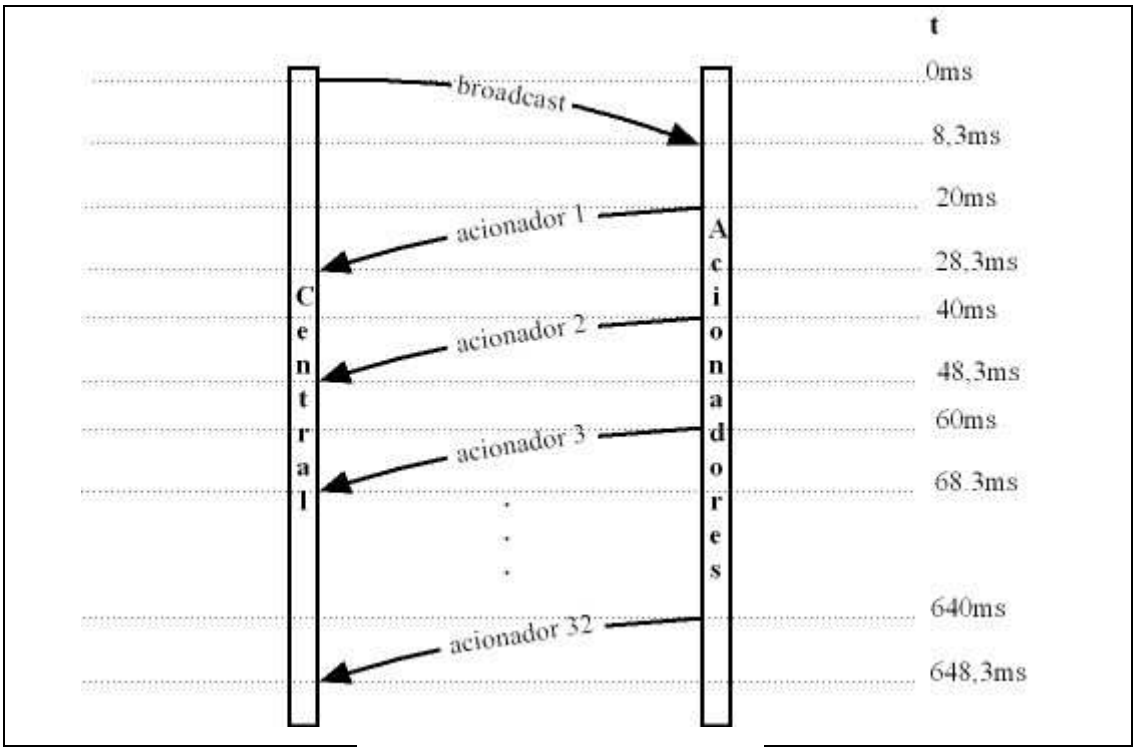


Figura 8 – Exemplificando *polling*

Os três bytes de comando são processados na central e nos acionadores. CMD2 e CMD3 estão reservados para implementações futuras. O quadro 3 mostra o que significa cada comando passado em CMD1.

Caractere	**Valor da Tabela ASCII**	**Sentido**	**Descrição**
A	65	Central -> Acionador	Requisição de *status* do acionador
L	76	Central -> Acionador	Liga o dispositivo
D	68	Central -> Acionador	Desliga o dispositivo
1	49	Acionador -> Central	O dispositivo está ligado
2	50	Acionador -> Central	O dispositivo está desligado

Quadro 3 – Comandos passados em CMD1

Os acionadores só responderão com os comandos “1” ou “2”. A central poderá requisitar o *status* do acionador através do comando “A” ou poderá requisitar mudança através dos comandos “L” (ligar) ou “D” (desligar).

Independente de qual valor contenha o byte CMD1, quem estiver lendo o pacote deverá seguir um padrão na leitura. Este padrão é representado pelo autômato na figura 9, que representa o acionador lendo os bytes.

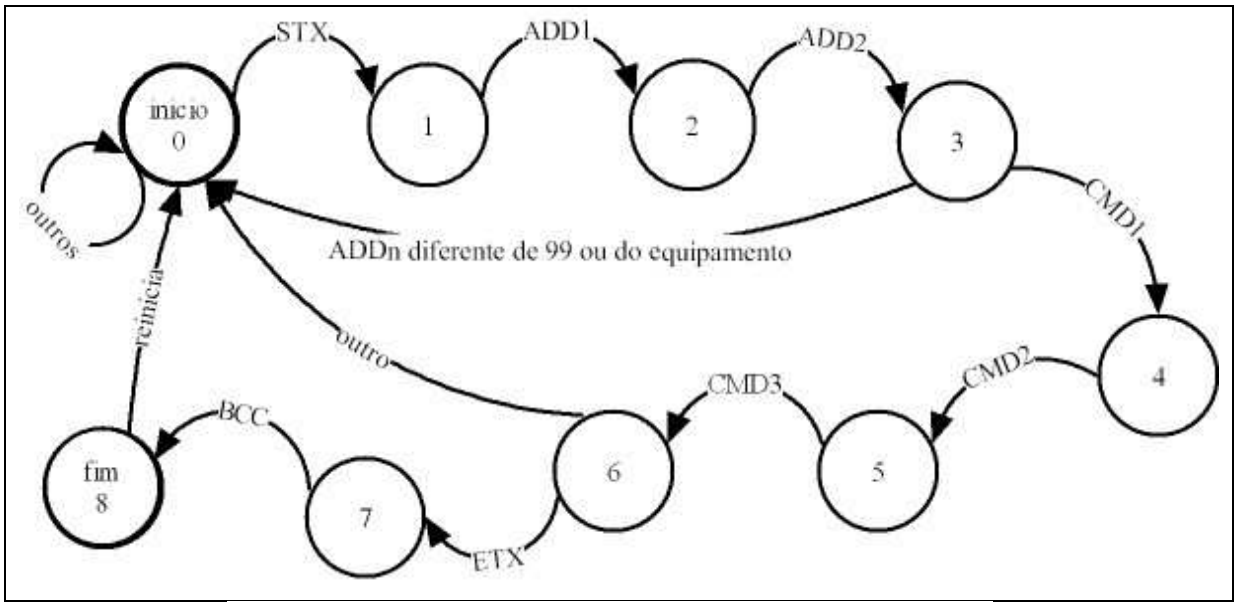


Figura 9 – Autômato de reconhecimento de pacotes no acionador

No autômato pode-se ver que o sistema deverá ficar no estado 0 até que receba um byte STX. Qualquer outro byte recebido será ignorado. No estado 1 o sistema aguarda o byte de endereço, seguido pelo segundo byte de endereço. No estado 3 o autômato é incrementado para acomodar um pequeno teste de comparação ao endereço. Se o endereço for do acionador ou se for um endereço de *broadcast*, que é o endereço 99, o sistema irá aguardar pelos bytes de comando até chegar no estado 6, caso contrário deverá retornar ao estado 0 pois o pacote é destinado a outro endereço. O estado 6 aguarda o byte ETX, que se não for recebido ou se for recebido um diferente, deverá voltar ao estado 0 também. Finalmente o último byte do pacote é o BCC, que assim que for recebido passará ao estado 8, onde o pacote é processado pelo acionador e respondido. É importante ressaltar que se ocorrer um evento de *timeout* em qualquer um dos estados, o autômato voltará ao estado 0.

A central também possui um autômato muito semelhante ao dos acionadores, mas a diferença está na comparação que é feita no estado 3. Na central o autômato continuará recebendo o pacote se o endereço for de 01 até 32 pois estes são os endereços dos transmissores da mensagem. O autômato da central pode ser visto na figura 10.

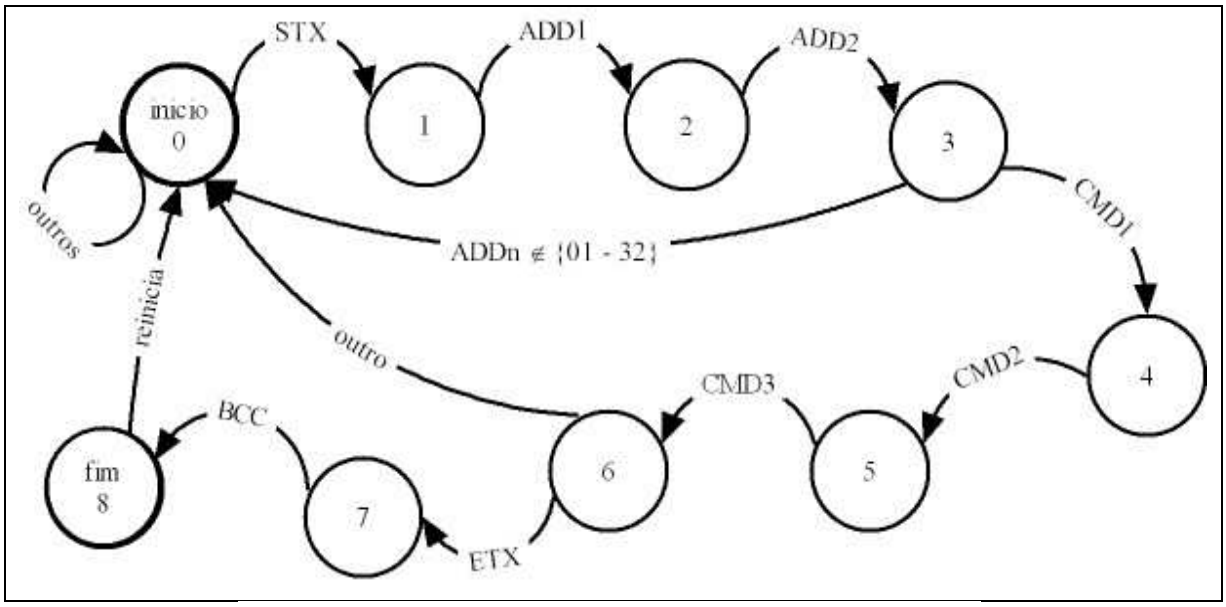


Figura 10 – Autômato de reconhecimento de pacotes na central

3.2.1.2 Protocolo PC-Central

Ao contrário do protocolo central-acionadores, o PC-central é cliente-servidor desde o início do projeto. O PC manda a requisição e a central responde. A diferença está nos tipos de requisições que podem ser feitas.

O tamanho do pacote do PC varia de acordo com a requisição que está sendo feita. O quadro 4 mostra o pacote referente aos comandos “A”, “L”, “D”, “H”, “I” e “O”.

B. 1	**B. 2**	**B. 3**	**B. 4**	**B. 5**	**B. 6**	**B. 7**	**B. 8**	**B. 9**	**B. 10**
STX	ADC1	ADC2	ADD1	ADD2	CMD1	CMD2	CMD3	ETX	BCC

Quadro 4 – Formato do pacote de comunicação PC → central

O byte 1 é o STX e junto com o byte 9, ETX, e 10, BCC, têm as mesmas funções do protocolo central-acionadores. Os bytes 2 e 3, ADC1 e ADC2, são o endereço da central que se destina o pacote pois em implementações futuras poder-se-á existir várias centrais. ADD1 e ADD2 são os endereços dos acionadores, para que estes possam ser controlados pelo PC também. CMD1, CMD2 e CMD3 são bytes de comandos, semelhantes àqueles vistos no protocolo central-acionadores.

No quadro 5 esta o pacote para o comando “S” que é o comando de ajuste de data e hora para a central.

B. 1	B. 2	B. 3	B. 4	B. 5	B. 6	B. 7 até 18	B. 19	B. 20
STX	ADC1	ADC2	ADD1	ADD2	S	hhmmssddmmaa	ETX	BCC

Quadro 5 – Pacote com o comando de ajuste de data e hora

Este pacote existe para ajustar facilmente pelo PC a hora da central, mas deve ser usado com cuidado para não prejudicar os horários do histórico de eventos da central. O quadro 6 mostra o pacote de alteração de nome de acionador no cadastro da central, que é passado pelo comando “N”.

B. 1	B. 2	B. 3	B. 4	B. 5	B. 6	B. 7 até 26	B. 27	B. 28
STX	ADC1	ADC2	ADD1	ADD2	N	Nome acionador	ETX	BCC

Quadro 6 – Pacote com o novo nome do acionador

O envio de nome de acionador para a central possui um tamanho fixo reservado ao nome, que é de 20 bytes. Caso o nome cadastrado no PC seja menor que isso, deverá ser preenchido com espaços em branco até completar o tamanho reservado. A central ao receber o pacote de alteração de nome, deve cadastrá-lo em sua EEPROM.

Em todos os 3 pacotes ADC é sempre 01 pois existe apenas uma central neste projeto. ADD deverá ser 00 se o comando for a central, ou um endereço de 01 a 32 se for um comando que deverá ser repassado ao acionador. CMD2 e CMD3 ficam reservados para implementações futuras e CMD1 é descrito no quadro 7 juntamente com os comandos “S” e “N”.

Caractere	Valor da Tabela ASCII	Descrição
A	65	Requisição de *status* dos acionadores
L	76	Liga algum acionador (necessário endereço do acionador)
D	68	Desliga algum acionador (necessário endereço do acionador)
H	72	Requisita hora da central
I	73	Requisita um registro do histórico dos acionadores presente na central
O	79	Registro do histórico recebido com sucesso, enviar próximo registro
S	83	Envio de Data e Hora para a central (pacote com 20 bytes)
N	78	Envio de nome do acionador (pacote com 28 bytes)

Quadro 7 – Comandos passados em CMD1 pelo PC

O autômato de reconhecimento do pacote vindo do PC é muito semelhante aos outros autômatos vistos até agora, apenas acrescentando-se dois estados a fim de receber o endereço da central.

Os pacotes enviados da central para o PC são bastante variáveis. No quadro 8 o pacote com o *status* dos acionadores que é enviado sempre que os comandos “A”, “L” ou “D” são recebidos pela central.

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5 até 36**	**Byte 37**	**Byte 38**
STX	ADC1	ADC2	1	ESTADOS	ETX	BCC

Quadro 8 – Formato do pacote de envio de status ao PC

ADC1 e ADC2 deverá ser o endereço da central, que no caso será sempre 01. O quarto byte é o caractere “1”, que indica ao PC que o resto do conteúdo serão estados dos acionadores daquela central. ESTADOS é um número fixo de caracteres seqüenciais onde o primeiro representa o acionador 1, o segundo é o acionador 2 e assim por diante até o acionador 32. Cada byte da seqüência de estados pode possuir apenas quatro valores: “1”, “2”, “0” ou “x”. O valor “1” indica que o dispositivo do acionador esta ligado, o valor “2” indica que o dispositivo esta desligado, o valor “0” indica que o acionador não existe e o valor “x” indica que o acionador existia mas parou de responder por algum motivo.

No quadro 9 o pacote com a data e hora da central.

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5 até 16**	**Byte 17**	**Byte 18**
STX	ADC1	ADC2	2	hhmmssddmmaa	ETX	BCC

Quadro 9 – Formato do pacote de envio da hora da central ao PC

O quarto byte indica que o pacote é um envio de data e hora para o PC poder sincronizar os relógios e os 12 bytes seguintes são referentes a data e a hora configurados na central. Este comando é especialmente importante para o PC poder sincronizar com o próximo comando que é o histórico do que aconteceu com os acionadores. O histórico é registrado usando a data e hora da central.

No quadro 10 o pacote com um registro do histórico.

B. 1	**B. 2**	**B. 3**	**B. 4**	**B. 5 até 16**	**B. 17**	**B. 18**	**B. 19**	**B. 20**	**B. 21**
STX	ADC1	ADC2	3	hhmmssddmmaa	ADD1	ADD2	status	ETX	BCC

Quadro 10 – Formato do pacote de envio de registro histórico ao PC

A fila de histórico da central guarda apenas os eventos ocorridos nos acionadores. Evento é qualquer mudança de estado, como por exemplo o ligar e o desligar de um dispositivo. Este pacote de envio de registro do histórico envia um registro de evento por vez, onde os bytes 5 até 16 indicam a data e hora do evento, os bytes ADD1 e ADD2 indicam o endereço do acionador que realizou o evento e o byte 19 é o novo estado do acionador, que pode ser “1” (ligou), “2” (desligou) ou “3” (perdeu contato). Assim que o PC recebe esse

pacote ele deve responder com o comando “O” que significa que recebeu o pacote e quer outro registro do histórico. Assim a central retira o registro da fila e envia o próximo.

O autômato de reconhecimento de pacotes do PC é ligeiramente diferente dos autômatos apresentados até agora. Ele está preparado para receber quantos caracteres forem necessários até que apareça um caractere ETX. Na figura 11 está o autômato. Os caracteres são armazenados em uma variável e só depois que a transmissão do pacote for concluída é que o pacote é analisado e processado.

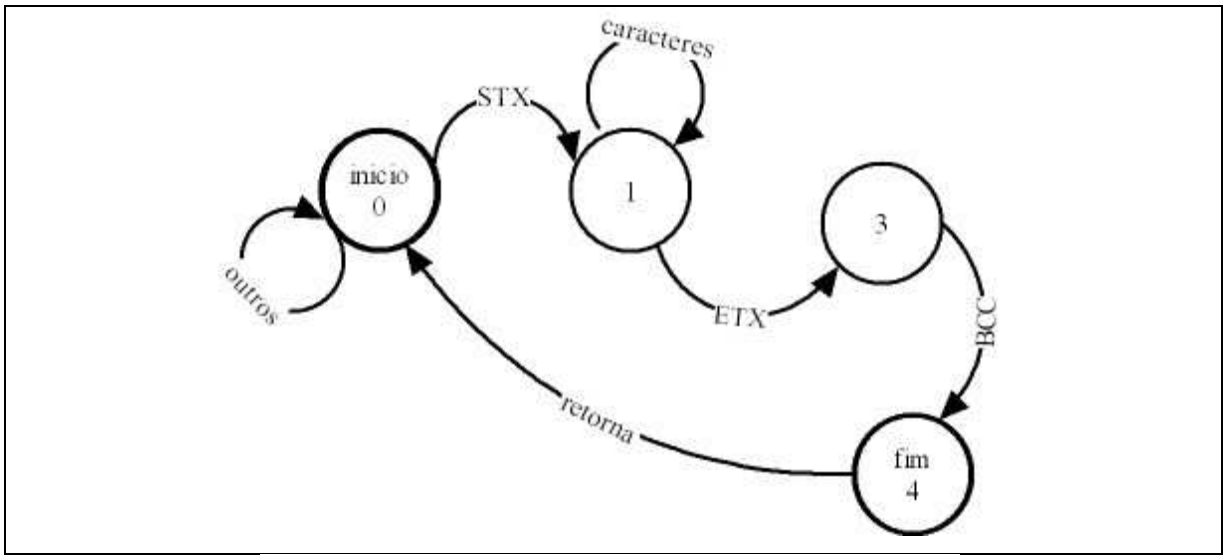


Figura 10 – Autômato de reconhecimento de pacotes no PC

3.2.2 Acionadores

Acionadores são equipamentos microcontrolados que devem ser instalados dentro de tomadas ou interruptores, como os da figura 11. Para respeitar o requisito de não interferir na estética da residência ele deve ter o tamanho reduzido e ter algumas características que os permitam se adaptar aos dispositivos elétricos.

A especificação dos acionadores é dividida em especificação física e especificação do software. Cada um desses assuntos é tratado em um tópico a seguir.

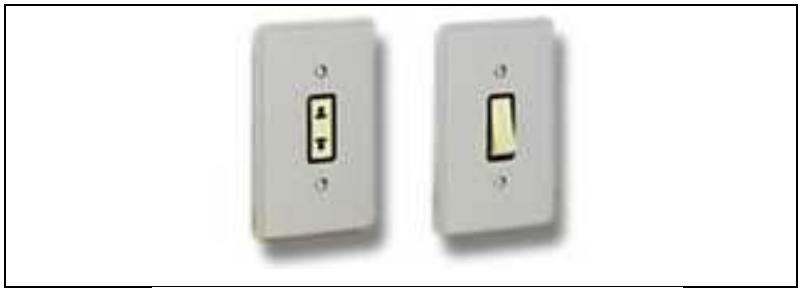


Figura 11 – Exemplo de tomada e interruptor

3.2.2.1 Especificação Física

Especificação física diz respeito a parte eletrônica do sistema. O acionador precisa possuir algum meio de acionar dispositivos elétricos, possuir uma interface RS-485 para comunicação com a central, ter algum tipo de entrada binária, possuir seguranças para evitar que queime o microprocessador e ter um tamanho reduzido de tal forma que caiba dentro de uma tomada convencional.

O esquema do acionador é apresentado na figura 12. No projeto são previstas uma entrada e uma saída, entretanto o esquema é mostrado com quatro entradas digitais, duas saídas, dois botões sobe-desce, termômetro e *dimmer* para futuras implementações.

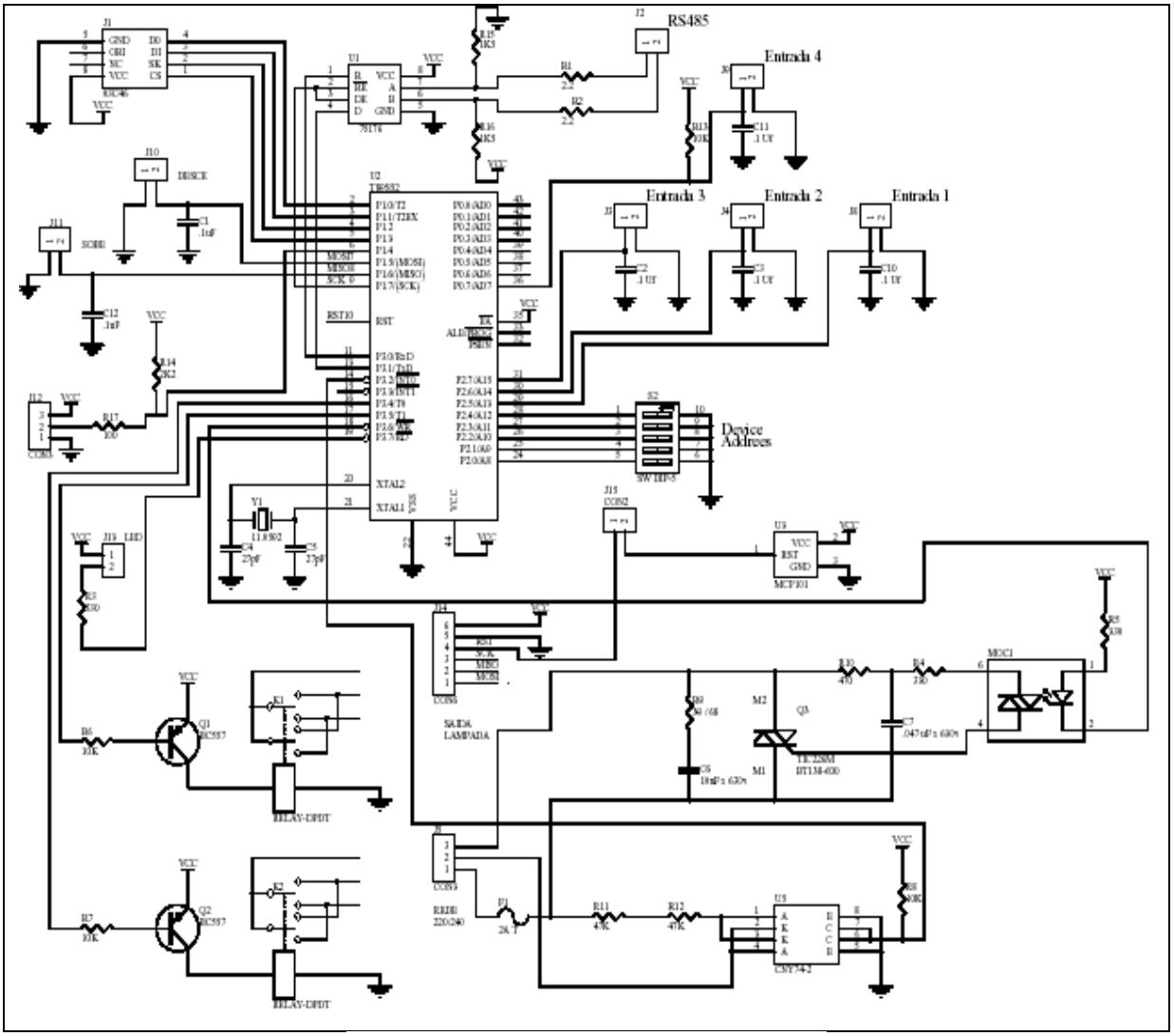


Figura 12 – Esquema elétrico do acionador

O microcontrolador é o componente U2 enquanto U1 é a interface RS-485 do sistema,

que é ligada nas entradas seriais de U1. As pontes ligadas na Port 2, bits 0 até 4, são os configuradores de endereço do acionador. Através de uma seqüência binária é possível alterar o endereço do acionador.

Para implementações futuras está esquematizado um controlador de intensidade luminosa (*dimmer*), que é a parte do esquema representado pelo acoplador ótico MOC1, pelo chaveador TIC226M e pelo detector de passagem por zero CNY74-2. Mas como não faz parte dos requisitos do trabalho, não serão dedicadas maiores explicações sobre essa parte.

O acionador não pode ser reduzido a ponto de comportar um transformador ligado diretamente na rede elétrica da residência (220/110 V), portanto, precisa ser alimentado com uma bateria externa. Para isso a tensão da bateria é passada no mesmo lado da comunicação RS-485 (logo precisa-se de um cabo com dois pares de fios). Essa tensão fornecida pela bateria de 12V precisa ser regulada no acionador para se obter 5V. Para isso é apresentado também um pequeno regulador de tensão na figura 13.

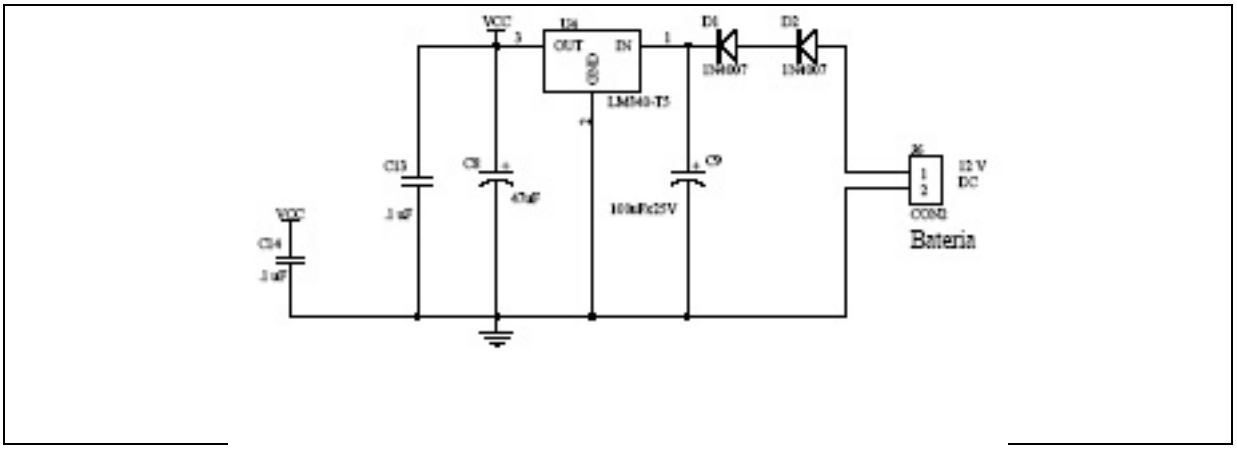


Figura 13 – Esquema elétrico do regulador de tensão do acionador

Com o esquema pronto pode-se usar um software de desenho da placa elétrica. As figuras 14 e 15 mostram os desenhos prontos das partes superior e inferior, respectivamente.

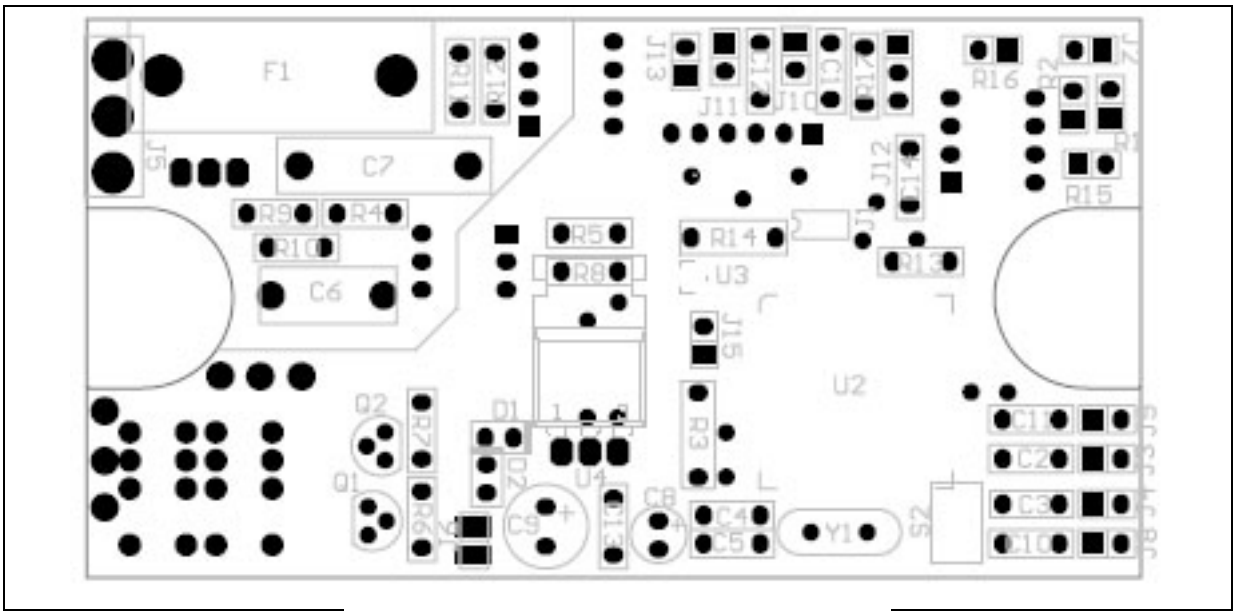


Figura 14 – Parte superior da placa acionadora

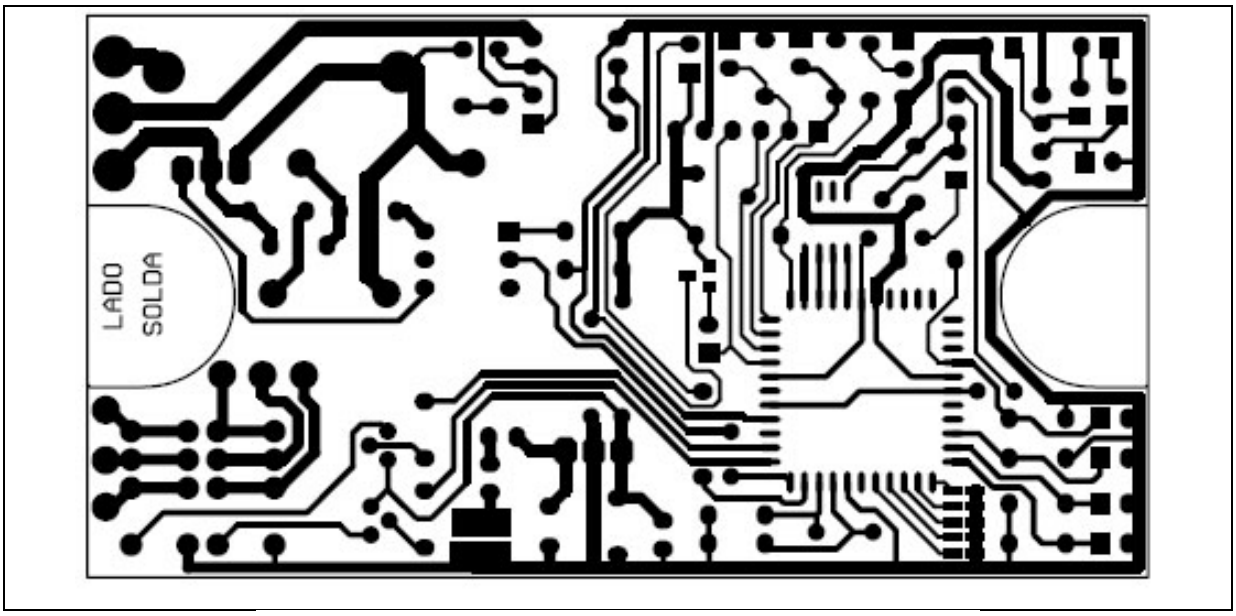


Figura 15 – Lado inferior da placa acionadora – lado da solda

É importante ressaltar a facilidade que o software proporciona pois ele pré-gera a placa com base no esquema elétrico, restando alguns ajustes depois. O software utilizado foi o Protel 99 SE (PROTEL, 2000).

3.2.2.2 Especificação do Software

São usados fluxogramas para especificar os algoritmos do acionador. O compilador utilizado não possui orientação a objetos e por isso não são discutidos diagramas de classe e

outros da UML.

O software deverá ficar num *loop* fazendo as verificações nas entradas do acionador (eventos) e na transmissão de dados. A figura 16 mostra o fluxograma principal do sistema, onde antes de iniciar o *loop* são iniciadas algumas variáveis para controle.

A inicialização de variáveis do microcontrolador é feita atribuindo valores aos registradores de configuração das interrupções, velocidade do *timer* interno e velocidade da transmissão serial. Logo em seguida é lido o endereço do acionador que, como visto no esquema elétrico, é configurado através de pontes (*jumpers*).

As variáveis de controle do sistema são *flags* e contadores para auxiliar os processos em outras rotinas. São variáveis globais e por isso precisam de um cuidado especial.

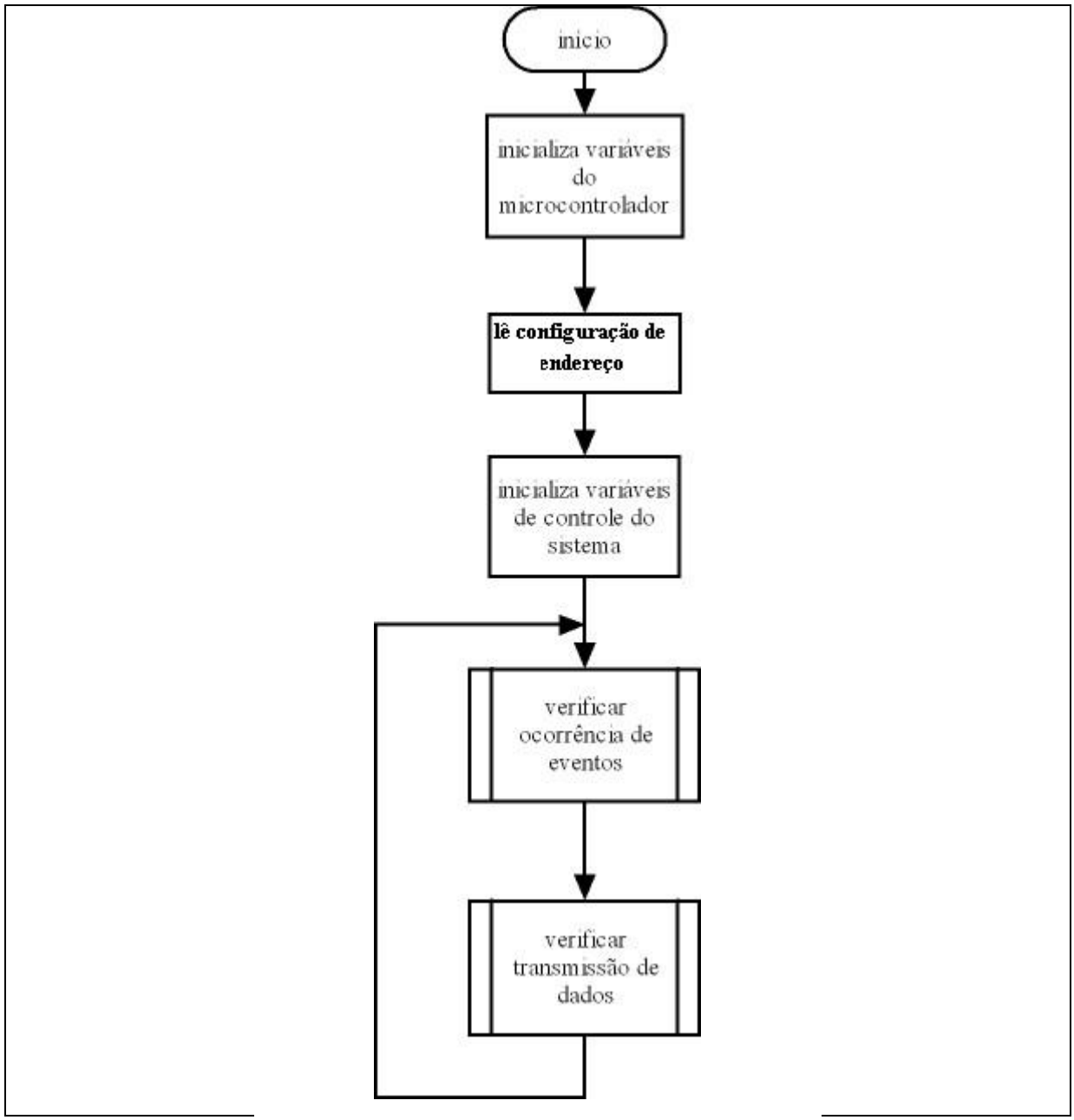


Figura 16 – Fluxograma – rotina principal do acionador

A sub rotina `verificar_ocorrência_de_eventos` irá verificar quando um interruptor mudou de estado. Se o interruptor for ligado ou se for desligado isso irá gerar um evento ao acionador. Esse evento muda o estado interno do acionador para o oposto do estado atual. Se estiver no estado ligado, ficará desligado, e vice-versa. O estado do interruptor não será sempre igual ao estado interno do acionador, pois o estado do acionador pode ser alterado pela central controladora, sem mexer no interruptor. A figura 17 mostra o algoritmo da verificação de eventos.

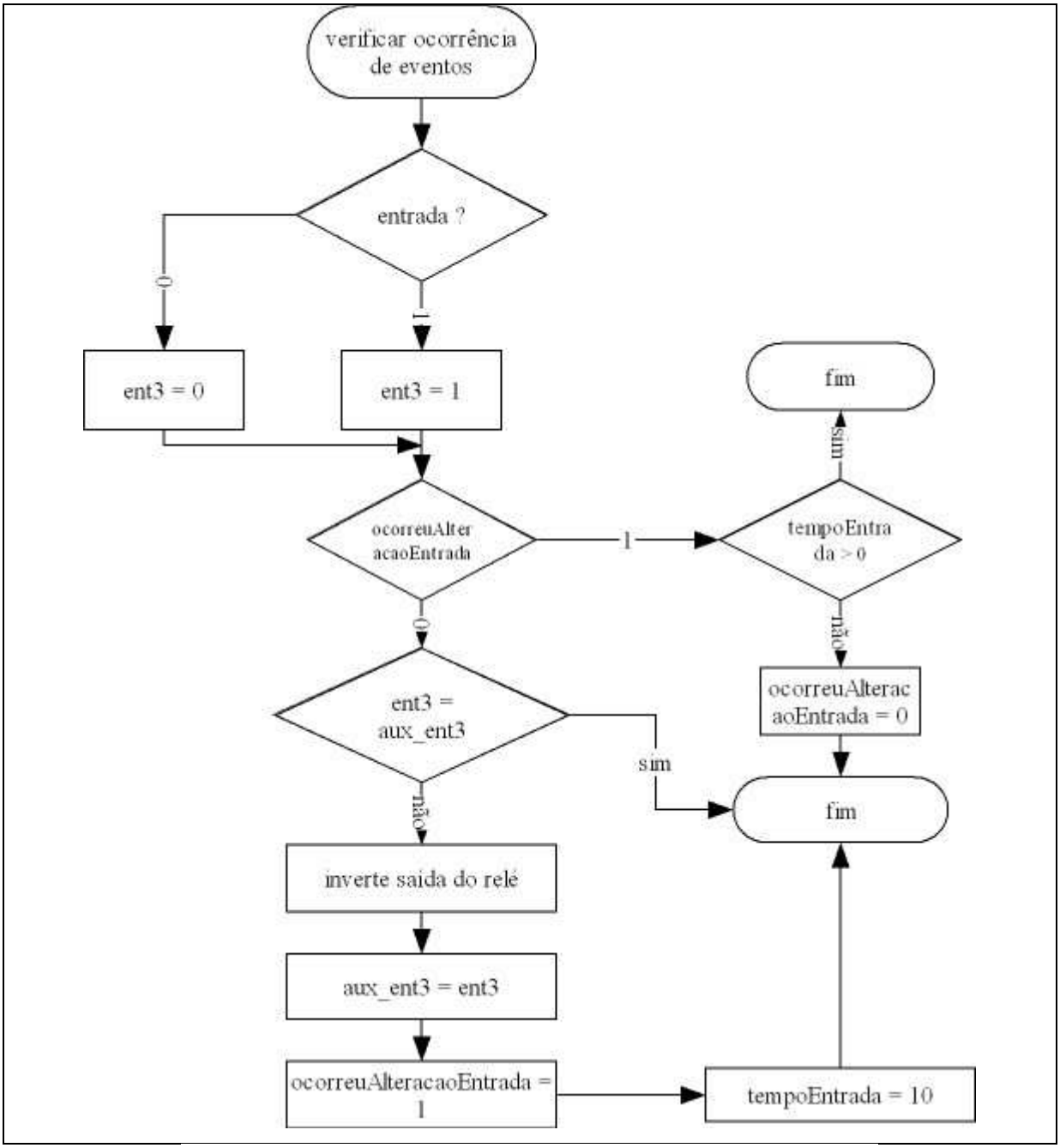


Figura 17 – Fluxograma – rotina verificação de eventos do acionador

A primeira pergunta feita é qual o estado da `entrada`, que é fisicamente o estado do interruptor. Então é atribuído um valor a `ent3` que é a variável que será comparada com o

estado anterior `aux_ent3` do acionador. Se forem diferentes então a saída, representada no sistema pelo relé, é invertida e o *flag* se ocorrência de alteração na entrada é acionado, impedindo que aconteça alguma alteração no sistema pelo interruptor nos próximos 10ms. Esse intertravamento é feito para evitar oscilações no chaveamento do interruptor, impedindo que o acionador deixe a lâmpada piscando momentaneamente. A variável `tempoEntrada` é subtraída de uma unidade a cada milésimo de segundo dentro da rotina de interrupção do `timer0` do microcontrolador. Este artifício evita de o acionador ficar parado esperando os 10ms em uma rotina `Delay(10)`.

Na figura 18 esta a rotina de verificação de transmissão de dados.

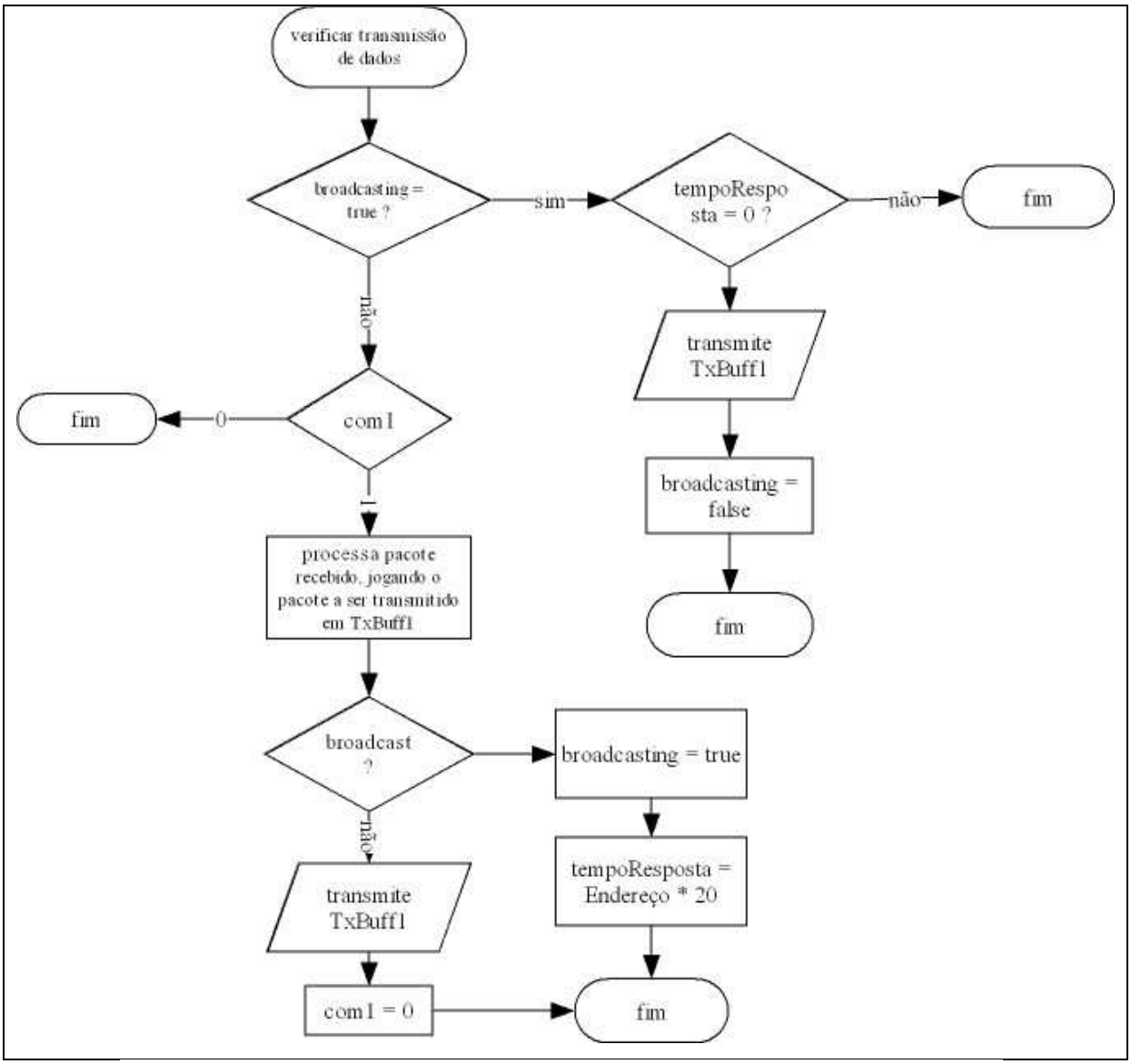


Figura 18 – Fluxograma – rotina verificação de transmissão de dados do acionador

A seguir uma explicação sobre as principais variáveis da rotina de transmissão de dados:

a) `broadcasting`: é um *flag* que indica ao sistema quando o acionador estiver em

modo de espera para um comando de *polling* da central;

b) `tempoResposta`: é o contador de tempo restante para o acionador responder ao comando de *polling* e sempre é iniciado com o produto do endereço do acionador por 20;

c) `com1`: *flag* indicando que um comando chegou ao microcontrolador pela porta serial;

d) `TxBuff1`: *buffer* guardando o pacote a ser transmitido pelo acionador.

O processamento do pacote recebido é feito pelo acionador de acordo com o comando recebido. Seguindo o protocolo, um comando “A” é uma requisição de *status*, “L” é ligar o acionador (a saída do acionador, que é o relé) e um “D” é para desligar. O acionador sempre responde apenas o seu *status* e por isso não há qualquer verificação do comando de entrada para transmitir o pacote.

É dentro da interrupção da porta serial do acionador que o pacote é validado. Caso o pacote recebido seja endereçado ao acionador, então o valor “1” é atribuído à variável `com1`, podendo então ser processado na rotina de verificação de transmissão de dados.

3.2.3 Central Controladora

A central controladora é a parte do projeto mais crítica. É nela que todos os acionadores serão controlados, que permitirá um acompanhamento através de uma Interface Homem Máquina (IHM), e que fará a comunicação com o software supervisor no PC. O Data Term é utilizado como hardware para a central e o software é especificado em fluxogramas pois, assim como o acionador, não comporta orientação a objetos.

3.2.3.1 Data Term

Data Term é um equipamento para aplicações diversas de coleta de dados, comunicação com outros dispositivos e Interface Homem Máquina (IHM). É desenvolvido pela Automasoft Sistemas e Serviços Ltda e, entre as principais características, destacam-se:

a) processador MCS51 T89C51RD2;

b) 64kb de flash interna;

c) 32kb de memória RAM externa;

d) um canal serial RS-485;

e) um canal serial RS-232;

f) display gráfico de 80x160 pixels;

g) teclado alfanumérico;

h) funções proprietárias para acesso ao hardware;

i) entradas e saídas digitais expansíveis.

A figura 19 apresenta uma foto do Data Term utilizado neste trabalho. Nela é possível ver as entradas seriais na parte inferior do equipamento, o display na parte frontal superior e o teclado alfanumérico.

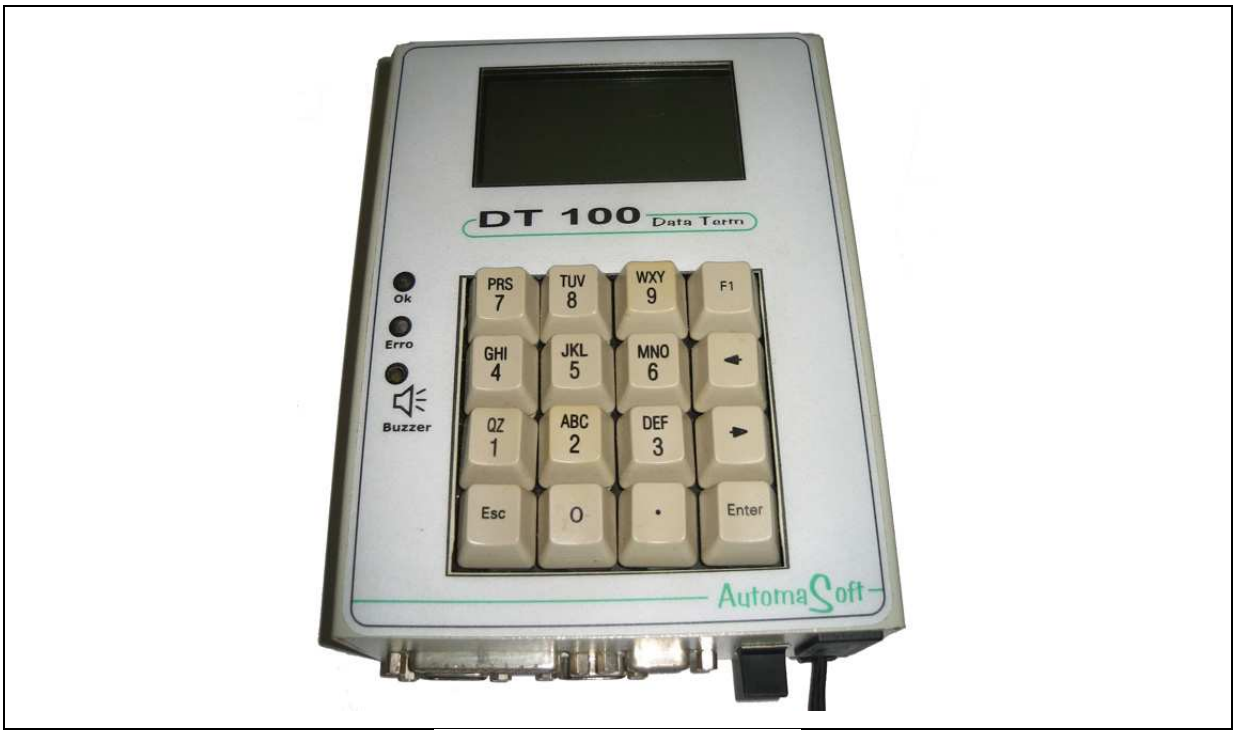


Figura 19 – Foto do Data Term

3.2.3.2 Software da Central

A central possui algumas telas que serão mostradas no *display* do Data Term. Na figura 20 está o desenho dessas telas com o seu respectivo número de identificação.

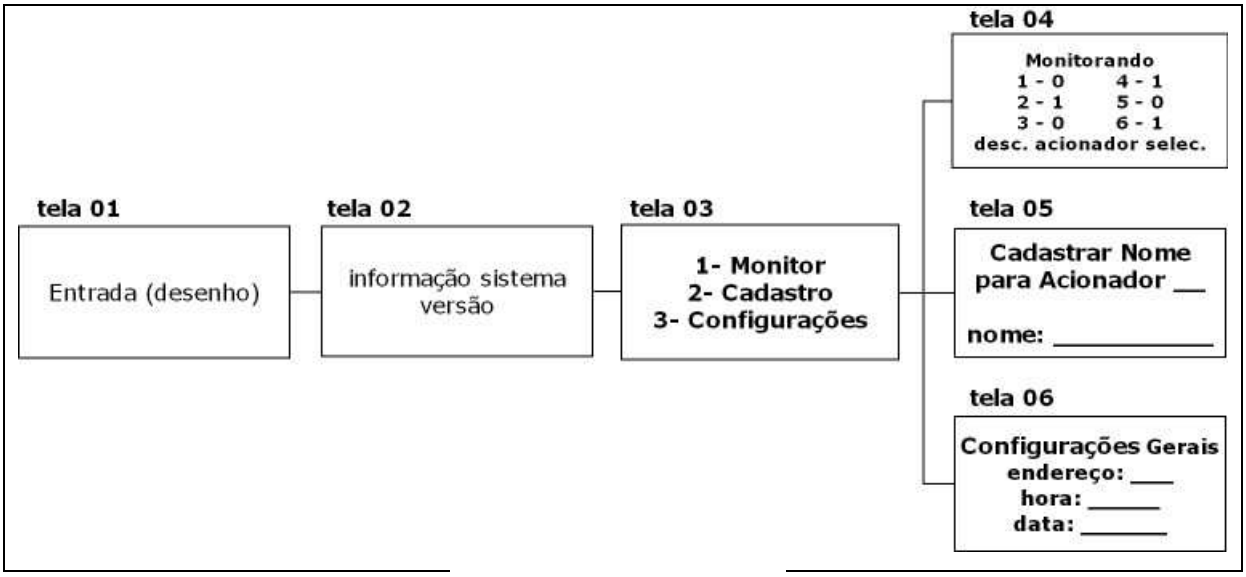


Figura 20 – Telas da Central

As telas 1 e 2 são apenas para apresentação da central. O menu principal é a tela 3 e é esta que leva às telas 4, 5 e 6. A tela 4 é a tela de monitoração dos acionadores, onde deverão ser mostrados todos aqueles que responderem ao comando de *polling* da central e, se clicarmos com as flechas para esquerda ou direita deverá ir selecionando acionador por acionador e aparecendo o nome cadastrado dele. A tela 5 serve para cadastrar os nomes dos acionadores que serão mostrados na tela 4 e finalmente a tela 6 serve para configurações da central, como o endereço (que em implementações futuras poderá ser diferente de 1), a data e hora.

Semelhante ao programa principal do acionador, o software da central também possui uma rotina principal que entrará em *loop* para chamar várias sub-rotinas. A Figura 21 mostra o fluxograma principal.

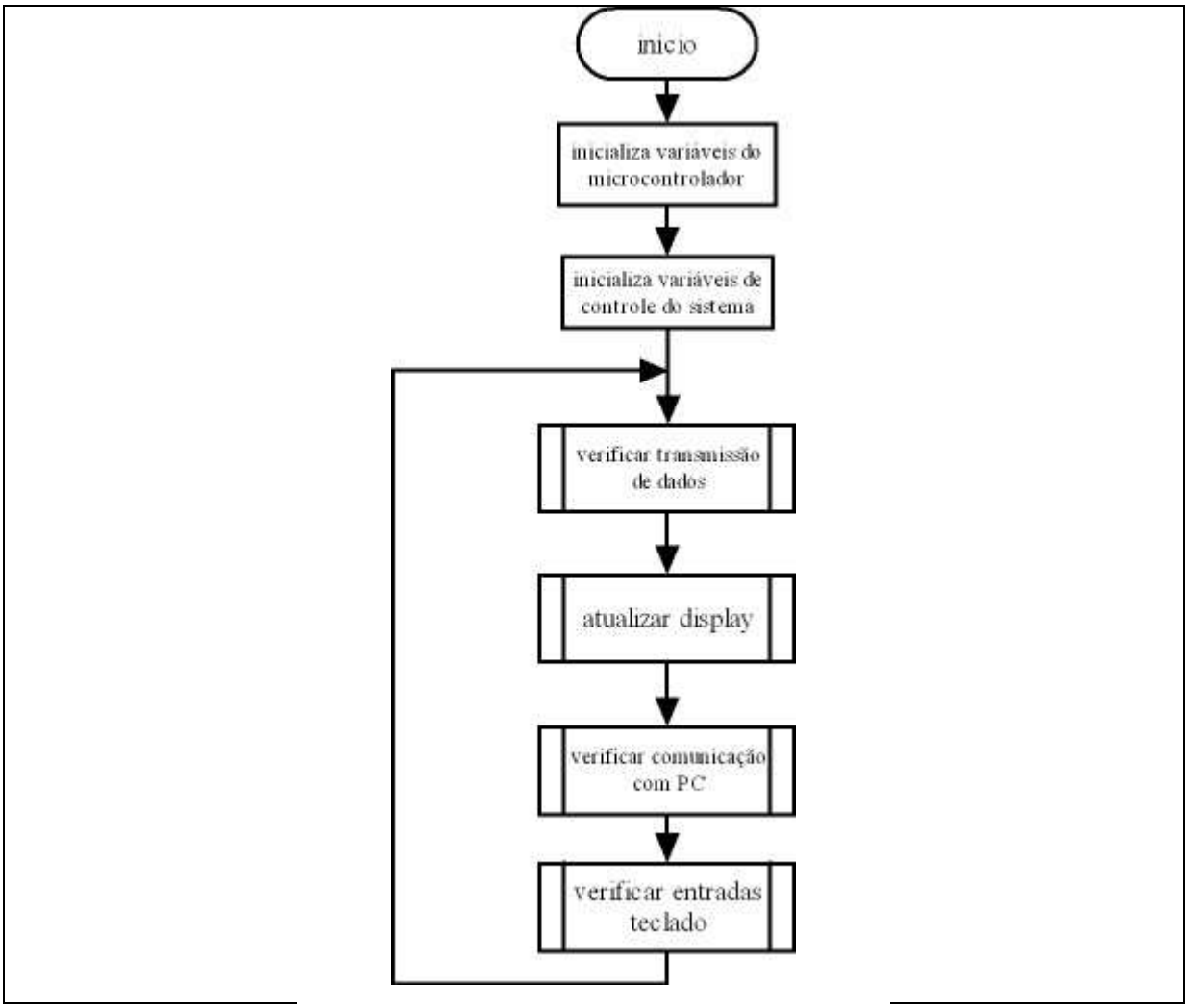


Figura 21 – Fluxograma – rotina principal da central

Antes de entrar no *loop* a central inicia as variáveis internas do microcontrolador para configurar as interrupções, as velocidades de transmissão e dos *timers*. Em seguida as variáveis do sistema são inicializadas.

O *loop* executa quatro grandes sub-rotinas, cada qual com seu respectivo fluxograma, descrito a seguir.

3.2.3.2.1 Sub-rotina Verificar Transmissão de Dados

A figura 22 mostra o fluxograma de verificação da transmissão serial entre central e acionadores.

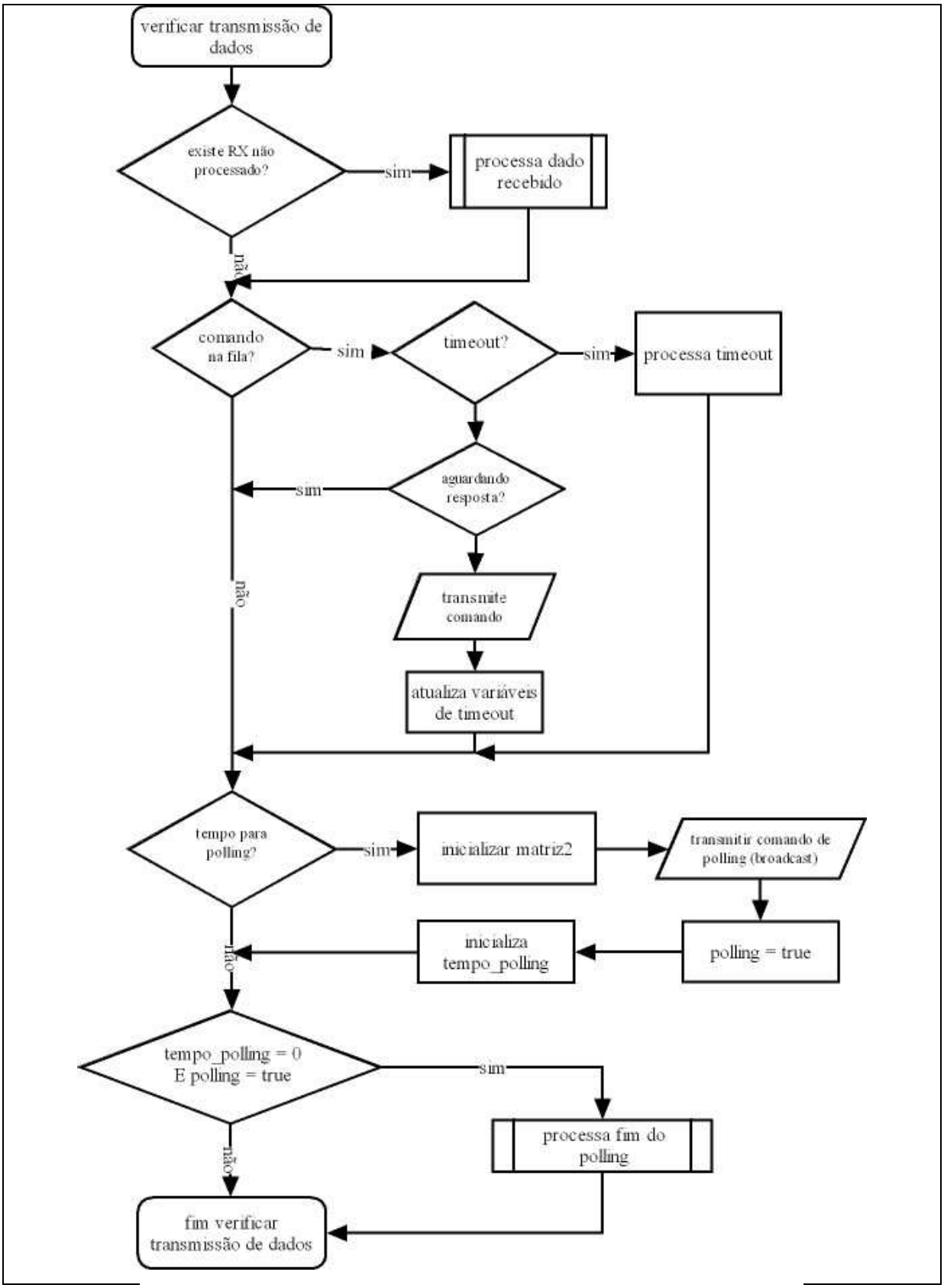


Figura 22 – Fluxograma – rotina verificar transmissão de dados da central

Na central existem duas matrizes (`matriz1` e `matriz2`) de trinta e duas posições. Cada valor da matriz representa um acionador, sendo o primeiro valor da matriz, sempre o

acionador 1 e seguindo em ordem crescente até o trigésimo segundo elemento da matriz. A `matriz1` representa o estado dos acionadores e a `matriz2` é o estado dos acionadores recebidos no comando de *polling*, para poder comparar com a `matriz1` e verificar se algum acionador não respondeu.

Tempo para *polling* é uma variável que deverá estar válida de um em um segundo pois ela sinaliza quando um novo *polling* deve acontecer.

`tempo_polling` é uma variável que conta o tempo de duração do *polling* para que a central possa controlar quando um dado é recebido durante o período de *polling.* Este tempo é contado no mesmo instante que o `tempo_para_polling`, só que este dura apenas 660ms.

`polling` é o sinalizador de que a central está em modo *polling.*

Existem duas filas circulares na central. Uma das filas é para dados recebidos e outra fila é para dados que devem ser enviados. Sempre que um pacote é recebido pela central, deve ser processado. A figura 23 mostra o fluxograma do processamento de dado recebido.

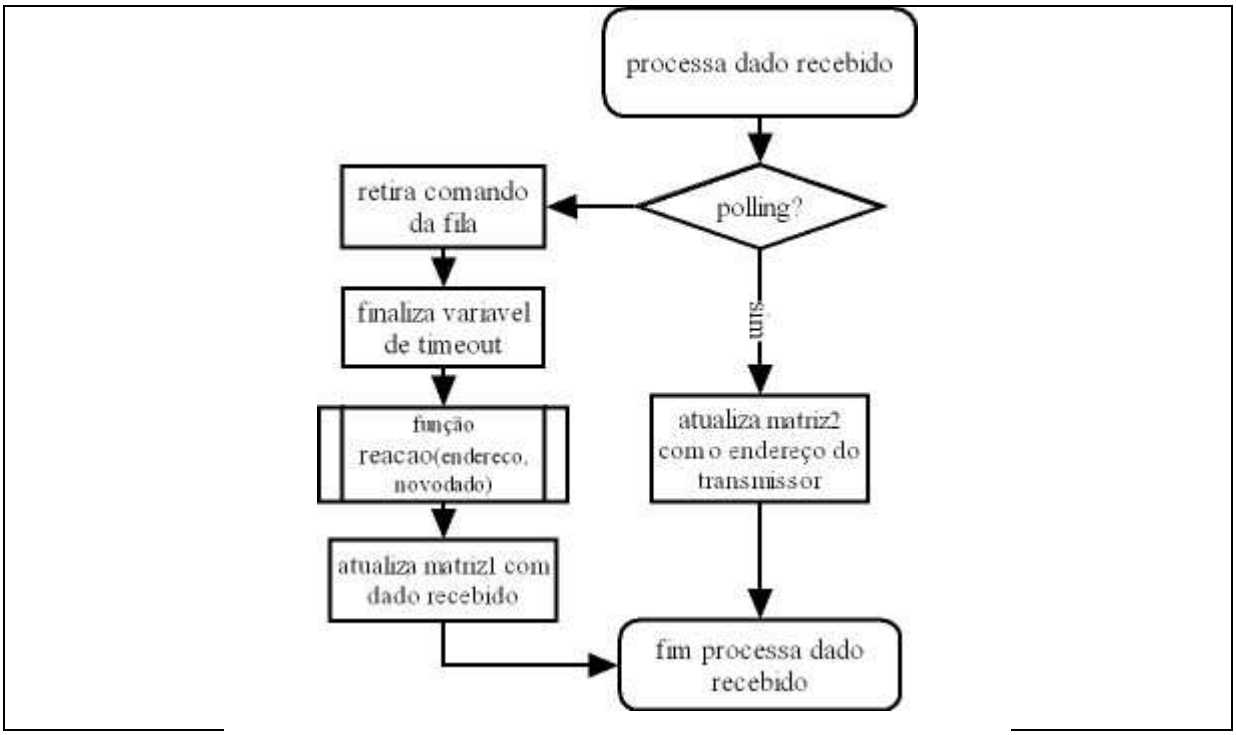


Figura 23 – Fluxograma – rotina processa dado recebido da central

Sempre que um dado é recebido a rotina verifica se a central está em modo de *polling*. Se estiver, então coloca na `matriz2` o valor recebido na posição referente ao endereço do acionador, se não retira o comando da fila de comandos a serem enviados, finaliza a variável de timeout, executa a `função_reação` e atualiza a `matriz1` com o dado recebido.

`Função_reação` é a função que atualiza a `matriz1` caso algum acionador não responda e é também, em implementações futuras, a função que acionará alarmes de acordo com ações

programadas a eventos.

Sempre que o período de tempo do *pollling* acaba, deve ser executada a sub-rotina `processar_fim_do_polling`. A figura 24 mostra o fluxograma desta sub-rotina.

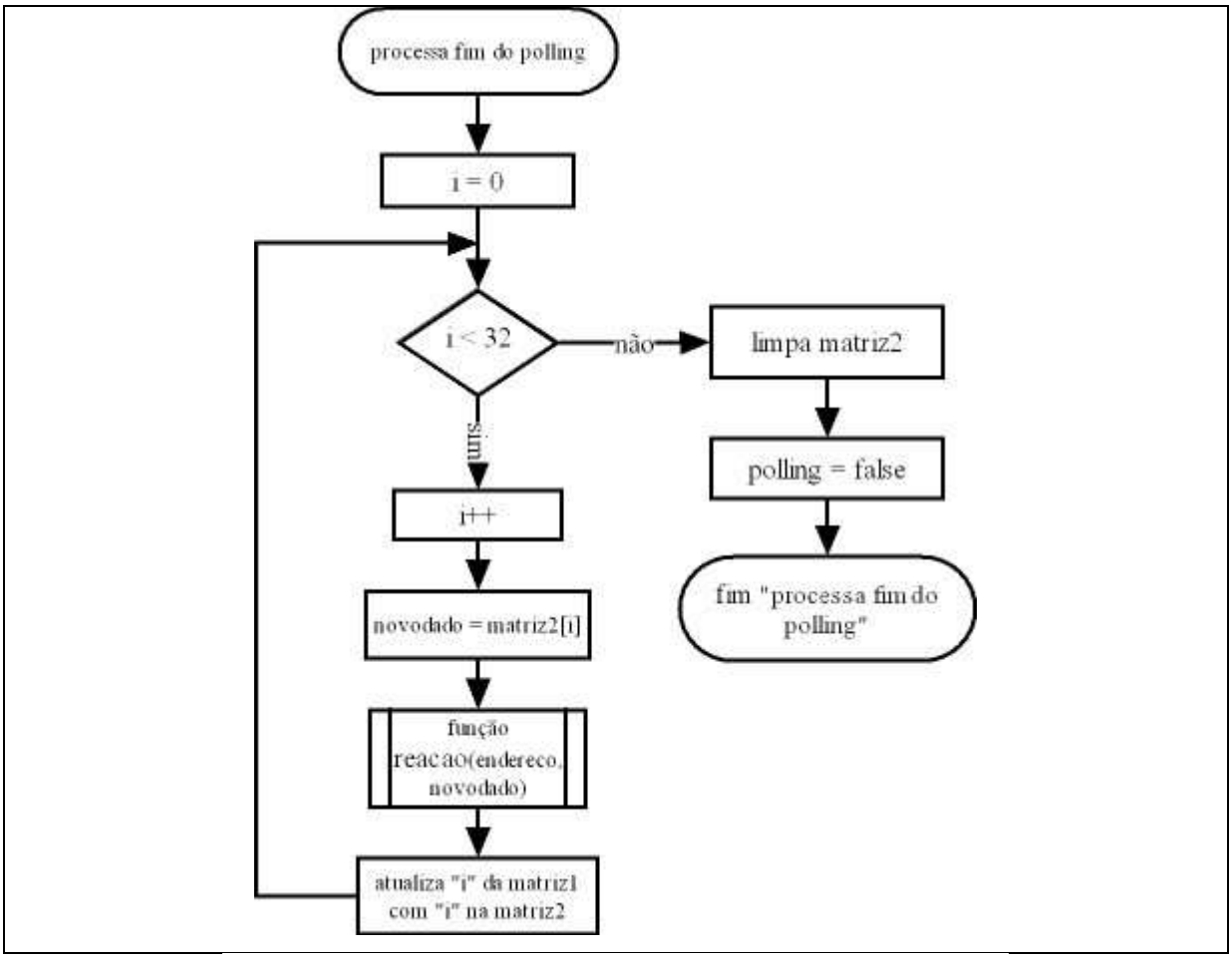


Figura 24 – Fluxograma – rotina processa fim do *polling* da central

Na rotina processa fim do *polling* é feita uma varredura na `matriz2` (onde dados do *polling* são colocados) e comparada com a `matriz1`. A função reação irá detectar se algum acionador que estava na `matriz1` não respondeu na `matriz2`, caracterizando perda de conexão (como mostrado na figura 25). No final, os dados da `matriz2` passam para a `matriz1` e caso algum acionador tenha perdido conexão, fica com o *status* "3" na `matriz1`. A `função_reação` também inclui na fila de eventos sempre que um acionador tiver seu estado alterado ou tiver perdido conexão.

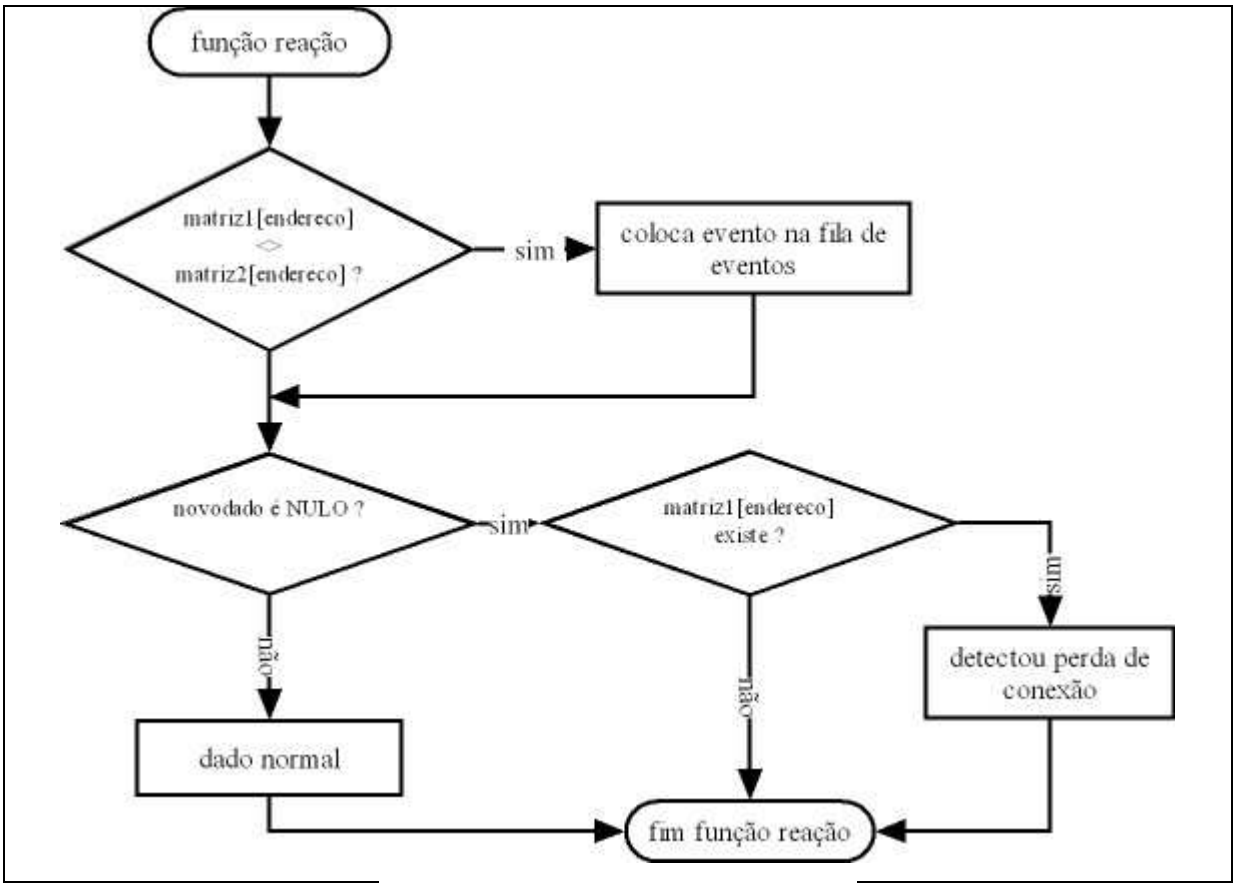


Figura 25 – Fluxograma – função reação

3.2.3.2.2 Sub-rotina Atualizar Display

A rotina de atualizar o *display* está demonstrada na figura 26. É usada uma variável `Display` para saber qual tela deverá ser mostrada. As telas estão na figura 20.

Nas telas 4, 5 e 6 é preciso posicionar o cursor para que o usuário saiba onde ele está no momento. Para isso cada uma das telas deve possuir alguma variável que guarde a posição.

Figura 26 – Fluxograma – rotina atualizar *display* na central

3.2.3.2.3 Sub-rotina Verificar Comunicação com PC

A rotina de comunicação de dados com o PC é mais simples do que com os acionadores. Aqui a central será o cliente e apenas responderá ao PC, sem se preocupar com *timeout* ou qualquer outro artifício de verificação do recebimento da resposta por parte do PC. A figura 27 mostra o fluxograma de comunicação com PC.

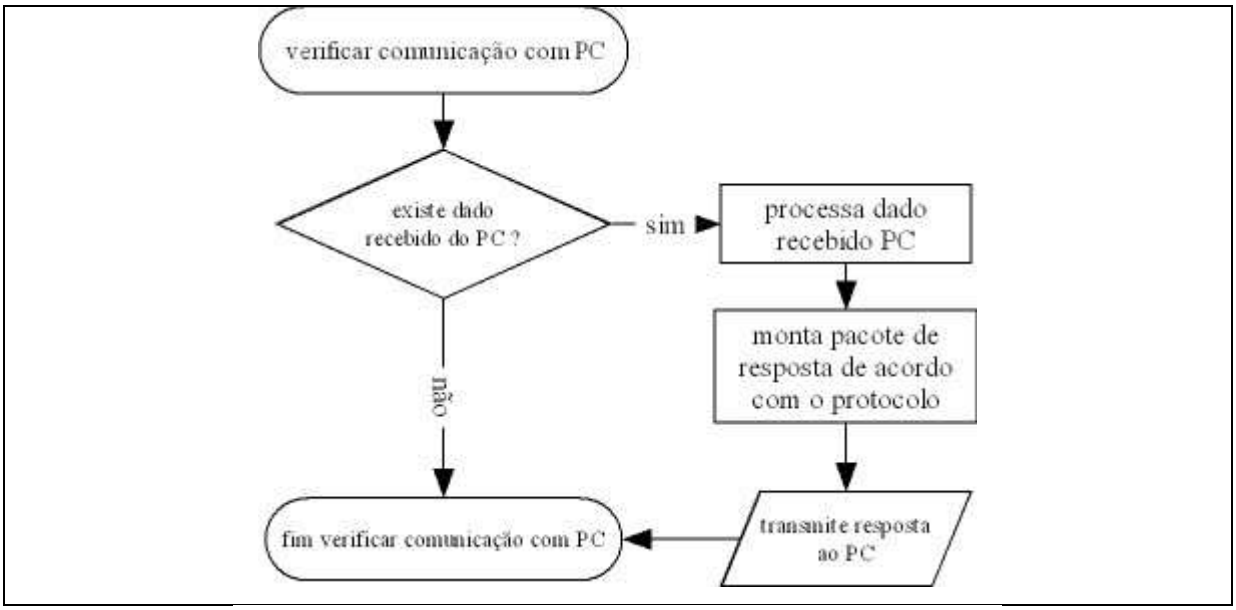


Figura 27 – Fluxograma – rotina comunicação com PC na central

O processamento de dado recebido pelo PC esta parcialmente descrito na especificação do protocolo. Caso um comando de ligar ou desligar algum acionador for recebido, este será enfileirado nos pacotes a serem transmitidos aos acionadores. Se um pacote de alteração de hora for recebido, a hora será alterada e finalmente se um pacote de alteração de nome de acionador for recebido será executada a rotina de gravação de nome na EEPROM do Data Term.

3.2.3.2.4 Sub-rotina Verificar Entradas Teclado

As entradas do teclado são tratadas no sistema de acordo com a tela que está sendo apresentada. Se estiver na tela 1 ou 2, nada acontece. Se estiver na tela 3, então ao pressionar as teclas “1”, “2” ou “3” deverá alterar a variável do display e então a nova tela se apresentará. Na tela “4”, monitoração, sempre que o número “1” ou “2” for pressionado, um comando de ligar ou desligar, respectivamente, deverá ser enfileirado na fila de comandos a serem transmitidos. Se as flechas forem pressionadas então o acionador selecionado é alterado pelo seguinte.

3.2.4 Supervisório PC

O sistema supervisório deverá comunicar serialmente com a central microcontrolada.

O sistema operacional usado é o Linux e o ambiente de desenvolvimento é o Kylix 3. Na figura 28 esta o diagrama de casos de uso.

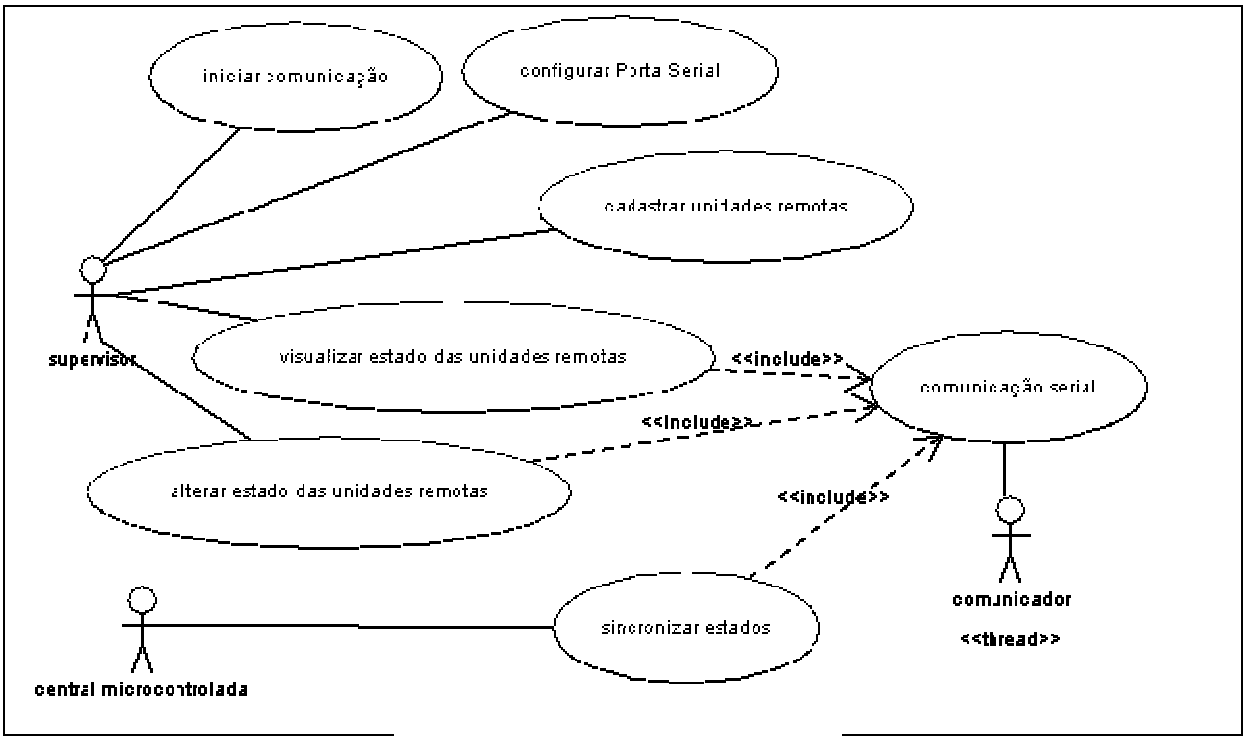


Figura 28 – Diagrama de casos de uso

Supervisor é o usuário humano que deverá operar o sistema. Central microcontrolada é a central que faz a comunicação com os acionadores e comunicador é a classe rodando em paralelo e que controla a comunicação de dados do sistema.

O supervisor pode iniciar a comunicação de dados, pode cadastrar os nomes dos acionadores através do sistema, precisa configurar a porta serial antes de iniciar a comunicação, pode visualizar os estados dos acionadores e alterá-los.

A central microcontrolada apenas faz o envio dos dados requisitados pelo supervisor enquanto o comunicador faz a transmissão de dados de entrada ou saída entre PC e central.

A figura 29 mostra o diagrama de classes do sistema supervisório. A principal classe é `SupervisorPrincipal` que fará a gerência das outras classes, principalmente das 32 `UnidadeRemota` que deverão ser instanciadas no início da execução. `UnidadeRemota` por sua vez faz o controle do que deverá ser mostrado ao supervisor do sistema, sendo que ela controla diretamente a interface `frmAcionador`.

`FilaTransmissao` é a classe que fará o intermediário entre dados de entrada e saída. Trata-se de duas filas que precisam de controle na inclusão e retirada dos dados, pois estes são feitos concorrentemente pelos formulários e pela classe `Comunicador`. Para evitar problemas de concorrência, é usada a classe `TCriticalSection` em cada inclusão ou retirada das filas.

`Comunicador` fará a transmissão de dados. Ele faz a leitura dos dados da fila de saída e se tiver algo, envia a central. E ao mesmo tempo, se receber algum pacote válido da central, colocará este na fila de entrada.

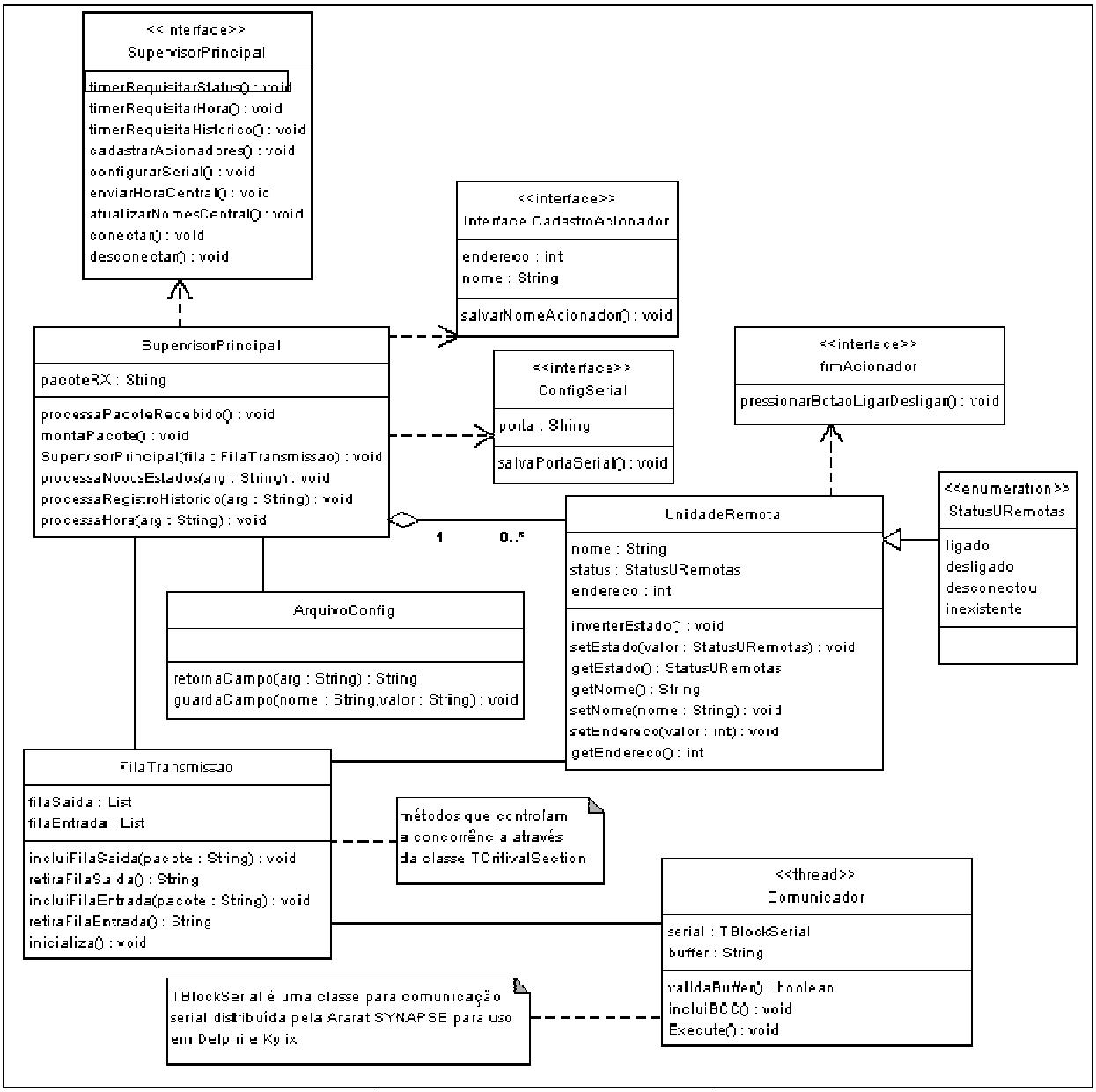


Figura 29 – Diagrama de classes

Como o sistema foi desenvolvido para usar arquivos, a classe `ArquivoConfig` existe para facilitar a gravação e recuperação dos dados. Os métodos `retornaCampo` e `guardaCampo` utilizam um mecanismo de rótulos para localizar facilmente os dados dentro do arquivo. O quadro 11 mostra um exemplo de dados com rótulos de identificação, onde os valores entre colchetes são os rótulos e o valor esta à direita.

```
[serialnome]COM1
[nomeacio32]Cozinha 02
[nomeacio01]Luz Garagem
[nomeacio02]Luz Cozinha 01
[nomeacio03]Luz Cozinha 02
[nomeacio04]Luz Quarto Casal
[nomeacio05]Luz Quarto S. 01
[nomeacio06]Luz Quarto S. 02
```

Quadro 11– Exemplo de arquivo com valores rotulados

As interfaces do sistema foram desenhadas no Kylix e podem ser vistas nas figuras 30 até 33.

A figura 30 mostra a tela principal. Estão ali os botões de conectar e desconectar, atualizar nomes na central, retornar a hora da central e atualizar a hora da central. Além dos botões, existe também o menu com as opções de configurar os acionadores e configurar a porta serial.

Os *timers* presentes na tela principal são referentes as requisições de *status* que deverão ser feitas a central a cada segundo. São eles também que descarregam as informações do histórico da central a cada minuto e que coordenam o envio dos nomes a central, quando clica-se no botão “Atualizar Nomes Central”.

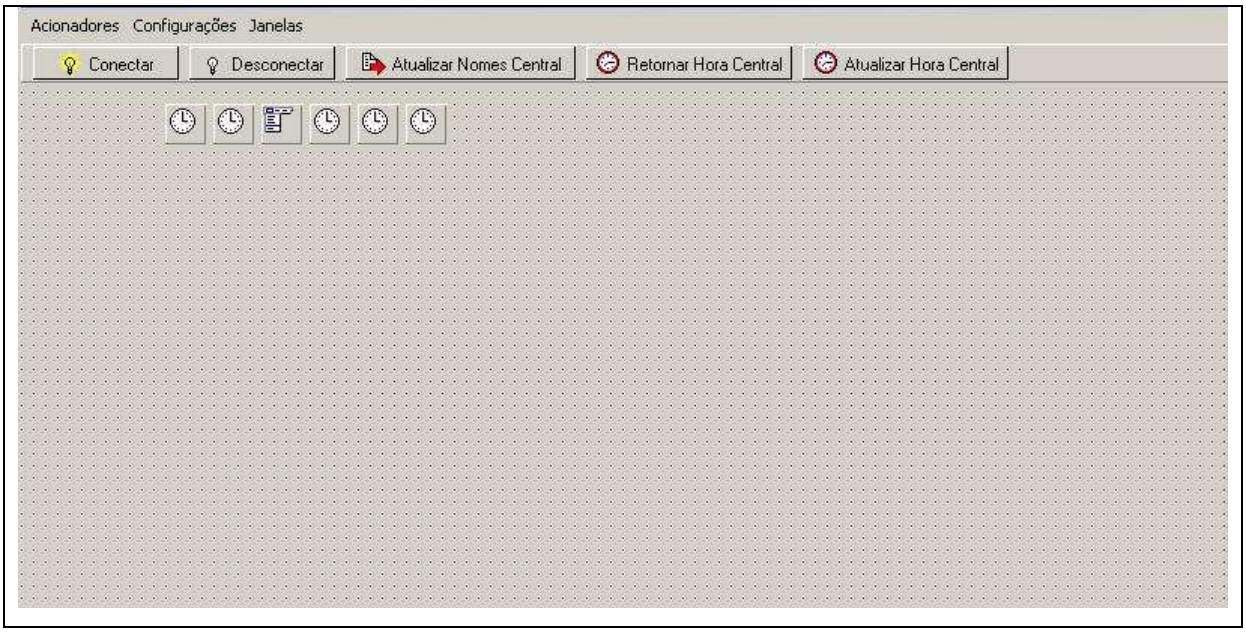


Figura 30 – Interface `SupervisorPrincipal`

A figura 31 mostra a interface do acionador. Nela é possível mandar os acionadores ligarem ou desligarem e é possível também ver a imagem referente ao cadastro do acionador e o nome, endereço e estado na barra *caption* do formulário.

Figura 31 – Interface `frmAcionador`

A figura 32 mostra a interface que serve para configurar a porta serial que o sistema deverá se conectar.

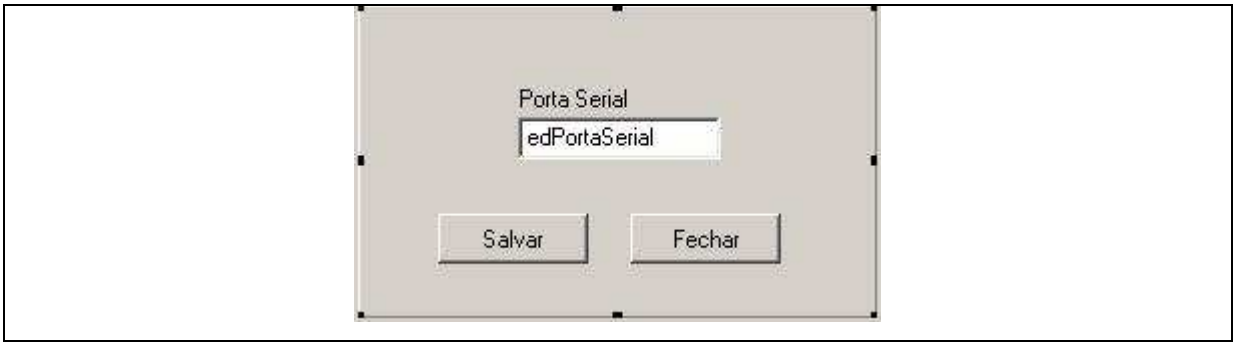


Figura 32 – Interface `ConfigSerial`

A figura 33 mostra a interface que faz o cadastro dos nomes de cada acionador e também da figura associada.

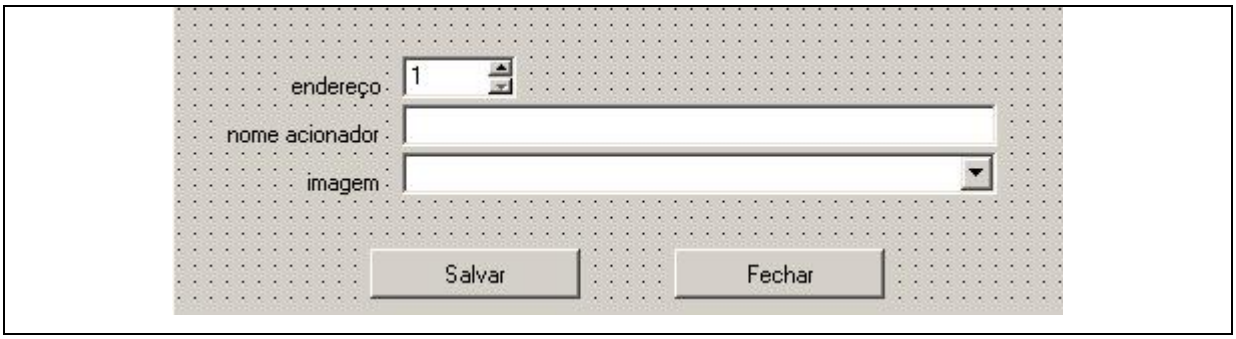


Figura 33 – Interface `CadastroAcionador`

A figura 34 mostra o diagrama de seqüência demonstrando o supervisor clicando no botão ligar/desligar da interface `frmAcionador`. O mesmo diagrama mostra o funcionamento do principal *timer* do sistema, que é o que controla o recebimento de todos os pacotes vindos da central.

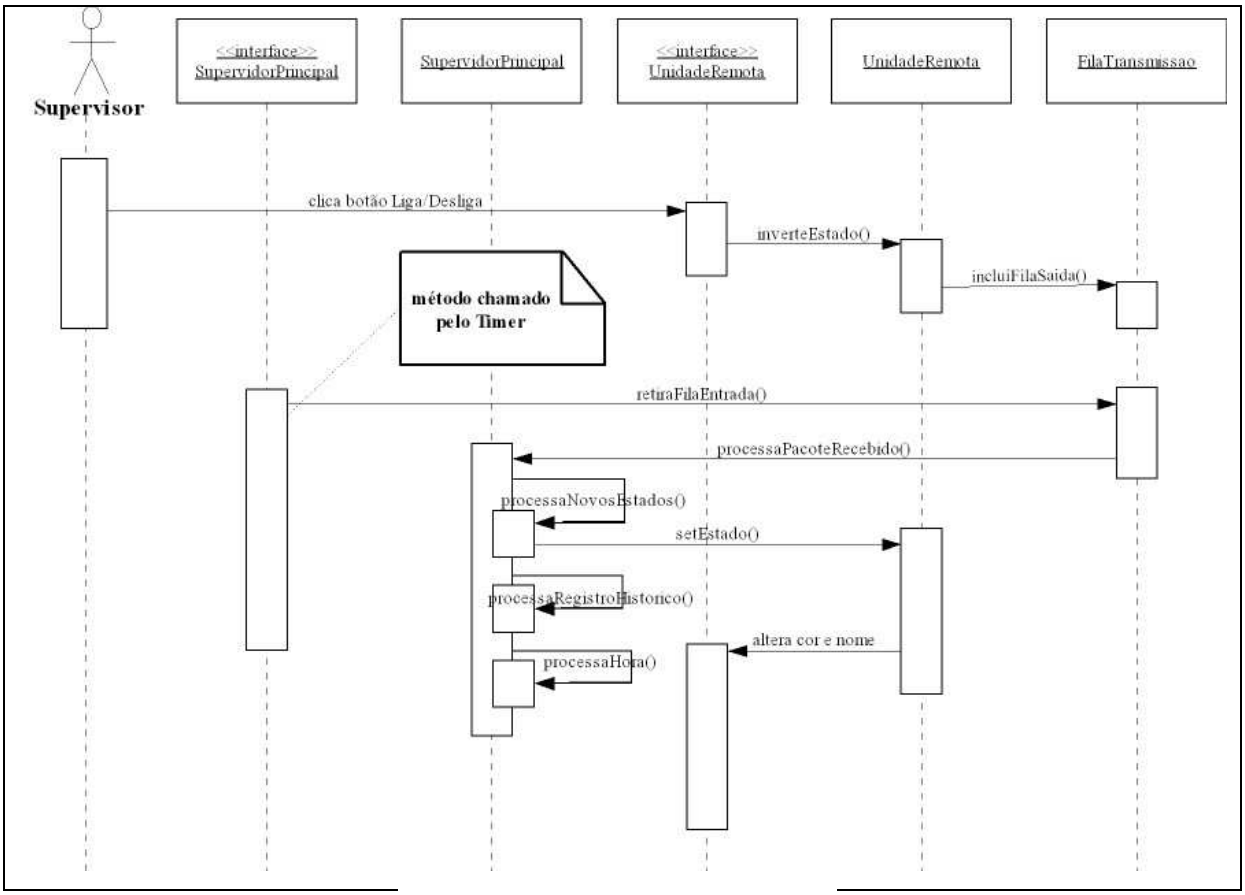


Figura 34 – Diagrama de seqüência 1

A figura 35 demonstra como a classe `Comunicador` deve processar as entradas e saídas de pacotes. É importante o uso contínuo da classe `FilaTransmissao` para que o software processe os pacotes através do *timer* do diagrama anterior.

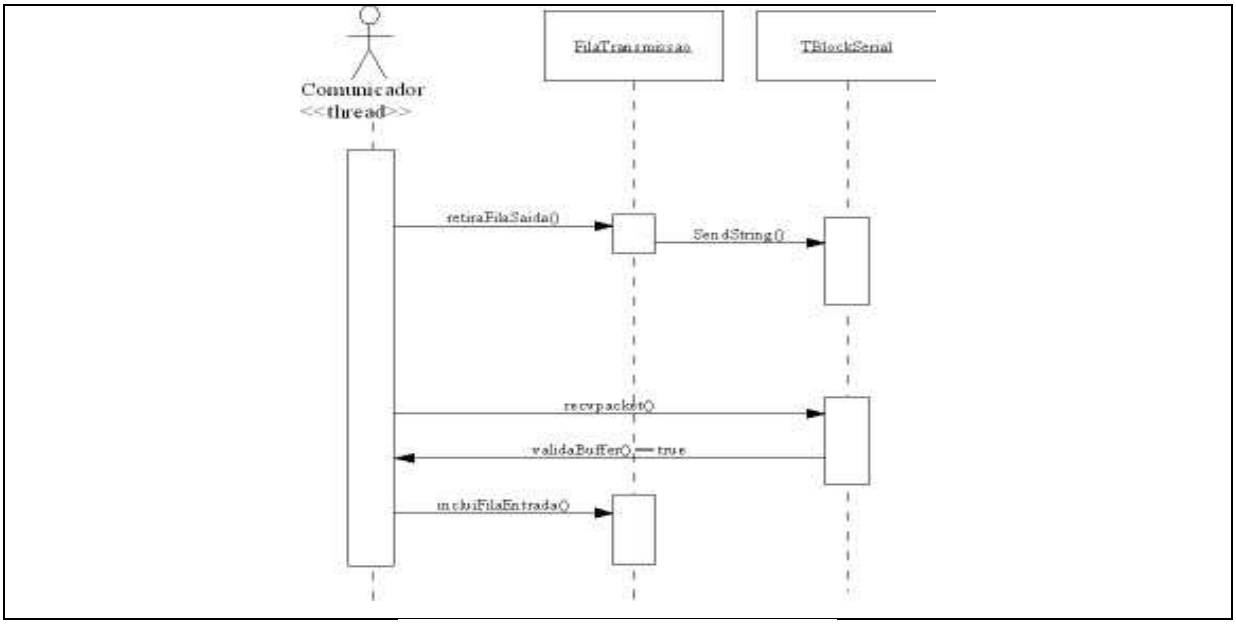


Figura 35 – Diagrama de seqüência 2

`TBlockSerial` é a classe para comunicação serial distribuída pela Ararat Synapse

(ARARAT, 2006) que funciona em Delphi e em Kylix e por isso é bastante adequada ao sistema que deve funcionar em Linux.

3.3 IMPLEMENTAÇÃO

A implementação esta dividida em três partes. Primeiramente será comentado sobre a implementação do acionador, seguido pela central e finalizando pelo software supervisor do PC. A especificação do protocolo está inclusa na implementação dos itens citados acima, portanto não é necessário um sub-tópico para ele.

Depois de comentada a implementação, é apresentado um estudo de caso do sistema completo e encerrando o capítulo com alguns comentários sobre os resultados obtidos no projeto.

3.3.1 Técnicas e ferramentas utilizadas

A seguir estão os comentários sobre as ferramentas e detalhes dos módulos do sistema. Cada um deles foi implementado individualmente e testado usando o software Listen32 (WIN-TECH, 1999) para validar o protocolo de comunicação serial.

Listen32 é um software que faz a monitoração do tráfego de dados em uma porta serial do PC e pode ser usado para enviar pacotes também. Por isso ele é adequado aos testes do protocolo dos três módulos do sistema.

3.3.1.1 Acionador

A implementação do acionador foi feita em dois passos: hardware e software.

O hardware que já estava especificado e foi apenas questão de corroer a placa e montar as peças. A figura 36 mostra o resultado final da montagem da placa acionadora, que ficou 97mm de comprimento por 53mm de largura, tamanho suficiente para caber nas caixas elétricas comuns em residências.

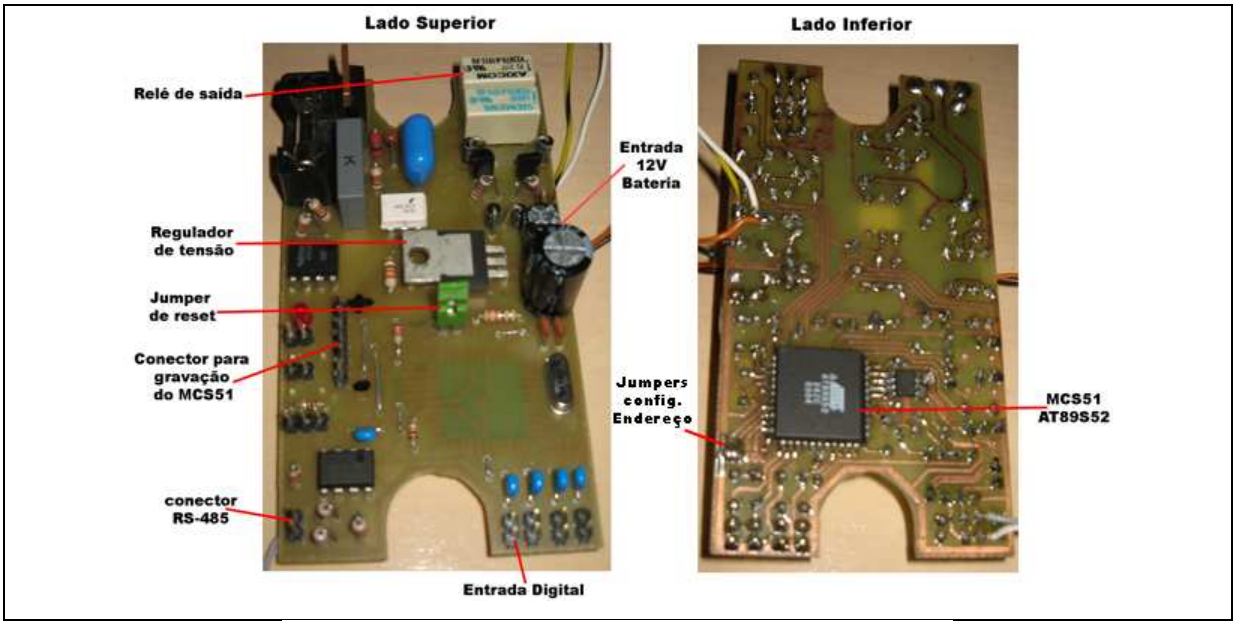


Figura 36 – Visão superior e inferior da placa acionadora

Tendo montado apenas uma placa acionadora, foi desenvolvido o software para ela. O ambiente de desenvolvimento usado é o uVision2 (KEIL SOFTWARE, 2003) que é um compilador para microcontroladores e que não possui recursos avançados como classes ou orientação a objetos. A linguagem usada no compilador é o C, adaptado a algumas características específicas dos microcontroladores como por exemplo a criação de variáveis do tipo `bit`.

No quadro 12 está a principal estrutura do código fonte do acionador: a função main e a função *loop*.

```
void main(void){
  int i;
  InitCpu();
  ler_adr();
  iniciaVariaveisSistema();
  for(i=0;i<10;i++){
    led_ativo = 1;
    Delay(100);
    led_ativo = 0;
    Delay(100);
  }
  Loop();
}
void Loop(void){
  while(1) {
    verificar_comunicacao();
    verificarOcorrenciaEventos();
  }
}
```

Quadro 12 – Principais rotinas do acionador

A função `InitCpu()` inicializa os registradores do microcontrolador que configuram a velocidade da transmissão serial, a velocidade do *timer* e quais interrupções estarão ativas.

`Ler_adr()` apenas lê o endereço configurado nos *jumpers* de configuração de endereço da placa e o `iniciaVariaveisSistema()` inicializa todas as variáveis do sistema, como por exemplo o status inicial da placa (desligada) e os *buffers* de entrada e saída.

No quadro 13, a listagem da função `verificar_comunicacao()`.Esta rotina faz o microcontrolador responder a um pacote normal e também gerencia o tempo de atraso na resposta quando um comando de *polling* é recebido.

```
void verificar_comunicacao(void){
  if(broadcasting) {
    if(tempoResposta == 0) {
      processaPacoteRecebido();
      xmit_com1();
      broadcasting = 0;
    }
  } else if(com1_ok) {
    com1_ok = 0;
    if(ehRequisicaoGeral) {
      ehRequisicaoGeral = 0;
      broadcasting = 1;
      tempoResposta = adr * 20;
    } else {
      processaPacoteRecebido();
      xmit_com1();
    }
  }
}
```

Quadro 13 – Função `verificar_comunicacao()`

`processaPacoteRecebido` é a função que irá verificar o tipo de pacote recebido. Se for um comando de ligar ou desligar, então o acionador liga ou desliga o relé de saída. Se for uma requisição de *status* ou como resposta aos comandos o acionador coloca no buffer `TxBuff1[]` o pacote a ser transmitido pela serial.

`xmit_com1()` é a função que envia o pacote colocado no buffer de transmissão `TxBuff1[]`. Esta função envia byte a byte o pacote, e faz o cálculo do BCC durante a transmissão, incluindo ele automaticamente no final.

A interrupção da porta serial coloca os bytes recebidos no *buffer* `RxBuff1[]` e se a transmissão estiver concluída, inclusive com a verificação do BCC, o *flag* `com1_ok` é colocado em 1.

`ehRequisicaoGeral` é outro *flag* que é colocado em 1 sempre que um pacote é recebido com o endereçamento 99 (endereço de *broadcast*). Esta variável é usada dentro da interrupção de recepção da serial.

Sempre que um comando de *broadcast* é recebido, a variável `tempoResposta` deve ser iniciada com o valor 20 vezes o endereço do microcontrolador. Esta variável é decrementada

em uma unidade por milissegundo dentro da interrupção do *timer0* até que chegue a zero.

No quadro 14 a rotina verificarOcorrenciaEventos() é mostrada. Esta função verifica quando um interruptor é ligado ou desligado e faz o travamento de alterações no interruptor contra ruídos.

```
void verificarOcorrenciaEventos() {
  getentradas();
  if(ocorreuAlteracaoEntrada) {
    if(tempoEntrada == 0)
      ocorreuAlteracaoEntrada = 0;
  } else {
    if(ent3_aux != ent3) {
      rele0 = !rele0; //invertendo saída do relé
      led_ativo = rele0;
      ent3_aux = ent3;
      ocorreuAlteracaoEntrada = 1;
      tempoEntrada = 10;
    }
  }
}
```

Quadro 14 – Função `verificarOcorrenciaEventos()`

`getentradas()` é a função que lê a porta referente ao interruptor `entrada4` e se estiver lendo 1, coloca 0 na variável `ent3`, caso contrário coloca 1 em `ent3`.

`ocorreuAlteracaoEntrada` é o *flag* que indica ao acionador quando uma alteração no interruptor aconteceu. Sempre que uma alteração acontecer, a saída do relé deve ser invertida (se estiver ligado, desliga-se; se desligado, liga-se) e a variável `tempoEntrada` é iniciada em 10, fazendo com que uma alteração só possa surtir efeito na saída a cada 10ms sem travar o acionador numa função do tipo `Delay(10)`. `tempoEntrada` também é decrementado em uma unidade no *timer0* do micro a cada milissegundo.

Com o software pronto foram feitos testes individuais para verificar se o acionador ligava e desligava elementos elétricos, seguidos de testes de comunicação serial usando o Listen32 como auxiliar para isso. Concluídos os testes o próximo passo é testar mais de um acionador ligado na mesma linha RS-485. A figura 37 mostra três acionadores interligados.

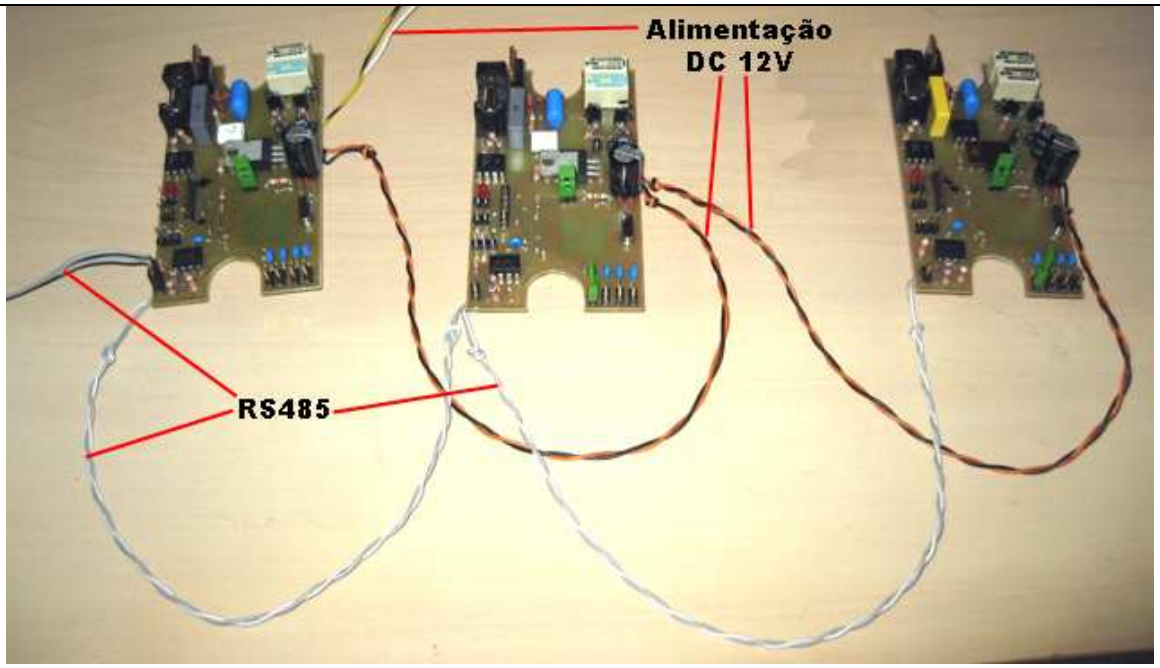


Figura 37 – Acionadores interligados

Os testes com três acionadores são concluídos com sucesso e a implementação da central é iniciada.

3.3.1.2 Central

Igual aos acionadores, a central é implementada usando o uVision2 (KEIL SOFTWARE, 2003), seguido de testes com o Listen32 e finalmente são realizados testes com os acionadores em si.

O Data Term possui uma série de funções proprietárias para envio e recebimento de dados por ambas portas seriais, gravação e recuperação de dados da EEPROM, escrita no *display* gráfico, telas para configuração de data e hora e funções gerais de formatação de dados. Essas funções não serão explicadas em detalhes.

O quadro 15 mostra a rotina principal da central `main()` e a função `Loop()` que chama os principais módulos do sistema.

```
void main(void){
  InitCpu(); //inicia registradores da cpu
  read_parametros();    //lê parametros da EEPROM
  ler_relogio();  //lê relógio e inicia variaveis
  InitCom2();  //configura UART externa para leitura da serial
  CKCON = 0x7F;
  Temp_buzina = 200;  //toca buzzer na inicialização do sistema
  EX0  = 1;
  dimmer = PM.PS.dimmer; //brilho do display
```

```
        Init_graph_display(); //inicia display
        ClearTela_graph();  //limpa display
        inicializa_variaveis_sistema();
        Loop();
      }
      void Loop(void){
        while(1) {
          verificar_transmissao_dados();
          atualizar_display();
          verificar_comunicacao_pc();
          verificar_entradas_teclado();
        }
      }
```

Quadro 15 – Funções `main()` e `loop()` da central

`inicializa_variaveis_sistema()` inicia as variáveis da central como por exemplo `matriz1`, `matriz2`, as filas de entrada e saída da serial, os buffers de transmissão e a fila de histórico de eventos.

`Loop()` é a rotina principal que chama os quatro grandes módulos da central.

Na implementação, as telas 5 e 6 (vistas na figura 20) usam funções proprietárias do Data Term para leitura de dados formatados, e, portanto, acabam não estando fiéis a especificação pois elas travam o sistema nas telas, impedindo a central de realizar todas as outras funções de transmissão serial e atualização de matrizes.

A posição de um acionador nas matrizes `matriz1` e `matriz2` será sempre o endereço dele subtraído de um. Isto se da por que uma matriz de 32 elementos começa com 0 e vai até 31. Por isso é comum ver no código fonte da central incrementos do endereço a ser transmitido e decrementos de endereço no pacote recebido.

A função `verificar_transmissao_dados()` é a mais crítica pois faz todo o controle de *polling* e de pacotes individuais a serem transmitidos aos acionadores. Ela está listada no apêndice A. Adicionalmente é esta função que recebe e processa os *status* de todos os acionadores usando as matrizes `matriz1` (principal) e `matriz2` (auxiliar) para verificação de perda de conexão e alteração de *status.* Alterações de *status* são considerados eventos e portanto devem ser colocados na fila de eventos com a data e hora registrada juntamente no novo estado e do endereço de quem gerou o evento.

Sempre que o tempo do *polling*, que é de 660ms, termina, é executada a função `processaFimPolling()`, que faz a atualização de valores na `matriz1`. Mais detalhes em como os valores são passados a matriz no quadro 16.

```
      void processaFimPolling(void) {
        char i = 0;
        for(i = 0; i<32;i++) {
          if(matriz1[i] != matriz2[i]) {
            atualizarTela = 1;
            //funcaoReacao(i,matriz2[i]);
```

```
          }
          if((matriz2[i] == '1') || (matriz2[i] == '2')) {
            if(matriz1[i] != matriz2[i]) {//gera evento (ligou ou desligou)
              adicionaEventoNoHistorico(i, matriz2[i]);
            }
            matriz1[i] = matriz2[i];
          } else if(matriz2[i] == '0') {
            if((matriz1[i] == '1') || (matriz1[i] == '2')){ //gera evento
(desconexão)
              adicionaEventoNoHistorico(i, '3');
              matriz1[i] = '3';
            }else if(matriz1[i] == '3')
              matriz1[i] = '3';
            else
              matriz1[i] = '0';
          }
          matriz2[i] = '0';
        }
      }
```

Quadro 16 – Função `processaFimPolling()` na central

`adicionaEventoNoHistorico()` é o que a função reação faz: adiciona um evento na fila do histórico. Esse evento ficará ali até o software PC requisitar por ele. A `matriz2` é colocada em zero no final do processamento de cada um dos campos, já inicializando ela para o próximo *polling*.

A `atualizar_display()` trata-se apenas de uma estrutura que verifica qual o valor da variável `Display`, e mostra a tela referente. Como já foi dito anteriormente, se a tela for 5 ou 6, então a central ficará bloqueada nessa função até que os cadastros tenham sido feitos.

A tela de monitoração (`Display` 4) possui algumas variáveis indicando qual dos acionadores está selecionado. Sempre que `atualizarTela` está em 1, a tela é redesenhada. Quando é mostrada, está tela faz uma varredura nos estados dos acionadores e imprime somente aqueles que estão com *status* diferente de “0”. O *status* “1” é impresso como “1”, o “2” é impresso como “0” e o “3” é impresso como “x”.

A função `verificar_comunicacao_pc()`tem o mesmo papel da verificação de transmissão dos acionadores. Ela fica aguardando até que receba um pacote vindo do PC, que então é processado, devolvendo a resposta ao software supervisor. Pelo fato de estarem sendo usadas funções proprietárias no Data Term, fica totalmente transparente o uso da segunda porta serial que na realidade usa uma UART externa.

No quadro 17 tem um exemplo de envio de *status* dos acionadores ao PC. `xmit_byte_com2()` é a função que envia o buffer em TxBuff2[] para a UART externa do Data Term, que por sua vez transmite ao PC.

```
void respondeStatusAcionadoresCOM2(void) {
  strncpy(&TxBuff[4], $matriz1[0], 32);
  TxBuff2[0] = STX;
  TxBuff2[1] = '0';
  TxBuff2[2] = '1';
  TxBuff2[3] = '1';
  TxBuff2[36] = ETX;
  TxBuff2[37] = 0x00;
  xmit_byte_com2();
}
```

Quadro 17 – Função `respondeStatusAcionadoresCOM2()` na central

O último dos módulos é a função `verificar_entradas_teclado()`que faz a verificação de entradas no teclado do Data Term. Esta rotina trabalha juntamente com a variável `Display`. Dependendo do seu valor, a entrada do teclado tem um processamento diferente.

Se `Display` for 3, então as entradas do teclado vão direcionar a um dos menus, alterando o valor da variável `Display`.

Se `Display` for 4, então entradas do teclado vão fazer com que pacotes sejam enviados aos acionadores com comandos de ligar ou desligar e o cursor em tela também será excursionado pelas flechas do teclado.

Os testes foram feitos com o Listen32 tanto na porta serial RS-485 quanto na porta serial RS-232. Depois de completamente validado os pacotes, foi integrado na linha RS-485 as placas acionadoras para comunicarem entre si. Com isto funcionando, foi iniciada a implementação do Software PC.

3.3.1.3 Software PC

Um dos principais objetivos deste trabalho é que o software supervisor do PC funcionasse em Linux. Mas apesar desse objetivo, mostrou-se pouco prático programar no Kylix e ter ao mesmo tempo que fazer ajustes no software da central que só podia ser editado em Windows. Então foi optado por desenvolver em Delphi 7 usando uma Aplicação CLX, que é o código fonte pronto para ser compilado tanto em Delphi quanto na sua versão Linux, o Kylix. O que facilitaria ainda mais isso é o componente serial TBlockSerial que funciona tanto em Delphi quanto Kylix.

As interfaces já haviam sido desenvolvidas na especificação, restando fazer as classes e ajustes finais.

A principal classe do supervisor é a `SupervisorPrincipal`, que deverá instanciar

todas as outras classes na sua criação. No menu tem duas opções, cadastrar a serial ou cadastrar os nomes dos acionadores. Independente de qual for escolhida, o *form* é chamado através do método `showModal`, impedindo que o sistema seja operado sem que essas janelas sejam fechadas.

`FormConfigSerial` é o formulário para cadastro da porta serial. Ele está demonstrado na figura 32. Ao clicar no botão salvar, o método `guardaCampo('serialnome',edPortaSerial.text)` da classe `TarquivoConfig` deverá ser invocado para salvar o nome da serial no arquivo de configurações. O botão fechar apenas fecha o formulário, voltando à tela principal.

FormCadastroAcionador é a interface para cadastro de nomes dos acionadores e está demonstrado na figura 33. Sempre que um novo endereço for selecionado, o método `retornaCampo('nomeacio'+endereco)` da classe `ArquivoConfig` é chamado, retornando o nome cadastrado do acionador. Ao alterar o nome, basta clicar em Salvar para gravar no arquivo de configurações da mesma forma que o nome da serial é gravado.

A classe `ArquivoConfig` grava e busca texto cadastrado no arquivo de configuração através de rótulos. Um exemplo do conteúdo do arquivo pode ser visto no quadro 18.

```
[serialnome]COM1
[nomeacio32]Cozinha 02
[nomeacio01]Luz Garagem
[nomeacio02]Luz Cozinha 01
[nomeacio03]Luz Cozinha 02
[nomeacio04]Luz Quarto Casal
[nomeacio05]Luz Quarto S. 01
[nomeacio06]Luz Quarto S. 02
[nomeacio07]Luz Banheiro
[nomeacio08]Luz Corredor Quartos
[nomeacio09]Luz Corredor Acesso
[nomeacio10]Luz Sala Estar
[nomeacio11]Luz Sala TV
[nomeacio13]Luz Escritorio
```

Quadro 18 – Exemplo do conteúdo do arquivo da classe `ArquivoConfig`

O formulário `SupervisorioPrincipal` possui a propriedade `FormStyle` igual a `fsMDIForm`, que significa que é um formulário *Multiple Document Interface* (MDI), ou seja, abre várias janelas dentro dele mesmo. Os formulários `TformAcionador` tem a propriedade `FormStyle` em `fsMDIChild`, fazendo com que seja uma janela filha da tela principal.

Para fazer todo o controle de envio de requisições de *status*, requisições de histórico da central e requisição de hora, são usados vários componentes `TTimer`, pois eles facilitam bastante o controle de tempo. Entretanto nenhum desses garante que os pacotes sejam enviados, pois quem transmite é a classe `TComunicador`, que é um processo paralelo. O

quadro 19 mostra o principal método da classe `TComunicador`.

Tudo que `TComunicador` faz é verificar sempre se existe algo na fila a ser transmitido. Se tiver algo, ele transmite, senão ele fica esperando por algum pacote vindo da central. Ele recebe o pacote e, se este passar no método `validaBuffer` (que faz a verificação do BCC), então o pacote é enfileirado na fila de entrada. O método Execute será executado até que a propriedade `Terminated` seja colocada a `true`;

```
procedure TComunicador.Execute;
var  x, buffer: string;  i: integer;  y: real;
begin
  inherited;
  buffer := '';
  while not terminated do begin
      x := ser.recvpacket(100);
      if(length(x) > 0) then begin
        if(x[1] = #2) then begin
          buffer := x;
        end else if(x[length(x)-1] <> #3) then
          buffer := buffer + x
        else  begin
          buffer := buffer + x;
          if(validaBuffer(buffer))then
            fila.IncluiFilaEntrada(buffer) else
            buffer := '';
        end;
      end else begin
        buffer := fila.RetiraFilaSaida;
        if(length(trim(buffer)) > 0) then  begin
          ser.SendString(buffer);
   end; end; end; end;
```

Quadro 19 – Método `Executa` da classe `TComunicador`

O objeto `timerPollingTimer`, da classe `Ttimer`, instanciado no formulário principal deverá, a cada segundo, enviar uma requisição de *status* à central. Ele faz isso colocando o pacote a ser enviado na fila, pelo método `IncluiFilaSaida(pacote)`.

O objeto `timerVerificacao` vai a cada segundo verificar se existe algum pacote na fila de entrada e, se exister, este pacote será processado. Um pacote de envio de evento do histórico da central é colocado no arquivo `LogHistoricoCentral.dat,` sempre no final. No quadro 20 está uma listagem parcial deste arquivo. Pode ser visto a formatação do pacote neste arquivo, onde os dois primeiros caracteres de cada linha representam a hora, seguido pelos minutos, segundos, dia, mês e ano. Então temos o endereço em dois caracteres e o ultimo é o estado que o acionador ficou naquele momento.

```
172400301006013
172401301006012
172403301006023
172404301006022
172933301006011
173009301006013
173011301006011
173030301006011
173051301006013
173052301006011
173128301006023
173129301006022
.
.
.
```

Quadro 20 – Listagem do arquivo `LogHistoricoCentral.dat`

O objeto `timerAtivaHistorico` requisita a central, a cada minuto, o histórico de eventos. Se existir, então o `timerRecebendoHistorico` ativa a cada 100ms requisitando o próximo evento do histórico até que receba um evento nulo, quando então ele se desativa.

O `timerNomesAcionadores` serve para enfileirar, a cada 100ms, comandos de envio de nomes dos acionadores. O motivo para esse tempo é que a central gasta algum tempo fazendo a gravação de dados na EEPROM, e como neste sistema não existe garantia de que o nome será corretamente cadastrado, é dado um tempo para manter uma boa margem de processamento na central.

São instanciadas 32 classes `TunidadeRemota`, cada qual com seu respectivo `TformAcionador`, que só será instanciado caso obtenha um status diferente de zero para o acionador. Os *status* dos acionadores são passados para os objetos `TunidadeRemota` através do objeto `SupervisorPrincipal`, com o método `processaPacoteRecebido`, evocado no `timerVerificacao` de segundo em segundo.

Sempre que um objeto de `TunidadeRemota` tem o estado alterado através do método `setEstado(novoEstado)`, ele altera a cor de fundo do `TformAcionador`, a imagem `imgAcionador` e altera também o que esta escrito na barra `Caption` do formulário.

A classe mais usada no sistema é `Tfila`, que possui duas matrizes se `string` sem tamanho fixo. Uma matriz é `filaEntrada` e outra `filaSaida`. Cada uma delas possui seus dois métodos `IncluiFilaSaida(string)`, `RetiraFilaSaida`, `IncluiFilaEntrada(string)` e `RetiraFilaEntrada`. No quadro 21 tem a demonstração dos dois métodos para a fila de entrada.

```
procedure TFila.IncluiFilaSaida(x: string);
begin
  try
    reserva2.Acquire;
    SetLength(filaSaida, (length(filaSaida) +1));
    filaSaida[high(filaSaida)] := x;
  finally   reserva2.Release;
  end;
end;
function TFila.RetiraFilaSaida: string;
var retorno: string;
begin
  try
    reserva2.Acquire;
    if(length(filaSaida) > 0) then begin
      retorno := filaSaida[0];
      if(length(filaSaida) = 1) then
        setlength(filaSaida, 0)
      else
        filaSaida := Copy(filaSaida, 1,Length(filaSaida));
    end;
    result := retorno;
  finally reserva2.Release;
  end;
end;
```

Quadro 21 – Métodos `retiraFila` e `incluiFila`

`reserva1` e `reserva2` são objetos da classe `TcriticalSection` que fazem o controle de acesso concorrente as áreas de leitura e gravação de dados importantes nas filas. Cada fila tem seu controle individual de seção.

Depois de concluído o sistema, foram realizados testes de transmissão serial com a central.

O código fonte que foi gerado em Delphi, quando compilado em Kylix funciona da mesma forma, sem precisar nenhum tipo de adaptação.

3.3.2 Operacionalidade da implementação

Os proprietários de uma residência desejam automatizar as luzes da cozinha e a luz da garagem, podendo ligá-las e desligá-las de outro cômodo. E ao mesmo tempo eles querem armazenar o tempo que cada luz ficou acesa, monitorar o estado atual das luzes e tudo isso através de seu computador localizado na sala.

3.3.2.1 Instalando os acionadores

Como a cozinha tem duas lâmpadas, cada uma delas terá uma placa acionadora. A garagem como possui apenas uma luz, ficará com um acionador. Cada acionador deve ser instalado dentro da caixa elétrica onde fica o interruptor da respectiva lâmpada, como visto na figura 38.

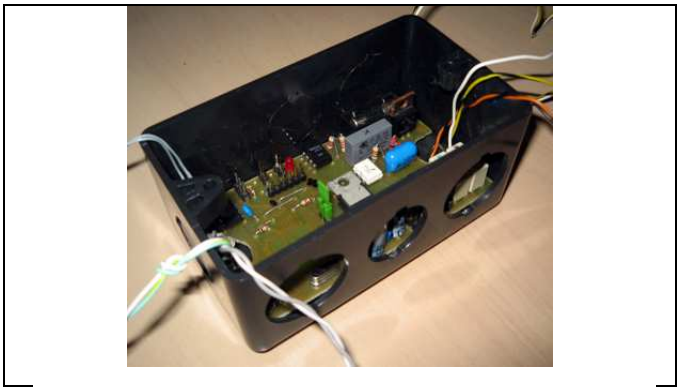


Figura 38 – Placa acionadora dentro da caixa elétrica

Colocando o interruptor na caixa, devidamente ligado na entrada digital da placa acionadora, e colocando a entrada positiva da rede elétrica em um contato do relé, com a saída para a lâmpada e esta por sua vez com o outro fio ligado no neutro, temos um equipamento apto a acender e apagar a lâmpada. A figura 39 mostra a caixa elétrica com o interruptor e a placa acionadora dentro.

Figura 39 – Placa acionadora dentro da caixa elétrica com interruptor

O desenho da figura 40 mostra como deve ser feita a ligação do interruptor no acionador e na lâmpada.

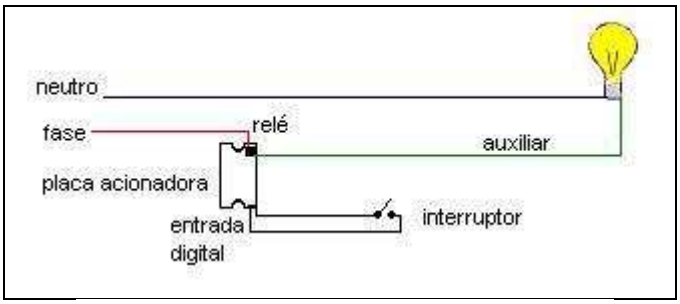


Figura 40 – Esquema de ligação do acionador

3.3.2.2 Usando a central

A central microcontrolada deverá ficar na sala onde é feita a monitoração dos acionadores e para isso deve estar ligada a eles através de sua entrada RS-485.

Ao ligar a central, ela soará um bipe e a tela inicial é mostrada durante um segundo, seguido da tela de apresentação e depois de mais um segundo, a tela do menu. As figuras 41 e 42 mostram fotos das telas iniciais e a figura 43 mostra o menu principal.

Figura 41 – Tela de entrada

Figura 42 – Tela de apresentação

Figura 43 – Tela do menu principal

No menu principal pode-se digitar três números referentes as opções apresentadas no menu. A opção 1 chamará a interface de monitoração dos acionadores. A opção 2 irá para a tela de cadastro de nomes para os acionadores e finalmente a opção 3 leva a configuração de data e hora da central.

Na figura 44 temos duas capturas de tela da monitoração de acionadores, cada uma com um acionador diferente selecionado. Sempre que um acionador estiver selecionado, o nome cadastrado dele aparece na parte de baixo da tela. As teclas “1” e “2” ligam e desligam o acionador selecionado enquanto as setas mudam a seleção. “ESC” volta ao menu principal.

A central deve, uma vez que conectada, encontrar sozinha os acionadores ligados a ela, portanto cada acionador deve ter seu endereço único. Na tela de monitoração, os acionadores são mostrados com seu endereço e seu estado. O número “1” indica acionador ligado, “0” indica acionador desligado e “x” indica perda de conexão.

Figura 44 – Tela monitoração de acionadores

As opções “2” e “3” do menu principal vão para telas de configuração mostradas nas figuras 45 e 46.

Figura 45 – Tela cadastro de acionadores

O cadastro dos acionadores é feito digitando-se o endereço e alterando o texto no campo abaixo. Assim que for digitado o nome, pressiona-se “ENTER” e então é salvo na memória da central. Um procedimento semelhante é feito na configuração de data e hora, bastando digitar a data e a hora seguido de “ENTER” que esta também será salva na memória.

Figura 46 – Tela configura data e hora

Enquanto nas telas de configuração a central não pode estar conectada ao PC, pois ocorrerá erro, portanto basta desconectar o software supervisor que não ocorrerão problemas.

3.3.2.3 Monitorando através do PC

O software do PC possui duas versões, uma para Linux e outra para Windows. Independente da versão escolhida elas funcionam exatamente da mesma forma.

A figura 47 mostra a tela principal do software PC.

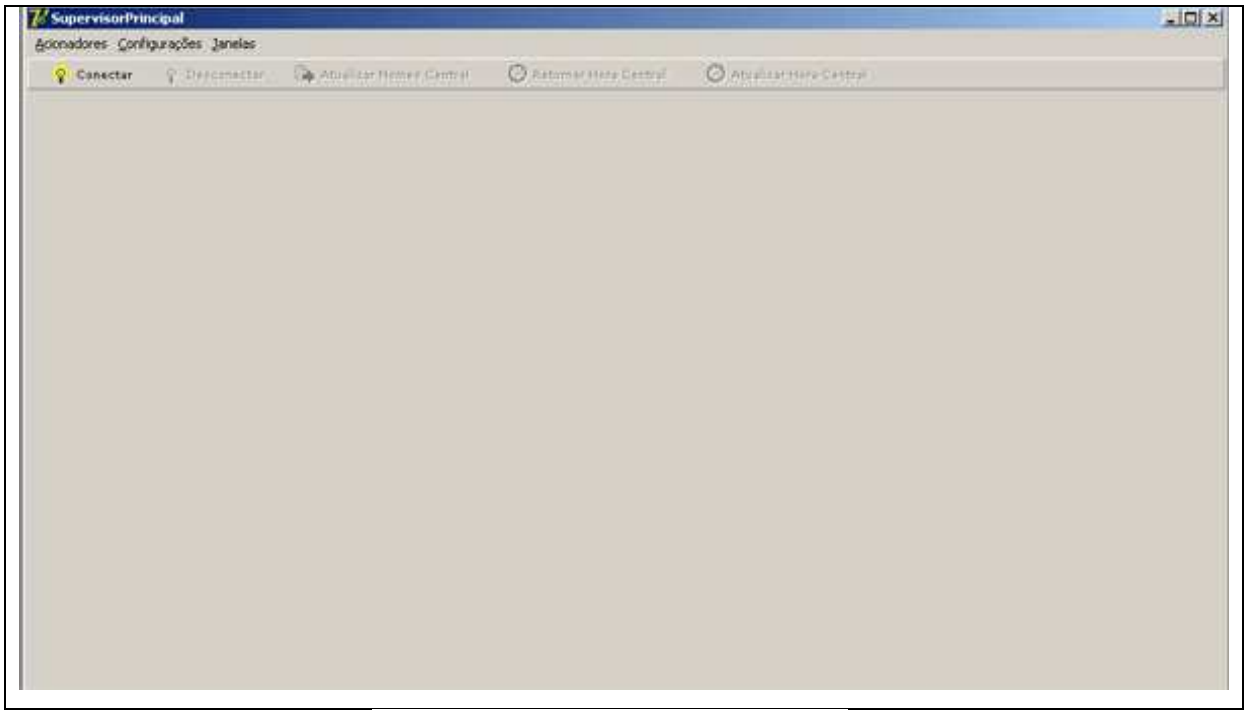


Figura 47 – Software PC – tela principal

Na tela principal existem alguns menus que servem para configurar os acionadores, configurar a porta serial e organizar as janelas abertas. Tem os botões de conexão com a central, de desconexão e outros. Ao clicar na configuração da porta serial aparece a janela da figura 48 para configurar o nome da porta serial. O Windows usa os nomes COM1, COM2, e

assim por diante. O Linux usa nomes ttyS0, ttyS1, e assim por diante.

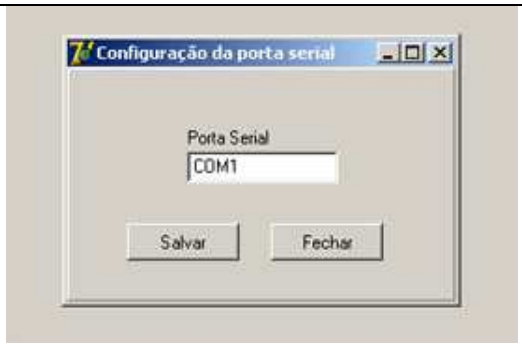


Figura 48 – Software PC – configuração da porta serial

Uma vez configurada a porta serial, recomenda-se cadastrar os nomes dos acionadores. Isto é feito no menu Acionadores → Cadastro, mostrado também na figura 49.

Figura 49 – Software PC – cadastro dos acionadores

A tela de cadastro serve para orientar quem estiver monitorando quais os acionadores que estão ligados ou não, sendo desnecessário decorar o endereço deles como precisa ser feito na central. O nome pode ter no máximo vinte caracteres e a imagem é selecionada entre algumas imagens padrões do sistema.

Voltando a tela principal e clicando no botão de conectar, se correr tudo bem os outros botões vão ficar habilitados e os acionadores vão aparecer na tela, como visto na demonstração da figura 50.

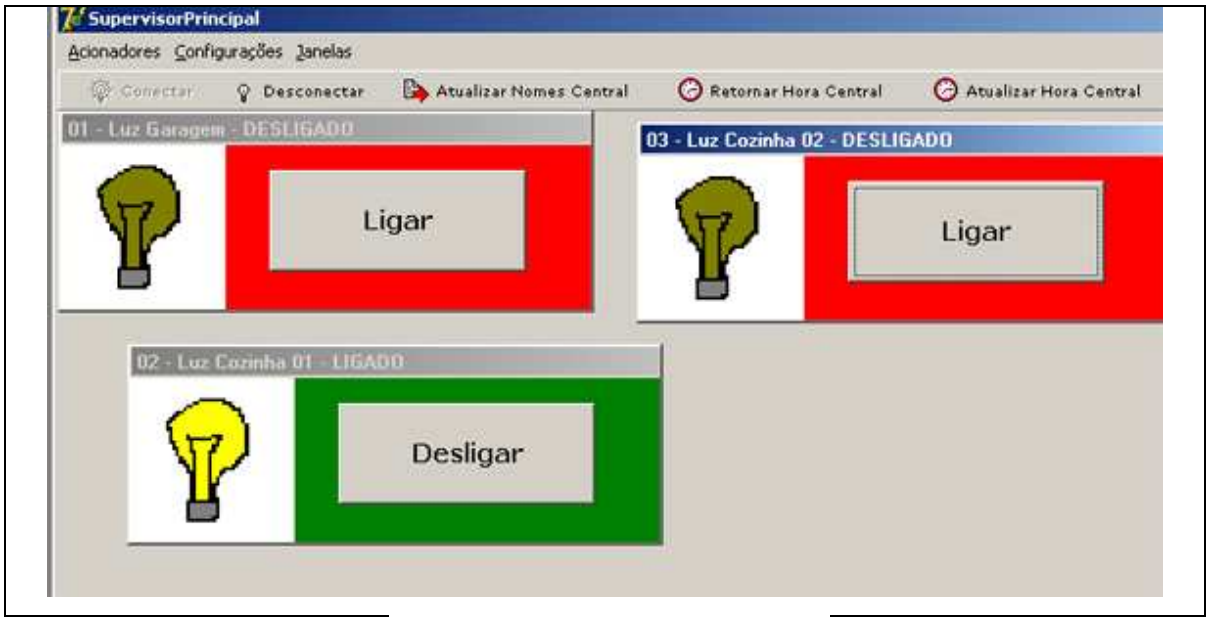


Figura 50 – Software PC – conectado

Cada acionador aparece em uma janela diferente, com seu respectivo nome e imagem cadastrada. As cores verde e vermelho significam que o acionador esta ligado ou desligado. A cor amarelo significa perda de conexão, como visto na figura 51.

Figura 51 – Software PC – perda de conexão com o acionador

Cada tela de acionador possui ainda um botão para poder alterar o estado dele. Se estiver ligado, o botão fará ele desligar e se estiver desligado, fará ligar.

O botão “Atualizar Nomes Central” da tela principal irá enviar a central todos os nomes cadastrados dos acionadores, evitando que os nomes sejam cadastrados na central.

O botão “Retornar Hora Central” irá verificar qual a hora na central e mostrar na tela, como pode ser visto na figura 52.

Figura 52 – Software PC – verificando hora da central

O botão “Atualizar Hora Central” enviará a hora do computador para a central, atualizando-a.

O software PC requisita de tempos em tempos todo o histórico de eventos presente na central, e armazena o mesmo no arquivo LogHistoricoCentral.dat, podendo este ser aberto em um editor de textos comuns e analisado.

3.4 RESULTADOS E DISCUSSÃO

O resultado geral do trabalho mostrou-se bastante satisfatório em vários aspectos. A placa acionadora funciona corretamente quando ligada a um interruptor e este for ligado ou desligado. A central microcontrolada consegue gerenciar os acionadores, inclusive armazenando o histórico de eventos corretamente e ao mesmo tempo fazer comunicação de dados com o PC. O software supervisório funciona por longos períodos de tempo (mais de 24 horas) sem precisar ser reiniciado e continua fazendo a coleta do histórico de eventos da central.

A placa acionadora ficou com um consumo de aproximadamente 40mA com os relés desatracados, e cada um dos dois relés consome 30mA adicionais, gerando um consumo máximo de 100mA por placa acionadora. Apesar desse valor parecer pequeno isoladamente, o limite de 32 acionadores poderá gerar um consumo de 3,2A, que pode ser um problema para alimentar todos eles pelo mesmo cabo. A sugestão é alimentar os acionadores em cabos diferentes.

A central microcontrolada processa duas comunicações seriais, atualiza o display e verifica entradas do teclado. Toda essa execução faz com que ela receba incorretamente alguns pacotes do *polling* vindo dos acionadores, colocando eles imediatamente no *status* “3” (perda de conexão). A perda de pacotes é de aproximadamente 5% dos recebidos, gerando um déficit muito grande de pacotes numa amostragem maior. A solução para isso seria considerar “perda de conexão” somente depois de perder dois ou três pacotes consecutivos pois é normal e aceitável receber um pacote incorreto a cada vinte. Outra solução para isso seria aumentar a velocidade da transmissão serial entre central e acionadores de 9600bps para 19200bps ou superior, fazendo com que a central tenha mais tempo para tratar o recebimento de uma mensagem antes que comece a receber a próxima.

O software supervisório PC projeta na tela exatamente os dados recebidos da central, por isso o status “perda de conexão” que é recebido sempre é mostrado na tela e, com a taxa de 5% de perdas, faz com que o usuário possa ter a impressão da tela estar piscando. A

solução é tratar isso na central microcontrolada ao invés de no PC, como já foi dito anteriormente.

O fato do PC não gerar um histórico próprio e ter que sempre recuperar o histórico de eventos da central é bastante adequado ao sistema pois mantém uma consistência dos eventos com relação ao horário que aconteceram. Isso é especialmente importante quando se deseja analisar os horários para fazer uma estimativa de tempo que cada acionador ficou ligado pois o horário da central não necessariamente será o mesmo horário do PC.

Adicionalmente a central detecta se algum acionador parar de responder (evento perda de conexão) por qualquer motivo. A central e o acionador não têm como saber se o equipamento ligado ao acionador está funcionando, ou se a lâmpada está queimada, logo é uma limitação ter que verificar se os dispositivos elétricos estão funcionando regularmente.

Os trabalhos corretatos de Besen (1996) e de Censi (2001) diferem principalmente na modularização deste trabalho. Aqui se a central parar de funcionar ainda será possível localmente ligar e desligar os acionadores através dos interruptores. Em ambos trabalhos correlatos os dispositivos elétricos são totalmente dependentes do funcionamento do microcontrolador central e se este falhar automaticamente os equipamentos elétricos deixarão de funcionar também. Outra diferença com relação ao de Besen (1996) é que nele só é possível monitorar com o PC ligado, enquanto no trabalho aqui desenvolvido pode-se monitorar tanto na central quanto no PC, independente do software PC estar em funcionamento. Ambos trabalhos têm o número de dispositivos elétricos limitados ao número de entradas e saídas do microcontrolador, enquanto aqui o número é limitado pela especificação do RS-485. No de Censi (2001) é possível controlar os acionadores através de e-mails pois o microcontrolador usado faz conexão com a Internet enquanto aqui o sistema funciona apenas localmente (apesar de ser possível implementar essas funcionalidades para web em trabalhos futuros).

4 CONCLUSÕES

O trabalho atendeu a todos os objetivos inicialmente traçados, alguns em sua totalidade e outros parcialmente. O maior triunfo deste é ter alcançado o objetivo principal que era fazer com que o sistema inteiro ficasse integrado.

A necessidade do software PC precisar funcionar em Linux foi tão bem sucedida que ele funciona tanto em Linux quanto em Windows, bastando recompilar o código fonte. É interessante pois fornece ao usuário final a possibilidade de escolher o que lhe for mais conveniente.

A interface da central de controle está relativamente intuitiva. Bastando algumas poucas instruções já é possível operá-la totalmente. Mas apesar disso, ela poderia ficar um pouco mais amigável, especialmente para usuários finais.

A possibilidade de ligar e desligar elementos elétricos à distância pela central e pelo software supervisório PC adicionam uma incrível facilidade e praticidade em automações residenciais e prediais. Pode-se instalar esse sistema em lugares como condomínios, prédios de escritórios ou de salas de aula, para controlar as luzes que estiverem acesas, e as tomadas que forem disponibilizadas para uso.

A eletrônica da placa acionadora foi uma grande dificuldade desde o início do projeto, mas graças a ajuda foi possível construí-la de forma que coubesse dentro de uma caixa elétrica, evitando que o acionador ficasse aparente onde instalado. Um dos requisitos era não interferir na estética do local, e este foi plenamente alcançado.

O uso do Data Term como hardware para a central foi determinante para a conclusão desse trabalho. Seria inviável construir um hardware que realizasse as mesmas funções que o Data Term no cronograma inicial.

Dentre os objetivos da automação residencial, apresentados no capítulo 2.2, o projeto conseguiu atender principalmente a questão da praticidade e conforto e indiretamente da economia. Outro fato interessante é que o custo final do sistema, com as 32 placas acionadoras, ficou em um valor aceitável para uma casa classe média, sendo pouco mais caro que uma central de alarme facilmente encontrada em lojas de eletrônica.

Durante a implementação dos softwares embarcados foi possível observar a diferença entre eles e um software para *desktop*. Primeiro que usar variáveis globais é a regra. Não existe orientação a objetos então foi necessário voltar a usar velhas técnicas de programação. E em informática embarcada é necessário, muitas vezes, se trabalhar em nível de *bit* para

aproveitar ao máximo o desempenho do microcontrolador. Da mesma forma muitas rotinas são “amarradas” de tal forma que fica difícil reaproveitar o código fonte sem fazer algumas alterações.

O sistema possui algumas limitações como o alto consumo elétrico das placas acionadoras que, quando ligadas todas as 32, pode gerar uma corrente alta demais. Outras limitações são a não detecção dos dispositivos elétricos estarem funcionando e o não tratamento de perdas de conexão momentâneas gerando trocas intermitentes de estados na tela do supervisório.

Por fim, o sistema permite muitas idéias para futuras expansões, sendo que ele será a base dos futuros projetos.

4.1 EXTENSÕES

A seguir algumas idéias de expansões.

a) Usar as entradas e saídas extras das placas acionadoras para controlar vários elementos elétricos como se fossem várias placas acionadoras.
b) Aperfeiçoar a comunicação de dados entre PC e central, para que várias centrais possam ser conectadas a um mesmo software supervisor.
c) Adaptar o sistema para que ele funcione como se fosse um alarme, fazendo com que a central dê respostas diante de certos eventos. É importante no caso de alarme que a central possua uma certa autonomia, logo a eletrônica dela precisaria ser re-analisada também.
d) Incluir na central um módulo de simulação de presença, fazendo com que determinados acionadores sejam ligados e desligados em horários pseudo-aleatórios, dando a impressão que a residência está habitada.
e) Fazer com que a monitoração e o controle dos acionadores seja possível pela Internet através de uma página com controle de usuários.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, Roterdan S. **automacao_residencial.ppt:** automação residencial, um pouco de história. Ribeirão Preto, 2003. 46 eslaides, color. PowerPoint for Windows 10.26. Arquivo baixado da Internet. Disponível em: <http://www.aureside.org.br/publicacoes/download/automacao_residencial.zip>. Acesso em: 2 nov. 2006.

ARARAT. **Synchronous TCP/IP library for Delphi, C++ Builder, Kylix and FreePascal.** [S.l., 2006]. Disponível em: <http://www.ararat.cz/synapse/>. Acesso em: 30 out. 2005.

ATMEL CORPORATION. **AVR 8-Bit RISC**. San Jose, [2006]. Disponível em: <http://www.atmel.com/products/AVR/>. Acesso em: 10 novembro 2006.

AXELSON, Jan. **Serial port complete:** programming and circuits for RS-232 and RS-485 links and networks. Madison, WI: Lakeview Research, 2000. 306 p.

BESEN, Nelson. **Sistema domótico para automação e controle de um cômodo residencial**. 1996. 69 f. Monografia (Especialização em Tecnologias em Desenvolvimento de Sistemas) – Universidade Regional de Blumenau, Blumenau.

CENSI, Angela. **Sistema para automação e controle residencial via e-mail**. 2001. 58 f. Monografia (Bacharelado em Ciências da Computação) – Universidade Regional de Blumenau, Blumenau.

COSTA, Eduard Montgomery Meira**. Introdução aos sistemas a eventos discretos e a teoria de controle supervisório**. Rio de Janeiro : Alta Books, 2005. 120 p.

DAAMEN, David. Sistemas de barramentos domésticos. **Elektor: eletrônica e microinformática,** Barueri-SP, n. 41, 2005. p. 8-13.

INTEL 8051. In: WIKIPEDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://en.wikipedia.org/wiki/Intel_8051>. Acesso em: 2 nov. 2006.

INTEL CORPORATION. **MCS 51/251 microcontrollers**. Santa Clara, [2006]. Disponível em: <http://www.intel.com/design/mcs51/>. Acesso em: 10 nov. 2006.

KEIL SOFTWARE. **UVision IDE & ebugger**. [S.l., 2003]. Disponível em: <http://www.keil.com/uvision2/>. Acesso em: 15 jun. 2005.

MICROCHIP TECHNOLOGY INC. **8-bit PIC Microcontrollers**. Chandler, [2006]. Disponível em: <http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=74>. Acesso em: 10 nov. 2006.

NICOLASI, Denys. **Microcontrolador 8051 detalhado**. 2. ed. São Paulo: Érica, 2001.

PROTEL. **Protel 99 SE**. [S.l., 2000]. Disponível em: <http://www.protel.com/>. Acesso em: 20 jun. 2005.

RS-485. In: WIKIPEDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://en.wikipedia.org/wiki/RS-485>. Acesso em: 2 nov. 2006.

SCADA. In: WIKIPEDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://en.wikipedia.org/wiki/SCADA >. Acesso em: 2 nov. 2006.

SOARES, Luiz Fernando G. **Redes locais**. Rio de Janeiro: Campus, 1986. 250 p.

SOARES NETO, Vicente. **Comunicação de dados:** conceitos fundamentais. São Paulo: Erica, 1993. 168 p.

SWEET, Michael R. **Serial programming guide for POSIX operating systems**. [S.l.]: Easy Software Products, 2005. Disponível em: <http://www.easysw.com/~mike/serial/serial.html>. Acesso em: 12 set. 2006.

TAFNER, Malcon Anderson; LOESCH, Claudio; STRINGARI, Sergio. **Comunicacao de dados usando linguagem C:** aplicacao em DOS e Windows**.** Blumenau: Ed. da FURB, 1996. 87 p.

WIN-TECH. **Listen32.** Lewisburg, [1999]. Disponível em: <http://www.win-tech.com/html/listen.htm>. Acesso em: 31 out. 2006.

APÊNDICE A – Função verificar transmissão de dados

O quadro 22 transcreve a função `verificar_transmissao_dados()` da central microcontrolada a fim de auxiliar nos comentários da implementação.

```
       void verificar_transmissao_dados(void) {
         struct comandoFila auxBufferRX;
         char auxBufferSaida[5];
         if(mens_com1_ok) {
           mens_com1_ok = 0;
           if(polling) {//atualiza matriz2 com dado recebido
             matriz2[(strtochar2elements(bufferRecepcao.endereco)-1)]                          =
bufferRecepcao.comando[0];
           } else {
             funcaoReacao(strtochar2elements(bufferRecepcao.endereco)-1,
bufferRecepcao.comando[0]);
             matriz1[(strtochar2elements(bufferRecepcao.endereco)-1)]                          =
bufferRecepcao.comando[0];
           }}
         if((posicaoRXFilaIni != 99)&&(posicaoRXFilaFim != 99)) { //se existir algo na fila de
dados recebidos
           //processa dado recebido
           if(polling) {
             //atualiza matriz2 com dado recebido
             auxBufferRX = filaComandosRX[posicaoRXFilaIni];
             matriz2[(strtochar2elements(auxBufferRX.endereco)-1)] = auxBufferRX.comando[0];
           } else {
             matriz1[(strtochar2elements(filaComandosRX[posicaoRXFilaIni].endereco)-1)]      =
auxBufferRX.comando[0];  //atualiza matriz1 com dado recebido
             funcaoReacao(strtochar2elements(filaComandosRX[posicaoRXFilaIni].endereco)-1,
auxBufferRX.comando[0]);
             retiraComandoRXFila();
             timeoutComando = 0;
             aguardandoResposta = 0;
             retiraComandoFila();
           }
         } else if((posicaoFilaIni != 99)&&(posicaoFilaFim != 99)&&(!polling)) { //se existir
Comando na fila
           if((aguardandoResposta == 1)&&(timeoutComando == 0)) { //estouro de timeout e estar
aguardando resposta
             //processa timeout
             retiraComandoFila();
             timeoutComando = 0;
             aguardandoResposta = 0;
             //coloca valor 'X' na matriz1 para o endereco que estourou timeout
           } else if(aguardandoResposta == 1) { //se estiver aguardando resposta mas não é
timeout ainda
             //não faz nada
           } else {
             //transmite comando
             strncpy(&auxBufferSaida[0], &filaComandos[posicaoFilaIni].endereco[0], 2 );
             strncpy(&auxBufferSaida[2], &filaComandos[posicaoFilaIni].comando[0], 3 );
             transmitir_string_com1(auxBufferSaida); //transmite comando ao acionador XX
             aguardandoResposta = 1;
             timeoutComando = 20;
           }} else if(tempoNovoPolling == 0) { //de 1 em 1 segundo será iniciado novo polling
           //inicializa matriz2 com 0 s
           transmitir_string_com1("99A00");  //transmite comando de polling (requisição de
status)
           tempoNovoPolling = 1000;
           tempoFimPolling = 650;
           polling = 1;
         } else if((tempoFimPolling == 0)&&(polling == 1)){ //a partir do momento que for
iniciado um polling, 640ms depois deverá cair aki, que é o tempo que todos os acionadores
tiveram para responder
           processaFimPolling(); //processa fim do polling
           polling = 0;
         }
}
```

Quadro 22 – função `verificar_transmissao_dados`