

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
CURSO BACHARELADO EM INFORMÁTICA

SISTEMA INTEGRADO E MULTIPLATAFORMA PARA CONTROLE REMOTO DE RESIDÊNCIAS

Prof. Fernando Gehm Moraes
Orientador

Alexandre Amory
Juracy Petrini Júnior

Trabalho final da disciplina de
Trabalho de Conclusão II

Porto Alegre, 4 de setembro de 2001

SISTEMA INTEGRADO E MULTIPLATAFORMA PARA CONTROLE REMOTO DE RESIDÊNCIAS

Alexandre Amory
Juracy Petrini Júnior

Sumário

| | | |
|------------|--|-----------|
| 1.1 | CONCEITOS BÁSICOS | 1 |
| 1.2 | REQUISITOS DE UM SISTEMA DE AUTOMAÇÃO DOMÉSTICA | 3 |
| 1.3 | BENEFÍCIOS DA DOMÓTICA..... | 5 |
| 1.3.1 | SEGURANÇA | 5 |
| 1.3.2 | CONFORTO..... | 6 |
| 1.3.3 | ENERGIA | 6 |
| 1.3.4 | COMUNICAÇÃO..... | 7 |
| 1.4 | MOTIVAÇÃO | 7 |
| 3.1 | FPGA | 13 |
| 3.1.1 | EVOLUÇÃO DOS DISPOSITIVOS PROGRAMÁVEIS..... | 13 |
| 3.1.2 | FPGA | 14 |
| 3.1.3 | BLOCOS LÓGICOS | 14 |
| 3.1.4 | ROTEAMENTO..... | 15 |
| 3.1.5 | CÉLULAS DE I/O | 16 |
| 3.1.6 | APLICAÇÕES DE FPGAS | 16 |
| 3.2 | VHDL | 17 |
| 3.2.1 | DESCRIÇÃO ESTRUTURAL | 19 |
| 3.2.2 | DESCRIÇÃO COMPORTAMENTAL..... | 19 |
| 3.2.3 | EXEMPLO DE DESCRIÇÃO VHDL..... | 20 |
| 3.2.4 | ESPECIFICAÇÃO DE VHDL | 21 |
| 3.3 | JAVA..... | 22 |
| 3.3.1 | MÁQUINA VIRTUAL JAVA | 22 |
| 3.3.2 | JAVA COMMUNICATIONS API (<i>COMMAPI</i>)..... | 22 |
| 3.3.3 | <i>JAVA SERVER PAGES</i> – JSP | 22 |
| 3.3.4 | COMPILADOR <i>JUST-IN-TIME</i> - JIT | 22 |
| 4.1 | INTRODUÇÃO..... | 23 |
| 4.2 | CARACTERÍSTICAS E REQUISITOS DE PROTOCOLOS..... | 25 |
| 4.3 | PROTOCOLOS ANALISADOS..... | 27 |
| 4.3.1 | <i>EUROPEAN HOME SYSTEMS</i> - EHS..... | 27 |
| 4.3.2 | <i>BYTE DATA LINK CONTROLLER</i> - BDLC | 31 |
| 4.3.3 | <i>LOCAL OPERATING NETWORK</i> - LON | 36 |
| 4.3.4 | <i>EUROPEAN INSTALLATION BUS</i> - EIB | 37 |
| 4.3.5 | X-10 | 38 |
| 4.3.6 | <i>CONSUMER ELECTRONICS BUS</i> - CEBUS | 40 |
| 4.4 | COMPARAÇÃO ENTRE OS PROTOCOLOS ESTUDADOS | 41 |
| 5.1 | APLICAÇÕES DE CAN | 43 |
| 5.2 | CONCEITOS BÁSICOS | 44 |
| 5.3 | ESPECIFICAÇÃO CAN..... | 45 |
| 5.3.1 | FORMATO DOS PACOTES | 46 |
| 5.3.2 | TÉCNICAS DE VERIFICAÇÃO E SINALIZAÇÃO DE FALHAS..... | 50 |
| 5.3.3 | ARBITRAÇÃO | 53 |
| 5.3.4 | BIT-TIMING | 54 |
| 5.3.5 | SINCRONISMO..... | 55 |
| 5.3.6 | CAMADA FÍSICA | 56 |
| 5.4 | IMPLEMENTAÇÃO..... | 58 |
| 5.4.1 | VARIAÇÃO QUANTO À INTEGRAÇÃO | 58 |
| 5.4.2 | VARIAÇÃO QUANTO À ARMAZENAMENTO DE MENSAGENS..... | 60 |
| 5.4.3 | VARIAÇÃO QUANTO AO FILTRO DE ACEITAÇÃO | 61 |
| 5.5 | ESTUDO DE ARQUITETURAS PARA PROTOCOLO CAN | 62 |
| 6.1 | ESCOLHA DO SISTEMA OPERACIONAL | 67 |
| 6.2 | ESCOLHA DA LINGUAGEM DE PROGRAMAÇÃO..... | 68 |
| 6.2.1 | JAVA | 69 |
| 6.2.2 | PERL | 70 |
| 6.2.3 | C..... | 71 |

| | | |
|------------|--|------------|
| 6.2.4 | ACTIVE SERVER PAGES - ASP | 72 |
| 6.2.5 | PHP : HYPERTEXT PREPROCESSOR | 73 |
| 6.2.6 | COMPARAÇÃO ENTRE LINGUAGENS DE PROGRAMAÇÃO | 73 |
| 6.2.7 | CONCLUSÃO | 74 |
| 6.3 | ARQUITETURA SERVIDOR/CAN..... | 75 |
| 6.4 | TIPO DE CONEXÃO ENTRE MÁQUINAS | 76 |
| 6.5 | ARQUITETURA CLIENTE/SERVIDOR | 76 |
| 7.1 | ARQUITETURA GERAL DO <i>SOFTWARE</i> | 79 |
| 7.1.1 | CLIENTE..... | 79 |
| 7.1.2 | SERVIDOR..... | 80 |
| 7.1.3 | <i>DEBUGGER (HARDWARE)</i> | 81 |
| 7.2 | ARQUITETURA DO SERVIDOR..... | 82 |
| 7.2.1 | COMUNICAÇÃO COM O CLIENTE..... | 82 |
| 7.2.2 | BANCO DE DADOS..... | 82 |
| 7.2.3 | COMUNICAÇÃO COM O BANCO DE DADOS / CONTROLADOR MESTRE (COMUNICAÇÃO SERIAL) | 87 |
| 7.3 | ARQUITETURA DO CLIENTE..... | 93 |
| 7.3.1 | AUTENTICAÇÃO DE USUÁRIOS NO SISTEMA | 93 |
| 7.3.2 | ATUALIZAÇÃO DA INTERFACE WEB..... | 95 |
| 7.3.3 | ATUALIZAÇÃO DO STATUS DA RESIDÊNCIA | 96 |
| 7.3.4 | MANUTENÇÃO DOS USUÁRIOS DO SISTEMA | 97 |
| 7.4 | ARQUITETURA DO <i>DEBUGGER</i>..... | 99 |
| 8.1 | DESCRIÇÃO DOS MÓDULOS HURRICANE..... | 103 |
| 8.1.1 | DESCRIÇÃO GERAL DOS MÓDULOS..... | 103 |
| 8.1.2 | MÓDULO CONTROLADOR..... | 107 |
| 8.1.3 | MÓDULO DE INTERFACE..... | 108 |
| 8.1.4 | MÓDULO CAN CORE | 109 |
| 8.1.5 | MÓDULO DE RECEBIMENTO | 110 |
| 8.1.6 | MÓDULO DE TRANSMISSÃO | 111 |
| 8.1.7 | MÓDULO DE CÁLCULO DE CRC..... | 112 |
| 8.1.8 | MÓDULO DE SINCRONISMO..... | 113 |
| 8.1.9 | MÓDULO DE STUFFING..... | 114 |
| 8.1.10 | MÓDULO DE CONTROLE DE ERROS..... | 115 |
| 8.1.11 | MÓDULO DE CONTADORES DE ERROS | 116 |
| 8.1.12 | RESUMO DE INFORMAÇÕES DO HURRICANE..... | 117 |
| 8.2 | ARQUITETURA DOS NODOS ESCRAVOS | 117 |
| 8.2.1 | NODO DE LÂMPADAS | 119 |
| 8.2.2 | RESUMO DE RELATÓRIO DE SÍNTESE | 120 |
| 8.2.3 | FORMAS DE ONDA DO OSCILOSCÓPIO..... | 121 |
| 8.2.4 | FORMAS DE ONDA POR SIMULAÇÃO | 122 |
| 8.3 | ARQUITETURA DO NODO MESTRE..... | 123 |
| 8.3.1 | RESUMO DE RELATÓRIO DE SÍNTESE | 125 |
| 8.3.2 | FORMAS DE ONDA POR SIMULAÇÃO | 125 |
| 8.3.3 | <i>TESTBENCH</i> | 130 |
| 8.4 | PACOTES DE DADOS X PACOTES DE CONTROLE..... | 132 |
| 8.5 | INTERFACE <i>HARDWARE/SOFTWARE</i> | 134 |
| 9.1 | INTERFACE PCI..... | 137 |
| 9.2 | PROTOCOLO <i>PLUG AND PLAY</i> | 137 |
| 9.3 | <i>ALIVE?</i>..... | 138 |
| 9.4 | TESTABILIDADE REMOTA..... | 138 |
| 9.5 | RECONFIGURAÇÃO REMOTA | 139 |
| 9.6 | MECANISMO DE TTL – TIME TO LIVE..... | 139 |
| 9.7 | CONTROLE DE ACESSO WEB POR MÚLTIPLAS PERMISSÕES..... | 140 |

Lista de Figuras

| | |
|---|----|
| FIGURA 1 – TELEAÇÃO | 2 |
| FIGURA 2 – BENEFÍCIOS DE DOMÓTICA | 5 |
| FIGURA 3 – APLICAÇÕES POSSÍVEIS DESTE PROJETO..... | 8 |
| FIGURA 4 – ARQUITETURA DO SISTEMA | 9 |
| FIGURA 5 – ESTRUTURA COMPLETA DO SERVIDOR | 10 |
| FIGURA 6 – ESTRUTURA DA APRESENTAÇÃO | 11 |
| FIGURA 7 – ESTRUTURA INTERNA DE UM FPGA..... | 14 |
| FIGURA 8 – ESTRUTURA DE UM BLOCO LÓGICO – XILINX XC3000 | 15 |
| FIGURA 9 – ESQUEMA DE UMA <i>SWITCH BOX</i> | 15 |
| FIGURA 10 – ROTEAMENTO EM FPGA..... | 16 |
| FIGURA 11 – EXEMPLO EM DIAGRAMA DE UMA DESCRIÇÃO ESTRUTURAL EM VHDL | 19 |
| FIGURA 12 – ESTRUTURA DO COMPONENTE CONT2 | 21 |
| FIGURA 13 – MODELO BÁSICO DE MÓDULOS PARA AUTOMAÇÃO..... | 23 |
| FIGURA 14 – ESTRUTURA INTERNA DE UM MÓDULO DE AUTOMAÇÃO INTELIGENTE..... | 24 |
| FIGURA 15 – ARQUITETURA DO PROTOCOLO EHS..... | 28 |
| FIGURA 16 – ESTRUTURA DA REDE EHS | 30 |
| FIGURA 17 – FORMATO DOS PACOTES EHS | 30 |
| FIGURA 18 – FORMATO DO PACOTE BDLC..... | 32 |
| FIGURA 19 – TIPOS DE <i>IN-FRAME RESPONSE</i> | 33 |
| FIGURA 20 – INTERFACE COM CPU | 33 |
| FIGURA 21 – GERENCIADOR DE PROTOCOLO | 34 |
| FIGURA 22 – INTERFACE MULTIPLEXADOR..... | 35 |
| FIGURA 23 – TOPOLOGIA LÓGICA EIB..... | 37 |
| FIGURA 24 – PACOTE EIB | 38 |
| FIGURA 25 – PACOTE DO PROTOCOLO X-10 | 39 |
| FIGURA 26 – TABELA DE COMANDO DO PROTOCOLO X-10..... | 39 |
| FIGURA 27 – MÉTODO DE DETECÇÃO DE ERROS DO PROTOCOLO X-10 | 39 |
| FIGURA 28 – FUNÇÕES DAS CAMADAS DE CAN..... | 45 |
| FIGURA 29 – FORMATO DE UM PACOTE DE DADOS CAN..... | 46 |
| FIGURA 30 – CAMPO DE ARBITRAÇÃO | 46 |
| FIGURA 31 – REQUISIÇÃO REMOTA DE DADOS..... | 48 |
| FIGURA 32 – PACOTE DE REQUISIÇÃO DE DADOS CAN..... | 49 |
| FIGURA 33 – PACOTE DE ERRO CAN | 49 |
| FIGURA 34 – PROCESSO DE SINALIZAÇÃO DE ERROS..... | 50 |
| FIGURA 35 – <i>BIT STUFFING</i> | 51 |
| FIGURA 36 – ESTADOS DE ERRO | 53 |
| FIGURA 37 – ARBITRAÇÃO CAN | 54 |
| FIGURA 38 – <i>BIT TIME CAN</i> | 55 |
| FIGURA 39 – SINCRONIZAÇÃO COM TRANSMISSOR LENTO..... | 56 |
| FIGURA 40 – SINCRONIZAÇÃO COM TRANSMISSOR RÁPIDO | 56 |
| FIGURA 41 – FORMA DA FIAÇÃO CAN..... | 57 |
| FIGURA 42 – TENSÃO NOMINAL..... | 57 |
| FIGURA 43 – INTERFERÊNCIA ELETROMAGNÉTICA | 57 |
| FIGURA 44 – <i>TRANSCEIVER</i> | 58 |
| FIGURA 45 – MÓDULO CAN <i>STAND-ALONE</i> | 59 |
| FIGURA 46 – MÓDULO CAN INTEGRADO..... | 59 |
| FIGURA 47 – MÓDULO CAN <i>SINGLE-CHIP</i> | 60 |
| FIGURA 48 – <i>BASICCAN</i> | 60 |
| FIGURA 49 – <i>FULLCAN</i> | 61 |
| FIGURA 50 – FILTRO SIMPLES | 61 |
| FIGURA 51 – MÚLTIPLOS FILTROS | 62 |
| FIGURA 52 – ARQUITETURA GERAL DO MÓDULO CAN..... | 62 |
| FIGURA 53 – ESTRUTURA DETALHADA DO MÓDULO CAN..... | 64 |
| FIGURA 54 – DETALHE DO BLOCO DE INTERFACE COM MEMÓRIA E APLICAÇÃO DO USUÁRIO..... | 65 |
| FIGURA 55 - ARQUITETURA DO SISTEMA..... | 79 |
| FIGURA 56 – TABELAS DO SISTEMA..... | 83 |
| FIGURA 57 - TABELA NÓDOS..... | 86 |
| FIGURA 58 – MÉTODOS DE ACESSO A BASE DE DADOS E PORTA DE COMUNICAÇÃO RS-232 | 88 |
| FIGURA 59 – PROBLEMA DE SEQUENCIAMENTO DE PACOTES..... | 89 |

| | |
|---|-----|
| FIGURA 60 – LISTA DE RECEBIMENTO DE PACOTES GRANDES | 90 |
| FIGURA 61 – TELA DE AUTENTICAÇÃO..... | 94 |
| FIGURA 62 - INTERFACE CLIENTE..... | 97 |
| FIGURA 63 – INTERFACE DE ADMINISTRAÇÃO..... | 98 |
| FIGURA 64 – INSERINDO USUÁRIOS | 98 |
| FIGURA 65 – ALTERANDO USUÁRIOS..... | 99 |
| FIGURA 66 – EXCLUINDO USUÁRIOS..... | 99 |
| FIGURA 67 – DEBUGGER..... | 101 |
| FIGURA 68- DISPOSIÇÃO HIERÁRQUICA DOS MÓDULOS..... | 104 |
| FIGURA 69 - INTERAÇÃO ENTRE OS BLOCO DO CAN <i>CONTROLLER</i> | 105 |
| FIGURA 70 – INTERAÇÃO ENTRE OS BLOCO DO CAN <i>CORE</i> | 106 |
| FIGURA 71 – PLATAFORMA NODO ESCRAVO (XS40) | 118 |
| FIGURA 72 – ESQUEMA XS40..... | 119 |
| FIGURA 73 – ARQUITETURA DO NODO DE LÂMPADAS..... | 119 |
| FIGURA 74 – MONITORAÇÃO PELO OSCIOSCÓPIO DE UM PACOTE CAN COMPLETO | 121 |
| FIGURA 75 –TRANSMISSOR X RECEPTOR CAN..... | 121 |
| FIGURA 76 – SIMULAÇÃO DE UM PACOTE CAN COMPLETO | 122 |
| FIGURA 77 – COMPARAÇÃO ENTRE BARRAMENTO, BITS DO TRANSMISSOR E DO RECEPTOR..... | 122 |
| FIGURA 78 – PROCESSO DE ENVIO DE PACOTES | 122 |
| FIGURA 79 – PROCESSO DE RECEPÇÃO DE PACOTES..... | 123 |
| FIGURA 80 – ARQUITETURA DO NODO MESTRE..... | 124 |
| FIGURA 81 – PLATAFORMA NODO MESTRE (VW300) | 125 |
| FIGURA 82 – CONVERSÃO CAN PARA SERIAL..... | 126 |
| FIGURA 83 – CONVERSÃO SERIAL PARA CAN..... | 127 |
| FIGURA 84 – TRANSMISSÃO SERIAL DO MESTRE | 127 |
| FIGURA 85 – RECEPÇÃO CAN NO MESTRE COM TRANSMISSÃO SERIAL | 128 |
| FIGURA 86 – ESCRITA E LEITURA NA FILA CAN | 129 |
| FIGURA 87 – ESCRITA E LEITURA DA FILA SERIAL..... | 129 |
| FIGURA 88 – ESTRUTURA DO <i>TESTBENCH</i> DO MESTRE..... | 130 |
| FIGURA 89 – BANCADA DE TESTE DO MESTRE | 132 |
| FIGURA 90 – DETALHES DA BANCADA DE TESTE | 132 |
| FIGURA 91 – FORMATO DE PACOTES DE DADOS..... | 133 |
| FIGURA 92 – FORMATO DE PACOTES DE CONTROLE | 133 |
| FIGURA 93 – INTERFACE <i>HARDWARE/SOFTWARE</i> | 134 |

Lista de Tabelas

| | |
|--|-----|
| TABELA 1 – TIPOS DE MEIOS FÍSICOS EHS..... | 29 |
| TABELA 2 – TABELA COMPARATIVA ENTRE PROTOCOLOS..... | 41 |
| TABELA 3 – LEGENDA DA TABELA COMPARATIVA DE PROTOCOLOS..... | 41 |
| TABELA 4 – COMPARATIVO ENTRE CAN PADRÃO E ESTENDIDO..... | 45 |
| TABELA 5 – CODIFICAÇÃO DO CAMPO DLC..... | 47 |
| TABELA 6 – TAXA DE TRANSFERÊNCIA X COMPRIMENTO DO BARRAMENTO..... | 58 |
| TABELA 7 – COMPARAÇÃO ENTRE DIVERSAS LINGUAGENS DE PROGRAMAÇÃO..... | 74 |
| TABELA 8 – TABELA USUARIOS..... | 84 |
| TABELA 9 – TABELA TABELANODOS..... | 85 |
| TABELA 10 - TABELA TABELADADOS..... | 86 |
| TABELA 11 - DESCRIÇÃO DA <i>PORT LIST</i> DO CAN <i>CONTROLLER</i> | 107 |
| TABELA 12 – DESCRIÇÃO DA <i>PORT LIST</i> DO CAN <i>INTERFACE</i> | 108 |
| TABELA 13 - DESCRIÇÃO DA <i>PORT LIST</i> DO CAN <i>CORE</i> | 109 |
| TABELA 14 - DESCRIÇÃO DA <i>PORT LIST</i> DO CAN RX..... | 110 |
| TABELA 15 - DESCRIÇÃO DA <i>PORT LIST</i> DO CAN TX..... | 112 |
| TABELA 16 - DESCRIÇÃO DA <i>PORT LIST</i> DO CRC CALC..... | 112 |
| TABELA 17 - DESCRIÇÃO DA <i>PORT LIST</i> DO SYNC..... | 113 |
| TABELA 18 - DESCRIÇÃO DA <i>PORT LIST</i> DO <i>STUFF HANDLER</i> | 114 |
| TABELA 19 - DESCRIÇÃO DA <i>PORT LIST</i> DO <i>ERROR FRAME</i> | 115 |
| TABELA 20 - DESCRIÇÃO DA <i>PORT LIST</i> DO <i>ERROR COUNTER</i> | 116 |
| TABELA 21 – RESUMO DE DADOS DOS MÓDULOS HURRICANE..... | 117 |
| TABELA 22 – RESUMO DE RELATÓRIOS DE SÍNTESE DO MESTRE..... | 125 |
| TABELA 23 – RELATÓRIO DO <i>TESTBENCH</i> DO MESTRE (CAN PARA SERIAL)..... | 130 |
| TABELA 24– RELATÓRIO DO <i>TESTBENCH</i> DO MESTRE (SERIAL PARA CAN)..... | 131 |

AGRADECIMENTOS

Durante o desenvolvimento deste trabalho tivemos ajudas essenciais que nos trouxeram contribuições tanto técnicas quanto emocionais. A essas pessoas, abaixo citadas, deixamos o nosso mais sincero obrigado.

Ao professor Fernando Moraes, que nos orientou durante essa jornada, agradecemos sinceramente à condução deste trabalho.

Ao professor Eduardo Bezerra, que mesmo afastado (em doutorado) teve uma contribuição inestimável neste do projeto.

Ao professor Ney Calazans, agradecemos por várias dicas dadas no decorrer deste trabalho.

Ao professor César Marcon, devido ao seu conhecimento da linguagem de programação Java.

A Letícia que nos ajudou muito na revisão de Português deste trabalho.

A todos membros do grupo Gaph, em especial Everton, José e Mesquita.

Ao grupo Sisc onde eu (Alexandre) trabalho como bolsista de IC, pois fiquei várias noites trabalhando nestes projeto usando recursos do grupo.

O objetivo deste trabalho é a implementação de um sistema de controle para aplicações domésticas, visando a automação de lares (domótica). Este sistema é acessado pela Internet, sendo os dispositivos da casa interligados pelo protocolo de controle CAN.

Os usuários (moradores) poderão controlar, monitorar e administrar seu lar a distância usando um navegador Web comum. Esta aplicação remota é o programa cliente. Localmente, um servidor traduz as instruções de/para o navegador Web e as distribui para as aplicações domésticas.

O sistema proposto neste trabalho apresenta como diferencial dos sistemas existentes as seguintes características: (i) atualização automática da interface do cliente; (ii) protocolo padrão de comunicação utilizado no *hardware*, o qual provê robustez e controle de erros; (iii) fácil inserção de novas aplicações na base de dados; (iv) mecanismos de autenticação de usuários garantindo a segurança do sistema; (v) interface de administração dos usuários do sistema. Além destas características, este trabalho apresente um sistema que integra *hardware* e *software*, com utilização de dispositivos programáveis do tipo FPGA.

Abstract

The objective of this work is to implement a control system for domestic applications – *domotics* or *smart houses*. This application is controlled through the Internet, being the home devices connected using the CAN control protocol.

The user remotely controls his house using a standard Web navigator. This remote application is a *client* program. Locally, the server translates the instructions from/to the Web navigator and distributes them to the domestic applications.

1 Introdução

Este trabalho é um estudo e implementação de um sistema que tem a finalidade de permitir que pessoas possam administrar seus lares remotamente, de uma forma simples, com um custo baixo (implantação e administração), utilizando uma plataforma portátil e flexível a avanços tecnológicos, facilitando a implementação de domótica em lares, escritórios e prédios.

Este capítulo caracteriza automação doméstica. A Seção 1.1 apresenta conceitos básicos em que esse trabalho está baseado. A Seção 1.2 mostra alguns requisitos de sistemas de domótica. A Seção 1.3 lista benefícios em termos de segurança, conforto, facilidade de comunicação, e economia de energia que o usuário de tal sistema de automação pode usufruir. Finalmente, a Seção 1.4 apresenta o que motiva o desenvolvimento deste projeto.

No Capítulo 0, a arquitetura do sistema é apresentada de forma genérica. Detalhes da arquitetura serão apresentados no Capítulo 6, onde apresentamos o sistema operacional, a linguagem de programação e a comunicação cliente/servidor usados. No Capítulo 3 apresentamos os conceitos de FPGA, VHDL e linguagem Java necessários para o acompanhamento da leitura deste trabalho. No Capítulo 4 apresentamos conceitos, requisitos e um estudo comparativo sobre protocolos de controle. Detalhes sobre o protocolo de controle CAN são apresentados no Capítulo 5. No Capítulo 7 e 8 são apresentados detalhes de implementação do sistema de *software* e *hardware*, respectivamente. Um conjunto de propostas para trabalhos futuros é apresentado no Capítulo 9. Uma lista de links relacionados a esse trabalho e referências bibliográficas encerram este relatório.

1.1 Conceitos Básicos

Domótica é a integração de tecnologias e serviços, aplicadas a lares, escritórios e pequenos prédios, com o propósito de automatizar e obter um aumento de segurança, conforto, comunicação e economia de energia [8] [9] [11] *link* [22].

Por automação entende-se a capacidade de se executar comandos, obter medidas, regular parâmetros e controlar funções de uma casa automaticamente.

A palavra domótica (“*domotique*”) surgiu na França, onde houveram as primeiras experiências relacionadas a domótica. Domótica é a contração da palavra *domus* do Latim

(equivalente a lar ou casa) com a palavra telemática. Outro sinônimo para domótica é casa inteligente (*smart house*), porém neste trabalho usaremos o termo domótica.

A domótica pode substituir o homem em diversas atividades rotineiras de forma a propiciar uma otimização nas condições de vida em uma casa. O próprio sistema zela pela satisfação dos moradores, sem que seja necessário a contínua intervenção dos mesmos.

O grau de controle alcançado pode ser variável, sendo uma função de custo, desejo pessoal dos moradores, estrutura do prédio e tecnologia usada. Casas que podem ter, por exemplo, ajuste automático de temperatura, escalonamento automático de tarefas rotineiras como ligar a cafeteira, acionamento automático de serviços de segurança e comunicação eficiente com o mundo externo têm vários benefícios que serão descritos na Seção 1.3.

Outro termo importante a se definir é o conceito de teleação (*teleaction*) [1] [2]. Teleação é a capacidade de se controlar algum dispositivo remotamente. A Figura 1 ilustra esse conceito.

Unindo os dois conceitos acima descritos (domótica e teleação) surgiu a idéia de interligar a rede interna de uma casa (domótica) com a rede externa à casa (Internet) de forma que os moradores da casa possam controlar, monitorar e administrar seu lar a distância, conforme pode ser visualizado na Figura 1. Uniremos neste projeto uma realidade atual (Internet e micro computadores domésticos) com uma tendência para o futuro (domótica).

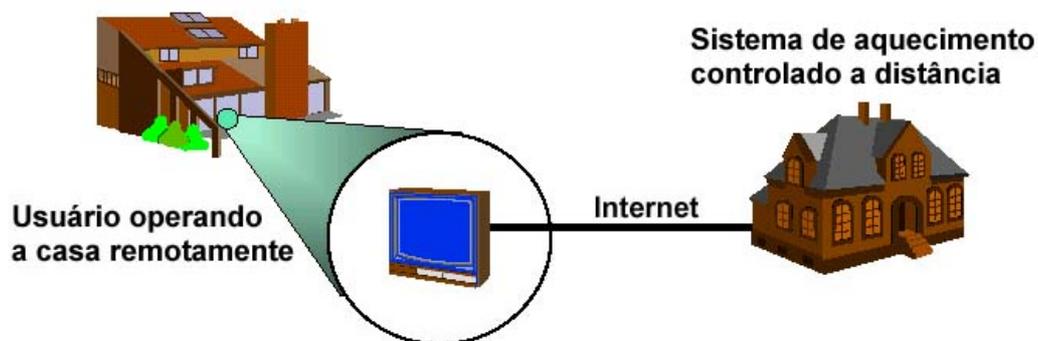


Figura 1 – Teleação

Com o enorme crescimento de novas aplicações que usam a Internet como meio de comunicação, novos produtos para aplicações de *infotainment* (informação + entretenimento) e comércio eletrônico estão surgindo. Com isso está havendo uma aproximação dos computadores aos lares. Isso tem estimulado o mercado SoHo (*Small office Home office*) a oferecer cada vez mais produtos para atender uma clientela cada vez mais exigente. Um outro efeito do aumento acelerado do mercado SoHo será a mudança dos lares de um simples ponto de acesso à Internet para um provedor de serviços e aplicações, como mostra a Figura 1.

Nos dias de hoje, existem algumas poucas empresas no mundo que desenvolvem produtos visando domótica [2]. Na Europa, principalmente, existem vários grupos que estão trabalhando no desenvolvimento de um protocolo doméstico padrão. Esses grupos, sem exceções, estão ligados a grandes fabricantes de eletrodomésticos e produtos de uso doméstico em geral. No Brasil, não se tem notícia de empresa que desenvolva produtos para domótica, mas existem revendedoras de empresas estrangeiras. Ainda assim, são raras essas empresas, pelo fato de que produtos visando domótica não serem acessíveis financeiramente à maioria da população.

1.2 Requisitos de um Sistema de Automação Doméstica

Para saber exatamente quais são as reais exigências que um sistema de automação doméstica deve ter é necessário uma equipe multidisciplinar, engenheiros, arquitetos, sociólogos, para identificarem o cenário doméstico, conforme [2]. Tal estudo terá a função de descrever a situação da domótica no futuro e definir por quais caminhos esse estudo deve seguir para alcançá-lo, no ponto de vista europeu.

O resultado deste estudo foi dividido em três partes: facilidades, vantagens e exigências técnicas.

Por facilidades entende-se todas as funções que o usuário gostaria de ter integrada num sistema, de modo a facilitar o seu cotidiano. Citamos as seguintes necessidades:

- manutenção doméstica;
- conservação e preparo de comida, preparo de roupas;
- comunicação e “home office”;
- entretenimento (áudio, vídeo, tv, rádio);
- controle de energia, aquecimento, água, iluminação, ou ventilação;
- segurança em relação a invasão, fogo, inundação, ou mau tempo.

Por vantagens entende-se os benefícios¹ que a domótica terá que propiciar para a satisfação do usuário final.

Essas necessidades e vantagens devem ser consideradas como pré-requisitos em um sistema de automação doméstica.

¹ benefícios serão detalhados na Seção 1.3.

Exigências técnicas são características que o desenvolvedor de sistema deve conhecer e saber contorná-los para implementar um sistema competitivo e com aceitação no mercado. As principais exigências são:

- baixo custo (sistema de fácil instalação e dispositivos baratos);
- *plug and play*;
- flexibilidade (sistema modular e extensível);
- integração com todas aplicações;
- integração com vários fabricantes (padronização);
- compatibilidade com vários meios (linha de energia, rádio-freqüência, par trançado, infra vermelho);
- confiabilidade;
- fácil utilização.

Baixo custo para o usuário final significa, primeiramente, que a instalação da rede deve ser fácil, de modo que necessite o mínimo possível de uma pessoa especializada para isso. Por outro lado, os dispositivos da rede devem ser baratos também. O usuário não vai pagar uma diferença de preço muito grande por uma lavadora de roupa conectada a rede se existe uma similar mais barata, mas que não pode ser conectada. Resumindo, o custo adicional para dispositivos de domótica devem ser extremamente baixo.

Os dispositivos devem ser *plug and play*, pois o próprio usuário deve ser capaz de fazer instalação da rede, de novos dispositivos e de reconfigurar a rede.

O ciclo de vida do sistema deve ser grande. O sistema instalado deve ser extensível. Deve ser possível integrar novos dispositivos e novas aplicações mesmo que esses ainda não existam. Isso implica que o sistema deve ser flexível a novas tecnologias.

Deve ser possível integrar todas as aplicações de um sistema de domótica. Não é uma boa estratégia instalar um sistema para controle de ar condicionado, outro para segurança, e assim por diante. As aplicações estando integradas possibilitam o compartilhamento de recursos e interoperabilidade.

Dispositivos de um fabricante devem ser conectados na mesma rede que dispositivos de um outro fabricante qualquer. Esta característica é a base de um sistema de automação.

O sistema deve ser confiável. Pelo fato do sistema integrar muitos dispositivos de diferentes características e fabricantes, aumentando-se as chances de que esse sistema se torne potencialmente sujeito a falhas. É desejável que o sistema inclua funções de monitoramento e teste, aumentando o grau de disponibilidade do mesmo.

Outro ponto muito importante é a facilidade de uso. A interface deve ser intuitiva e ter formas similares de executar funções diferentes, para facilitar o aprendizado do mesmo. Levando em conta que o sistema deve tornar algumas tarefas de casa independentes de sua intervenção.

1.3 Benefícios da Domótica

Os benefícios de domótica concentram-se em quatro classes: segurança, conforto, economia de energia e comunicação. Esses itens serão descritos abaixo e podem ser visualizados na Figura 2.



Figura 2 – Benefícios de Domótica

1.3.1 Segurança

Trata de proteger pessoas e pertences em casos de eventualidades como invasão, vazamento de água, vazamento de gás, incêndio, doenças, etc. Pode-se destacar como aplicações:

- alarmes técnicos: inundação, gás, queda de energia;
- fogo e fumaça: detecção rápida, alerta a moradores, chamada de bombeiros;

- invasão e assalto: comunicação à polícia, sistema de câmeras, foto das pessoas que passaram pela frente da porta dianteira ou tocaram a campainha;
- alarme médico: monitoramento e diagnóstico remoto de sinais vitais;
- simulação de presença: ligar música e luzes aleatoriamente.

1.3.2 Conforto

Abaixo exemplifica-se alguns controles relacionados a conforto:

- luz automática: acionamento por presença, som, hora ou luz ambiente;
- persianas: controle automático por presença de luz ambiente e chuvas, abertura automática de persianas pelo acionamento do despertador;
- centralização: ligar/desligar o sistema com um único botão;
- controle de temperatura: temperatura interna mantém-se sempre a um nível agradável;
- programação de eletrodomésticos: pode-se programar para que a cafeteira e o aquecimento da banheira liguem 10 minutos antes que o despertador seja acionado;
- abertura de portões.

1.3.3 Energia

Controles inteligentes podem evitar desperdício de energia.

- iluminação: desliga luzes automaticamente quando não houver pessoas em determinado ambiente;
- controle de temperatura: poder controlar aquecedores e ar condicionado de forma a minimizar o consumo de energia;
- controle de eletrodomésticos: acionar eletrodomésticos como lavadoras de roupa em horários que as tarifas são menores ou fora de horários de pico.

1.3.4 Comunicação

- segurança: chamada automática a bombeiros e polícia;
- entretenimento: interligação de áudio e vídeo, sinal de videofone na televisão;
- mensagens: envio de mensagens texto para distribuição no destino (*Broadcast*);
- *chat*: bate-papo entre o administrador remoto e pessoas no interior do lar, escritório, etc;
- conectividade: interligação entre casas, escritórios e prédios que utilizam a domótica (WAN).

1.4 Motivação

Tem-se por motivação deste trabalho o desejo de obter experiência no desenvolvimento de *hardware* (protocolo de controle), *software* (comunicação Cliente/Servidor e RS-232C), integração harmônica entre o *hardware* e o *software* e o desenvolvimento da interface da aplicação (*home page* da casa).

Outra grande motivação está no apelo comercial e futurístico da idéia. Esse trabalho pode dar continuidade a trabalhos futuros visando uma versão comercial. A longo prazo, poderíamos vislumbrar uma empresa que, baseado neste trabalho, implementasse desde a *home-page* (a interface), até soluções de conectividade e controle de casas. Atualmente, uma empresa que forneça tal serviço ainda não existe, a nosso conhecimento, no Brasil.

Para ilustrar o apelo futurístico desta aplicação mostramos na Figura 3 algumas situações possíveis em uma casa com um sistema de domótica interligado à Internet.

Digamos que Maria está saindo do seu trabalho (caso 1). Ela poderia usar o seu PDA (*personal digital assistant*) sem fio com acesso a Internet para conectar-se a sua casa e, por exemplo, ajustar o termostato geral e deixar a casa bem iluminada para quando ela chegar em casa.

No caso 2, o medidor de nível de óleo do sistema de aquecimento estava com nível baixo. Imediatamente o sistema manda uma mensagem para a distribuidora de óleo para que seja feito o reabastecimento.

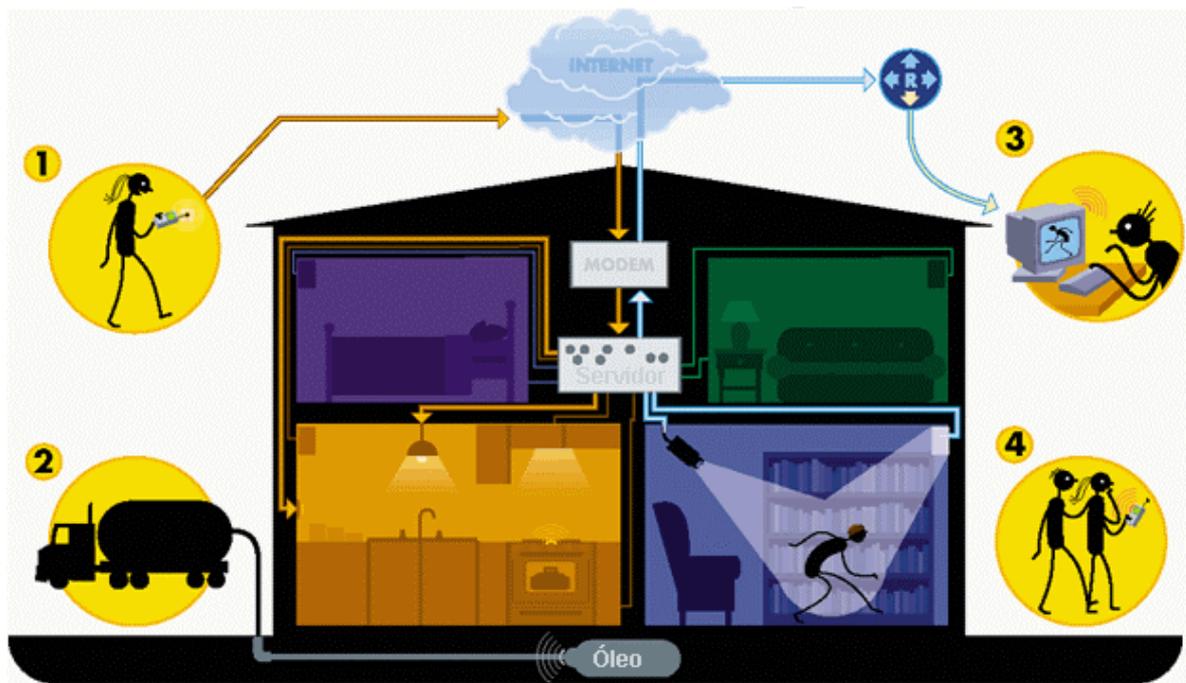


Figura 3 – Aplicações possíveis deste projeto

Já no caso 3, João está trabalhando quando de repente recebe uma mensagem urgente comunicando uma invasão à sua casa. Imediatamente ele se conecta à casa e, através das câmeras de segurança, consegue ver a imagem do invasor. Ao mesmo tempo que o sistema já havia enviado uma mensagem à polícia notificando a invasão.

O caso 4, digamos que João e Maria decidem sair para ir ao cinema. Ao saírem da casa o sistema avisa ao PDA que esqueceram de ativar o sistema de alarme. Então, pelo próprio PDA, João aciona o sistema e verifica se todas as portas e janelas foram bem fechadas.

Os quatro exemplos citados acima podem parecer muito longe da realidade atual, porém não estão. Já existem PDAs com acesso a Internet (*link* [15]) e, durante a execução desse trabalho, encontramos alguns sistemas de supervisão comercial com objetivo similar ao objetivo desse trabalho (*links* [16] [17] [18] [19] [20] [21]). Isso vem provar que já existem tecnologias convergindo para o objetivo deste projeto.

Visamos como meta do desenvolvimento deste projeto usar tecnologia de baixo custo ou de livre distribuição de modo que uma possível comercialização do produto venha a ser viável. Durante as Seções seguintes poderá ser notado que detalhes como a linguagem de programação e o sistema operacional foram escolhidos de forma e tornar uma versão comercial deste projeto viável, visto que esta é uma motivação importante deste projeto.

Outra motivação ao desenvolvimento deste trabalho foi a criação de uma nova aplicação para computadores e Internet. Pensamos que essa aplicação poderá contribuir de forma a fazer com que o uso de computadores faça cada vez mais parte do dia-a-dia do usuário comum.

2 Arquitetura do Sistema

O sistema desenvolvido baseia-se na comunicação entre dois computadores através do protocolo de comunicação HTTP, um destes computadores é denominado de Servidor e o(s) outro(s) computador(es) denominado(s) Cliente(s). Internamente à casa existirá uma outra rede utilizando o protocolo CAN² responsável em interligar os diversos periféricos da casa. Esta rede interna, através de um Controlador Mestre, comunica-se com o computador Servidor através da porta RS-232C. A Figura 4 ilustra a estrutura geral do sistema.

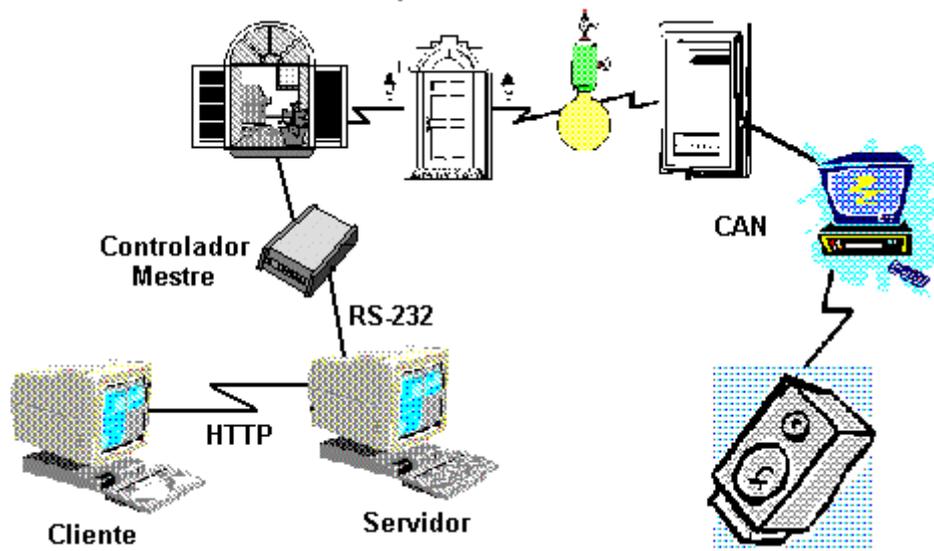


Figura 4 – Arquitetura do Sistema

O computador Cliente é responsável por permitir que o usuário possa de maneira amigável, interagir com sua residência. Para que isso seja possível, no computador cliente, temos um *software* responsável pelo envio de sinais de controles que neste caso iremos denominar de pacotes de controle. Estes pacotes serão recebidos pelo Servidor que se encarregará de tratá-los de maneira adequada. Devido ao crescimento e popularização da Internet no mundo, o *software* utilizado no lado Cliente será um navegador Web.

Sendo assim, o navegador Web é responsável em enviar os pacotes de controle ao Servidor através do protocolo HTTP, e ao mesmo tempo é responsável em manter a interface com o usuário atualizada a medida que o Servidor envia atualizações.

O Servidor é responsável por receber pacotes (Figura 5 - 1) de controle que serão enviados a partir do navegador Web que estiver rodando na máquina Cliente. O Servidor por sua vez

² Detalhes do protocolo CAN serão analisados no Capítulo 5

interpretará estes pacotes de controle recebidos do Cliente e é responsável em atualizar uma base de dados e repassar estes pacotes de controle através da porta RS-232C para o Controlador Mestre da rede interna da casa.

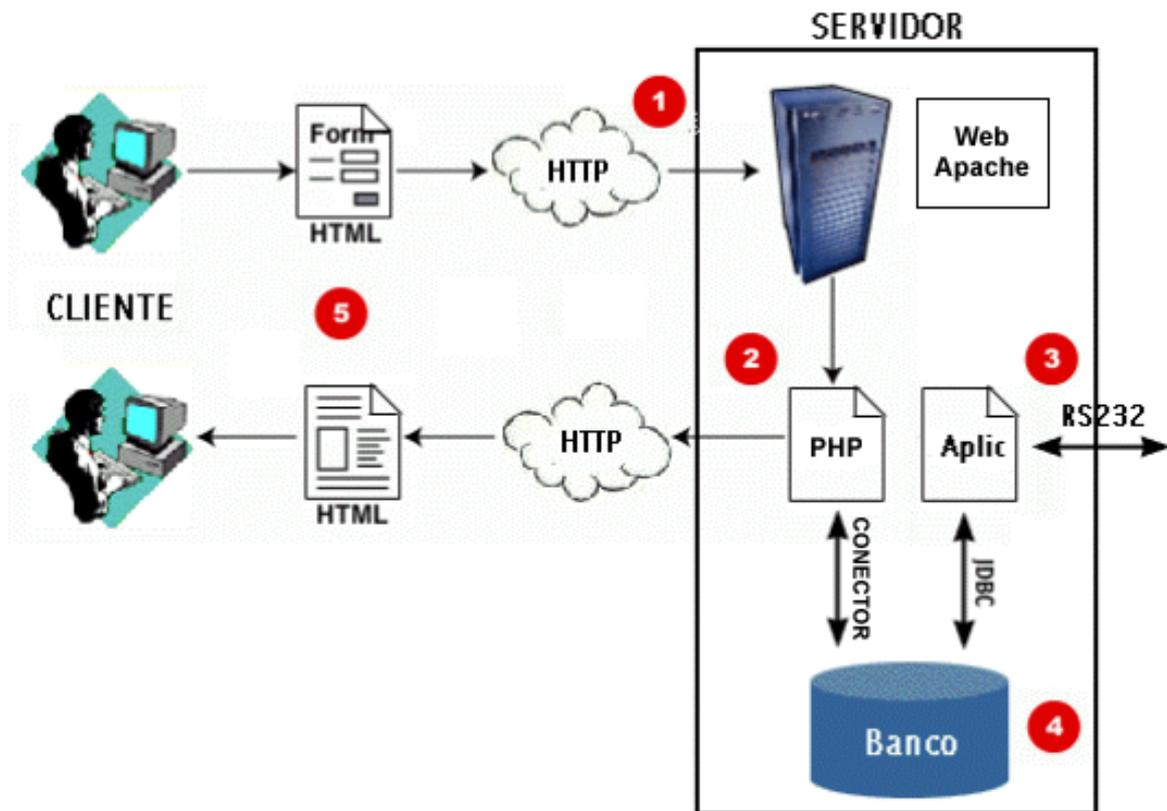


Figura 5 – Estrutura completa do servidor

Para viabilizar o desenvolvimento da tarefa atribuída ao computador Servidor, utilizaremos um servidor Web Apache ([link \[33\]](#)), responsável em disponibilizar a interface (a partir deste momento começaremos a tratar esta interface como sendo a *homepage* da residência) ao(s) Cliente(s) (navegador Web). Juntamente com o servidor Web é utilizada a linguagem de programação PHP, [link \[28\]](#) (Figura 5 - 2) cuja responsabilidade é receber os pacotes de controles enviados pelo usuário através do navegador Web, atualizando uma base de dados (Figura 5-4) contida no servidor. Esta base de dados tem como principal função manter o estado atualizado sobre cada aplicação doméstica que está sendo controlada remotamente.

Enquadra-se como responsabilidade do Servidor a atualização, de maneira transparente ao usuário, da interface do navegador Web (Cliente). Em outras palavras, isso possibilita que o usuário tenha sempre o último estado (Figura 5 - 5) sobre sua residência sem a necessidade de ficar acionando um comando de atualização da interface.

Ainda no contexto do Servidor, existe uma aplicação (Figura 5 - 3) escrita em Java, responsável pela comunicação com a porta serial RS-232C possibilitando o envio de pacotes de

controle ao Controlador Mestre e também do recebimento dos pacotes de controle provenientes do Controlador Mestre.

Outro objetivo principal da aplicação Java é garantir a consistência das aplicações domésticas da residência de acordo com o seu respectivo status no banco de dados, pois este último reflete diretamente os comandos de atualização enviados pelo usuário a sua casa.

A autenticação de usuários e criptografia dos dados são mecanismos utilizados para garantir a segurança do sistema, impedindo que pessoas desautorizadas tenham acesso ao sistema. Como solução para esta preocupação, utiliza-se o conceito de login e criptografia de dados garantindo à segurança do sistema.

O Controlador Mestre da rede residencial fará a codificação dos dados recebidos pelo Servidor para o protocolo CAN. Além disso, tarefas como endereçamento dos pacotes, gerenciamento do barramento e decodificação dos dados recebidos dos periféricos para o servidor, também serão tarefas do Controlador Mestre.

A proposta é instalar na casa uma rede serial. Essa rede interligará todos os periféricos e nela tráfegarão os comandos e dados necessários para se atingir o objetivo de automatizar a casa. Estes periféricos serão controlados pelo Controlador Mestre que receberá comandos do Servidor.

O protocolo CAN será implementado utilizando um *core* disponível [38], descrito em VHDL [3], e implementado em placas de prototipação que emularão alguns periféricos de uma casa real. A Figura 6 apresenta uma proposta para simular o controle de periféricos de uma casa.

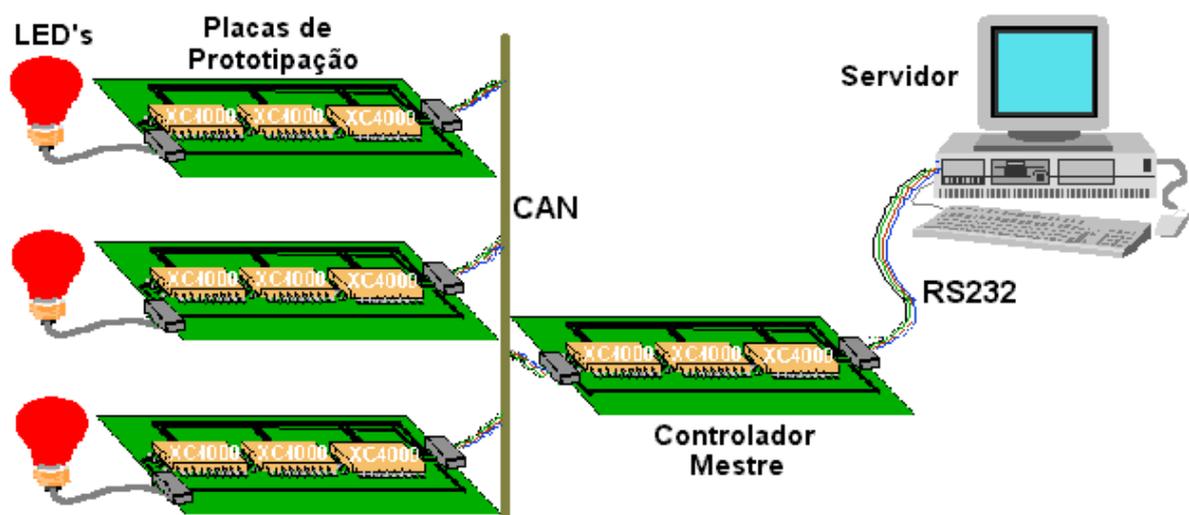


Figura 6 – Estrutura da apresentação

Apesar do projeto ser especificado com a flexibilidade necessária para que possa monitorar e acionar grande parte dos periféricos comuns em uma casa, esse projeto se limitará a ter como cargas LEDs e displays de sete segmentos que simbolizarão os periféricos reais da casa, conforme

Figura 6. O acionamento de cargas maiores como lâmpadas, demandam um circuito que comporte maior potência. O conhecimento da montagem de tal circuito não faz parte do escopo do atual trabalho. As placas de prototipação poderão enviar dados ao servidor através de chaves e botões. Para simular a fiação da casa usaremos o cabos seriais usados para comunicação de RS232C

3 Definições

O objetivo deste capítulo é abordar definições fundamentais que são a base deste projeto, portanto a compreensão destes conceitos é fundamental. A Seção 3.1 define o que é FPGA. A Seção 3.2 define a linguagem VHDL. Na Seção 3.3 definem-se alguns conceitos relacionados à linguagem de programação Java.

3.1 FPGA

A tecnologia VLSI abriu as portas para a implementação de circuitos digitais poderosos e de baixo custo. Porém o processo de manufatura desses circuitos perdura meses. Isso resulta em um alto preço a não ser que haja produção em volumes muito grandes.

Na indústria eletrônica é vital que novos produtos alcancem o mercado o mais rápido possível, e para isso reduzir o tempo de desenvolvimento e produção é essencial. *Field Programmable Gate Array* (FPGA) surgiu como solução para esse problema, porque prevê implementação imediata e com baixo custo de prototipação.

3.1.1 Evolução dos Dispositivos Programáveis

Dispositivos programáveis são circuitos de propósito geral que podem ser configurados para uma grande variedade de aplicações.

Um dos primeiros tipos de dispositivo programável foram as *Programmable Read-Only Memory* (PROM). As PROMS consistem de uma matriz de células que pode ser programada somente uma vez. Esse dispositivo é usado para implementar tabelas verdade.

Outro dispositivo programável, específico para implementação de circuitos lógicos, é o *Programmable Logic Device* (PLD). Esse dispositivo é baseado em uma matriz de portas E conectada a uma matriz de portas OU. PLDs possibilitaram um grande avanço na implementação de circuitos lógicos, porém, devido a sua estrutura simples (limitação para lógica seqüencial), só se podia desenvolver pequenos circuitos lógicos.

O *Mask Programmable Gate Arrays* (MPGA) é um outro tipo de dispositivo programável que pode interconectar elementos de acordo com especificações do usuário. A maior vantagem dos

MPGAs sobre os PLDs é que eles possuem uma estrutura mais genérica permitindo a implementação de circuitos maiores.

3.1.2 FPGA

Assim como um MPGA, um FPGA (*Field Programmable Gate Array*) é um circuito programável composto por um conjunto de células lógicas ou blocos lógicos alocados em forma de uma matriz [6] [7]. Em geral, a funcionalidade destes blocos assim como o seu roteamento, são configuráveis por *software*. A Figura 7 ilustra a organização interna de um FPGA com arquitetura de roteamento baseada em canais horizontais e verticais (exemplo: Xilinx família XC4000).

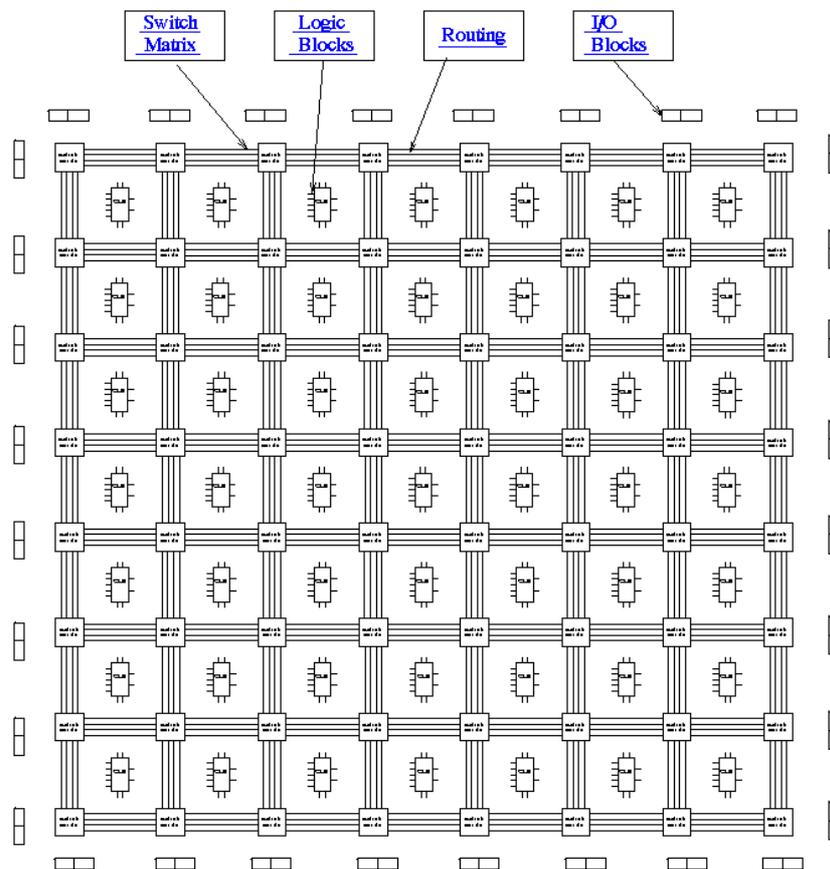


Figura 7 – Estrutura interna de um FPGA

3.1.3 Blocos Lógicos

As funções lógicas são implementadas no interior dos blocos lógicos. A arquitetura de um bloco lógico pode ser desenvolvida de várias formas e ter vários recursos. Cada fabricante e família de dispositivos pode ter uma arquitetura diferente. Porém, é importante que essa escolha vise a maior versatilidade possível. A Figura 8 apresenta a arquitetura interna de um bloco lógico de um FPGA XC 3000 fabricado pela Xilinx.

Em algumas arquiteturas os Blocos Lógicos possuem recursos sequenciais tais como *flip-flop* ou registradores. No CLB da Figura 8, por exemplo, há dois *flip-flops*.

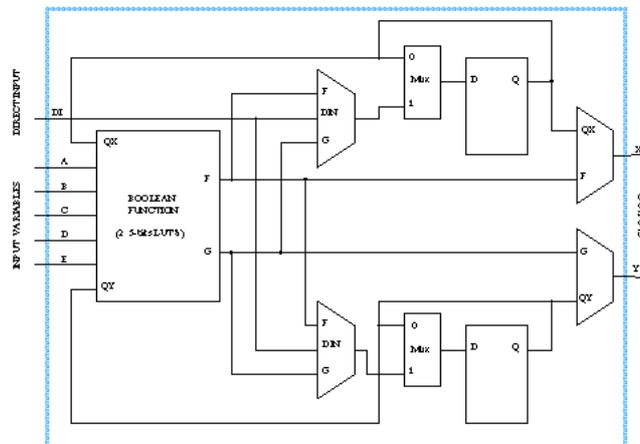


Figura 8 – Estrutura de um bloco lógico – Xilinx XC3000

3.1.4 Roteamento

Roteamento é responsável pela interconexão entre os blocos lógicos. As conexões físicas entre os fios são feitas ora com transistores de passagem controlados por bits de memória (PIP) ora com chaves de interconexão (*Switch Matrix*).

Os recursos de roteamento da série XC4000 da Xilinx possuem:

Conexões Globais: formam uma rede de interconexão em linhas e colunas de cinco fios, que se ligam através de chaves de interconexão. Esta rede circunda os blocos lógicos (CLBs) e os blocos de E/S (**IOBs**);

Matrizes de Conexão (*Switch Matrix*): são chaves de interconexão que permitem o roteamento entre os canais de roteamento (Figura 9). Estas conexões são programáveis na fase de roteamento automático, executada pelo *software* de projeto do fabricante do FPGA.

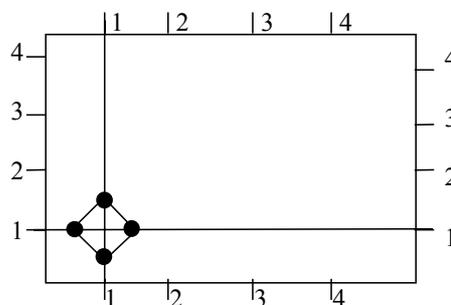


Figura 9 – Esquema de uma *Switch Box*

Conexões Diretas: são conexões entre CLB's vizinhos e permitem conectar blocos com menor atraso, pois não utilizam os recursos globais de roteamento.

Linhas Longas: são conexões que atravessam todo o circuito sem passar pelas *switch matrix* e são utilizadas para conectar sinais longos e com restrições de *skew* entre múltiplos destinos (Figura 10)

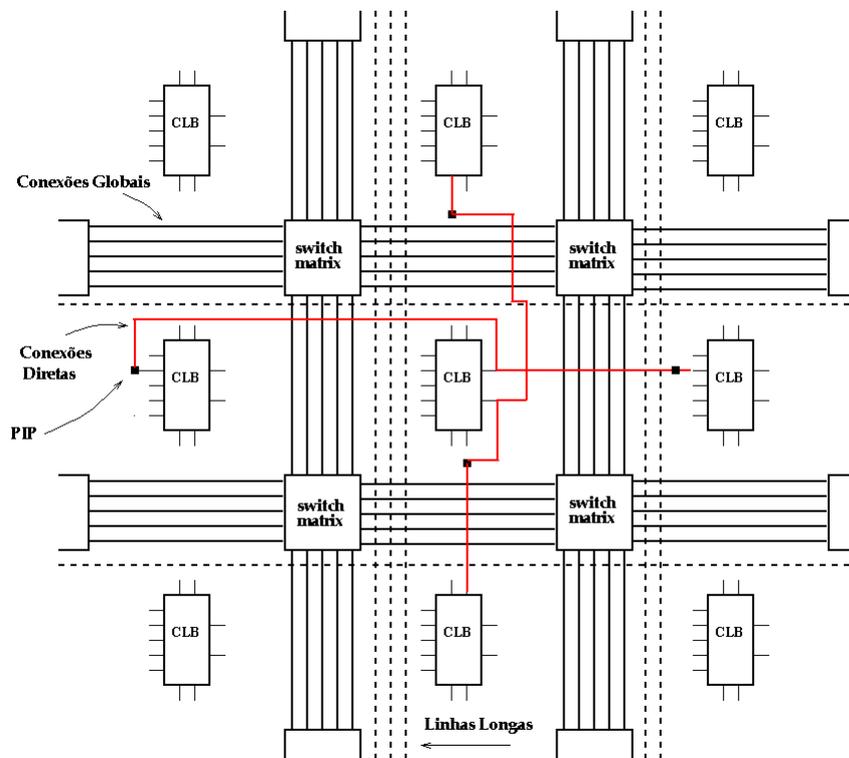


Figura 10 – Roteamento em FPGA

3.1.5 Células de I/O

As *I/O Cells* ou *I/OBs* (*I/O Blocks*)³ são blocos que se localizam na periferia do FPGA, como mostra a Figura 7, e têm a função de fazer a interface com o mundo externo. Os blocos de E/S podem ter capacidade para usar *buffers tristate* e *flip-flops* para os sinais da saída.

3.1.6 Aplicações de FPGAs

- Implementação de Lógica Randômica - Lógica randômica era usualmente desenvolvida em PLA⁴, porém FPGAs atuais podem armazenar o equivalente a vários PLAs e com melhor desempenho elétrico;

³ Blocos de Entrada e Saída de Dados

⁴ *Programmable Logic Array* (Matriz de Programação Lógica)

- Integração de Componentes - Pode-se diminuir a área de uma placa de circuito impresso integrando para dentro de um FPGA vários componentes simples da placa (lógica de cola);
- Prototipação - FPGAs foram idealmente desenvolvidos visando aplicações de prototipação. O baixo custo e curto tempo de implementação de aplicações traz grandes vantagens sobre a técnica tradicional. Assim, prototipação pode ser feita rapidamente e sem custos adicionais em caso de mudanças no projeto;
- Reconfiguração de *Hardware* - Devido a sua grande flexibilidade de interconexão, FPGAs são aplicados em sistemas onde o *hardware* pode ser reconfigurado durante funcionamento. Existe a possibilidade de se fazer atualização de *hardware* remotamente, sem que seja necessário trocar de placa.

3.2 VHDL

VHDL é uma linguagem textual para descrever sistemas digitais. Pode-se, através de um descrição VHDL, descrever o comportamento de um circuito ou componente digital [3] [5].

Uma HDL é uma linguagem que descreve a funcionalidade de um dado circuito. Essa descrição poderá ser verificada em um simulador. A diferença é que esse sistema descrito em HDL poderá ser implementado em *hardware* em um dispositivo programável FPGA ou outro tipo de ASIC.

Para manter o crescimento do desenvolvimento de dispositivos cada vez maiores e mais complexos, os métodos de desenvolvimento e ferramentas para projetos de sistemas digitais também deveriam acompanhar esse crescimento. Para resolver esse problema foi reunido um grupo de especialistas em sistemas digitais, incluindo empresas, universidades e órgãos militares. O foco principal era aprimorar o processo de desenvolvimento, aumentar a produtividade do desenvolvedor e diminuir o ciclo de desenvolvimento de sistemas digitais. Os dois maiores avanços resultantes desse trabalho foi a criação das HDLs e de seus sistemas de síntese lógica.

Existem dezenas de outras linguagens de descrição de *hardware* (Verilog, Abel, Handel-C, AHDL, etc.), porém VHDL foi adotada para esse projeto por ser uma linguagem padrão. Com isso, ela é a mais largamente usada e com mais documentação disponível. Além disso, essa linguagem é a que os componentes do grupo mais dominam.

A complexidade de sistemas digitais modernos conduz cada vez mais ao uso de altos níveis de abstração durante o projeto. Linguagens de descrição de *hardware*, como VHDL, têm sido

crescentemente empregadas para descrever o comportamento desses sistemas digitais [3]. Essa descrição pode ser puramente comportamental, usada somente para fins de documentação e descrição funcional simulável, ou pode ser sintetizável.

Para se chegar a uma descrição sintetizável são necessárias várias etapas de refinamento da descrição. Esses refinamentos visam alterações na descrição de forma a alcançarem o subconjunto de instruções HDL específicas que a ferramenta de síntese suporta.

Entre os aspectos que favorecem o desenvolvimento de *hardware* usando HDLs podemos citar:

- *time-to-market* – Nos tempos atuais a evolução de tecnologias está acontecendo cada vez mais rápido. Se há dez anos atrás um produto demorava 6 meses para ser desenvolvido, mas permanecia no mercado por 2 anos, hoje um produto não permanece mais de 18 meses, logo o seu desenvolvimento deve levar bem menos tempo. Isso tem forçado o estudo de novas técnicas para diminuir o ciclo de desenvolvimento de sistemas digitais. O processo de síntese lógica automatizada ataca esse problema mecanizando etapas mais abstratas de desenvolvimento.
- menor ciclo de desenvolvimento – O ciclo de desenvolvimento pode ser reduzido com o uso de VHDL, devido à eliminação de geração, manutenção de esquemáticos e pela diminuição de erros de desenvolvimento pelo uso de simulação nos ciclos iniciais do projeto;
- menor custo de desenvolvimento – Diretamente ligado ao tópico anterior;
- aumento de qualidade no desenvolvimento – Essa vantagem é alcançada pelo fato que VHDL facilita o rápido experimento com diferentes arquiteturas e técnicas de implementação, e pela capacidade das ferramentas de síntese otimizarem um projeto tanto para área mínima quanto para velocidade máxima;
- evolução da tecnologia – Novos dispositivos surgem com mais capacidade e mais recursos internos;
- gerenciamento do projeto – Projetos em VHDL são mais fáceis de serem gerenciados que os projetos baseados em esquemático. Eles facilitam a estruturação de componentes (*top-down*), facilitam a documentação e são necessárias menos pessoas para desenvolver e verificar, sendo também mais simples modificar o projeto;
- VHDL é independente de tecnologia e fabricante, porém sabe-se que na prática não é independente de ferramenta de síntese e de simulação.

As desvantagens de se usar VHDL apontam, basicamente, para o aprendizado de uma metodologia nova e complexa. Citamos desvantagens tais como:

- Mudança de cultura;
- Aprendizado e treinamento;
- Escolha de uma ferramenta de desenvolvimento;
- Circuito é menos otimizado que esquemático;
- Ferramentas de síntese ineficientes.

3.2.1 Descrição Estrutural

Um sistema digital pode ser descrito como um módulo com entradas e saídas, onde o valor das saídas é uma função dos valores de entrada. A Figura 11 (a) ilustra esse conceito, onde o módulo F possui duas entradas A e B e uma saída Y. Em VHDL o módulo F é chamado de *entidade* e as entradas e saídas são chamadas de *portas*.

Um modo de descrever a função de um módulo é descrever a função de seus sub-módulos e suas conexões. Cada sub-módulo é uma *instância* de uma entidade, e as conexões que os conectam são chamados de *sinais*. A Figura 11 (b) mostra como a entidade F pode ser formada por instâncias das entidades G, H e I. Este tipo de descrição é chamada de descrição estrutural.

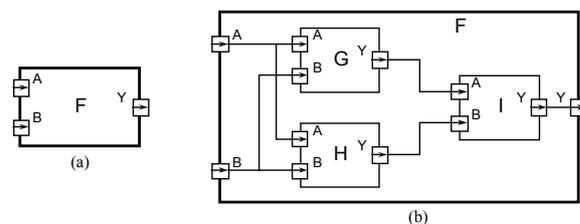


Figura 11 – Exemplo em diagrama de uma descrição estrutural em VHDL

3.2.2 Descrição Comportamental

Em muitos casos não é necessário descrever a estrutura interna de um componente, uma descrição que tenha o mesmo resultado funcional é o suficiente. Esse tipo de descrição é chamada de descrição *funcional* ou *comportamental*.

Para ilustrar essa idéia, suponha que a função da entidade F seja um ou exclusivo. A descrição comportamental de F poderia ser uma função como $Y = \overline{A} \cdot B + A \cdot \overline{B}$.

3.2.3 Exemplo de Descrição VHDL

Nesta Seção, descreveremos um simples exemplo de um contador de dois bits.

Logo abaixo está a descrição da entidade. A entidade especifica a interface externa do componente, incluindo os pinos de entrada e saída.

```
entity cont2 is
  port ( clock : in bit;
         q1, q0 : out bit);
end cont2;
```

Esse trecho de programa especifica uma entidade com uma entrada e duas saídas do tipo bit que assume valores 0 ou 1.

A função que um componente executa é especificada na *arquitetura* do componente. O trecho de programa abaixo especifica uma arquitetura comportamental para a entidade cont2.

```
architecture comportamental of cont2 is
begin
  count_up: process (clock)
    variable count_value : natural := 0;
  begin
    if clock = '1' and clock'event then
      count_value := (count_value + 1) mod 4;
      q0 <= bit'val(count_value mod 2);
      q1 <= bit'val(count_value / 2);
    end if;
  end process count_up;
end comportamental;
```

Dentro da arquitetura é definido um processo. Um processo é um bloco que é executado em paralelo com outros processos da mesma arquitetura. Os comandos dentro de um processo são executados seqüencialmente. O processo count_up é executado uma vez a cada mudança de valor no sinal de clock. O sinal clock é o ativador do processo count_up. A variável count_value, que é inicializada com zero, contém o valor da contagem.

Uma versão estrutural que executa a mesma função que a especificada na arquitetura comportamental pode ser vista logo abaixo. Cont2 é composto por dois *flip-flops* e por um inversor. A Figura 12 descreve a estrutura da entidade cont2.

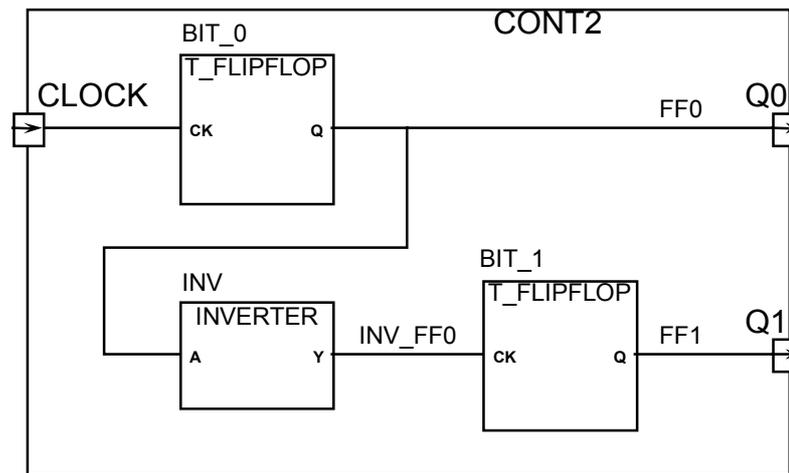


Figura 12 – Estrutura do componente Cont2

```

architecture estrutura of cont2 is
  component t_flipflop
    port (ck : in bit; q : out bit);
  end component;
  component inverter
    port (a : in bit; y : out bit);
  end component;
  signal ff0, ff1, inv_ff0 : bit;
begin
  bit_0 : t_flipflop port map (ck => clock, q => ff0);
  inv : inverter port map (a => ff0, y => inv_ff0);
  bit_1 : t_flipflop port map (ck => inv_ff0, q => ff1);
  q0 <= ff0;
  q1 <= ff1;
end estrutura;

```

Nesta arquitetura dois componentes e três sinais internos são declarados. Cada componente é então instanciado e as portas desses componentes são interligados. Por exemplo, bit_0 é uma instancia do componente t_flipflop e a porta ck deste é conectada à porta clock da entidade cont2.

3.2.4 Especificação de VHDL

Como esse trabalho não tem a pretensão de ser um manual de VHDL, e sim situar o leitor no assunto, não descreveremos detalhes da estrutura da linguagem VHDL. Como em outras linguagens, existem regras que devem ser seguidas pelo programador para se gerar um programa válido. A especificação detalhada dessas regras podem ser encontradas em [3] [4] [5], sendo que no *link* [27] encontra-se uma especificação gratuita da linguagem.

3.3 Java

Java ([17], [25] e *link* [14]) é uma linguagem de programação utilizada principalmente para o desenvolvimento de aplicações que rodem em ambientes distribuídos, como por exemplo a Internet, apesar de possibilitar aos programadores o desenvolvimento de aplicações que rodem em computadores isolados.

3.3.1 Máquina Virtual Java

Máquina hipotética (*software*) responsável em interpretar e executar o *bytecode* gerado pelo compilador da linguagem Java.

A Máquina Virtual Java [17] [25] é dependente da plataforma, devido à necessidade de interpretar o *bytecode* e gerar instruções para a máquina real executar.

3.3.2 Java Communications API (*CommAPI*)

Pacote de comunicação em Java desenvolvido pela *Sun Microsystems* (*link* [32]) que permite o desenvolvimento de aplicações com suporte a porta de comunicação serial RS-232C e portas paralelas padrão IEEE 1284.

3.3.3 Java Server Pages – JSP

Tecnologia desenvolvida com o objetivo de facilitar o desenvolvimento de aplicações Web dinâmicas, escritas em Java.

Java Server Pages (*link* [36]) consiste basicamente na construção de páginas Web utilizando HTML para a definição da interface, e juntamente com o HTML existe a inclusão de linhas de código Java, responsáveis em definir a parte dinâmica da aplicação. O código Java é identificado através de indicadores especiais de bloco.

3.3.4 Compilador *Just-in-time* - JIT

A máquina virtual Java vem acompanhada opcionalmente de um compilador especial denominado *just-in-time compiler* [17] [25] responsável pela compilação dinâmica dos *bytecodes* em códigos executáveis. Utilizado como alternativa para trechos de programas onde o interpretador Java deve executar muitas vezes, pois sabe-se que a execução de um código executável binário é muito mais rápida do que a sua execução através da interpretação. Exemplo: bloco de código dentro de um laço.

4 Protocolos de Controle

A crescente demanda de comunicação, conectividade e fluxo de informações impulsionou a criação de vários protocolos de comunicação. De acordo com as áreas de aplicação para as quais os protocolos são designados, eles podem ser diferenciados. O propósito dessa Seção é mostrar as métricas usadas para escolher o protocolo a ser implementado neste trabalho.

A Seção 4.1 introduz o leitor ao conceito de protocolo de controle. A Seção 4.2 apresenta características, parâmetros e requisitos de protocolos em geral. A Seção 4.3 apresenta um resumo com características principais de alguns dos protocolos estudados. A Seção 4.4 faz um resumo comparativo entre todos os protocolos estudados.

4.1 Introdução

Protocolo é a especificação de um conjunto de regras em que diversos equipamentos respeitam para trocar informações. Mais especificamente, protocolos de controle são empregados, principalmente na indústria, como linguagem de comunicação entre os módulos processadores responsáveis pelo controle de atuadores e monitoração de sensores. A Figura 13 ilustra esse conceito. Dentro de cada módulo processador que está ligado no meio de comunicação deve existir um sub-módulo (P de protocolo) que contém o algoritmo da “linguagem” que os processadores entendem. Esse algoritmo é chamado de protocolo de comunicação.

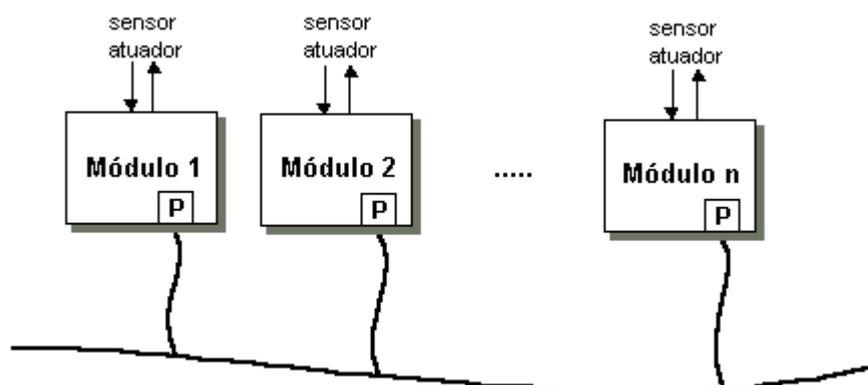


Figura 13 – Modelo básico de módulos para automação

A área de protocolos de controle tem ganho grande importância devido ao crescimento do mercado de automação (industrial, predial, doméstica, de escritório, entre outros). Outro fator importante é a taxa de crescimento da tecnologia dos dispositivos, que tem oferecido a cada ano mais funcionalidade com menores preços. Devido a isso, o desenvolvimento de sensores e atuadores inteligentes⁵ (Figura 14) tem sido impulsionado, exigindo a definição de padrões para os protocolos de comunicação. De um ponto de vista mais genérico, o mercado tem exigido soluções mais competitivas, que são caracterizadas por sistemas de controle altamente flexíveis às necessidades do consumidor. A chave para atingir essa meta é a padronização das interfaces de comunicação.

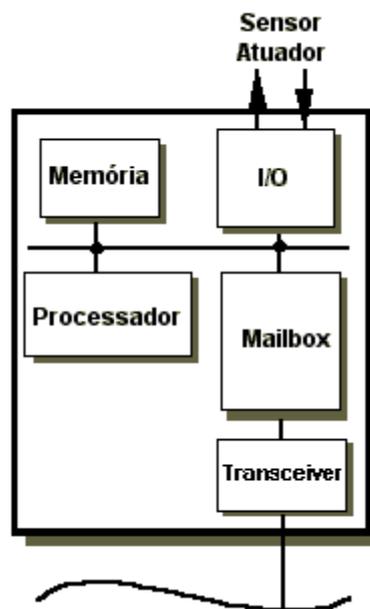


Figura 14 – Estrutura interna de um módulo de automação inteligente

Quando vamos fazer uma pesquisa de protocolos de comunicação para ser implementado em algum projeto, deve-se conhecer, *a priori*, a aplicação alvo. Hoje em dia existem em grande número de protocolos disponíveis. Cada um destes possui características que o “protocolo do concorrente não tem” ou simplesmente para fins de proteger seu sistema e restringir o consumidor aos produtos de um único fabricante. Devido ao fato de termos muitos protocolos disponíveis, os fechados⁶, proprietários e não padronizados tem perdido espaço para os protocolos abertos, certificados e padronizados.

Contudo, conhecer somente a aplicação do protocolo que será desenvolvido não basta, pois para uma aplicação específica, ainda existirão dezenas de possibilidades. Algumas características

⁵ Com a alta taxa de integração que, hoje em dia, dispositivos eletrônicos possuem é viável a construção de sensores e atuadores inteligentes que incorporam dentro de um único *chip* elementos de memória, de protocolo e de processamento para verificar limites de sensores, fazer linearização de sinais, etc.

⁶ sem especificação técnica do protocolo.

técnicas devem ser investigadas para se escolher corretamente o protocolo. A próxima Seção trata justamente disso, explicando características técnicas que devem ser estudadas.

4.2 Características e Requisitos de Protocolos

O estudo das características de protocolos depende muito da aplicação destes. Por exemplo, se procurarmos por um protocolo para ser usado em uma grande indústria metalúrgica esse protocolo deve ter longo alcance para poder se estender por toda empresa, poder ligar vários nodos, ter grande imunidade a ruídos usando técnicas de detecção e correção de erros, ter alta disponibilidade e tolerância a falhas, uma vez que grandes valores e vidas humanas podem estar em jogo. Já um protocolo para aplicação de entretenimento, basicamente só usado dentro do ambiente doméstico, não necessita ter alta disponibilidade, precisa ter grande taxa de transferência para transportar vídeo e som, um alcance curto já será o suficiente, compatibilidade entre vários fabricantes, entre outras características.

Esses dois exemplos citados acima, bem ortogonais, nos dão idéia que as características e requisitos de um protocolo podem variar muito, dependendo exclusivamente de sua aplicação. Porém, como dito na Seção anterior, conhecendo-se apenas a aplicação não se faz uma escolha adequada de protocolo. A seguir descrevemos mais algumas características técnicas que devem ser avaliadas:

- custo/benefício da técnica de cabeamento. A técnica de cabeamento e conexão é um dos itens que mais influenciam no custo total de instalação de uma rede;
- custo de manutenção e facilidade de diagnosticar problemas;
- confiabilidade – as técnicas de detecção e correção de erros são adequadas a sua aplicação? O ambiente onde a rede será instalada possui muito ruído? Confiabilidade pode ser aplicada aos dados, com técnicas como CRC, e no meio de comunicação;
- disponibilidade – existem aplicações onde paradas de sistema não são toleráveis. Um exemplo seria uma rede de processadores de um avião, onde a segurança de centenas de pessoas está em jogo. Nesse tipo de aplicação, o protocolo deve conter técnicas de tolerância a falhas visando minimizar a probabilidade de falhas do sistema;
- flexibilidade – capacidade de modificação do layout do sistema;
- compatibilidade – é importante que vários fabricantes suportem o protocolo escolhido para que você não fique dependendo de um só fabricante, tendo maior liberdade de escolha de equipamentos e de suporte técnico;

- parametrização – é um requisito diretamente relacionado com flexibilidade. Corresponde à facilidade de inserção de novos nodos na rede;
- variabilidade de aplicações – o mesmo protocolo pode ser empregado em aplicações diferenciadas?
- protocolos de alto nível – existem camadas de alto nível (aplicação) para o protocolo?
- metodologias, ferramentas para teste e diagnóstico;
- interface com PC;
- *drivers* de *hardware* e *software*;
- taxa de comunicação – a taxa máxima de comunicação do protocolo é compatível com o tempo de resposta do seu sistema?
- tempo de latência – atende as suas necessidades?
- técnica de endereçamento – suporta *broadcasting* e *multicasting*?
- metodologia de amostragem - amostragem baseada em *polling* ou baseada em eventos?
- a rede será multi-mestre ou com um único mestre? Uma rede que suporta múltiplos mestres tem uma maior disponibilidade. Em uma rede com um único mestre corre-se o risco de, se o mestre falhar, todo o sistema entrar em colapso;
- topologia – barramento, anel, estrela, hierárquico, entre outros. Topologia está relacionada com o número máximo de nodos da rede e com o comprimento máximo;
- resposta em tempo real – seu sistema exige resposta em tempo real?
- técnica de acesso ao meio – redes com múltiplos mestres devem ter um algoritmo de acesso ao meio que evite colisões e que, preferencialmente, tenha suporte a prioridade;
- requisição remota – um nodo pode pedir um dado para outro nodo na rede. Essa característica é útil para se verificar se um nodo está em funcionamento;
- confirmação de recebimento de pacotes por parte do destinatário - reconhecimento (*acknowledge*);
- sincronização inter-processador – deve-se manter sincronismo em uma rede onde dispositivos possuem velocidades diferentes;

- número de mensagens – número máximo de mensagens que o sistema suporta;
- comprimento físico da rede;
- número de bytes de dados – número máximo de bytes que um pacote pode transmitir;
- facilidade de integração – é um item relacionado ao custo total do sistema;
- maior aceitação no mercado – é mais fácil encontrar suporte e pessoal técnico;
- potencial de proliferação;
- padronização – verificar se o protocolo estudado é certificado e padronizado;
- *Plug and Play* – maior facilidade de instalação.

4.3 Protocolos Analisados

Esta Seção tem por objetivo explicar resumidamente as características dos principais protocolos analisados para a implementação deste trabalho. Muitos desses protocolos não possuíam documentação disponível e gratuita, por isso não conseguimos detalhá-los suficientemente. Porém acreditamos que estes deveriam constar neste trabalho para que pudéssemos fazer comparações e aprender mais sobre as características dos protocolos de controle. Essa Seção também é especialmente interessante para pessoas que estejam fazendo pesquisa de protocolos, pois neste documento o leitor encontrará uma fonte com referências resumidas de vários protocolos, facilitando seu trabalho de pesquisa.

Os protocolos analisados foram: EHS, BDLC, LON, EIB, X-10, CeBus e CAN. CAN porém, será explicado em detalhes no Capítulo 5.

4.3.1 *European Home Systems* - EHS

O protocolo EHS foi criado em 1992 por uma comissão de grandes empresas européias do ramo de eletrodomésticos [1] [2]. Esse esforço garantiu a criação de um protocolo aberto e com uma vasta gama de aplicações, permitindo que equipamentos de diferentes fabricantes comuniquem-se de forma que possam compartilhar recursos.

Hoje em dia o protocolo já chegou a maturidade. Já existem produtos em *hardware* e *software*. As maiores empresas européias do ramo eletrodoméstico já incluíram EHS em seus produtos.

A especificação completa deste protocolo chama-se *EHS specification* R1.2 e pode ser encontrada nos *links* [23] [24] [25].

4.3.1.1 Arquitetura

O modelo de comunicação EHS é semelhante a estrutura do modelo OSI. EHS especifica a camada física, de enlace, de rede e de aplicação, conforme Figura 15.

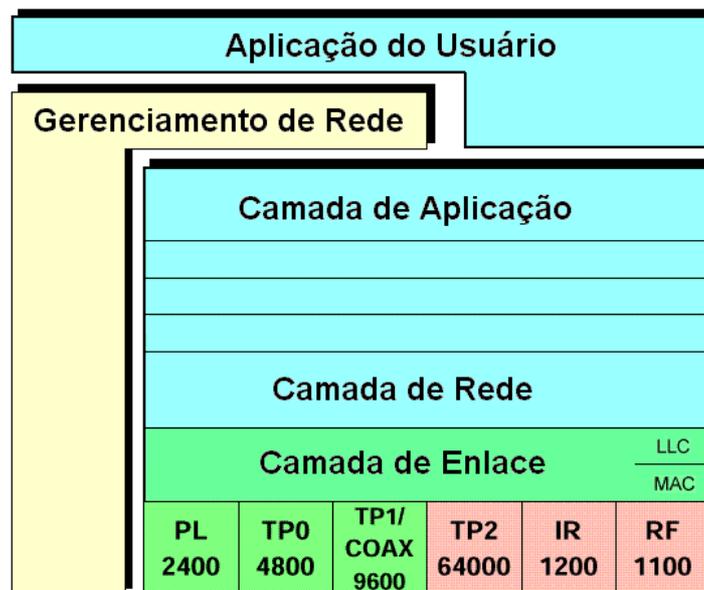


Figura 15 – Arquitetura do protocolo EHS

A camada de aplicação traduz a “linguagem” da aplicação em pacotes de dados capazes de circular na rede.

A camada de rede está relacionada ao roteamento e endereçamento dos pacotes.

A camada de enlace, dividida em MAC e LLC, gerencia a conversão de bits, regras de acesso a rede, recebimento e envio de pacotes e mecanismos de repetição.

Várias camadas físicas estão definidas devido ao grande número de aplicações que o protocolo abrange. Rede elétrica, infra-vermelho e rádio podem ser usados como canal de comunicação de baixa velocidade sem a necessidade de cabeamento extra. Um exemplo seria gerenciamento de aquecedores, ar condicionados e acionamentos remotos em geral. Par trançado e cabo coaxial podem ser usados quando se requer alta velocidade, por exemplo, aplicações de vídeo, áudio e segurança. As características de cada meio físico suportado pelo protocolo são mostradas na Tabela 1.

4.3.1.2 Características do Protocolo

- Plug and Play;

- Interoperabilidade;
- Expansionabilidade e configuração automática.

4.3.1.3 Meios Físicos

Uma parte importante de um sistema de automação doméstica é o meio de comunicação. A especificação EHS, versão 1.2, cobre seis meios para transportar informações sendo que outros meios ainda poderão vir a ser acrescentados.

| Meio Físico | Par Trançado tipo1 TP1 | Par Trançado tipo1 TP2 | Cabo Coaxial CX | Linha de Energia PL | Rádio RF | Infra-Vermelho IR |
|----------------------------|---------------------------|----------------------------------|-----------------------------------|---------------------|----------------------------|-------------------|
| Aplicação | Propósito geral, controle | telefonia, ISDN, dados, controle | Áudio, Vídeo, TV, dados, controle | controle | telefone sem fio, controle | controle remoto |
| Taxa de transmissão | 9.6 Kbps | 64 Kbps | 9.6 Kbps | 2.4 Kbps | 1.2 Kbps | 1.1 Kbps |
| Acesso | CSMA/CA | CSMA/CD | CSMA/CA | CSMA/ack | CT2 | - |
| Alimentação | 35 V | 35 V | 15 V | 230 Vac | - | - |
| Codificação | - | TDM | FDM | - | FDM | - |
| Topologia | Livre | barramento | barramento | livre | livre | livre |
| Unidades | 128 | 40 | 128 | 256 | 256 | 256 |
| Alcance | 500 m | 300 m | 150/50 m | casa | 50/200 m | sala |

Tabela 1 – Tipos de meios físicos EHS

Os meios mais importantes para o protocolo são o linha de energia e par trançado (TP1). Em um sistema onde o custo é prioridade o uso de linha de energia tem uma grande vantagem em relação a outros meios. Não é necessário cabeamento extra, pois todas as casas possuem um cabeamento da rede elétrica.

Vale comentar que o meio por linha de força usa uma técnica de detecção e correção de erros chamada FEC (*Forward Error Correction*) que adiciona 6 bits de codificação para cada 8 bits de dados. Tal redundância é necessária devido ao elevado nível de ruído que esse meio possui. Os outros meios possuem outro método de detecção de falhas chamado CRC (*Cycle Redundancy Check*).

4.3.1.4 Estrutura da Rede

EHS provê várias implementações de camadas físicas. Com isso a estrutura da rede pode ser formada por várias sub-redes, sendo cada uma baseada em uma camada física. Para interligar todas

as sub-redes usam-se roteadores, formando um único sistema de controle, como mostra Figura 16. Gateways são usados para interligar a rede EHS em uma rede não EHS.

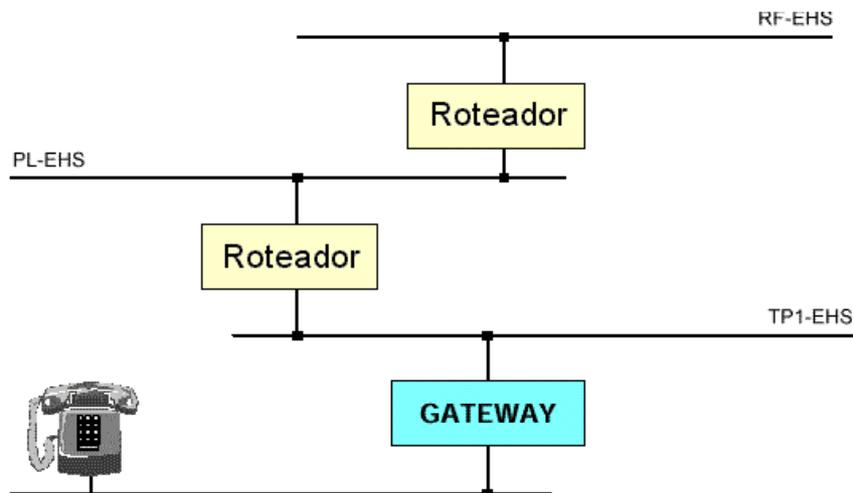


Figura 16 – Estrutura da rede EHS

4.3.1.5 Formato dos Pacotes

Não conseguimos um material que detalhasse a função de cada campo, porém podemos destacar o endereço de origem e destino do pacote, FCS como técnica de detecção de falhas e a prioridade da mensagem, como campos auto explicativos. A Figura 17 ilustra o formato do pacote EHS.

| | | | | | | | | |
|-----------|-----------|--------------|------------------|--------------|-------------|--------------|------|-----|
| Preâmbulo | Cabeçalho | End. da Casa | Controle de Link | Cód. de Prio | End. Origem | End. Destino | Dado | FCS |
| 2 | 2 | 2 | 1 | 1 | 2 | 2 | n | 2 |

Figura 17 – Formato dos pacotes EHS

4.3.1.6 Funções do Gerenciamento de Rede

EHS provê e integra várias funções de gerenciamento de rede. São explicados abaixo algumas dessas funções.

Registro

Quando uma nova unidade é instalada no sistema, a sua primeira função é executar o processo de registro. O registro é um processo automático responsável por carregar um novo endereço físico à unidade.

Inscrição

Depois do processo de registro, o processo de inscrição será iniciado. Durante essa etapa a unidade recém instalada informa a todas outras unidades instaladas quais são suas funções. Essa etapa é necessária para que todo o sistema tenha conhecimento dos recursos disponíveis de forma que possa ser compartilhado.

Esses dois processos não necessitam de intervenção do usuário, sendo uma das técnicas do protocolo para atender ao requisito *plug and play*.

4.3.1.7 Conclusões

EHS é um protocolo criado baseado nos requisitos e restrições que automação doméstica exige. Porém, percebemos que se trata de um protocolo muito amplo e com várias camadas de implementação (aplicação, rede, enlace e físico), por esse motivo não será o protocolo selecionado para este trabalho.

4.3.2 **Byte Data Link Controller - BDLC**

O módulo *Byte Data Link Controller* (BDLC) é um módulo de comunicação serial que permite ao usuário mandar e receber mensagens por uma rede *Society of Automotive Engineers* (SAE) J1850.

Este protocolo foi desenvolvido especificamente para aplicações automotivas, porém foi estudado por possuir fácil acesso a documentação completa e gratuita, através do *link* [26], e ser simples de implementar.

4.3.2.1 Características

Como características básica podemos citar:

- compatível com SAE J1850, que é um protocolo automotivo;
- modulação *Variable Pulse Width* (VPW) de 10,4 Kbps;
- filtro de ruído digital;
- detecção de colisão;
- geração e verificação de *Cyclic Redundancy Check* (CRC);
- dois modos de conservação de energia com *wake-up* automático;
- recebe e transmite blocos;

- recebimento em quatro vezes a velocidade normal (41,6 Kbps).

4.3.2.2 Formato do Pacote

A função dos blocos do pacote BDLC é mostrada na Figura 18 e citada logo abaixo.

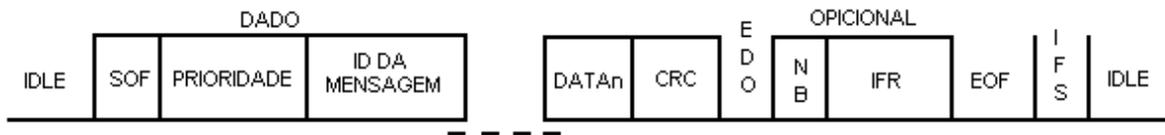


Figura 18 – Formato do pacote BDLC

- SOF – início do pacote (*start of frame*);
- DATA – bloco de dados. Este bloco deve conter pelo menos um byte de dados;
- CRC – bits usados em controle de erros. O polinômio gerador é $x^8 + x^4 + x^3 + x^2 + 1$;
- EOD – fim dos dados (*end of data*);
- IFR – *in-frame response*. Explicado na Seção 4.3.2.3;
- EOF – fim do pacote (*end of frame*);
- IFS – separador entre pacotes (*inter-frames separator*).

4.3.2.3 In-Frame Response

In-Frame response especifica formas de troca de informações entre transmissor e receptor. BDLC suporta 4 modos diferentes que podem ser visualizados na Figura 19.

Abaixo segue uma descrição dos tipos de respostas de requisição de dados e métodos de reconhecimento (*acknowledgement*) existentes no protocolo SAE J1850:

- Tipo 0 – Dado é enviado mas não é requisitado resposta;
- Tipo 1- Usado em *broadcast*. É necessário que um dos destinos responda para se validar o pacote enviado;
- Tipo 2- Usado em *broadcast*. É necessário que todos os destinos respondam para se validar o pacote enviado;
- Tipo 3 – Usado em requisição de dados. Receptor retorna dados para o transmissor do pacote de requisição.

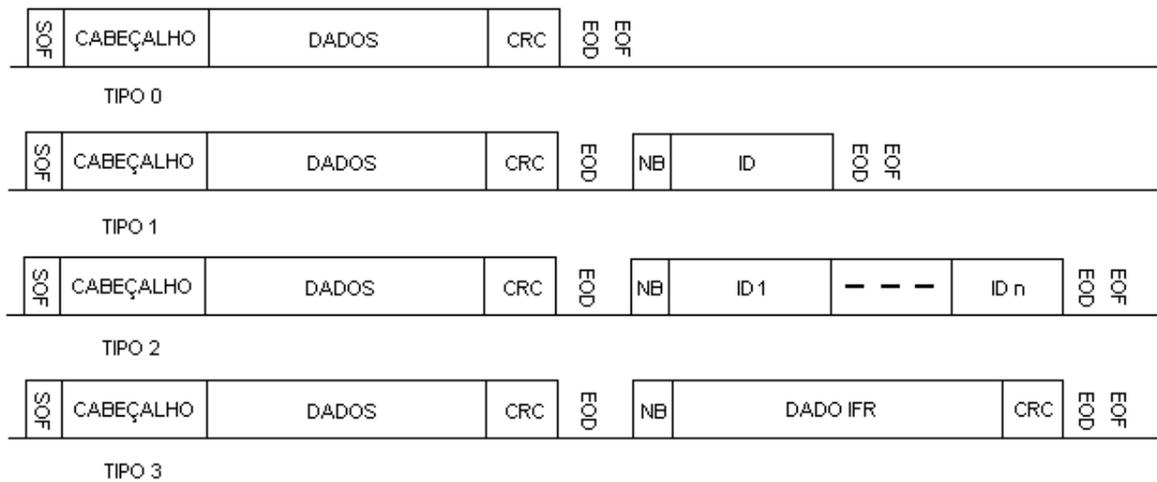


Figura 19 – Tipos de *in-frame response*

4.3.2.4 Diagrama de Blocos

Os próximos itens desta Seção tratam da divisão interna do protocolo: Interface com CPU, Gerenciador de protocolo, *Interface MUX*.

Interface com CPU

Esse bloco tem a função básica de fazer a interface do módulo BDLC com a CPU. Ele é composto de cinco registradores que podem ser visualizados na Figura 20.

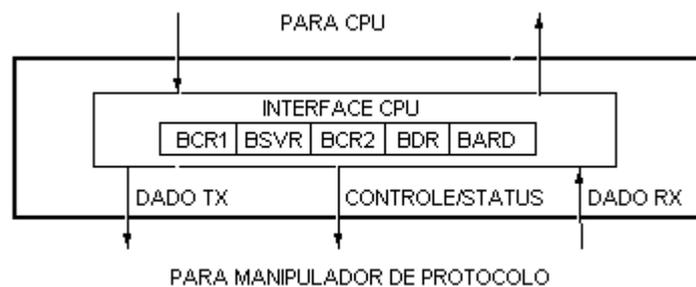


Figura 20 – Interface com CPU

- BCR1 (*Control Register 1*) configura e controla o BDLC. Suas funções incluem seleção do clock, habilitação de interrupções e indicação de mensagens que devem ser ignoradas.
- BSVR (*State Vector Register*) indica o estado atual de operação do BDLC.
- BCR2 (*Control Register 2*) controla parâmetros de transmissão. Quando a CPU envia ou recebe um dado, esse dado é passado byte a byte para essa interface sendo armazenado temporariamente no registrador BDR (*Data Register*). Bytes que serão transmitidos pela CPU devem ser primeiramente escritos no BDR para chegarem ao barramento. Bytes recebidos do barramento serão lidos pela CPU também por esse registrador.

- BARD (*Analog Roundtrip Delay Register*) Configura o BDLC para os diferentes tipos de *transceivers*.

Gerenciador de Protocolo

A Figura 21 ilustra a estrutura interna do gerenciador de protocolo.

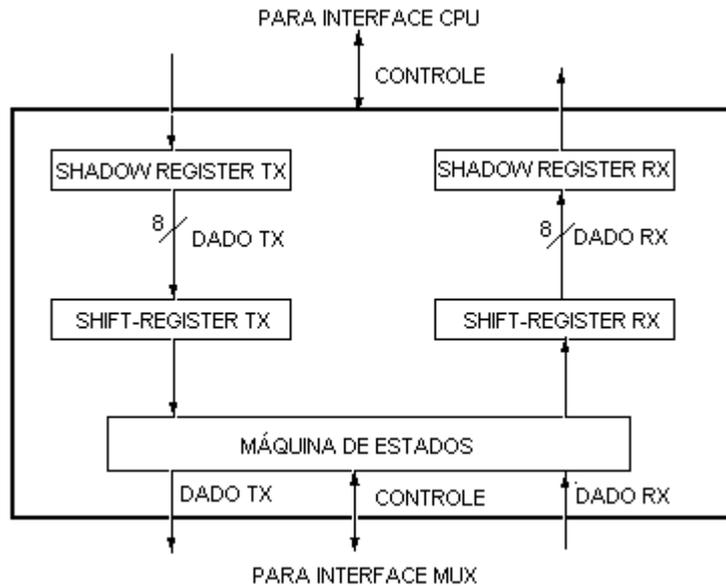


Figura 21 – Gerenciador de protocolo

- A máquina de estados tem a função de controlar todas as operações do protocolo. Suas funções são controle de acesso ao barramento, formação do pacote, detecção de colisão, arbitração, geração e verificação de CRC e detecção de erros.
- O *Shift-Register RX* tem a função de receber serialmente os dados do barramento e enviar para o *Shadow-Register RX* paralelamente.
- O *Shift-Register TX* tem a função de receber paralelamente os dados do *Shadow-Register TX* e enviá-los serialmente à máquina de estados.
- O *Shadow-Register RX* é um *buffer* entre o *Shift-Register RX* e o BDR. Analogamente, o *Shadow-Register TX* é um *buffer* entre o BDR e o barramento.

Interface MUX

A Figura 22 ilustra a estrutura interna do MUX interface.

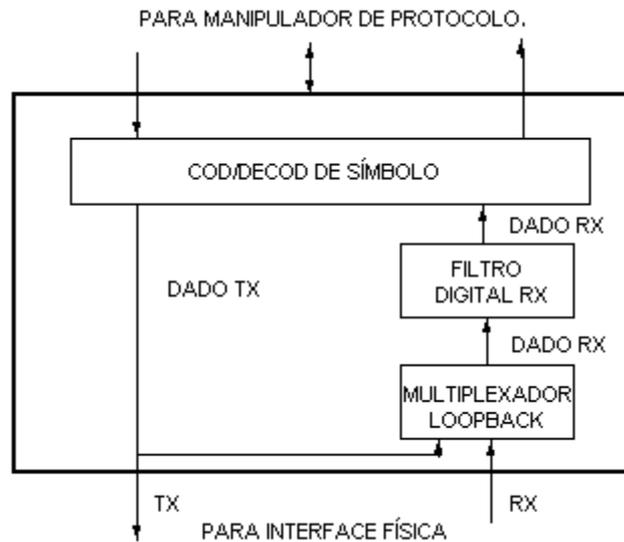


Figura 22 – Interface multiplexador

- O codificador e decodificador de símbolo tem a função, em modo de transmissão, de receber dados serialmente do gerenciador de protocolo e codificá-los para o barramento. Analogamente o decodificador recebe os dados do barramento, decodifica-os e envia-os ao gerenciador de protocolo.
- O filtro de ruído tem a função de verificar se o sinal de entrada é um ruído. Essa verificação é baseada no tempo de duração do sinal. Sinais curtos serão descartados, pois o circuitos interpreta-os como um ruído.
- O Multiplexador Loopback tem a função de isolar o módulo BDLC da interface física se o sinal de transmissão estiver conectado ao bloco de recebimento.

4.3.2.5 Método de Acesso ao Barramento

O algoritmo de acesso ao barramento baseia-se em CSMA/CD+AMP que é um método não destrutivo que permite que mensagens com prioridade maior sejam transmitidas. Os nodos que perderam a disputa pelo meio físico simplesmente ficam em estado de leitura aguardando o meio ficar liberado novamente para começarem a transmitir novamente.

O tratamento de colisão ocorre da seguinte forma: o nodo envia um bit para o barramento, lendo o estado do barramento após o envio. Se o valor lido foi diferente do valor enviado, é sinal que esse nodo perdeu o acesso ao barramento para um nodo com mensagem de prioridade maior. O nodo que perdeu o acesso entrará em estado leitura do barramento até que o barramento seja novamente liberado. O nodo que ganhou a disputa pelo barramento continuará a enviar a sua mensagem.

4.3.2.6 Conclusão

Apesar de possuir documentação completa, gratuita, incluir fluxogramas e ser simples de implementar, o que mais nos desmotivou a usar esse protocolo é sua baixa taxa de transferência de dados. Com uma taxa de 10,4 Kbps ficaria impossível mantermos o compromisso de desenvolver um protocolo versátil para domótica. Aplicações como transferência de imagem seriam prejudicadas por essa baixa taxa.

4.3.3 **Local Operating Network - LON**

LON [13] [8] é um protocolo específico para automação predial, desenvolvido por Echelon Corporation⁷. Existem famílias de dispositivos Motorola e Toshiba que dão suporte a esse protocolo.

LON especifica as camadas física, de enlace e de aplicação do modelo OSI. A comunicação entre as camadas é baseada no princípio de troca de variáveis, onde, por exemplo, na camada de aplicação o usuário faz uso do protocolo somente lendo e escrevendo variáveis.

Uma rede LON baseia-se numa estrutura hierárquica, onde existem nodos, sub-redes e domínios. Há no máximo 127 nodos por sub-rede e 255 sub-redes por domínio. Isso totaliza 32385 nodos na rede.

O método de acesso ao meio é CSMA. O barramento é controlado por um único mestre baseado em *polling*. Por ser baseado em *polling* com um mestre, aplicações baseadas em eventos e que exigem resposta em tempo real não são suportadas.

A taxa de transferência suportada é de 78 Kbps com barramento de 1300m e 1.25 Mbps a 300m. Há suporte a vários meios físicos, entre eles par trançado, cabo coaxial, linha de energia, rádio-frequência e infra vermelho.

Como vantagens desse protocolo destacamos a taxa de transferência, a estrutura hierárquica e o uso de linha de energia como meio físico. Porém sua desvantagem é o algoritmo simples de acesso ao barramento e o fato de ter mestre único. Para completar as desvantagens, citamos o fato de ser um protocolo proprietário da Echelon e ter pouca documentação especificando o protocolo.

⁷ Local Operating Network (<http://www.lonworks.echelon.com/>)

4.3.4 European Installation Bus - EIB

EIB [13] [8] foi desenvolvido pela Siemens nos anos oitenta visando mercado europeu de automação predial. Atualmente é suportado pela EIBA⁸. O processador Motorola 68HC05 provê interface para esse protocolo.

O protocolo EIB é baseado na técnica de acesso ao meio CSMA/CA. Esta técnica permite que qualquer nodo da rede seja mestre, ou seja, a qualquer momento qualquer nodo pode tentar acessar o barramento quando esse estiver livre. O tratamento de colisão ocorre da mesma forma que o protocolo BDLC.

EIB suporta redes de estrutura hierárquica, consistindo de nodos, sub-redes, áreas e sistema geral. Cada sub-rede pode ter até 256 nodos, cada área pode ter até 15 sub-redes e o sistema pode ter até 15 áreas. Isso totaliza $(255 \times 16) \times 15 + 255 = 61455$ nodos por sistema. A topologia lógica EIB pode ser visualizada na Figura 23.

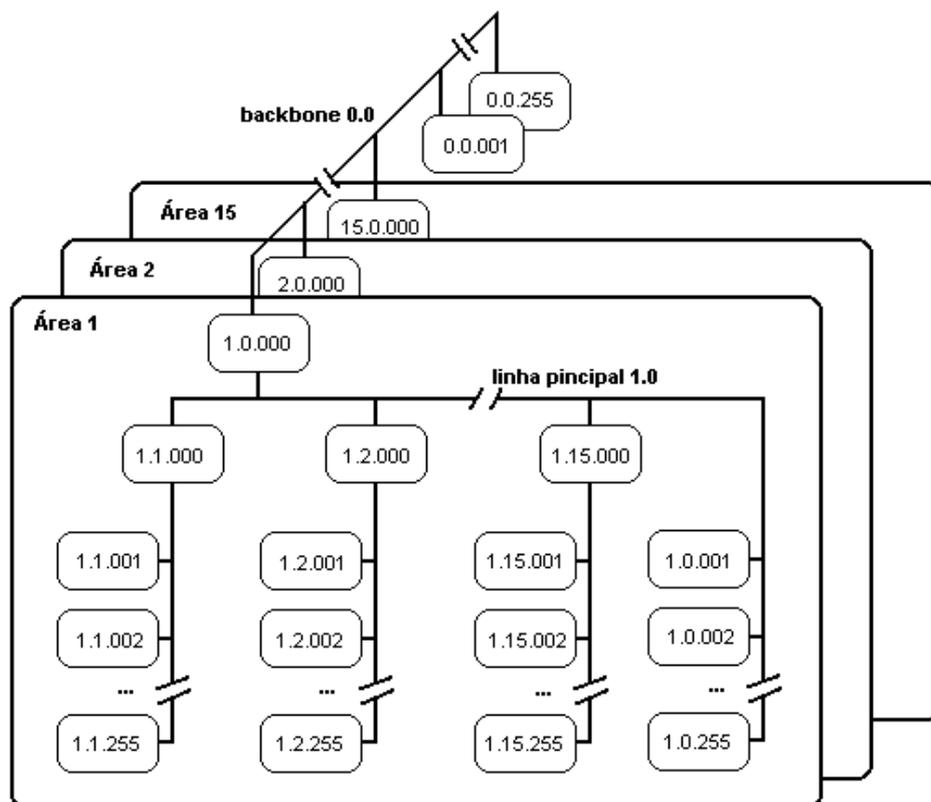


Figura 23 – Topologia lógica EIB

O pacote EIB é mostrado na Figura 24. Pode-se acrescentar o fato que o campo de endereço possui tanto o endereço do(s) destinatário(s) quanto do remetente do pacote.

⁸ European Installation Bus Association (<http://www.eiba.com/>)

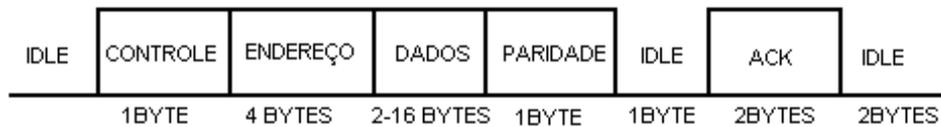


Figura 24 – Pacote EIB

Os meios físicos de comunicação disponíveis são:

- EIB.TP – Par Trançado. A taxa de transferência é de 9600 Bit/s. O comprimento máximo por sub-rede é de 1000m.
- EIB.PL – Linha de Energia. Usa modulação SFSK (Spread Frequency Shift Keying). A distância máxima entre dois dispositivos, sem necessidade do uso de repetidores, é de 600m devido ao alto nível de ruído que esse meio possui.
- EIB.RF – Rádio Freqüência. Sem retansmissores tem-se um alcance de 300m.

O ponto forte desse protocolo e o método de acesso ao meio, que possibilita uso em aplicações críticas em tempo real. Porém o que nos levou a rejeitar esse protocolo foi a falta de literatura detalhada que especificasse o protocolo, o fato de ser um protocolo mais restrito à Europa e a taxa de transferência de 9600bps. Também descobrimos que equipamentos que utilizam esse protocolo são em torno de 10 a 100 vezes mais caros que equipamentos similares.

4.3.5 X-10

O Sistema X-10⁹ é um protocolo de controle baseado em linha elétrica residencial. O sistema consiste basicamente de um controlador (transmissor) e um interruptor remoto (receptor). Os controladores X-10 enviam sinais digitalmente codificados aos módulos de recepção através da rede elétrica de 120VAC já existente.

O interruptor remoto pode ser conectado a qualquer saída de tensão. Um eletrodoméstico é conectado a um receptor, que continuamente monitora a linha elétrica por sinais codificados. Os sinais são recebidos e interpretados por um circuito eletrônico embutido no receptor.

Um comando é transmitido em onze ciclos de linha AC. Um comando válido deve começar com o “*Start Code*” 1110, e demora dois ciclos, meio para cada bit, para ser enviado. Os próximos quatro ciclos são para enviar os quatro bits do campo “*House Code*” que, por causa do complemento explicado abaixo, deve ser duplicado. Os últimos cinco bits é o “*key code*”. A Figura 25 mostra o formato do pacote x-10.

⁹ X-10 Home Page (<http://www.x-10.com/>)

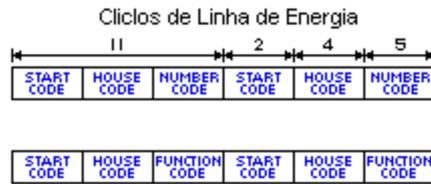


Figura 25 – Pacote do protocolo X-10

O controlador envia um série de sinais para o receptor, de acordo com a Figura 26.

| HOUSE CODES | | | | KEY CODES | | | | | | |
|-------------|----|----|------------------------|-----------|----|----|----|----|-----|---|
| | H1 | H2 | H4 | H8 | D1 | D2 | D4 | D8 | D16 | |
| A | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| B | 1 | 1 | 1 | 0 | 2 | 1 | 1 | 1 | 0 | 0 |
| C | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 0 |
| D | 1 | 0 | 1 | 0 | 4 | 1 | 0 | 1 | 0 | 0 |
| E | 0 | 0 | 0 | 1 | 5 | 0 | 0 | 0 | 1 | 0 |
| F | 1 | 0 | 0 | 1 | 6 | 1 | 0 | 0 | 1 | 0 |
| G | 0 | 1 | 0 | 1 | 7 | 0 | 1 | 0 | 1 | 0 |
| H | 1 | 1 | 0 | 1 | 8 | 1 | 1 | 0 | 1 | 0 |
| I | 0 | 1 | 1 | 1 | 9 | 0 | 1 | 1 | 1 | 0 |
| J | 1 | 1 | 1 | 1 | 10 | 1 | 1 | 1 | 1 | 0 |
| K | 0 | 0 | 1 | 1 | 11 | 0 | 0 | 1 | 1 | 0 |
| L | 1 | 0 | 1 | 1 | 12 | 1 | 0 | 1 | 1 | 0 |
| M | 0 | 0 | 0 | 0 | 13 | 0 | 0 | 0 | 0 | 0 |
| N | 1 | 0 | 0 | 0 | 14 | 1 | 0 | 0 | 0 | 0 |
| O | 0 | 1 | 0 | 0 | 15 | 0 | 1 | 0 | 0 | 0 |
| P | 1 | 1 | 0 | 0 | 16 | 1 | 1 | 0 | 0 | 0 |
| | | | All Units Off | | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | All Lights On | | 0 | 0 | 0 | 1 | 1 | 1 |
| | | | On | | 0 | 0 | 1 | 0 | 1 | 1 |
| | | | Off | | 0 | 0 | 1 | 1 | 1 | 1 |
| | | | Dim | | 0 | 1 | 0 | 0 | 1 | 1 |
| | | | Bright | | 0 | 1 | 0 | 1 | 1 | 1 |
| | | | All Lights Off | | 0 | 1 | 1 | 0 | 1 | 1 |
| | | | Extended Code | | 0 | 1 | 1 | 1 | 1 | 1 |
| | | | Hail Request | | 1 | 0 | 0 | 0 | 1 | 1 |
| | | | Hail Acknowledge | | 1 | 0 | 0 | 1 | 1 | 1 |
| | | | Pre-Set Dim | | 1 | 0 | 1 | X | 1 | 1 |
| | | | Extended Data (analog) | | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | Status-on | | 1 | 1 | 0 | 1 | 1 | 1 |
| | | | Status-off | | 1 | 1 | 1 | 0 | 1 | 1 |
| | | | Status Request | | 1 | 1 | 1 | 1 | 1 | 1 |

Figura 26 – Tabela de comando do protocolo X-10

A vantagem de se usar linha elétrica da casa como meio de transmissão é que não se necessita cabeamento extra para instalação do sistema. Isso reduz o custo de sua implementação. Porém, linha de energia não é um meio de comunicação ideal devido ao ruído, flutuações e interferências que este meio possui. Para conter essas interferências são necessárias técnicas adicionais para verificar erros nos pacotes. X-10 usa o método “1 e complemento” que consiste em enviar, por exemplo, um bit 1 e logo após o seu complemento 0. Isso traz uma duplicação de dados enviados e uma diminuição do desempenho do sistema, porém tem a vantagem de ser de fácil implementação. A Figura 27 ilustra essa técnica.



Figura 27 – Método de detecção de erros do protocolo X-10

As grandes vantagens do protocolo X-10 são sua simplicidade e o uso linha elétrica como meio físico. Isso leva a uma diminuição de custo de implementação e facilidade de manutenção e instalação, o que muitas vezes pode ser feito pelo próprio consumidor. Essas vantagens foram marcantes para X-10 ser o protocolo específico para automação doméstica mais famoso e usado. Porém, como neste trabalho temos que manter um compromisso com a oportunidade de aprendizado prático, decidimos não optar pelo uso deste protocolo justamente por ser simples ao ponto de dificultar a aplicação deste protocolos para aplicações mais exigentes a nível de protocolo como entretenimento, onde uma alta taxa de transmissão é exigida.

4.3.6 Consumer Electronics Bus - CEBus

CEBus¹⁰ foi desenvolvido visando o mercado norte-americano. Este protocolo apresenta as seguintes características:

- Arquitetura aberta;
- Expansível;
- Comunicação e Controle Distribuído. Não necessita de um controlador centralizado;
- Plug and Play.

CEBus possui os seguintes meios de comunicação:

- PLBus – Linha elétrica;
- TPBus – Par trançado. Normalmente usado para aparelhos de baixa potência;
- IRBus – Infra-vermelho a 10kbps/s com frequência portadora de 70 a 80 KHz;
- RFBus – Rádio frequência que opera a 902MHz;
- CXBus – Cabo coaxial. Normalmente usado em circuito de TV.

CeBus pareceu, segundo as informações que dispúnhamos, ser um protocolo muito bem especificado, visando o crescimento futuro que o ramo de automação doméstica está tendo. Sua flexibilidade e facilidade de instalação são o seu diferencial. Porém a escassez de material técnico gratuito impossibilitou maiores análises do protocolo. A especificação do padrão¹¹ é vendida à U\$42,00.

¹⁰ Consumer Electronics Bus (<http://www.cebus.org/>)

¹¹ CEBus Standard (EIA-600)

4.4 Comparação entre os protocolos estudados

Nesta Seção montamos uma tabela comparativa (Tabela 2) que resume as características levantados nas seções anteriores.

| | LON | CeBus | EIB | BDLC | EHS | X10 |
|--|---------------------|----------------------|---------------------|-------------|------------------------|---------------------|
| Taxa de Tx | 78Kbps a 1.25Mbps | * | 9.6Kbps | 10.4Kbps | *** | * |
| Nro max. Nodos | 32385 | * | 11520 | * | *** | * |
| Método acesso ao meio | CSMA controlado | * | CSMA/CA | CSMA/CD | *** | * |
| Alcance | 1300m | * | 1000m | * | *** | * |
| Aberto | Não | Sim | * | * | Sim | Não |
| Aplicação Típica | Automação Doméstica | Automação Doméstica | Automação Doméstica | Automotiva | Automação Doméstica | Automação Doméstica |
| Topologia | Hierárquico | * | Hierárquico | Barramento | Barramento em níveis | * |
| Métodos de verificacao de erros | * | * | <i>Checksum</i> | CRC | CRC e FEC para PL | Compleme nto |
| Flexibilidade dos pacotes | * | Muito boa | Boa | Boa | Muito boa | Ruim |
| Complexidade de implementação | * | * | * | pequena | alta | pequena |
| Plug and Play | Não | Sim | Não | Não | Sim | Sim |
| Padrão | Não tem | EIA-600 | Não tem | SAE J1850 | Encaminha do | * |
| Meios físicos suportados ** | PT,CX,PL, RF e IR | PT,CX,PL, RF,IR e FO | PT,PL,RF | Serial | PT1,PT2, PL,RF,CX e IR | PL |

Tabela 2 – Tabela comparativa entre protocolos

* Documentação encontrada não foi suficiente para julgarmos este item.

** Legenda dos meios físicos suportados pode ser visualizada na Tabela 3.

*** Ver Tabela 1 na Seção sobre EHS.

| Sigla | Legenda |
|--------------|--------------------------------------|
| PT | Par trançado |
| CX | Cabo coaxial |
| PL | <i>Power line</i> (linha de energia) |
| RF | Rádio frequência |
| IR | Infra vermelho |
| FO | Fibra ótica |

Tabela 3 – Legenda da tabela comparativa de protocolos

5 Controller Area Network - CAN

Controller Area Network é um protocolo serial que suporta controle em tempo real distribuído, com um grande nível de segurança [10] [13] [12] [14] [15]. Foi originalmente desenvolvido para aplicação automotiva, mas hoje em dia já existem muitas outras aplicações que serão citadas na Seção 5.1.

Existe muito material sobre CAN disponível gratuitamente na Internet. Entre os principais citamos os *links* [10] [11] [12].

Este capítulo não tem a função de ser um manual sobre CAN, e sim de dar ao leitor uma visão das características principais do protocolo que será a base desse trabalho.

Na Seção 5.1 citamos algumas aplicações do protocolo CAN. Na Seção 5.2 apresentamos conceitos básicos sobre CAN. A Seção 5.3 apresenta em detalhes a especificação do protocolo CAN, bem com suas características. A Seção 5.4 apresenta algumas variações de implementações existentes. Na Seção 5.5 é feito o estudo a nível de **diagrama de blocos da arquitetura que será implementada no Trabalho de Conclusão 2**.

5.1 Aplicações de CAN

Uma das principais preocupações que tivemos para escolher o protocolo a ser implementado é a flexibilidade a nível de aplicações que ele pode ter. Por esse motivo, escolhemos um protocolo que não restrinja as aplicações somente ao âmbito deste trabalho.

Entre as principais aplicações citamos:

- veículos (marítimo, aéreo, terrestre) – carros de passeio, *off-road*, trens, sistema de semáforo (trens e carros), eletrônica marítima, máquinas agrícolas, helicópteros, transporte público;
- sistema de controle industrial – controle de planta, de maquinário, de robôs, sistema de supervisão;
- automação predial – controle de elevadores, de ar condicionado, de iluminação;
- aplicações específicas – sistemas médicos, telescópios, simuladores de vôo, satélites artificiais, entre outros;

Neste trabalho propomos uma nova aplicação a CAN. Essa aplicação é um sistema de supervisão de domótica via *web* usando o protocolo CAN como rede de controle da casa.

5.2 Conceitos Básicos

Nesta Seção alguns conceitos básicos são explicados para facilitar o entendimento das seções seguintes.

CAN possui as seguintes **características**:

- priorização de mensagens;
- garantia do tempo de latência (tempo real);
- flexibilidade de configuração;
- recepção *multicast* com sincronização;
- várias técnicas para manter consistência de dados;
- multi-mestre;
- técnicas de detecção e sinalização de erros;
- desligamento automático de nodos com defeitos;

CAN é dividido em diferentes **camadas** de acordo com o modelo OSI /ISO:

- Camada de enlace:
 - sub-camada LLC (*Logical Link Control*);
 - sub-camada MAC (*Medium Access Control*);
- Camada física;

As funções de cada camada podem ser visualizadas na Figura 28, sendo explicadas na Seção 5.3.

| Camada de Enlace |
|---|
| LLC - Serviços de transferência de dados e requisição remota; - Filtro de aceitação; - Notificação de overload; - Recuperação. |
| MAC - En/De-Capsulamento de dados; - De-/Codificação de pacotes; - Bit stuffing; - Gerenciamento de acesso ao meio; - Detecção de erros; - Sinalização de erros; - Reconhecimento; - Serialização. |
| Camada Física |
| - En-/Codificação de bit; - Temporização de bit; - Sincronismo. |

Figura 28 – Funções das camadas de CAN

O barramento CAN possui dois valores. **dominante**, equivalente ao nível lógico 0, e o **recessivo**, equivalente ao nível lógico 1. Na ocorrência de duas escritas simultâneas no barramento, uma escrita dominante (0) e outra recessiva (1), o barramento estará com valor dominante, pois, como o próprio nome diz, esse tem prioridade em relação ao recessivo.

Existem duas especificações sobre o protocolo CAN. A primeira é chamada *Standard CAN* e a segunda *Extended CAN*. A Tabela 4 faz um comparativo entre as duas especificações [12].

| | Standard CAN | Extended CAN |
|--|---------------------|---------------------|
| Nome da Especificação | CAN 2.0 A | CAN 2.0 B |
| Número de bits do campo de identificação | 11 | 29 |

Tabela 4 – Comparativo entre CAN padrão e estendido

5.3 Especificação CAN

As seções subsequentes têm como objetivo detalhar a especificação padrão do protocolo CAN [12] [13].

Na Seção 5.3.1 apresentamos o formato e função dos pacotes CAN. A Seção 5.3.2 descreve as técnicas de detecção de erros existentes no protocolo CAN. A Seção 5.3.3 explica o algoritmo de arbitragem usado na disputa pelo barramento. Nas Seções 5.3.4 e 5.3.5 é explicada a técnica de

ajuste de ponto de amostragem de sinal. Finalmente, na Seção 5.3.6 são apresentadas algumas características referentes ao nível físico de CAN.

5.3.1 Formato dos Pacotes

Cada mensagem CAN consiste em uma seqüência de bits que é dividida em campos. Cada campo tem uma função diferente como descrita a seguir.

CAN possui quatro tipos de mensagens: dados, requisição remota, erro e *overload*. O formato e função de cada pacote é explicado nas seções seguintes.

5.3.1.1 Pacote de Dados

A Figura 29 ilustra o formato do pacote de dados CAN.

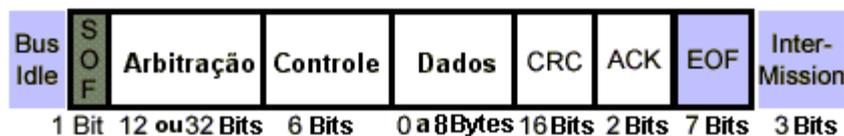


Figura 29 – Formato de um pacote de dados CAN

- SOF – 1 bit - dominante – Marca o início de uma mensagem. Quando o barramento se encontra ocioso¹², uma borda de descida do SOF sincroniza os nodos da rede (*hard synchronization*).
- Arbitração e Controle – Esses campos são sub-divididos de acordo com a Figura 30.

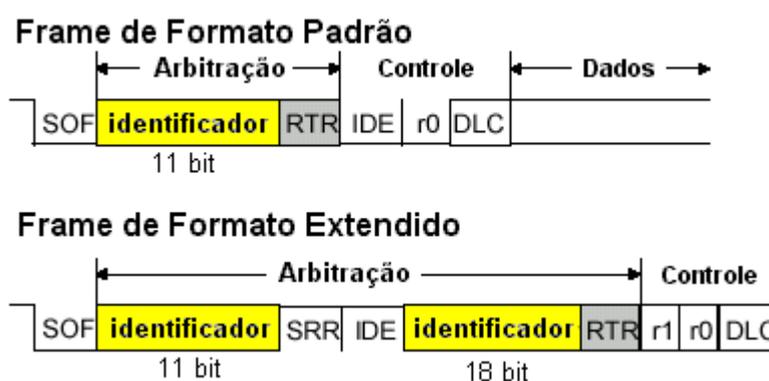


Figura 30 – Campo de arbitração

¹² *Bus Idle*

- Identificador – 11 ou 29 bits - É o endereço lógico e a prioridade da mensagem. Valores menores tem maior prioridade. O formato padrão possui 11 bits, enquanto o estendido possui 29. A Seção 5.3.3 explica a técnica de arbitração;
- RTR – 1 bit – O RTR (*Remote Transmission Request*) identifica se a mensagem é de dados (dominante) ou de requisição de dados (recessivo);
- IDE – 1 bit – O IDE (*Identifier Extension*) identifica se a mensagem é do formato padrão (dominante) ou estendido (recessivo);
- SRR – 1 bit – recessivo - (*Substitute Remote Request*) ;
- r0 e r1 – 2 bits – dominantes – São bits de reserva;
- DLC – 4 bits – O DLC (*Data Length Code*) informa o número de bytes que será transmitido. Originalmente CAN suporta o envio de até 8 bytes de dados em um único pacote. Porém, em aplicações específicas, pode-se fazer uso de até 16 bytes de dados por pacote. A Tabela 5 mostra a codificação do campo DLC.

| Nro Bytes | DLC3 | DLC2 | DLC1 | DLC0 |
|-----------|------|------|------|------|
| 0 | d | d | d | d |
| 1 | d | d | d | r |
| 2 | d | d | r | d |
| 3 | d | d | r | r |
| 4 | d | r | d | d |
| 5 | d | r | d | r |
| 6 | d | r | r | d |
| 7 | d | r | r | r |
| 8 | r | x | x | x |

Tabela 5 – Codificação do campo DLC

- Dados – 0 a 64 bits – Contém o dado da mensagem;
- CRC – 16 bits – CRC contém o *checksum* dos bits precedentes da mensagem. Esse *checksum* é usado para fazer detecção de erros. Possui distância de hamming de 6, o que significa que pode identificar até 5 falhas no pacote. Em caso de erro no pacote, esse será descartado e o receptor transmite um pacote de erro para sinalizar o erro e requisitar retransmissão. O polinômio gerador é $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ formado por 15 bits. O último bit é o delimitador de CRC que é fixo em recessivo;
- ACK – 2 bits – É composto pelo bit ACK Slot e pelo bit Delimitador de ACK. Transmissores enviam ambos os bits em recessivo. Um receptor indica que recebeu a

mensagem enviando um bit dominante no ACK Slot. Isso indica ao transmissor que ao menos um nodo recebeu a mensagem corretamente;

- EOF – 7 bits – todos recessivos - EOF (*End Of Frame*) delimita o fim de uma mensagem. *Bit stuffing* é desativado enquanto EOF está ativo;
- IDLE – 0 ou mais bits – recessivos – Sinaliza que o barramento está livre. Qualquer nodo pode começar uma transferência de mensagem;
- *Intermission* ou IFS (*InterFrame Space*) – 3 bits – recessivos – IFS é o tempo necessário para que um nodo transfira um pacote corretamente recebido do barramento para área de armazenamento local (*mailbox*). É o tempo mínimo entre a transmissão de dois pacotes (tempo interno para nodo processar o pacote).

5.3.1.2 Pacote de Requisição de Dados

É possível que um nodo de destino requisiite dados da origem. Para isso o nodo de destino envia uma requisição de dados (RFR¹³) com o identificador que pertence ao dado requerido. O nodo que tem este dado devolverá um pacote de dados, respondendo à requisição.

A Figura 31 ilustra o funcionamento de RFR. O nodo 1 envia uma requisição de dados com o identificador do mesmo, indicado pela marca (1) na figura. O nodo que responde por esse dado (identificador) é o nodo 2. O nodo 2, então, envia o dado requisitado (marca (2)) ao barramento e os nodos 1 e 4 lêem este dado (marca (2)).

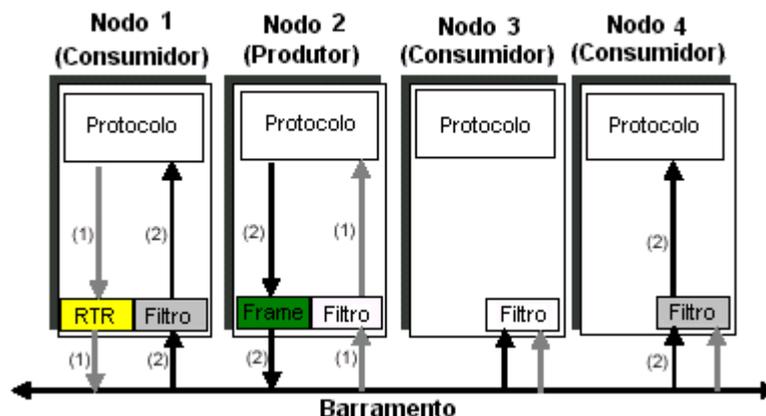


Figura 31 – Requisição remota de dados

Existem duas diferenças entre pacotes de dados e pacotes RFR. Primeiramente o bit RTR é transmitido dominante em pacotes de dados e recessivo em RFR. A outra diferença é que não

¹³ Remote Frame Request

existe campo de dados em RFR. Quando um pacote de dados e um RFR com mesmo identificador são transmitidos no mesmo instante, o pacote de dados ganha a disputa devido ao bit RTR dominante. Assim, o nodo que requisitou o dado recebe-o imediatamente. A Figura 32 ilustra o formato deste pacote.

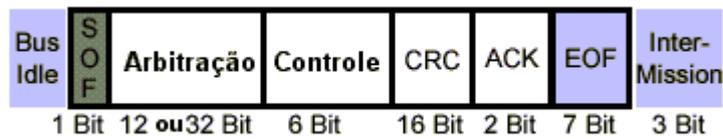


Figura 32 – Pacote de requisição de dados CAN

5.3.1.3 Pacote de Erro

Um pacote de erro é gerado por qualquer nodo que detecte um erro. Sua função, por tanto, é notificar a ocorrência de falhas. O pacote de erro é formado por dois campos: flag de erro e delimitador de erro. A Figura 33 mostra o formato do pacote de erro.

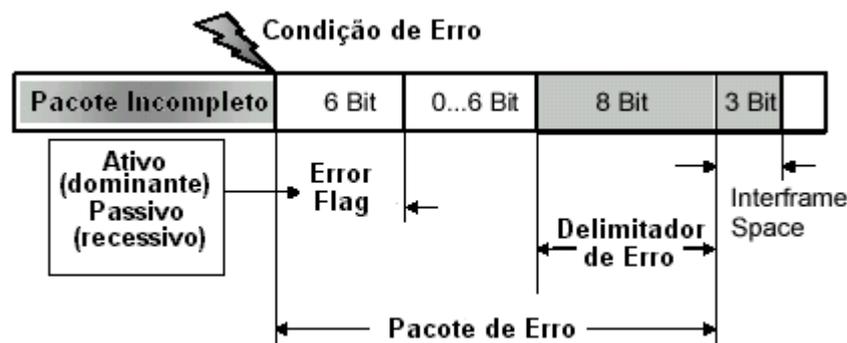


Figura 33 – Pacote de erro CAN

- *Flag* de Erro – 6 bits – É o campo que sinaliza a existência de um erro. Existem dois tipos de *flag* de erros: *flag* erro ativo e *flag* erro passivo. Um nodo em estado de erro ativo envia um *flag* de erro ativo (dominante), enquanto um nodo em estado de erro passivo envia um *flag* de erro passivo (recessivo). A diferença entre os estados é melhor explicada na Seção 5.3.2.6.

O campo de *flag* de erro é enviado pelo nodo que detectou o erro (receptor 2 na Figura 34) e, se esse nodo estiver em estado ativo (dominante), sobrescreverá o dado corrompido. Quando os outros nodos da rede recebem a seqüência de 6 bits dominantes referente ao *flag* de erro, irá ocorrer uma violação de *bit stuffing* e todos os nodos (receptor 1) enviarão ao mesmo tempo um outro *flag* de erro. Os próximos 6 bits são esse segundo *flag* de erro que é enviado, em dominante, pelos outros nodos da rede.



Figura 34 – Processo de sinalização de erros

Se o nodo que identificou o erro está em modo erro passivo, ele enviará um *flag* de erro recessivo e mais o delimitador, também em recessivo. Portanto erros sinalizados por nodos em estado de erro passivo não afetarão o barramento, isolando assim nodos que tenham uma taxa de erros grandes.

- Delimitador de Erro – 8 bits (recessivos) - É transmitido pelo nodo que enviou o dado que continha erro. Sua função é recomeçar comunicações no barramento após uma falha.

5.3.1.4 Pacote de *Overload*

É usado, basicamente, para sinalizar a um transmissor que o receptor não está pronto para receber as próximas mensagens, portanto o transmissor deve esperar para fazer a próxima transmissão.

O formato do pacote é idêntico ao pacote de erro com a diferença que *Overload Flag* é sempre dominante e que o envio de um pacote de *overload* ocorre sempre após os 3 bits de IFS, ou seja, depois do fim de um mensagem.

5.3.2 Técnicas de Verificação e Sinalização de Falhas

Esta Seção apresenta as técnicas de detecção de erros implementadas no protocolo CAN.

5.3.2.1 Bit Stuffing

Nos pacotes de dados e de RFR é usada a técnica de *bit stuffing* para assegurar a não existência de um número maior que 5 bits de mesmo nível lógico consecutivo. Isso é usado para garantir a sincronização de todos os nodos da rede e também aumenta a imunidade a ruído.

Se um nodo detectar uma seqüência de 5 ou mais bits com mesmo nível lógico, será gerado um pacote de erro no próximo bit.

Nos pacotes de dados e de RFR os bits de delimitação de CRC, ACK Slot e EOF não são computados pela técnica. Os pacotes de erro e de *overload*, por serem fixos, não são codificados com *bit stuffing*.

A Figura 35 apresenta a forma de codificação e decodificação da técnica *bit stuffing*.

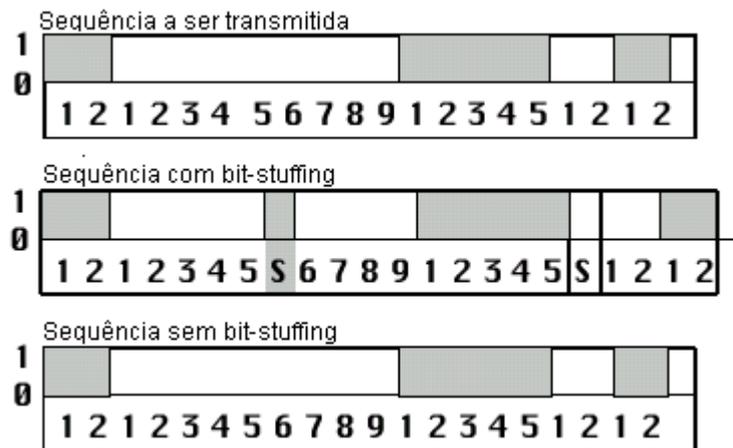


Figura 35 – *Bit Stuffing*

5.3.2.2 Bit Error

Um nodo que envia um bit ao barramento também monitora o barramento. Um *bit error* ocorre quando o valor monitorado é diferente do valor enviado. Para sinalizar essa falha um pacote de erro é gerado. Porém, nos campos de arbitração e *Ack Slot*, não é considerado uma falha, pois nestes campos deve ser possível que um bit dominante sobrescreva um bit recessivo.

5.3.2.3 CRC Error

Ocorre quando o CRC recebido não é idêntico ao CRC calculado. Neste caso um pacote de erro deve ser enviado logo após o delimitador de *Ack*. CRC é calculado desde o SOF até o fim do campo de dados. CRC é enviado nos pacotes de dados e de RFR.

5.3.2.4 Form Error

Ocorre quando um bit de valor fixo (*Ack Slot*, delimitador de CRC e EOF) contém um valor ilegal.

5.3.2.5 Erro de Reconhecimento

Quando o transmissor envia um pacote, ele manda o *Ack Slot* em recessivo, porém ao lê-lo, o receptor vai mudar esse bit para dominante para que o transmissor reconheça o recebimento da mensagem. Se o transmissor não receber o *Ack Slot* em dominante significa que nenhum nodo da rede leu o dado. Isso causará um envio de um pacote de erro para sinalizar essa falha no próximo bit.

5.3.2.6 Estados de Erros

Para prevenir que distúrbios permanentes possam afetar o barramento, CAN possui um algoritmo de confinamento de erros que é baseado em dois registradores que, através deles, controla-se o estado atual de um nodo. Existem três estados possíveis :

Erro Ativo – É o estado inicial dos nodos quando é inicializado. Neste estado os nodos podem enviar *flag* de erro ativo.

Erro Passivo – Quando já existem mais de 127 erros no REC¹⁴ ou no TEC¹⁵ o estado passa de erro ativo para erro passivo. Neste estado o nodo só pode enviar *flag* de erro passivo. Essa característica é importante, pois um nodo com muitos distúrbios não terá mais influência no barramento.

Para evitar que um nodo em erro passivo bloqueie o barramento com mensagens de alta prioridade é acrescentado um tempo de espera de 3+8 bits recessivos entre a transmissão de pacotes. Os 3 primeiros bits são o IFS e os 8 bits são o atraso propriamente dito.

Se o número de erros diminuir abaixo de 127, então o nodo voltará ao estado erro ativo.

Desconectado – Quando o TEC acusa mais de 255 erros o nodo será desconectado do barramento. O nodo só sairá desse estado através de um *reset*.

A Figura 36 mostra a transição dos estados de erro.

¹⁴ Contador de erros de recepção (*Receive Error Counter*)

¹⁵ Contador de erros de transmissão (*Transmit Error Counter*)

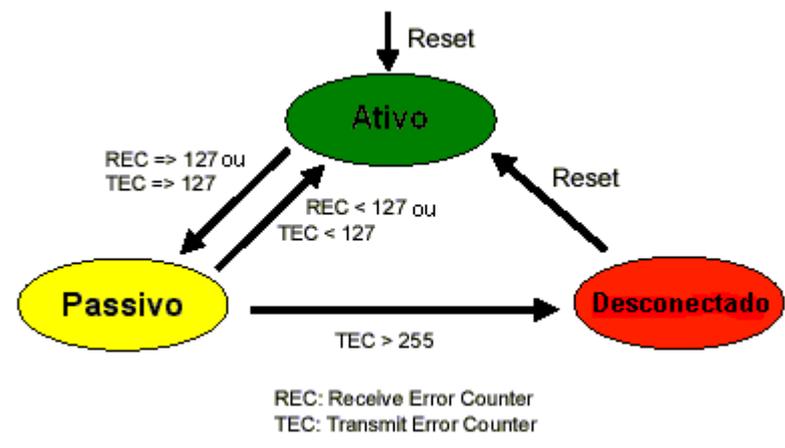


Figura 36 – Estados de erro

5.3.2.7 Análise de Detecção de Erros

A probabilidade [13] de não se detectar um falha em um nodo CAN padrão é:

$$p < 4.7 \times 10^{-11} \times \text{taxa de erro}$$

Exemplo: Suponha que ocorra um erro a cada 0.7s a uma taxa de transferência de 500Kbits/s, que a rede esta em funcionamento 8 horas por dia, 365 dias por ano. Com isso conseguimos um valor médio aproximado de **uma falha não detectada a cada 1000 anos**.

Em pacotes estendidos a probabilidade de não detectar uma falha geralmente é maior do que em pacotes padrões.

5.3.3 Arbitragem

Se dois ou mais nodos começam a transmissão no mesmo instante, a colisão das mensagens é evitada pela implementação do protocolo CSMA/CD+AMP para decidir disputas de acesso ao barramento. O campo de identificação dos pacotes conterá não somente a identificação do dado, mas também a prioridade que essa mensagem terá. Identificadores de menor valor terão maior prioridade.

Enquanto os nodos estiverem enviando os mesmos bits de identificação, nada acontece. No entanto, quando um nodo enviar um bit de identificação de prioridade menor, esse perderá a disputa pelo barramento.

A Figura 37 mostra como ocorre a arbitragem. Até o bit 6 todos os nodos enviaram o mesmo valor. Porém, no bit 5 o nodo 2 transmitiu um bit recessivo, perdendo assim a disputa para o nodo 3 e 1. No bit 2 o nodo 1 enviou um bit recessivo e entrou em modo de escuta. Desta forma, o nodo 3 ganhou a disputa pelo barramento e seguiu transmitindo o pacote.

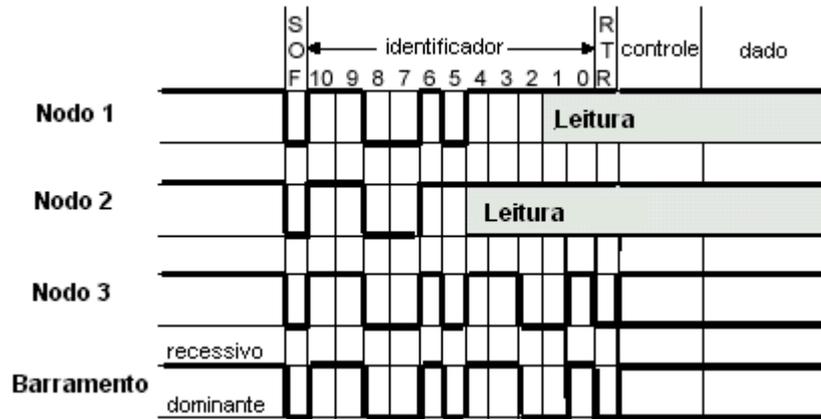


Figura 37 – Arbitragem CAN

5.3.4 Bit-Timing

Uma rede CAN consiste de vários nodos, cada um com o seu *clock* individual. Por esse motivo podem ocorrer perdas de fase nos diferentes nodos. Os nodos CAN possuem um algoritmo de sincronização que **compensa perdas de fase enquanto estão recebendo um pacote**.

Um *Bit Time* é o período de transmissão e recepção de um bit. Todos os nodos da rede devem ter o mesmo *bit time*. *Bit time* é formado por quatro segmentos: *sync_seg*, *prop_seg*, *phase_seg1* e *phase_seg2*. Cada segmento é formado por múltiplos *Time Quanta* (tq). O *Time Quantum* é uma unidade de tempo fixa (menor porção de tempo usada no nodo CAN) derivada do *clock* do sistema.

- SYNC_SEG – 1 tq – É usado para sincronizar os vários nodos. Espera-se que a borda de descida em uma recepção ocorra neste segmento (caso ideal);
- PROG_SEG – 1..8 tq – É um campo de largura variável usado para compensar o atraso de sinais através da rede;
- PHASE_SEG1 – 1..8 tq – É usado para compensar o erro de fase e pode ter seu comprimento aumentado durante a resincronização;
- PHASE_SEG2 – 1..8 tq – É usado para compensar o erro de fase e pode ter seu comprimento diminuído durante a resincronização;
- SAMPLE POINT ¹⁶ – É o instante onde o nível do barramento deve ser lido para ser interpretado como valor do respectivo bit.

¹⁶ Ponto de Amostragem

A Figura 38 mostra o formato de um Bit Time.



Figura 38 – *Bit Time CAN*

Também podemos observar na Figura 38 que a borda de descida do sinal de entrada ocorre no segmento *sync_seg* do *bit time*. Está a situação ideal de comunicação, pois não existe diferença de fase entre o transmissor e o receptor.

5.3.5 Sincronismo

Duas técnicas de sincronização são suportadas: *Hard Synchronisation* e *Soft Synchronisation*.

- *Hard Synchronisation* – É acionado na borda de descida quando o barramento está livre, que é interpretado como SOF. Essa sincronização inicia a lógica interna de *bit time*.
- *Soft Synchronisation* – É usado para ajustar o *bit time* enquanto o nodo CAN está recebendo um pacote. Quando o transmissor é lento em relação ao receptor o *bit time* é aumentado. Quando o transmissor é mais rápido o *bit time* é diminuído.

A cada borda de descida inicia-se a lógica de *bit time*. Idealmente essa borda é esperada no segmento *sync_Seg*, conforme Figura 38. Porém pode acontecer, devido a alteração de fase, que a borda de descida venha a acontecer durante o segmento *phase_Seg1* ou *phase_Seg2*.

Se a borda de descida do sinal de entrada for detectada quando o *bit time* estiver no segmento *phase_Seg1* (Figura 39) interpreta-se que o **transmissor é lento** em relação ao receptor. Neste caso o segmento *phase_Seg1* é adicionado de RJW pulsos para que o sinal de amostra esteja no ponto adequado (ponto 2). Se não houvesse um sistema de sincronização, o sinal seria amostrado no ponto 1 da figura.

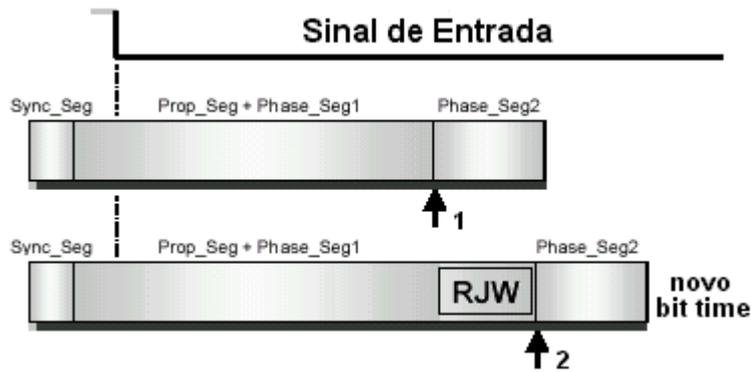


Figura 39 – Sincronização com transmissor lento

Caso contrário, se a borda de descida coincidissem com o segmento *phase_Seg2*, como ilustrado na Figura 40, poderíamos interpretar que o **transmissor é mais rápido** que o receptor. Neste caso o segmento *phase_Seg2* do bit anterior é diminuído de RJW pulsos de forma que o novo *bit time* comece mais cedo do que começaria (ponto 3). Com isso consegue-se que o ponto de amostragem esteja localizado no ponto 2 ao invés do ponto 1.

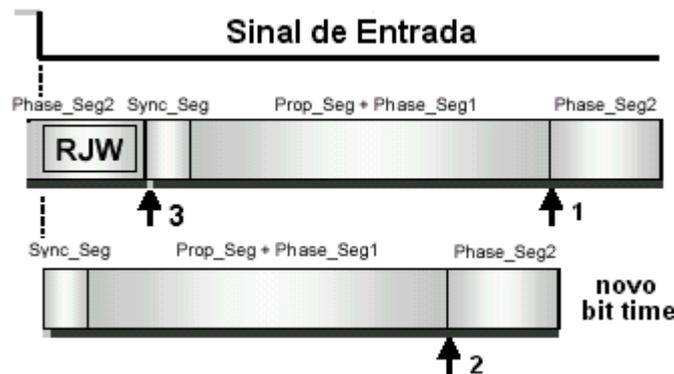


Figura 40 – Sincronização com transmissor rápido

5.3.6 Camada Física

A fiação no barramento CAN é feita a dois fios: CAN_L e CAN_H. Essa técnica é usada para minimizar efeitos de interferências eletromagnéticas no barramento. A Figura 41 ilustra a fiação do barramento.

CAN usa a técnica de sinalização por tensão diferencial. Isso quer dizer que o nível do sinal, dominante ou recessivo, é determinado pela diferença de tensão entre CAN_H e CAN_L. Em nível recessivo, CAN_H não pode ter uma tensão maior que 0.5V em relação a tensão em CAN_L. Se a tensão em CAN_H for pelo menos 0.9V maior que CAN_L, então o barramento está em nível dominante. A Figura 42 ilustra esta explicação.

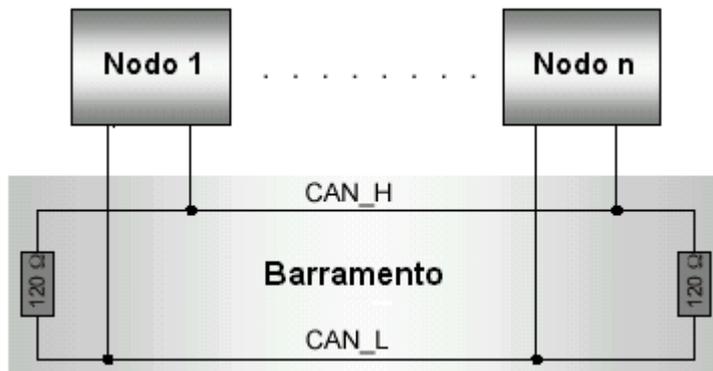


Figura 41 – Forma da fiação CAN

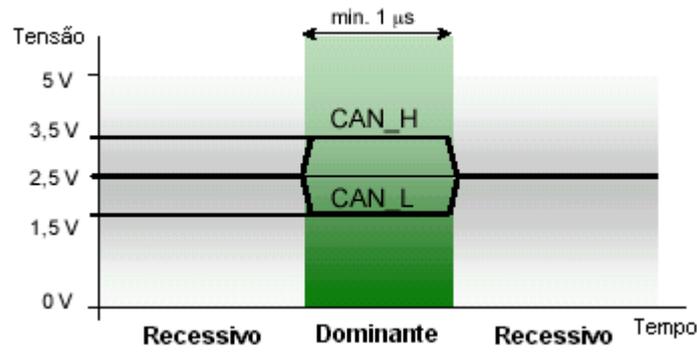


Figura 42 – Tensão nominal

Com esta técnica de fiação consegue-se minimizar efeitos de interferência eletromagnética¹⁷ porque essa interferência vai afetar ambos os fios CAN_L e CAN_H da mesma forma, permanecendo a tensão diferencial igual. A Figura 43 ilustra esse efeito no barramento.

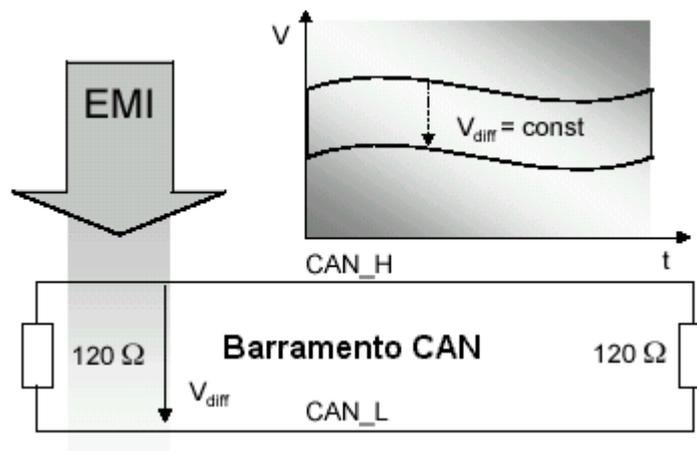


Figura 43 – Interferência eletromagnética

¹⁷ EMI (Electromagnetic Interference)

Para se fazer a interface do controlado CAN com o barramento é necessário que exista um *transceiver* para fazer a codificação do sinal do controlador para sinal de tensão diferencial, do barramento (Figura 44).

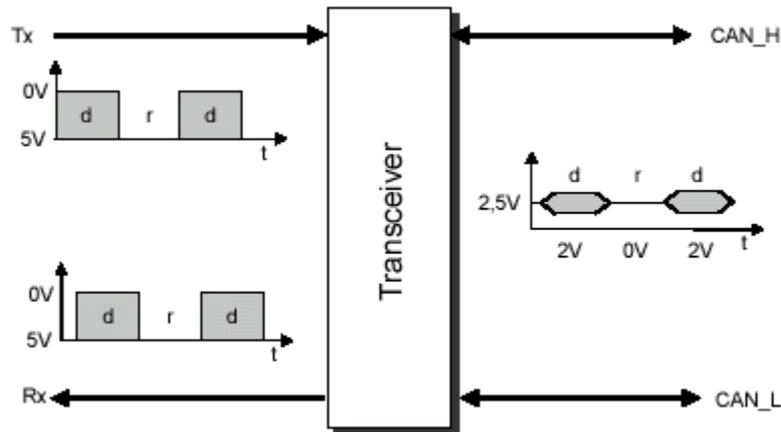


Figura 44 – *Transceiver*

A taxa de transferência máxima de uma rede CAN é de 1Mbit/s. Porém essa taxa varia de acordo com o comprimento da rede devido à resistência dos fios do barramento. A Tabela 6 mostra essa relação.

Pode-se conseguir maiores comprimentos para uma mesma taxa caso exista na rede elementos chamados de repetidores, que têm a função de reforçar o sinal elétrico para que esse possa percorrer maiores distâncias.

| Taxa (Kbit/s) | Comprimento(m) |
|---------------|----------------|
| 1000 | 30 |
| 500 | 100 |
| 250 | 250 |
| 125 | 500 |
| 62.5 | 1000 |

Tabela 6 – Taxa de transferência x comprimento do barramento

5.4 Implementação

Apesar da especificação CAN ser bem direcionada, o desenvolvedor tem liberdade de escolha na implementação de certos itens de um controlador CAN.

5.4.1 Variação Quanto à Integração

Quanto a integração de componentes podemos citar três variações:

- *Stand-alone* – Controlador CAN é separado do microcontrolador e do *transceiver*. É desenvolvido para fazer interface com diversos processadores diferentes permitindo reutilização de código. Um código desenvolvido para um periférico CAN integrado pode não funcionar em outra CPU. A Figura 45 ilustra essa idéia.

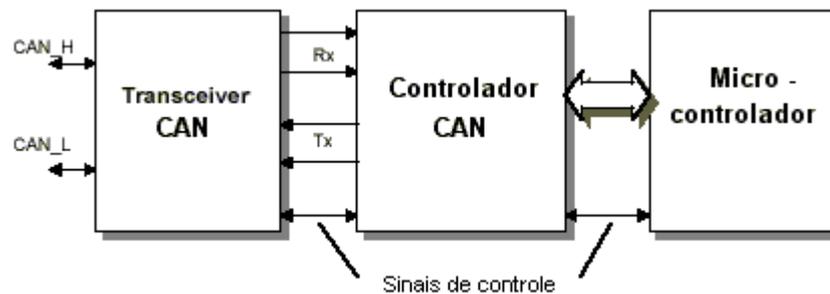


Figura 45 – Módulo CAN *Stand-alone*

- Integrado – O Controlador CAN é integrado com o microcontrolador. Módulos CAN integrados são mais baratos que *Stand-alone* não só pelo preço do módulo, mas também pela maior facilidade de desenvolver a placa de circuito impresso, uma vez que terá componentes a menos na placa. Também existe a vantagem da diminuição, em geral pela metade, da carga da CPU uma vez que o endereçamento ao controlador CAN vai ser feito por endereçamento interno que é mais rápido que endereçamento externo. A Figura 46 ilustra esse conceito.

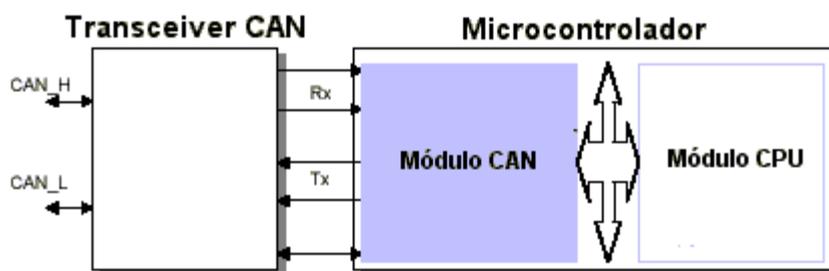


Figura 46 – Módulo CAN Integrado

- *Single Chip* – Integra um único *chip* o *transceiver*, o controlador CAN e o microcontrolador. Esse nível de integração é relativamente novo, pois possui um problema de implementação que é a integração de diferentes tecnologias em um mesmo *chip*. A Figura 47 ilustra esse conceito.

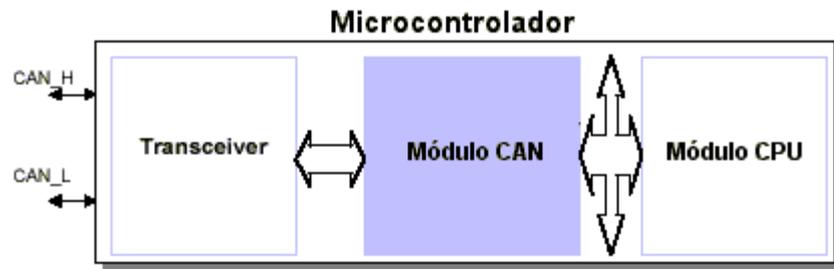


Figura 47 – Módulo CAN *Single-Chip*

5.4.2 Variação Quanto à Armazenamento de Mensagens

Quanto ao armazenamento das mensagens, existem duas diferentes implementações: *BasicCAN* e *FullCAN*.

- *BasicCAN* – É uma implementação de CAN com *buffers* intermediários para fazer a interface entre o controlador CAN e o microcontrolador. O tipo mais simples é o com dois *buffers* para recepção e um para envio de mensagens. Podem existir variações de número de *buffers*. A Figura 48 ilustra essa arquitetura.

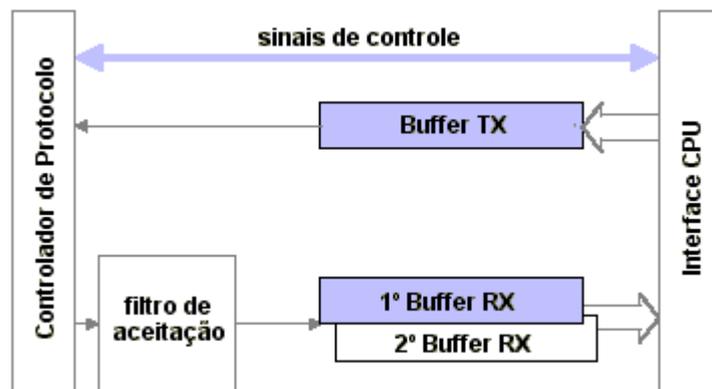


Figura 48 – *BasicCAN*

- *FullCAN* – A interface entre o microcontrolador e o controlador CAN é dada por uma memória RAM *dual-port*.

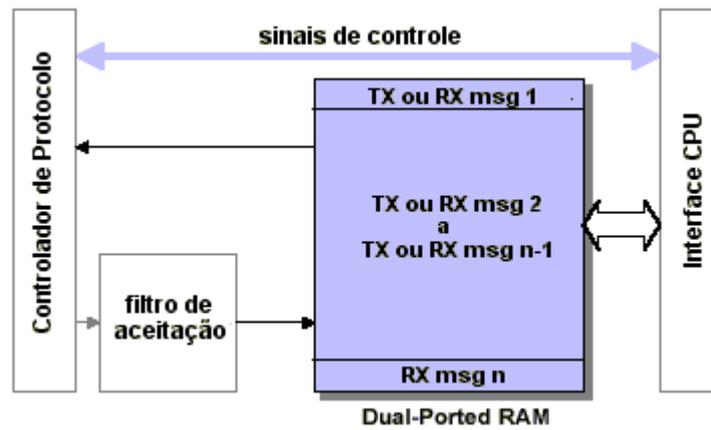


Figura 49 – *FullCAN*

Uma das diferenças entre *FullCAN* e *BasicCAN* é que, em *FullCAN*, a aplicação não precisa lidar com algoritmo de filtro de mensagem e transmissão de resposta de requisição de dados. O próprio *hardware* do controlador tem recursos suficiente para executar essas tarefas. Outra diferença é que um nodo *FullCAN* terá um número limitado de mensagens (em geral de 16 a 64) que podem ser usadas ao mesmo tempo. Em *BasicCAN* o número de mensagens é definido pela aplicação.

5.4.3 Variação Quanto ao Filtro de Aceitação

A função do filtro de aceitação é verificar se o dado que está sendo recebido é importante para o microcontrolador do nodo. Se for aceita, a mensagem será gravada no *buffer* de recebimento, caso contrário será descartada. O filtro é implementado comparando-se o identificador da mensagem que está chegando com os filtros existentes. O uso de filtros no controlador CAN libera o microcontrolador da tarefa de seleção de dados. Basicamente existem duas implementações de filtros: filtro simples e múltiplo. A escolha de uma implementação ou outra está baseada no custo do dispositivo e *overhead* de área que o projeto deseja possuir.

- Filtro Simples – Uma máscara é usada para comparar os bits de identificação que estão no barramento. Por ter somente uma máscara o nodo receberá somente uma mensagem. Ver Figura 50.

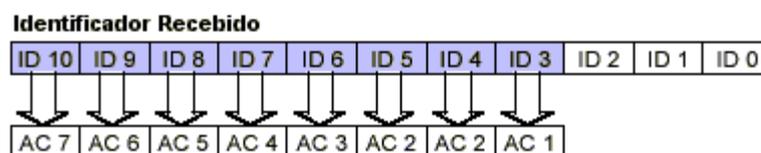


Figura 50 – Filtro simples

- Filtro Múltiplo – Múltiplas máscaras são usadas na comparação da mensagem de entrada. Desta forma o nodo pode receber várias mensagens diferentes. Ver Figura 51.

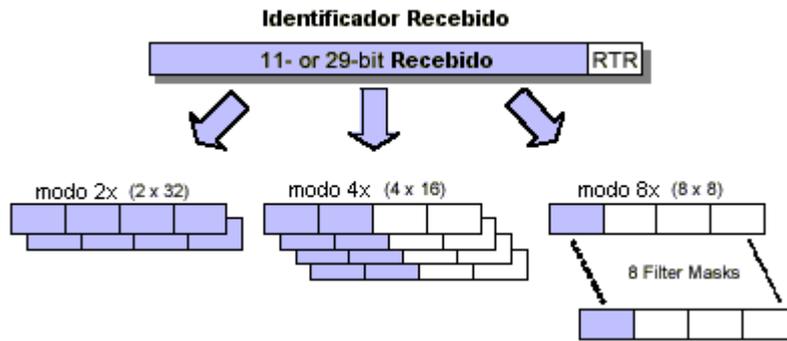


Figura 51 – Múltiplos filtros

5.5 Estudo de Arquiteturas para Protocolo CAN

Esta Seção apresenta uma arquitetura em diagrama de blocos do controlador do protocolo CAN.

Para melhor definirmos a arquitetura do controlador CAN usamos como fonte de pesquisa manuais das implementações comerciais de módulos CAN. Esses manuais foram obtidos na Internet onde alguns dos *links* pesquisados foram [1] [2] [3] [4] [5] [6] [8] [9]. Uma referência particularmente interessante para esse trabalho é [10], pois trata-se de uma proposta muito parecida com a deste projeto, porém aplicado a automação de processos usando protocolo Alnet da empresa Altus.

Por possuir maior flexibilidade de aplicação optamos por uma implementação *Stand-Along* com *mailboxes* localizados na memória do sistema e não internamente ao módulo. A Figura 52 ilustra esta estrutura.

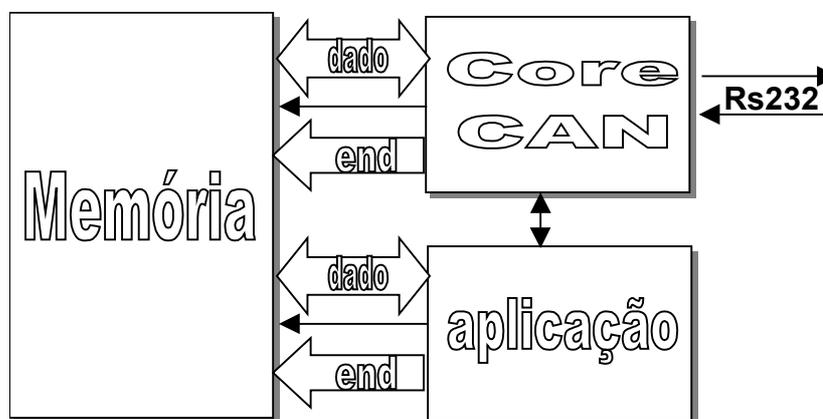


Figura 52 – Arquitetura geral do módulo CAN

Nesta figura podemos verificar que a troca de mensagem entre a aplicação do usuário e o módulo CAN ocorre através da memória. Apenas alguns sinais de controle os interligam diretamente. É importante salientar que a aplicação do usuário é o mestre da memória, por isso ela tem a necessidade de informar ao módulo CAN quando essa vai ter acesso a memória. Para essa implementação é desejável o uso de memória *dual-port*.

No caso de envio de uma mensagem a aplicação do usuário escreve na posição relativa a *mailbox* de transmissão o dado a ser transmitido. Então sinaliza, através dos sinais de controle, para iniciar o envio. Desta forma, o módulo CAN lê o dado da memória e envia-o pela rede CAN.

Durante a recepção de dados, o dado recebido será armazenado temporariamente no módulo CAN. Quando todos os bits do identificador do pacote for recebido, é verificado se esse pacote é aceito. Se não for aceito, o pacote será rejeitado. Caso seja aceito, o pacote continuará a ser recebido até o final, e então será gravado na *mailbox* de recebimento localizada na memória RAM e sinalizado à aplicação do usuário o recebimento de uma mensagem.

A Figura 53 mostra a estrutura detalhada do módulo CAN. A função de cada bloco é citada a seguir :

- Tx – bloco que executa o envio de dados. Também é responsável por fazer *bit-stuffing* e arbitração;
- Rx – bloco que executa o recebimento de dados. Também é responsável por remover *stuffing*, fazer sincronização e *timing* do nodo;
- Tx/Rx *Shift Register* – faz a conversão de paralelo para serial (transmissão) e serial para paralelo (recebimento);
- CAN FSM – é a parte de controle do módulo CAN. É responsável pelo gerenciamento de ações e tomadas de decisão. Entre suas principais ações citamos :
 - identificação e classificação de erros;
 - controle do Tx/Rx *Shift Register*;
 - controle da interface de aplicação do usuário e memória;
 - geração de endereços para memória;

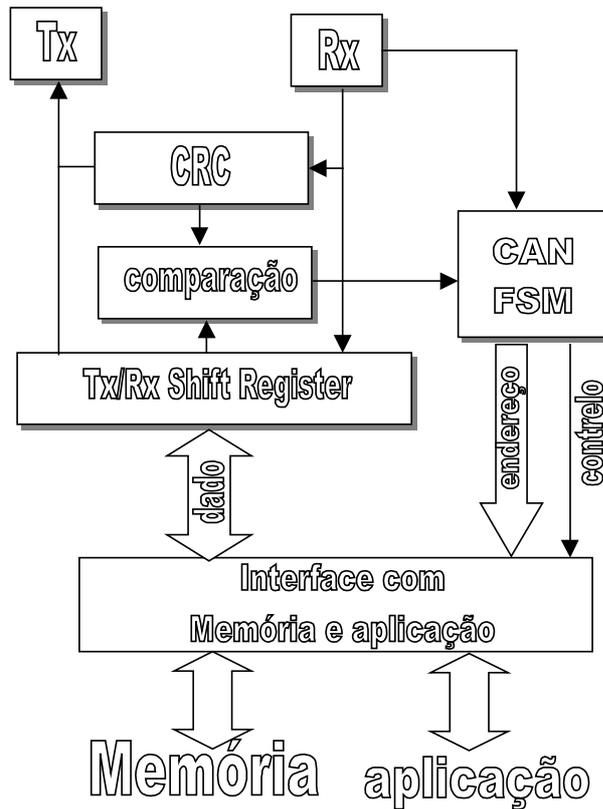


Figura 53 – Estrutura detalhada do módulo CAN

A Figura 54 expande as funcionalidade do bloco de interface com memória e aplicação do usuário.

O bloco *mailbox* é uma área de armazenamento temporário de envio e transmissão de dados interna ao módulo CAN. O bloco aceitação verifica se, em caso de recebimento, o pacote interessa a esse módulo. o bloco erro é responsável pela alteração dos registradores de contagem de erros. Controle é responsável pela sinalização de dados entre CAN e aplicação do usuário.

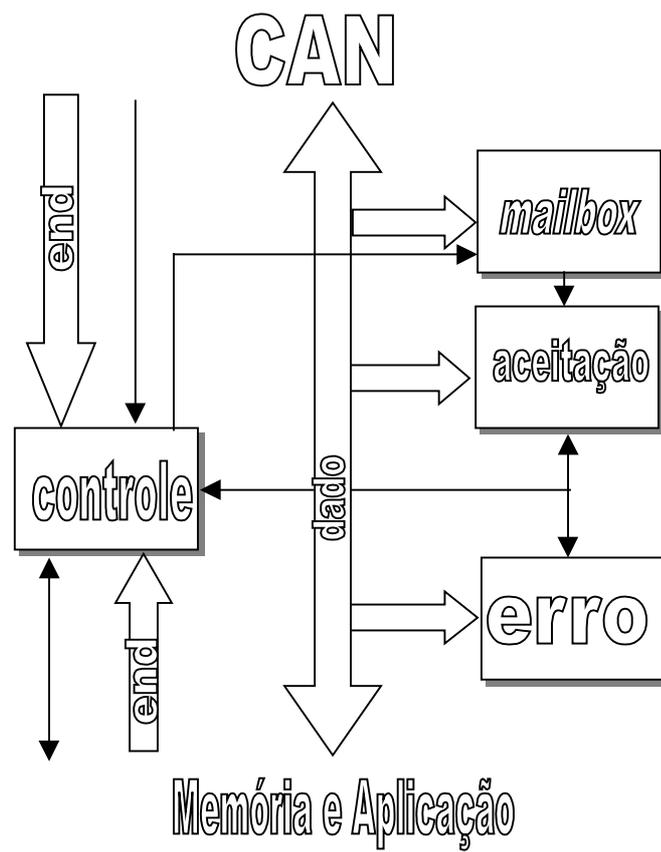


Figura 54 – Detalhe do bloco de interface com memória e aplicação do usuário

6 Arquitetura do software

Neste capítulo procuramos descrever com maiores detalhes as necessidades e o processo de escolha das melhores tecnologias para a implementação da arquitetura do sistema em nível de *software*.

6.1 Escolha do Sistema Operacional

O desenvolvimento de nosso trabalho necessita de um sistema operacional robusto, estável, flexível e seguro, garantindo desta forma que não existam limitações e empecilhos na implementação do trabalho proposto.

Devemos lembrar que o trabalho baseia-se na comunicação cliente/servidor, sendo assim, o sistema operacional deve proporcionar um suporte ao desenvolvimento de aplicações em rede.

O sistema operacional deve suportar as mais diversas linguagens de programação, permitindo desta forma uma maior flexibilidade na escolha da linguagem para a implementação do trabalho.

Outro aspecto de vital importância para a escolha do sistema operacional é a capacidade do mesmo em gerenciar a porta de comunicação serial RS-232C, pois o servidor depende desta porta para comunicação com o protocolo.

Características como por exemplo, facilidade do uso, predominância no mercado, custo, facilidade de instalação, etc, também devem ser avaliadas.

De acordo com as características e necessidades citadas acima e com base em estudos sobre os sistemas operacionais, foram escolhidos dois sistemas operacionais capazes de fornecer recursos ao desenvolvimento de nosso trabalho. Os sistemas operacionais são : Windows NT ([18] e *link* [13]) e o Linux ([19][20][21][24] *link* [13]).

Ambos os sistemas operacionais possuem características importantes que são muito bem tratadas como: estabilidade do núcleo, desempenho, robustez, flexibilidade, segurança, suporte a comunicação cliente/servidor (suporte a redes e protocolos), compiladores disponíveis, suporte a porta de comunicação serial RS-232C, etc.

A eficiência proporcionada por ambos os sistemas operacionais no desenvolvimento de nosso trabalho é garantida, no entanto, optamos pelo desenvolvimento em Windows NT,

principalmente devido à facilidade de instalação do ambiente (sistema operacional, linguagens de programação, etc), predominância no mercado e utilização do Windows NT nos laboratórios que foram desenvolvidos o trabalho.

6.2 Escolha da Linguagem De Programação

Atualmente existem inúmeras linguagens de programação que são utilizadas para o desenvolvimento de soluções informatizadas das demandas de mercado. No entanto, as linguagens de programação normalmente especializam-se no desenvolvimento de determinados “perfis” de *software*.

O levantamento das prováveis linguagens de programação é um fator importantíssimo que deve ser levado em consideração no desenvolvimento de nosso trabalho, sendo assim, seremos capazes de escolher, de maneira segura, a melhor linguagem para o desenvolvimento do trabalho.

Devido à natureza de nosso trabalho, a linguagem de programação a ser escolhida deverá obrigatoriamente dar suporte a utilização da porta de comunicação serial RS-232C, e juntamente possibilitar a comunicação cliente/servidor. Mecanismos de segurança como criptografia e autenticação de usuários também devem ser suportados pela linguagem.

Dentre as inúmeras linguagens de programação existentes no mercado, selecionamos aquelas linguagens que de antemão sabemos que possivelmente podem atender nossas necessidades, principalmente porque as mesmas são amplamente utilizadas no desenvolvimento de aplicações Internet.

Linguagens como estas possuem fortes princípios de conexões do tipo cliente/servidor (cliente é normalmente o navegador Web utilizado pelo usuário e o servidor irá rodar o programa que irá tratar os comandos vindos do navegador), inclusive também em muitos casos dão suporte à utilização da porta de comunicação serial RS-232C e controle de segurança e acesso.

Abaixo segue um levantamento das potenciais linguagens de programação a serem utilizadas no desenvolvimento de nosso trabalho, para cada linguagem foram levantadas as principais características, auxiliando neste caso para termos uma visão ampla das vantagens e desvantagens de cada linguagem, permitindo a escolha segura pela linguagem de programação que mais se adapte ao contexto do trabalho.

6.2.1 Java

Java [17][25] *link* é uma linguagem de programação totalmente orientada a objetos, projetada para o desenvolvimento de aplicações em ambientes distribuídos, como a Internet.

A aparência de Java é semelhante a linguagem C++, no entanto é muito mais simples do que C++. Java pode ser utilizado para criar aplicações completas que podem rodar em um único computador ou ser distribuídas entre servidores e clientes na rede.

Java possui a sintaxe semelhante a linguagem C, possibilitando a facilidade de adaptação para novos programadores que por ventura conhecem a linguagem C.

Java não carrega consigo algumas características da linguagem de programação C++, aumentando assim a simplicidade de codificação. Podemos citar como exemplo, herança múltipla em C++, Java não traz isso consigo devido a falta de uso prático deste recurso e principalmente devido a dificuldade de utilização.

O tamanho das aplicações desenvolvidas em Java são realmente muito pequenas, proporcionando sua maior utilização no desenvolvimento de aplicações de redes.

Java é uma linguagem orientada a objetos, facilitando assim a definição de interfaces de comunicação e aumenta a reusabilidade de código.

Os programas desenvolvidos em Java são portáteis, uma vez desenvolvido o código fonte, o mesmo é compilado gerando um código intermediário, denominado de *bytecode*. Este *bytecode* é interpretado pela Máquina Virtual Java que neste caso responsabiliza-se em gerar as instruções ao *hardware* específico.

Java possui uma extensa biblioteca de rotinas para o desenvolvimento de aplicações de rede, sendo de muito fácil utilização. Suporta o desenvolvimento de aplicações que utilizam o protocolo TCP/IP, HTTP, FTP, etc.

A linguagem impossibilita que determinadas operações realizadas pelo programador possam causar a parada de funcionamento do sistema, tornando a linguagem robusta. Por exemplo, a referência a objetos nulos. A máquina virtual se encarrega de verificar a integridade das operações para evitar estes tipos de acontecimentos comuns em outras linguagens de programação.

Devido à utilização de Java para o desenvolvimento de aplicações em ambientes de rede e ambientes distribuídos, muitos mecanismos de segurança foram agregados visando o desenvolvimento de aplicações seguras.

Java é uma linguagem de programação interpretada. Devido a isso, o *just-in-time compiler* foi introduzido para o aumento de performance das aplicações Java.

O desenvolvimento de aplicações multitarefa é simplificado devido à facilidade de manipulação de *threads* em Java.

Java possui ferramentas que auxiliam na documentação, aumentando a legibilidade e reutilização de classes e métodos.

Recentemente, a *Sun Microsystems* desenvolveu o pacote `javax.comm`¹⁸, responsável em permitir que aplicações escritas em Java possam enviar e receber dados através das portas de comunicação serial e paralela.

6.2.2 Perl

Perl [22] é uma linguagem de programação utilizada principalmente para a manipulação de textos e arquivos.

O objetivo principal da criação da linguagem Perl foi devido a necessidade de seu inventor, Larry Wall, de criar relatórios a partir de arquivos texto com facilidade e em um pequeno espaço de tempo, pois o mesmo levava muito tempo desenvolvendo um programa em C ou em outras linguagens para solucionar um problema simples de ser resolvido se a linguagem de programação utilizada fosse mais prática e direta.

Além da facilidade de utilização, Perl possui como característica importante a sua portabilidade. Perl surgiu primeiramente para sistemas operacionais Unix. Devido ao seu sucesso entre programadores, foram criadas versões de Perl para o Amiga, Atari ST, Macintosh, VMS, OS/2 e inclusive MS-DOS, Windows NT e Windows 95/98.

Perl é uma linguagem interpretada, no entanto, devido a otimizações internas ao interpretador, Perl possui um boa performance durante sua execução. Qualquer requisição de execução de um *script* Perl verifica a sintaxe do *script* antes de sua execução.

Perl possui a sintaxe semelhante a sintaxe utilizada na linguagem C, possibilitando a facilidade de adaptação de programadores C à linguagem Perl.

Perl é uma linguagem fracamente tipada, não necessita da declaração prévia de variáveis antes de suas utilização e muito menos a necessidade de utilização de *casting* explícitos.

Ainda relacionada a facilidade de utilização, Perl utiliza expressões regulares para a manipulação de informações, facilidade na utilização de operações de entrada e saída,

¹⁸ Java™ CommunicationsAPI - <http://java.sun.com/products/javacomm/index.html>

características avançadas para processamento de texto, enfim, uma biblioteca de funções muito completa para as mais diversas necessidades.

Não podemos deixar de salientar a vantagem de utilização de Perl devido à vasta gama de documentações sobre a linguagem. Atualmente existem inúmeros livros e documentações na Internet capazes de ajudar desde programadores novatos até programadores experientes na linguagem.

Perl não possui custo algum para sua utilização, é totalmente de graça e possui o código fonte aberto, permitindo que qualquer pessoa possa alterá-lo de acordo com suas necessidades.

Infelizmente Perl possui algumas desvantagens, entre elas podemos citar :

- Permite a visualização do código fonte, impedindo os programadores de esconder o código;
- *Scripts* escritos em Perl necessitam da existência do interpretador para que sua execução seja possível;
- Perl não possibilita o acesso a funções de baixo nível, reduzindo a liberdade do programador nestes tipos de caso.

6.2.3 C

Inúmeras são as características da linguagem de programação C, iremos citar aquelas que acreditamos serem relevantes dentro do contexto de desenvolvimento de nosso trabalho.

Arquivos binários gerados pelo compilador C a partir de um código fonte são em geral de pequeno tamanho e de rápida execução, pois já contém as instruções de máquina específicas para a plataforma, que são executadas diretamente.

Elevado número de bibliotecas em diferentes plataformas. Essas bibliotecas implementam as mais diversas funcionalidades e são utilizadas de acordo com a necessidade de desenvolvimento do programador. Exemplo: bibliotecas matemáticas.

Linguagem fortemente tipada, exigindo a definição prévia de tipos de dados antes de sua utilização.

Linguagem de programação estruturada, facilitando a compreensão dos programas escritos.

Apesar de C ser uma linguagem de programação de alto nível, C permite aos programadores a capacidade de programação em baixo nível, como por exemplo a manipulação de bits, aumentando a possibilidade de desenvolvimento de uma gama maior de aplicações pela linguagem.

A linguagem permite a utilização de estruturas de dados como *arrays*, *structures* e *unions*, além de manipulação de ponteiros para áreas de memória, alocação de memória dinâmica, operações de entrada e saída, etc.

O desenvolvimento de aplicações em C, utilizando execuções concorrentes é possível através da utilização de *threads*.

C possibilita o desenvolvimento de aplicações que necessitam comunicar-se entre si através da rede, isto é possível através da utilização de *sockets*.

A linguagem C é capaz de gerenciar e manipular informações através de portas de comunicação serial e paralela, enviando e recebendo bits.

C tornou-se uma linguagem de programação de extrema utilização em todo mundo principalmente devido a sua capacidade de ao mesmo tempo ser uma linguagem de alto nível mas permitir o desenvolvimento de operações de baixo nível.

Como já foi visto, C permite a realização de operações baixo nível, como por exemplo a manipulação de memória através de ponteiros, contudo, a linguagem de programação C não possui nenhum mecanismo de segurança para impedir e garantir a integridade do sistema. Em outras palavras, facilmente em C podemos invadir a área de memória de outra aplicação causando a pane no sistema.

6.2.4 Active Server Pages - ASP

Active Server Pages link [29] é uma tecnologia desenvolvida pela *Microsoft link* [29] que permite o desenvolvimento de aplicações dinâmicas na Internet.

ASP segue o mesmo conceito das *Java Server Pages* (Seção 3.3.3). No caso de páginas ASP, a parte dinâmica da página é construída utilizando-se a linguagem *VB Script*, ao contrário do JSP, cuja parte dinâmica é programada em Java.

VB Script é a denominação dada pela Microsoft à linguagem de programação responsável pela parte dinâmica das páginas ASP. Semelhante ao Basic, possui basicamente as mesmas características.

A principal característica e facilidade que páginas ASP proporcionam é a sua capacidade de conexão com banco de dados, utilizando a tecnologia ADO, facilmente pode-se construir uma página Web para exibir por exemplo a ficha cadastral de um cliente através de um navegador Web. No entanto, sua portabilidade é bastante limitada.

6.2.5 PHP : Hypertext Preprocessor

PHP *link* [28] segue o mesmo contexto de JSP's e ASP's, no entanto, sua sintaxe é semelhante a linguagem C.

PHP é utilizada principalmente devido a rapidez e facilidade no desenvolvimento de aplicações, agregando características da linguagem Perl.

PHP não é uma linguagem fortemente tipada, no entanto, permite ao programador a declaração de tipos caso haja necessidade, tornando a linguagem fortemente tipada somente no caso da necessidade desta característica e como em C e Java, permite a utilização de *casting* explícitos.

Utilização de expressões regulares para a manipulação de textos, assemelhando-se mais uma vez com a linguagem Perl.

Permite a utilização de estruturas de dados como *arrays*, *structures*, entre outras, no entanto, não é capaz de realizar manipulação de memória através de ponteiros.

PHP possibilita ao programador o desenvolvimento de programas orientados a objeto, permitindo a definição de classes, métodos, construtores, herança simples, etc.

PHP possui uma biblioteca com aproximadamente 1000 funções, proporcionando a versatilidade no desenvolvimento de aplicações Web.

PHP possui suporte de conexão a diferentes tipos de banco de dados de maneira simplificada.

Como característica de linguagens de programação utilizadas na Internet, suporta comunicações entre aplicações através de *sockets* de maneira simplificada e de fácil utilização.

6.2.6 Comparação entre Linguagens de Programação

Visando auxiliar na escolha da linguagem de programação mais apropriada ao desenvolvimento do trabalho, fizemos um quadro comparativo (Tabela 7) com as principais características das diversas linguagens de programação estudadas.

| | Java | Perl | C | ASP | PHP |
|---|------------------------|----------------|-----------|--------------|----------------|
| Sintaxe | Semelhante a C | Semelhante a C | C | VB Script | Semelhante a C |
| Simplicidade | Intermediário | Fácil | Difícil | Fácil | Fácil |
| Tamanho | Muito pequeno | Pequeno | Pequeno | Pequeno | Pequeno |
| Suporte a orientação a objetos | Sim | Não | Não | Não | Sim |
| Portabilidade | Sim | Sim* | Sim* | Sim* | Sim |
| Suporte a redes (TCP/IP, Sockets, etc) | Sim | Não | Sim | Não | Sim |
| Robusto | Sim | Sim | Não | Sim | Sim |
| Segurança | Sim | Sim | Sim | Sim | Sim |
| Interpretado | Interpretado/Compilado | Interpretado | Compilado | Interpretado | Interpretado |
| Performance | Muito Boa | Mediana | Ótima | Mediana | Mediana |
| Multithreads | Sim | Não | Sim | Não | Sim |
| Suporte a comunicação Serial | Sim | Não | Sim | Não | Não |

Tabela 7 – Comparação entre diversas linguagens de programação

*Dependendo da plataforma pode requerer pequenas mudanças no código.

6.2.7 Conclusão

A escolha da linguagem de programação apropriada é um passo fundamental para o sucesso do desenvolvimento de nosso trabalho.

Conforme descrito na Seção 2, referente a arquitetura do sistema, necessitamos de uma linguagem de programação que permita o desenvolvimento de aplicações Internet. Além disso, necessitamos obrigatoriamente de uma linguagem que possibilite ao computador Servidor enviar e receber dados através da interface serial RS-232C, pois este é o único método de comunicação com os periféricos da casa.

No entanto é importante salientar que a linguagem de programação para o desenvolvimento da *homepage* não necessita obrigatoriamente ser a mesma linguagem utilizada para o desenvolvimento da interface de comunicação através da porta RS-232C.

Devido ao pré-requisito de comunicação utilizando a interface RS-232C, podemos descartar primeiramente todas as linguagens que não permitem gerenciar e controlar a interface de comunicação serial RS-232C. Sendo assim, linguagens como Perl, ASP, PHP são descartadas de nossas opções, no entanto, estas linguagens são utilizadas freqüentemente no desenvolvimento de aplicações Internet, sendo assim, uma delas será escolhidas para o desenvolvimento da *homepage*.

Dentre as linguagens de programação estudadas, as linguagens que suportam comunicação serial resumem-se à apenas duas: Java e C.

Ambas as linguagens são capazes de proporcionar o desenvolvimento de nosso trabalho, no entanto, visando tornar o trabalho portátil, flexível, legível e robusto, será utilizada para o desenvolvimento a linguagem Java, pois além de cumprir com todos os requisitos exigidos, Java é totalmente portátil entre plataformas devido a presença da Máquina Virtual Java.

Java possui ferramentas de documentação importantes para melhorar a documentação do sistema, facilitando a legibilidade do código através de API's geradas por uma ferramenta de documentação.

A Máquina Virtual Java permite um controle sobre as operações executadas pelo sistema, garantindo a integridade do sistema, tornando-o robusto e conseqüentemente impedindo que falhas no sistema operacional possam acontecer por sua causa.

Java possui uma vasta API com a definição de classes e métodos capazes de propiciar o desenvolvimento do sistema de maneira simplificada, aumentando a legibilidade do código e evitando a necessidade de desenvolvimento de rotinas que não estão implementadas na linguagem C.

Enfim, a utilização de Java como linguagem de programação aumenta a flexibilidade de desenvolvimento do sistema além de possuir maiores vantagens sobre a linguagem C.

A linguagem Java foi escolhida para o desenvolvimento da aplicação que tem o controle sobre a interface de comunicação serial RS-232C, no entanto, a escolha da linguagem de programação para o desenvolvimento da *homepage* deve ser feita, onde classificamos como linguagens candidatas : Java, Perl, ASP e PHP.

Para o desenvolvimento da *homepage*, foi escolhida a linguagem PHP devido as suas vantagens que serão vistas na Seção 7.1.2.

Outras linguagens se fazem necessárias (HTML e Javascript) independentemente se a linguagem de programação principal for PHP, Perl, Java ou ASP. Nas Seção 7.1.1 e 7.3.2 são explicados com maiores detalhes a coexistência entre todas estas linguagens para o desenvolvimento da *homepage*.

6.3 Arquitetura Servidor/CAN

Visando permitir a comunicação entre o servidor e o controlador mestre, devemos determinar um protocolo de comunicação entre ambos, permitindo que o controlador mestre no momento que receber uma seqüência de bits consiga interpretá-la de maneira correta, e os envie aos diversos periféricos da residência. Este tópico será descrito na Seção 8.5.

6.4 Tipo de Conexão entre máquinas

De acordo com a arquitetura do sistema que está sendo proposto, devemos determinar qual será o tipo de conexão entre máquinas apropriado à necessidade de trabalho. Seguem abaixo os tipos de conexão em estudo:

- Computador cliente juntamente com o computador servidor poderão ser máquinas equipadas com placas de Fax/Modem em cada um deles, na qual a conexão irá se realizar através de uma conexão ponto-a-ponto através da rede de telefonia convencional ou através da conexão por rede de telefonia celular.
- Computador servidor poderá ser uma máquina conectada diretamente à Internet, o que permite que qualquer cliente em qualquer lugar do mundo possa ter acesso ao sistema.

Com o advento da Internet a cabo, e mais recente, Internet por microondas, está se tornando comum nos lares a existência de computadores conectados diretamente à Internet com altas velocidades de transmissão, com custos relativamente baixos se comparados a outros meios de acessos menos eficientes e que são responsáveis em distribuir o acesso à Internet aos demais computadores do lar.

Em nosso trabalho adotamos como padrão uma máquina conectada diretamente à Internet devido a grande tendência mundial deste tipo de conexão.

6.5 Arquitetura Cliente/Servidor

Antes de começarmos com o estudo da arquitetura cliente/servidor, torna-se importante deixar claro o que é uma arquitetura cliente/servidor e como esta arquitetura irá definir os caminhos do desenvolvimento de nosso trabalho.

A arquitetura cliente/servidor [23] consiste basicamente na comunicação entre dois *softwares* denominados neste caso de cliente e servidor. Cada um destes *softwares* tem sua função específica, na qual o servidor é responsável por aceitar requisições de informações dos clientes e retornar os resultados do processamento aos clientes responsáveis por exibir estas informações de maneira amigável aos usuários.

A comunicação cliente/servidor está relacionada à forma como o cliente irá montar os pacotes de informações (provenientes da interação do usuário com o cliente) a serem enviados ao servidor e vice-versa.

A definição do protocolo de comunicação irá permitir que tanto o cliente quanto o servidor possam trocar mensagens inteligíveis para ambos.

A escolha do protocolo de comunicação cliente/servidor pode ser realizada de duas formas. A primeira escolha refere-se ao desenvolvimento de um protocolo de comunicação proprietário da aplicação. A segunda escolha seria a utilização de protocolos de comunicações já existentes no mercado e que são utilizados nas mais diversas aplicações Internet.

Devido ao nosso trabalho utilizar um navegador Web como sendo o *software* localizado no computador Cliente. E no computador Servidor possuímos um servidor Web Apache, utilizaremos o protocolo de comunicação HTTP [16] (*HyperText Transfer Protocol*) por se tratar de um protocolo de transferência padrão na Internet e atender nossas necessidades de desenvolvimento.

O HTTP também é um protocolo que está em constante evolução, sendo que há várias versões em uso e outras tantas em desenvolvimento.

A utilização de um protocolo proprietário está totalmente descartada, pois para isso necessitaríamos da construção de um novo protocolo de comunicação, além da necessidade de construção de uma aplicação específica tanto para o cliente quanto para o servidor. Estas aplicações seriam responsáveis por interpretar e tratar as informações que trafegariam neste novo protocolo.

Outro ponto importante a ser levantado é que o desenvolvimento de uma aplicação cliente específica para o controle da residência obrigaria o usuário a carregar consigo a aplicação em algum meio de armazenamento para possibilitar a sua instalação na máquina. A utilização de um navegador Web neste caso descarta esta necessidade, pois atualmente a grande maioria dos sistemas operacionais já possui um navegador Web incorporado.

7 Implementação do Software

Durante o decorrer deste capítulo, explicaremos a implementação de *software* deste projeto.

A parte de *software* é dividida em quatro Seções: arquitetura geral do *software* (Seção 7.1), arquitetura do servidor (Seção 7.2), arquitetura do cliente (Seção 7.3) e arquitetura do *debugger* (Seção 7.4).

7.1 Arquitetura Geral do Software

Como pode ser visto na Figura 55, a arquitetura do sistema é composta de três grandes blocos que interagem entre si. Estes blocos são o cliente, o servidor e o *hardware*. Cada bloco possui uma ou mais funcionalidades, sendo que a troca de informações entre blocos garante o perfeito funcionamento do sistema.

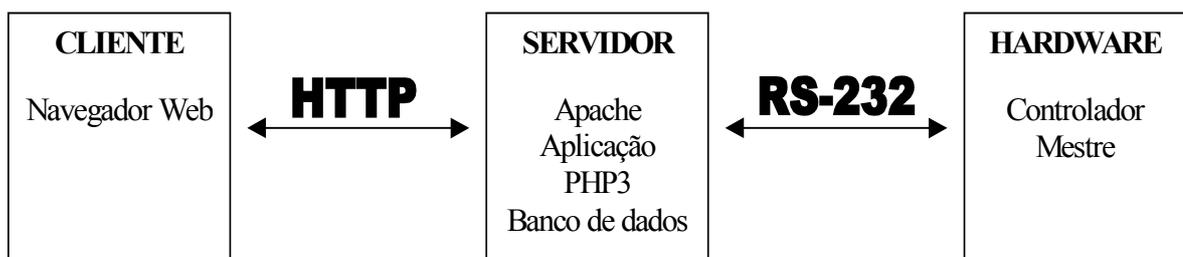


Figura 55 - Arquitetura do Sistema

7.1.1 Cliente

A figura do cliente é representada por um navegador Web. Sua principal função é exibir ao usuário o status de sua residência, além de possibilitar a interação por parte do usuário, permitindo que o mesmo possa realizar comandos remotos na sua residência, como por exemplo desligar a lâmpada da sala, ou capturar a imagem de vídeo de uma câmera na entrada da casa.

Devido a aplicação cliente se tratar de um navegador Web, conforme visto na Seção 2, o protocolo de comunicação utilizado para comunicação entre o computador cliente e computador servidor é o protocolo HTTP.

Para o desenvolvimento da interface, utilizou-se três linguagens de programação:

- HTML – responsável por montar estaticamente a interface da aplicação no cliente;

- Javascript *link*[37] – responsável por tornar a interface dinâmica a medida que o usuário interage com a mesma. Também utilizada para gerar atualizações na interface quando ocorreram mudanças na aplicação final (residência);
- PHP *link*[28] - Devido a grande interação entre cliente-servidor, linguagens de geração de páginas dinâmicas tiveram que ser utilizadas.

Apesar da linguagem PHP ser responsável pela geração de páginas dinâmicas, o funcionamento da linguagem ocorre na máquina servidor, acoplado juntamente com o servidor Web. Maiores detalhes sobre a utilização de PHP serão discutidos no Seção 7.3.

Como característica importante, citamos a utilização de controle de sessão no cliente, o que garante a segurança da residência, pois somente através de autenticação eletrônica o usuário possui controle da residência.

Para auxiliar o desenvolvimento da interface no cliente, pode-se utilizar um editor de texto qualquer, no entanto, utilizou-se como ferramenta de desenvolvimento o *software HomeSite link*[34]. Este *software* foi escolhido devido às facilidades que o mesmo proporciona no desenvolvimento de aplicações utilizando as linguagens de programação empregadas no cliente.

7.1.2 Servidor

O servidor é o computador que está localizado junto à residência do usuário. Dentre as funcionalidades realizadas pelo servidor, citamos :

- envio e recepção de pacotes de controle através da porta RS-232;
- interpretação dos pacotes de controle enviados pelo controlador mestre;
- monitorar a base de dados em busca de alterações no status da aplicação;
- atualização da base de dados;
- atualização da interface no cliente.

A base de dados, no contexto do sistema, serve como *gateway* entre o computador cliente e o controlador mestre. Ela também é responsável em manter o status da residência, sendo que qualquer alteração no status da residência deve ser gravada na base de dados. A partir do momento que ocorre uma alteração na base de dados, o *software* que está rodando no computador servidor é

capaz de gerar comandos de atualização para a interface cliente, ou enviar pacotes de controle através da porta RS-232, gerando neste caso a atualização dos periféricos dentro da residência.

O *software* desenvolvido para rodar no servidor e realizar algumas das funcionalidades citadas acima foi totalmente desenvolvido em Java. Este *software* é capaz de comunicar-se com a porta de comunicação RS-232, assim como acessar a base de dados existente no servidor.

A base de dados utilizada no sistema foi desenvolvida em *Access*, pois o sistema operacional que está sendo utilizado para desenvolvimento do sistema pertence à família MS – Windows. O acesso a base de dados através do *software* desenvolvido ocorre com a utilização de JDBC (Java Database Connectivity) [17][25], utilizando o driver denominado JDBC-ODBC [17] [25].

Como parte da arquitetura necessária para alcançar o objetivo do trabalho existirá instalado um servidor Web, rodando e ativo, localizado na residência. Sua finalidade é armazenar a interface da residência. O servidor Web escolhido para suportar nosso sistema foi o servidor Web Apache. O principal motivo pela escolha deste servidor Web podem ser justificadas pelas seguintes características: robusto, confiável, largamente utilizado na internet, de fácil instalação e principalmente por suportar diferentes plataformas de *hardware*.

Devido as características gerais da arquitetura do sistema, é necessário a utilização de linguagens que permitam a geração de páginas dinâmicas e acesso a banco de dados via Web.

Conforme descrito na Seção 6.2.5, a linguagem PHP provê as características citadas acima, além de: (i) possuir compatibilidade com servidores Web Apache (normalmente linguagens de geração de páginas dinâmicas rodam acopladas a um servidor Web); (ii) ser compatível com um grande número de servidores Web existentes no mercado, assim como um grande número de plataformas computacionais; (iii) suportar um grande número de banco de dados diferentes disponíveis no mercado, garantindo assim a portabilidade do sistema entre diferentes plataformas, servidores Web e fabricantes de banco de dados.

Dentre as razões citadas, devemos salientar que a linguagem PHP foi totalmente projetada para o desenvolvimento de aplicações Web. Como parte de seu projeto inicial, o objetivo da linguagem é proporcionar aos programadores o desenvolvimento de sistemas Web de maneira fácil, simples e rápida.

7.1.3 Debugger (Hardware)

O *hardware* é representado pelo controlador mestre, juntamente com os nodos acoplados a ele via barramento CAN. Sua principal função é distribuir aos nodos os pacotes de comandos recebidos através da interface RS-232, assim como enviar para o computador servidor pacotes de controle.

Devido a necessidade de verificar o que o computador servidor está enviando através da porta RS-232, foi desenvolvido um *software* de *debug* que mostra na tela todas as informações que chegam até ele, logo, a partir deste momento podemos verificar se a implementação do *software* no computador servidor está de acordo com o protocolo CAN explicado na Seção 8.5. O *software* de *debug* também possibilita que pacotes de controle sejam enviados ao servidor, neste caso, simulando o envio de pacotes como se fosse o próprio controlador mestre, permitindo assim o teste das funcionalidades realizadas pelo *software* residente no servidor.

7.2 Arquitetura do Servidor

Esta Seção apresenta o servidor, dividido em três partes: comunicação com o cliente, banco de dados e comunicação com o controlador mestre.

7.2.1 Comunicação com o cliente

Como descrito na Seção 2, toda a comunicação com o computador cliente ocorre através do protocolo HTTP. A distribuição das informações através deste protocolo é feita através de um servidor Web Apache, versão 1.3.14, rodando no servidor. Todas as requisições que partirem do computador cliente serão recebidas pelo servidor Web. O servidor é responsável por gerenciar estas requisições e respondê-las ao longo do tempo ao(s) cliente(s).

A *homepage* da residência é armazenada no disco rígido do computador, cujo diretório é diretamente mapeado no servidor Web, permitindo assim a disponibilização da *homepage* na internet. Para o desenvolvimento da *homepage*, foram utilizadas as linguagens de programação HTML, Javascript e PHP, conforme visto na Seção 7.1.1.

A linguagem PHP é executada no servidor Web e gera saídas de texto no formato HTML / Javascript, que neste caso quando enviadas através do protocolo HTTP ao cliente (navegador Web) serão interpretadas, montando a interface visual e um conjunto de funcionalidades oferecidas ao usuário. O detalhamento completo sobre a funcionalidade do computador cliente serão explicadas no Capítulo 7.3.

7.2.2 Banco de dados

O banco de dados utilizado para o desenvolvimento do sistema foi o *Access*, devido ao desenvolvimento do trabalho ocorrer sobre a plataforma Windows NT. O banco de dados *Access* apresenta ótima compatibilidade, desempenho e interface de fácil interação. A utilização de banco

de dados em *Access* não compromete a portabilidade do sistema, pois todos os tipos de dados utilizados no banco de dados são comuns entre fabricantes de banco de dados, o que garante que através de comandos SQL possamos migrar o banco de dados atualmente feito em *Access* para outros sistemas de banco de dados.

Na Figura 56 apresentamos as tabelas utilizadas no sistema, assim como seus respectivos relacionamentos. A seguir, descreveremos detalhadamente todas as tabelas e relacionamentos existentes no banco de dados criado para suportar o sistema proposto no trabalho. A Figura 56 ilustra no canto superior esquerdo a tabela **Usuarios**. Sua principal função é armazenar os usuários que possuem permissão de acesso à *homepage* da residência e conseqüentemente o acesso às aplicações desta. Também indica quais usuários possuem acesso a interface de manutenção de usuários. Na Tabela 8 informações mais detalhadas são fornecidas.

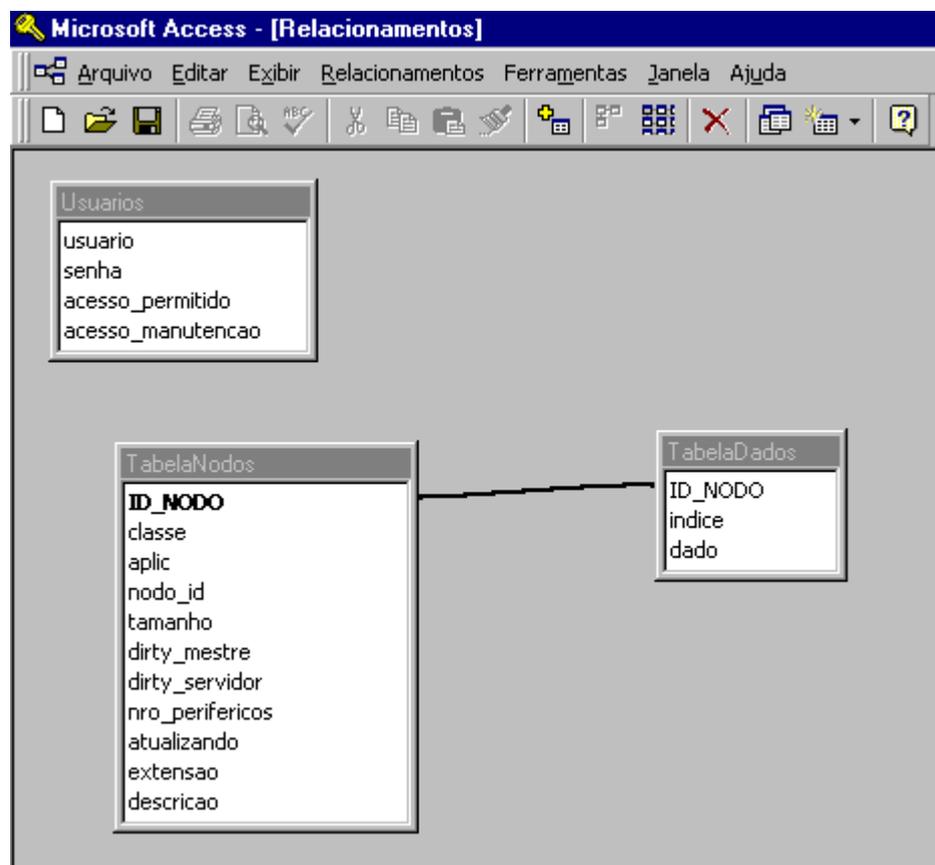


Figura 56 – Tabelas do Sistema

| Coluna | Tipo | Tamanho | Descrição |
|---------|------|---------|-----------------|
| Usuario | char | 10 | Nome do usuário |

| | | | |
|-------------------|------|----|--|
| Senha | char | 32 | Senha do usuário criptografada pelo algoritmo md5 |
| acesso_permitido | byte | - | Indica se o usuário tem acesso a <i>homepage</i> |
| acesso_manutencao | byte | - | Indica se o usuário tem acesso a interface de manutenção de usuários |

Tabela 8 – Tabela Usuarios

As colunas *acesso_permitido*, *acesso_manutencao* são definidas como do tipo byte, no entanto, para o sistema essas colunas tem significado boleano, ou seja, verdadeiro ou falso. Neste caso representamos verdadeiro pelo número 1 e falso pelo número 0.

Visando aumentar a segurança e a confiabilidade do sistema, a senha do usuário é criptografada e armazenada no banco de dados. O algoritmo de criptografia utilizado pelo sistema é o *md5*, com *fingerprint* de 128 bits (*link* [30]). O algoritmo *md5* é um algoritmo do tipo irreversível e sua principal utilização é comparação de *strings* criptografadas. Seu funcionamento é simples, basta uma *string* qualquer ser informada e o algoritmo automaticamente retorna outra *string* de exatamente 32 bytes. A senha do usuário é armazenada criptografada. A todo momento que o usuário realizar uma tentativa de login no sistema, a senha digitada é criptografada e comparada com a senha existente no banco de dados, se as *strings* forem iguais e o usuário possuir no banco de dados permissão de acesso ao sistema (coluna *acesso_permitido*), o usuário é aceito pelo sistema.

No trecho de código abaixo mostramos como ocorre a criptografia de uma *string* através do algoritmo *md5* utilizando a linguagem PHP.

```
$senha = "futebol"; // a variavel senha recebe a string "futebol"
$senha_cripto = md5($senha); // a variavel senha_cripto recebe o resultado da
criptografia da string contida na variavel senha ("futebol")
```

Outra tabela existente no banco de dados é a tabela denominada **TabelaNodos**. Esta tabela é responsável em armazenar as informações referentes ao periféricos existentes na residência. A Tabela 9 representa a estrutura de dados utilizada na TabelaNodos.

| Coluna | Tipo | Tamanho | Descrição |
|---------|----------|---------|--|
| ID_NODO | long int | - | Identificador do NODO (chave primária) |
| classe | long int | - | Classe |

| | | | |
|-----------------|----------|----|--|
| aplic | long int | - | Aplicação |
| nodo_id | long int | - | Nodo id |
| tamanho | long int | - | Tamanho dos dados |
| dirty_mestre | byte | - | Dirty em relação ao mestre |
| dirty_servidor | byte | - | Dirty em relação ao servidor |
| nro_periféricos | long int | - | Número de periféricos |
| atualizando | byte | - | Indica se o registro está sendo atualizado |
| extensao | char | 5 | Em caso de pacotes maiores que 8 bytes indica com qual extensão o arquivo deve ser salvo |
| descricao | char | 50 | Descrição do tipo de aplicação |

Tabela 9 – Tabela TabelaNodos

Os campos desta tabela de nodos são descritos abaixo.

- *ID_NODO* – é a chave primária da tabela e sua principal função é permitir o relacionamento entre a TabelaNodos com a TabelaDados. Informações mais detalhadas sobre este relacionamento serão apresentadas nas próximas páginas;
- *classe* – representa a classe da aplicação dentro da residência. Exemplo: iluminação, refrigeração;
- *aplic* – representa a aplicação propriamente dita. Exemplo: dentro da classe iluminação existe a aplicação lâmpadas, *dimmers*, etc;
- *nodo_id* – representa um conjunto finito de aplicativos de uma mesma aplicação;
- *tamanho* – indica o número bytes utilizado para que a aplicação seja controlada;
- *dirty_mestre* – indica que o controlador mestre está desatualizado em relação a base de dados;
- *dirty_servidor* – indica que o estado da base de dados foi alterado e o servidor deve enviar comandos para a atualização da interface cliente;
- *nro_periféricos* – indica o número de periféricos existentes para uma aplicação;
- *atualizando* – indica que o registro está sendo atualizado e garante que aquele registro não seja lido antes de sua atualização ser considerada concluída;
- *extensao* – o envio/recepção de pacotes maiores que 8 bytes ocorrem normalmente para arquivos transportando imagens, vídeos, etc. Neste caso existe a necessidade de sabermos qual é o tipo de arquivo que trafega para uma determinada aplicação, pois no término do processo de envio, o

arquivo deve ser salvo com a extensão correta para que depois esse arquivo possa ser interpretado pela aplicação cliente o navegador Web;

- *descricao* – descrição que indica ao servidor como ele deve fazer a atualização da interface do cliente.

As colunas *dirty_mestre*, *dirty_servidor* e *atualizando* como no caso da tabela Usuarios, representam informações booleanas, sendo assim, o número 0 representa falso e o número 1 verdadeiro.

Na Figura 57 apresentamos a tabela TabelaNodos com três tipos diferentes de aplicações cadastradas.

| ID_NODO | classe | aplic | nodo_id | tamanho | dirty_mestre | dirty_servidor | nro_periferico | atualizando | extensao | descricao |
|---------|--------|-------|---------|---------|--------------|----------------|----------------|-------------|----------|-----------|
| 2 | 1 | 1 | 0 | 1 | 1 | 0 | 6 | 0 | | lampada |
| 3 | 1 | 1 | 2 | 3 | 0 | 0 | 3 | 0 | | dimmer |
| 4 | 6 | 6 | 10 | 888 | 0 | 1 | 0 | 0 | jpg | camera |

Figura 57 - TabelaNodos

A **TabelaDados** é a terceira e última tabela do banco de dados. A responsabilidade desta tabela é armazenar os dados referentes as aplicações existentes na residência.

Na Tabela 10 detalhamos informações sobre a TabelaDados.

| Coluna | Tipo | Tamanho | Descrição |
|---------|-------------|---------|---|
| ID_NODO | long int | - | Identificador do NODO ID (chave estrangeira) |
| indice | long int | - | Posição do byte em relação ao pacotes de bytes transmitidos |
| dado | int | - | Bytes transmitidos |

Tabela 10 - Tabela TabelaDados

Abaixo explicamos detalhadamente o significado de cada coluna existente na TabelaDados.

- *ID_NODO* – utilizada para identificar qual é a aplicação que o registro está associado na TabelaNodos;

- *indice* – indica qual é a posição do byte dentre todos os bytes existentes para aquela aplicação específica;
- *dado* – representa o dado da aplicação.

O relacionamento criado entre as tabelas TabelaNodos e TabelaDados existe para permitir identificar, quais são os dados existentes para cada aplicação. Desta forma, a coluna denominada ID_NODO em ambas as tabelas é utilizada para efetivar este relacionamento.

Esta organização do banco de dados garante independência das aplicações na atualização da base de dados. Em outras palavras, pacotes de controles que chegam ao servidor são gravados diretamente na base de dados, não importando que tipo de aplicação está enviando estes pacotes.

A estrutura de banco de dados planejada e utilizada atualmente não é a mesma estrutura planejada na primeira versão do banco de dados. A primeira versão desenvolvida tornou-se inviável, pois para cada tipo de aplicação existente deveria haver uma estrutura de banco de dados proprietária de acordo com esta aplicação, logo, para cada novo tipo de aplicação adicionada ao sistema, era necessário preparar a base de dados e inclusive modificar o programa Java que roda no servidor para suportar esta nova aplicação, já que as estruturas de banco de dados também foram criadas neste momento.

Devido a estas exigências técnicas (necessidade de refazer todo o banco de dados a cada aplicação), tornou-se inviável a utilização desta estrutura. Como solução a este problema, planejou-se a estrutura de banco de dados atual, a qual que é independente da aplicação, pois informações referentes às aplicações são sempre armazenadas na TabelaNodos e os dados referentes a estas aplicações são armazenados na TabelaDados.

Nesta Seção apresentamos e explicamos as estruturas de dados utilizadas neste trabalho. Na próxima Seção iremos descrever como o sistema desenvolvido em Java interage com o banco de dados.

7.2.3 Comunicação com o banco de dados / controlador mestre (comunicação serial)

Como já fora dito na Seção 7.1.2, o banco de dados serve de *gateway* entre a aplicação cliente e o *hardware*, tendo a responsabilidade de guardar o *status* das aplicações da residência. Também já foi visto que toda a comunicação com o controlador mestre (*hardware*)

ocorre através da porta de comunicação serial padrão RS-232, enviando e recebendo pacotes de controle.

Devido à necessidade de implementação destas funcionalidades, desenvolveu-se um *software* responsável pelo controle/atualização do banco de dados, juntamente com o controle da porta de comunicação RS-232. O *software* que roda no servidor foi desenvolvido totalmente em Java. O acesso à base de dados acontece através dos métodos da biblioteca JDBC, utilizando o driver JDBC-ODBC. Para a implementação da comunicação serial utilizando RS-232 utilizamos o pacote desenvolvido pela *Sun Microsystems* denominado de *CommAPI* definido na Seção 3.3.2.

Devido a utilização da linguagem Java, todo o desenvolvimento é orientado a objetos, sendo que todas as operações realizadas pelo *software* são executadas através da chamada de métodos destes objetos. A utilização da linguagem Java também garante a portabilidade do sistema entre diferentes plataformas. Para que o sistema seja portado para outra plataforma, a única mudança que deve ser feita no código fonte é na linha de código que faz a carga do *driver* de banco de dados. As Seções seguintes detalham os dois componentes deste *software*:

- Comunicação serial, liga o *software* ao *hardware*;
- Acesso a base de dados.

A Figura 58 ilustra o relacionamento dos métodos utilizados pela aplicação para a comunicação com o banco de dados e porta de comunicação RS-232 que serão apresentados nas Seções seguintes:

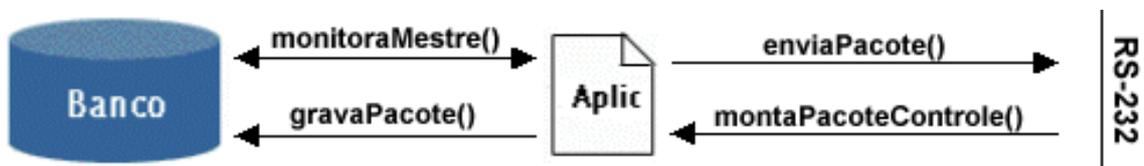


Figura 58 – Métodos de acesso a base de dados e porta de comunicação RS-232

7.2.3.1 Comunicação serial

Este módulo gerencia o envio e a recepção de pacotes de controle em paralelo. A qualquer momento o controlador mestre pode enviar pacotes, neste caso o *software* deve estar preparado para recebê-los.

Sendo assim, utilizou-se o conceito de *threads* para possibilitar a monitoração da porta serial, pois ao mesmo tempo em que a porta serial é monitorada, outras operações devem ser realizadas pelo *software*, logo, somente com a utilização de *threads* podemos tratar eventos em paralelo.

Para conseguir classificar os bytes que chegam na porta serial e montar um pacote de controle, seja ele de qual for o tipo, criou-se um método chamado de *montaPacoteControle()*, o qual

sempre recebe o byte proveniente da porta serial. Internamente a este método, mecanismos de controle e sequenciamento foram implementados para que o pacote de controle seja identificado dentro da seqüência de bytes que chegam. Atualmente classificamos os pacotes como sendo de dois tipos :

- Pacotes Pequenos – Pacotes com até 8 bytes de tamanho
- Pacotes Grandes – Pacotes maiores que 8 bytes.

Devido a existência destes dois tipos de pacotes, foram criadas duas implementações diferentes para suportá-los.

No caso de pacotes pequenos, à medida que os bytes chegam na porta serial, e o método *montaPacoteControle()* é chamado, o método gerencia a montagem do pacote. De acordo com a seqüência dos bytes, juntamente com as informações contidas nestes bytes, consegue-se interpretar e extrair informações destes (segundo a especificação do protocolo CAN) e montar a classe chamada *PacoteControle*, que possui os seguintes atributos (Seção 8.4): Classe, aplicação, nodo id, tamanho, RTR, dados.

No final do processo, quando o pacote de controle é montado, temos divididas as informações listadas acima em forma de atributos. Logo, para obtermos as informações da classe basta utilizarmos os métodos específicos para cada atributo. Por exemplo: para obtermos a Classe do pacote de controle que acaba de ser montado, devemos simplesmente chamar o método *getClasse()* e assim sucessivamente para cada atributo (*getAplicacao()*, *getNodeId()*, etc).

No caso de pacotes grandes a implementação foi realizada de maneira diferenciada, pois para pacotes pequenos, todos os bytes referentes a um pacote chegam na porta serial em seqüência. No entanto, para pacotes grandes, nem sempre isso ocorre, pois bytes de outros pacotes podem chegar misturados aos de pacote grande. Neste caso, o *software* deve implementar uma lógica que encaminhe o pacote para a “fila” correta, até que esta fila atinja o tamanho do pacote, montando assim o pacote grande. A Figura 59 representa graficamente este problema. Note que o servidor está recebendo uma sequencia de pacotes com Identificador A, que são fragmentos (sub-pacotes) de um pacote grande. Porém o quarto pacote recebido é de outra origem, possivelmente um pacote de controle comum (pacote pequeno).



Figura 59 – Problema de sequenciamento de pacotes

A lógica implementada no *software* servidor funciona da seguinte forma. Sempre que o sistema identificar que um pacote grande está para ser recebido, este pacote é adicionado em uma

fila de pacotes grandes. A partir deste momento, já possuímos uma identificação deste pacote na fila. Todas as informações que chegaram através da serial com este identificador, são gravadas em forma de atributos em sua respectiva entrada na fila. Desta forma é possível ao sistema gerenciar o recebimento dos bytes de dados para cada pacote grande. A cada pacote que chega através da serial é feita uma verificação se este faz parte de um pacote grande. Sendo um pacote grande, automaticamente é feita uma busca pelo identificador na lista de pacotes grandes para verificar se o pacote já possui entrada. Em caso positivo, recupera-se o contexto atual do pacote e adiciona-se os novos bytes de dados (nova remessa de dados) que estão chegando a partir da posição do último byte que foi gravado. Em caso negativo, o pacote grande que está sendo enviado é adicionado na lista. Se não for pacote grande o tratamento é feito como descrito anteriormente para o caso de pacotes pequenos.

A Figura 60 representa a fila de pacotes grandes. Percebe-se que com o nodo da lista com ID A estão todos os pacotes com este ID. Em cada um desses pacotes está contida até 8 bytes de dados que representam um parte dos dados enviados pelo processo de envio de pacotes grandes. O último pacote recebido (ID A e Seq4) é identificado com pertencente a lista de ID A, sendo o mesmo adicionado a esta lista. Ainda n Figura 60, podemos visualizar a existencia de recebimento de mais duas mensagens de pacotes grandes (ID B e C).

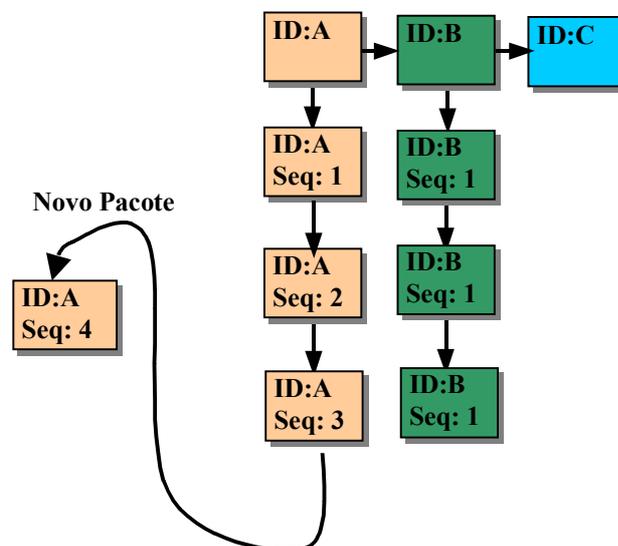


Figura 60 – Lista de recebimento de pacotes grandes

Como visto anteriormente, explicamos a recepção de pacotes de controle enviados pelo controlador mestre, agora iremos tratar do caso inverso, o sistema enviando pacotes de controle para o controlador mestre.

Durante o envio de pacotes pelo sistema, possuímos duas situações distintas: o envio de pacotes pequenos e o envio de pacotes grandes. Devido a necessidade de envio de pacotes pela

porta serial, foi criado um método denominado de *enviaPacote()*. Este método recebe os seguintes parâmetros: Classe, aplicação, nodo id, RTR, tamanho, dados.

Todos estes parâmetros são declarados ao sistema como array de bytes, pois trabalhando diretamente com este tipo de dado o processo de envio através da porta serial é simplificado pois a mesma é apta para trabalhar diretamente com o tipo byte.

Dentro da lógica do método *enviaPacote()* é feita a distinção através do parâmetro *tamanho* se o pacote que está sendo enviado é um pacote grande, neste caso maior que 8 bytes, ou um pacote pequeno. É também internamente ao método *enviaPacote()* que a montagem dos pacotes a serem enviados pela serial é realizada, de acordo com as normas do protocolo CAN.

Se o pacote for do tipo pacote pequeno, os métodos de envio de bytes pela porta serial existentes na biblioteca *CommAPI* são invocados e os bytes são transmitidos diretamente de acordo com a seqüência especificada pelo protocolo CAN.

No caso de pacotes grandes, a lógica utilizada para o envio de pacotes utilizando CAN é um tanto quanto mais complexa, pois como já foi visto na Seção 8.4, os dados devem ser fragmentados várias vezes em tantos pacotes quanto forem necessários, pois cada pacote pode transportar até no máximo 8 bytes de dados, sendo assim, se quisermos transmitir 20 bytes de dados, o envio destes dados serão divididos em três pacotes, cada um contendo 8 + 8 + 4 bytes de dados. Esta lógica é implementada pelo método *enviaPacoteMaior()*. Este método é um método privado que somente é invocado internamente ao método *enviaPacote()*, sendo assim, quando pacotes grandes devem ser enviados, o método *enviaPacote()* identifica que é um pacote grande e automaticamente invoca o método interno *enviaPacoteMaior()*, que por sua vez é responsável em montar os pacotes seguindo o protocolo CAN e posteriormente enviá-los através da porta serial.

A seguir veremos com detalhes a interface de comunicação com o banco de dados implementada no *software* existente no servidor.

7.2.3.2 Acesso a base de dados

Na questão do acesso a base de dados, temos dois eventos que automaticamente necessitam que ocorra o acesso a base de dados:

- Chegada de pacotes através da serial;
- Envio de pacote através da porta serial.

Chegada de pacotes através da serial

Devido a esta interação do sistema com a base de dados, criou-se uma classe denominada *BancodeDados*, cujo principal objetivo é prover métodos que interajam com a base de dados de acordo com as necessidades do sistema.

Quando um pacote é considerado montado, também passa a ser responsabilidade deste método avisar ao banco de dados que existe um pacote de controle disponível e que o mesmo deve ser salvo na base de dados. Como consequência da chegada de pacotes de controles, necessitamos atualizar a base de dados com as informações provenientes deste pacote. Visando alcançar este objetivo, criou-se um método na classe *BancodeDados* denominado *gravaPacote()*. Internamente a este método é realizada uma atualização sobre o registro específico referente à aplicação que teve mudança de status por algum motivo. Este registro encontra-se localizado na tabela do banco de dados denominada *TabelaNodos* que pode ser vista através da Figura 56.

Este método recebe como parâmetro as seguintes informações provenientes do pacote de controle que acaba de ser montado :

- *classe* – classe da aplicação;
- *aplicação* – tipo da aplicação;
- *nodo id* – nodo id da aplicação;
- *tamanho* – tamanho em número de bytes da área de dados;
- *dados* – array de bytes contendo os dados;
- *dirty_mestre* – indica que a atualização deve ser propagada ao controlador mestre (houve atualização do status da residência através do computador cliente);
- *dirty_servidor* - indica que a atualização deve ser propagada ao computador cliente (houve atualização do status da residência através do controlador mestre).

Para o registro específico, são atualizadas informações tais como o tamanho atual do pacote, os dados referentes a este pacote e se a atualização deve ser propagada ao controlador mestre ou ao computador cliente (interface Web).

É importante salientarmos que os dados referentes a uma aplicação específica estão armazenados na tabela *TabelaDados*, sendo assim, a atualização dos dados ocorre sobre esta tabela. Para esta atualização põem-se em prática o relacionamento que existe entre as tabelas *TabelaNodos* e *TabelaDados*, pois a partir do *ID_NODO* obtido a partir do registro da aplicação (*TabelaNodos*), conseguimos encontrar todos os dados referentes àquela aplicação (*TabelaDados*).

No início do processo de atualização das informações referentes a uma aplicação, a primeira operação que é realizada é modificar o status da coluna *atualizacao* indicando que este registro está sendo atualizado, após isso, as demais operações de banco de dados são realizadas. Ao final do processo de atualização, quando todas as operações de atualizações referentes a esta aplicação já estão concluídas, o status da coluna *atualizacao* volta ao normal indicando que este registro não está mais sendo atualizado.

Envio de pacote através da porta serial

O envio de pacotes através da porta serial para o controlador mestre deve ocorrer sempre que a coluna *dirty_mestre* da tabela *TabelaNodos* estiver com o seu valor booleano em *true*.

Na classe *BancodeDados* criou-se o método *monitoraMestre()*, sua função principal é procurar no banco de dados por aplicações que estejam com o status desatualizado em relação ao controlador mestre (coluna *dirty_mestre* com valor igual a *true*) e enviar estas informações (classe, aplicação, *nodoid*, dados, etc) através da porta serial utilizando o método descrito anteriormente chamado *enviaPacote()*.

Para que constantemente o status do banco de dados em relação ao controlador mestre seja monitorado, o método *monitoraMestre()* deve ser chamado de tempos em tempos. Para que isto seja possível, novamente utilizou-se o conceito de *threads*, sendo assim, criou-se uma *thread* que é executada em segundo plano, e que em um intervalo definido de tempo invoca o método *monitoraMestre()*, que por sua vez envia os pacotes de controle ao controlador mestre.

7.3 Arquitetura do cliente

Neste Capítulo descreveremos o processo de funcionamento da interface Web no cliente. Essa interface possui quatro funcionalidades básicas:

- Autenticação de usuários no sistema;
- Atualização da interface Web;
- Atualização do status da residência;
- Manutenção dos usuários do sistema.

7.3.1 Autenticação de usuários no sistema

Conforme Figura 61, o acesso ao sistema ocorre mediante a autenticação do usuário através de uma tela onde deve-se obrigatoriamente informar um usuário e senha, seguindo o mesmo conceito utilizado em redes corporativas.

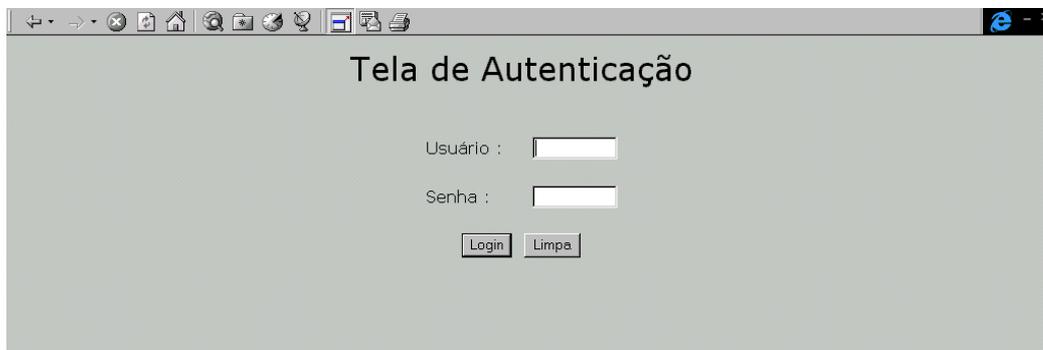


Figura 61 – Tela de Autenticação

O controle da sessão do usuário no navegador Web, é realizado com a utilização de *cookies* (*link* [35]) de sessão. *Cookies* são informações que um site Web envia ao seu browser, permitindo que o site Web possa ler estas informações quando desejar.

Os *cookies* são classificados em dois tipos:

- Cookies de sessão – ficam armazenados na memória do computador e duram até que a sessão do browser seja fechada.
- Cookies persistentes – ficam armazenados no disco rígido do computador, por isso são denominados persistentes.

No processo de autenticação no sistema, o usuário informa ao sistema um usuário e senha. Neste momento o sistema criptografa com *md5* a senha digitada e acessa a base de dados procurando se existe realmente o usuário. Se o usuário existe, compara a senha digitada que já está criptografada com a senha contida na base de dados. Se forem iguais, o usuário tem permissão de acessar a residência caso a coluna *acesso_permitido* possua o valor 1 (verdadeiro).

Quando ocorre sucesso na autenticação do usuário, automaticamente é criado um *cookie* de sessão e o usuário é encaminhado para a interface de controle da residência. Se por algum motivo ocorrer algum problema na autenticação do usuário (ex.: senha incorreta), o *cookie* não é criado e o usuário não terá acesso ao sistema.

O encerramento da sessão do usuário pode ocorrer de duas formas. A primeira delas é fechando todas as instâncias do navegador Web que estiverem ativas, pois o *cookie* de sessão permanece ativo enquanto o navegador Web estiver ativo, neste caso, fechando os navegadores garantimos o término da sessão no sistema. Outra maneira possível é através da opção Logout, que encontra-se na interface interna do sistema.

É importante salientar que caso nenhuma das duas alternativas acima seja tomada, a sessão continuará ativa. Neste caso pode-se voltar normalmente a interface de controle da residência sem a necessidade de autenticação.

7.3.2 Atualização da interface Web

A atualização de interface Web, que ocorre quando há modificações no status da residência. A interface Web possui os seguintes frames:

- *Frame 1* – Bibliotecas Javascript;
- *Frame 2* – Programa PHP de leitura da base de dados;
- *Frame 3* – Programa PHP para atualização da base de dados;
- *Frame 4* – Menu de opções;
- *Frame 5* – Aplicações encontradas na residência.

Os *frames* 1, 2 e 3 são invisíveis para o usuário, pois se tratam de *frames* responsáveis unicamente pelo processamento de informações que ocorrem em segundo plano na interface cliente.

No *frame 2*, existe um programa desenvolvido em PHP cuja principal função é ler a base de dados contida no servidor e verificar quais dos registros da base de dados estão desatualizados em relação ao computador cliente (coluna *dirty_servidor*), com base nestes registros, o programa PHP gera comandos Javascripts que atualizam a interface Web. Como se fosse uma *thread*, este programa é invocado de tempos em tempos garantindo a atualização da interface Web. Com a utilização de comandos Javascript para atualização da interface, juntamente associada com a monitoração constante da base de dados sem a necessidade de interação do usuário, a atualização da interface ocorre de maneira automática sem que o usuário precise clicar em um botão de “Atualizar” a todo instante. Esta é uma importante vantagem do sistema, uma vez que o usuário é automaticamente informado de tudo que é monitorado na residência, possibilitando, inclusive, o recebimento de mensagens de alarme na interface.

O programa PHP acessa o banco de dados através de um conector de banco de dados fornecido pela própria linguagem, no caso da migração do sistema para outras plataformas, esta linha poderá ser modificada de acordo com o banco de dados existente na nova plataforma.

O programa PHP, gera comandos Javascripts de atualização e também é capaz de acessar bibliotecas de funções também escritas em Javascript.

Devido a dinâmica utilizada para a atualização da interface cliente, criou-se uma biblioteca de funções Javascript. Esta biblioteca está localizada no *frame 1* e atualmente agrupa um número determinado de funções úteis para as aplicações já desenvolvidas. A medida que novos controles da

residências são implementados na interface, esta biblioteca Javascript deve incluir as novas funcionalidades que serão necessárias.

Com a crescente utilização do navegador Web Internet Explorer, a *homepage* da residência foi construída baseando-se na versão 4.0 deste navegador Web. Devido aos avançados recursos de Javascript utilizados para permitir a atualização da *homepage* sem a interação do usuário, é possível que nem todas as funcionalidades empregadas sejam suportadas por outros navegadores Web.

7.3.3 Atualização do status da residência

Toda a atualização do status da residência ocorre antes sobre o banco de dados, sendo assim, a interface cliente comunica-se diretamente com o banco de dados de acordo com a interação do usuário com a interface cliente. Após este passo, o programa desenvolvido em Java responsabiliza-se em repassar as mudanças da base de dados através da porta serial.

Neste primeiro momento, construiu-se três aplicações diferentes para a residência :

- Lâmpadas;
- *Dimmers*;
- Captura de vídeo.

Na aplicação de lâmpadas, controla-se determinadas lâmpadas da residência.

Dimmers são responsáveis em mensurar a grandeza de determinadas aplicações. Por exemplo: um termômetro para medir a temperatura da cozinha, lâmpadas com controle de intensidade, temperatura da geladeira, etc.

Captura de vídeo representa a imagem de determinada peça da residência que é capturada por uma câmera de vídeo e enviada ao computador cliente.

Como vemos na Figura 62, dependendo da aplicação, a interface oferece um conjunto de informações ao usuário. As informações que a interface pode exibir ao usuário são as seguintes :

- Descrição da aplicação – descreve qual é a aplicação que está sendo controlado;
- Ação – indica a ação que pode ser comandada sobre uma determinada aplicação;
- Localização – localização da aplicação dentro da residência;
- Estado atual – corresponde ao estado atual da aplicação dentro da residência;
- Data e hora – indica a data e o horário referente ao status da aplicação.

A interação que acontece entre o usuário e a residência ocorre através da coluna **ação**, permitindo ao usuário interagir diretamente com uma determinada aplicação. No caso de lâmpadas, podendo ligar e desligá-las. Para *dimmers*, podemos atualizar o seu valor de acordo com o tipo de *dimmer*.

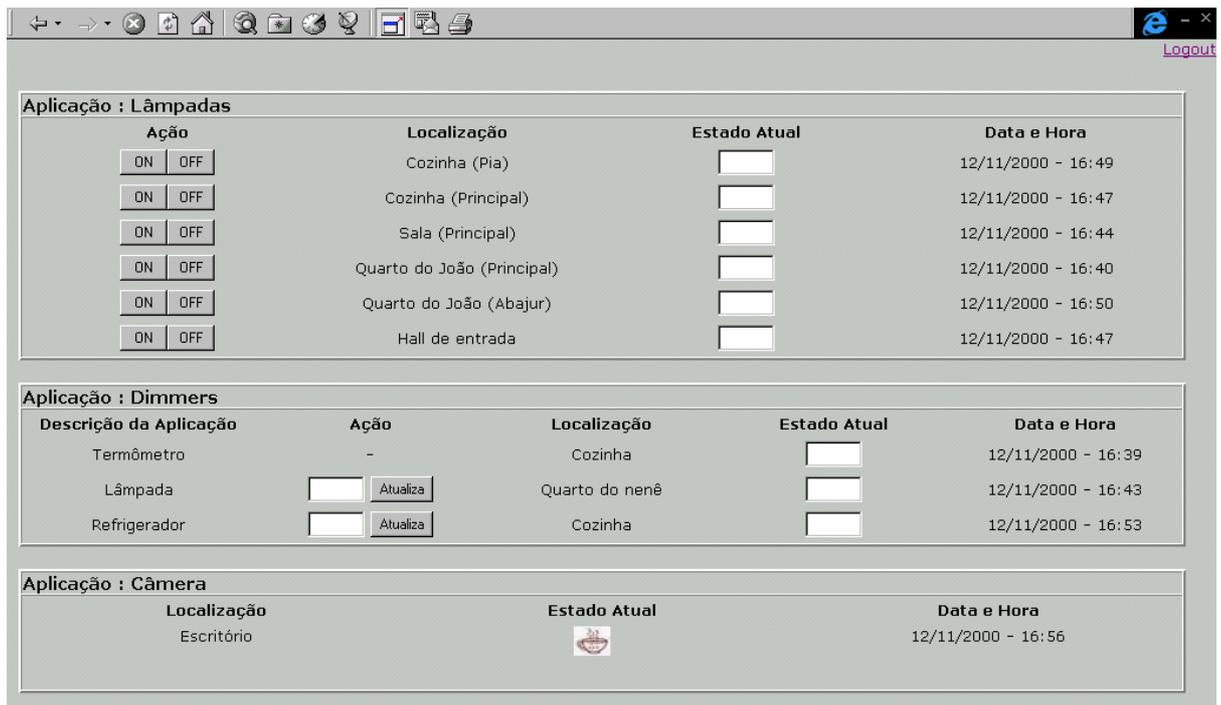


Figura 62 - Interface Cliente

No momento que o usuário realiza algum comando, devemos atualizar a base de dados. Para que isso seja possível, quando o comando é realizado, o novo status da aplicação é capturado através de comandos Javascript e é enviado para processamento no *frame 3*, cuja responsabilidade é atualizar a base de dados, indicando que a aplicação está com seu status desatualizado em relação ao controlador mestre (*dirty_mestre*).

7.3.4 Manutenção dos usuários do sistema

Visando permitir que novos usuários sejam adicionados ao sistema, possibilitando o acesso dos mesmos ao controle da residência, criou-se uma interface de manutenção de usuários.

A disponibilização desta interface ocorre somente para o(s) usuário(s) que possuir(em) permissão para executar tal operação. Isto só é possível através da existência da coluna *acesso_manutencao* da tabela *Usuarios*.

Na interface de manutenção dos usuários, estão disponibilizadas as opções de: inclusão, alteração, exclusão

Na Figura 63 apresentamos a interface de administração de usuários do sistema.

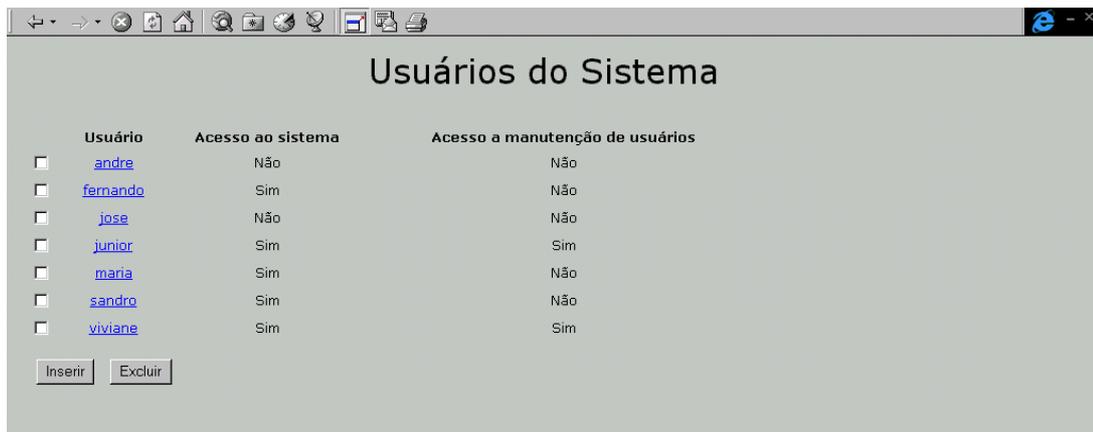


Figura 63 – Interface de Administração

7.3.4.1 Inclusão

Permite ao usuário que possui permissão de manutenção (visando facilitar a compreensão, chamaremos este usuário de administrador) incluir novos usuários ao sistema.

Esta opção está disponível através do botão inclusão. Como podemos ver na Figura 64, deve-se informar o nome, e senha do novo usuário, sendo que a senha deve ser informada duas vezes.

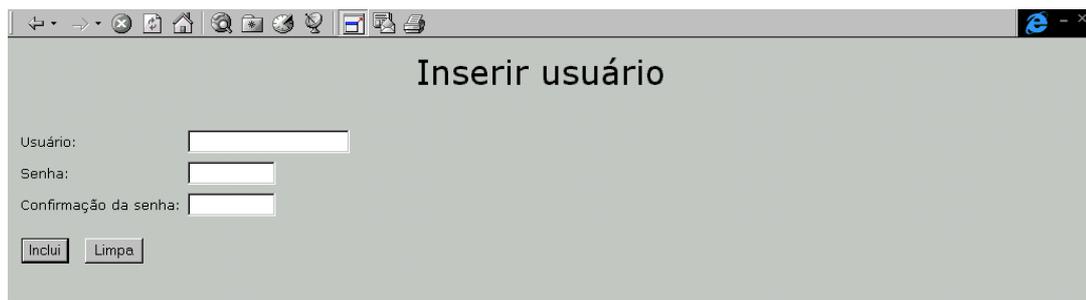


Figura 64 – Inserindo usuários

Por configuração *default*, no momento da inclusão de um usuário, automaticamente o mesmo possui permissão de acesso ao sistema e não possui permissão de acesso a interface de manutenção. Para que estas opções sejam alteradas, deve-se executar a operação de alteração.

7.3.4.2 Alteração

A partir desta opção o administrador pode alterar os dados referentes a um usuário cadastrado no sistema.

Para alterar informações de um usuário, basta clicar sobre o seu nome para que a tela de alteração seja exibida conforme Figura 65.

Figura 65 – Alterando usuários

Na alteração, o administrador pode modificar informações como por exemplo o nome do usuário, sua senha, permissão de acesso ao sistema ou até mesmo torná-lo um administrador do sistema.

7.3.4.3 Exclusão

Permite ao administrador excluir usuários permanentemente da base de dados.

Para isso basta que o administrador selecione os usuários que deseja excluir e clique no botão excluir como pode ser visto na Figura 66.

| Usuário | Acesso ao sistema | Acesso a manutenção de usuários |
|--|-------------------|---------------------------------|
| <input type="checkbox"/> andre | Não | Não |
| <input type="checkbox"/> fernando | Sim | Não |
| <input checked="" type="checkbox"/> jose | Não | Não |
| <input type="checkbox"/> junior | Sim | Sim |
| <input checked="" type="checkbox"/> maria | Sim | Não |
| <input checked="" type="checkbox"/> sandro | Sim | Não |
| <input type="checkbox"/> viviane | Sim | Sim |

Figura 66 – Excluindo usuários

7.4 Arquitetura do *Debugger*

O *debugger* na implementação de *software* simula o comportamento do *hardware*, que comunica-se diretamente com o computador servidor através da porta de comunicação serial RS-232C. O *debugger* tem por finalidade exibir na tela todos os pacotes de controle que partem do computador servidor. O *software* de *debugger* possibilita também que pacotes de controles sejam enviados para o computador servidor.

Como característica do envio de pacotes, existe a possibilidade de envio de pacotes pequenos (menores de 8 bytes) utilizado normalmente para o controle de lâmpadas e *dimmers*. Outra possibilidade é o envio de pacotes grandes (maiores que 8 bytes).

Aplicações que utilizam o envio de pacotes grandes normalmente trabalham diretamente com arquivos, neste caso, por exemplo, enviando a imagem capturada por uma câmera de vídeo.

Para pacotes pequenos, o usuário informa diretamente na interface do *debugger* as informações sobre este pacote (classe, aplicação, nodoid, dados, etc). Também é possível que pacotes grandes sejam enviados utilizando o mesmo processo, no entanto, visando simular o envio de pacotes com tamanhos maiores, foi adicionado ao debugger a capacidade de enviar arquivos, desta forma, podemos facilmente enviar uma imagem ao computador servidor e visualizarmos sua exibição na interface Web.

Na Figura 67 apresentamos a interface desenvolvida para o *software* de *debugger*. Podemos visualizar na interface campos como por exemplo classe, aplicação, nodo ID, RTR. Estes campos deverão ser preenchidos, pois para a montagem dos pacotes, essas informações são essenciais.

Os dados propriamente ditos podem ser obtidos de duas maneiras, a primeira é preenchendo o campo denominado de dados (obrigatoriamente deve ser preenchido o campo de DLC que indica a número de bytes de dados), a segunda maneira é utilizando um arquivo, desta forma o arquivo representa os dados que devem ser enviados. Para isso deve-se clicar sobre o botão “Arquivo” e selecionar o arquivo desejado. Após informado os dados, deve-se selecionar a partir de onde os dados serão obtidos (campo de dados ou arquivo), para isso existe uma opção na interface do *debugger*.

No lado direito do *debugger* podemos visualizar uma lista descrita como status da porta serial. Neste lista que serão exibidos todos os pacotes que chegam até o *software* de *debugger*, desta forma podemos ver o que o computador servidor estaria enviando ao controlador mestre.

O *software* de *debugger* foi desenvolvido em Java e devido as características de linguagens de programação orientadas a objetos, ocorre a reutilização de bibliotecas de classes, desta forma, o *software* de *debugger* neste caso seria apenas um interface visual mais aprimorada sobre toda a biblioteca de classes que fora desenvolvida para o *software* que roda no servidor. Em outras palavras, reutilizamos todas as classes responsáveis pelo envio, recepção, codificação e decodificação de pacotes, ficando sem utilização apenas as classes que acessam o banco de dados, pois no contexto que se encontra o *software* de *debugger* isso não é necessário.

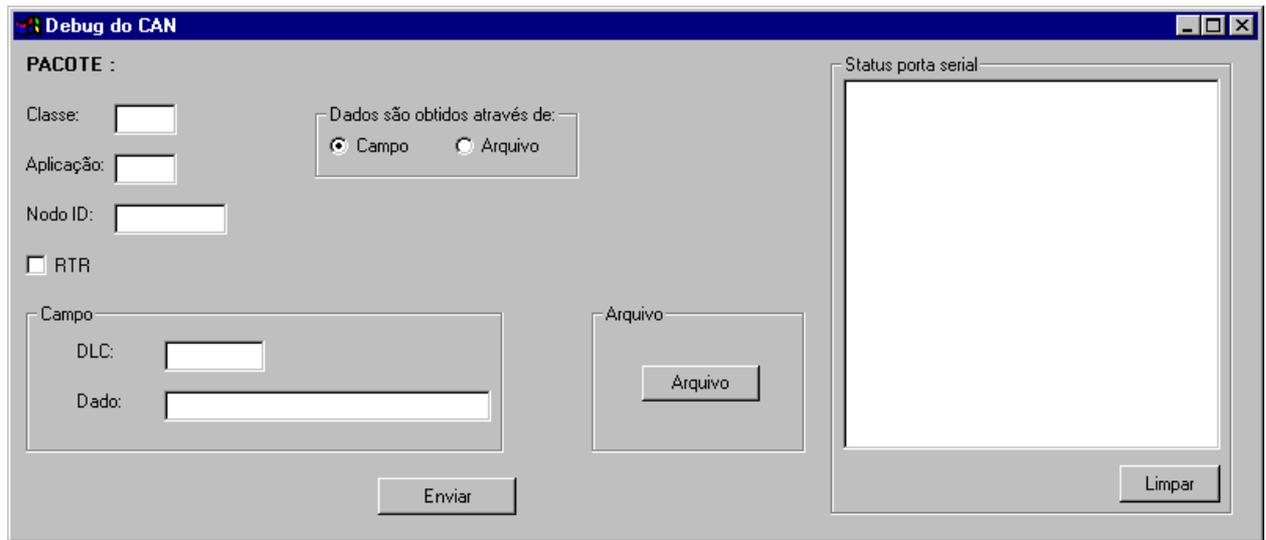


Figura 67 – Debugger

8 Implementação de *Hardware*

Esta Seção tem por objetivo detalhar a implementação do sistema de *hardware* do projeto.

Na Seção 8.1 descrevemos a implementação do HurriCANE. Na Seção 8.2 discute-se a implementação dos nodos de aplicação (escravos) desenvolvidos neste projeto. A Seção 8.3 apresentamos a arquitetura e desenvolvimento da placa de aquisição de dados (nodo mestre). Na Seção 8.4 é detalhado as alterações lógicas proposta ao protocolo CAN de modo que o mesmo suportasse dados maiores que 8 bytes entre outros serviços adicionais. Finalmente, na Seção 8.5, é apresentado o protocolo que é responsável pela comunicação entre *software* e *hardware*.

8.1 Descrição dos Módulos HurriCANE

Pelo fato do protocolo CAN ser razoavelmente complexo de ser implementado e, principalmente, de dispormos de somente um semestre para desenvolvê-lo, optamos por basear o trabalho em um código já existente, propondo algumas modificações e acrescentando certas funcionalidades específicas para aplicação em domótica (Seção 8.4). Outro fato que motivou o uso de um código pronto e testado amplamente é que a fase de teste e ajustes do sistema necessita de muito tempo.

HurriCANE é um módulo descrito em VHDL que implementa o protocolo CAN. Esse módulo foi desenvolvido na ESA¹⁹ por Luca Stagnaro, onde o código pode ser encontrado em [38].

Durante essa Seção descreveremos a funcionalidade de cada módulo do HurriCANE. Na Seção 8.1.1 apresentamos uma breve descrição dos módulos que compõe o HurriCANE, dando uma visão da interligação dos mesmos. Da Seção 8.1.2 até a 8.1.11 é descrito em maiores detalhes a função de cada módulo CAN. Em cada uma dessas Seções são apresentadas as descrições da *port list* e as áreas de ocupação²⁰ no FPGA **xvc300-5-bg352 [28]** da família Virtex 300 da Xilinx.

8.1.1 Descrição Geral dos Módulos

A organização hierárquica dos módulos HurriCANE pode ser visualizado na Figura 68. Cada cubo representa uma entidade, onde cada entidade possui sua função própria.

¹⁹ *European Space Agency* (Agência Espacial Européia)

²⁰ Esses resultados foram calculado sem esforço máximo de síntese.

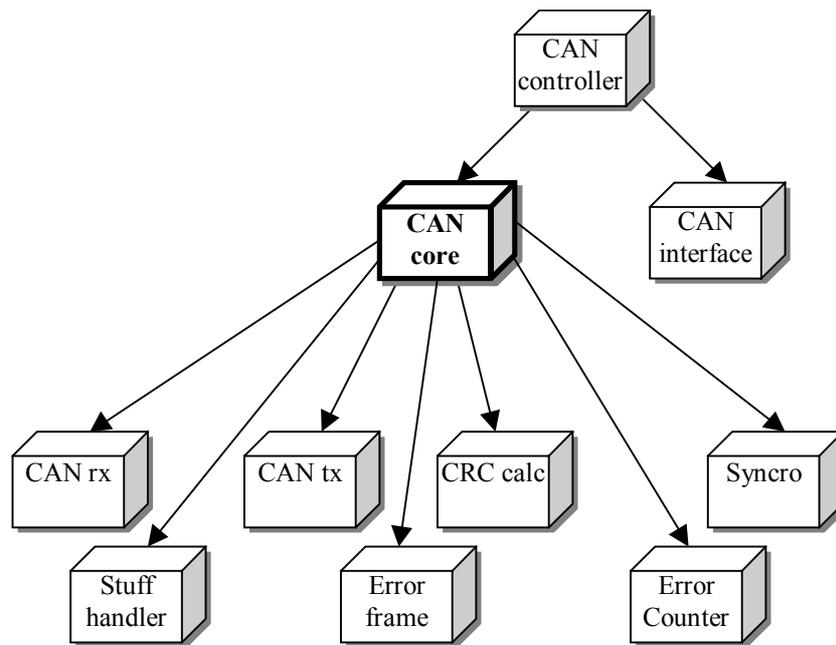


Figura 68- Disposição hierárquica dos módulos

A seguir é feita uma breve descrição de cada módulo:

- Módulo Controlador – Une duas entidades *CAN core* e *CAN interface*;
- Módulo de Interface – Provê ao usuário do módulo uma interface padrão entre dispositivos, baseando-se em comandos de escrita (transmissão) e leitura (recepção). Esse módulo é especialmente importante caso use-se o HurriCANE com um processador comercial, pois essa interface facilitará o controle e o tráfego de dados entre a aplicação do usuário e o componente CAN;
- Módulo CAN Core (*CAN Core*) – Esse é o principal módulo do protocolo. Ele interliga todos os módulos que estão abaixo na hierarquia. Pelo fato de não ter sido proposto a este projeto o desenvolvimento da interface entre CAN e um processador, *usaremos somente este módulo*;
- Módulo de Recebimento (*CAN Rx*) – Responsável por receber os pacotes destinados a esse módulo;
- Módulo de Transmissão (*CAN Tx*) – Responsável por transmitir ao meio os pacotes deste módulo;
- Módulo de Cálculo de CRC (*CRC Calc*) – Gerador de CRC;

- Módulo de Sincronismo (*Syncro*) – É o módulo mais importante da descrição. Ele é responsável pela geração de sinais de controle aos outros módulos e de fazer a amostragem do sinal;
- Módulo de Stuffing (*Stuff Handler*) – Gerador de *bit stuffing*;
- Módulo de Controle de Erros (*Error Frame*) – Responsável pela geração do pacote de sinalização de erros e pelo controle do módulo de contadores de erros. Esse módulo recebe dos módulos restantes suas sinalizações de erro;
- Módulo de Contadores de Erros (*Error Counter*) – Controla os contadores de erro de transmissão e de recepção. Com base no valor desses contadores o usuário sabe o estado em que o nodo CAN está (ativo, passivo ou *bus off*).

O diagrama de blocos de interação do *CAN Controller* é apresentado na Figura 69.

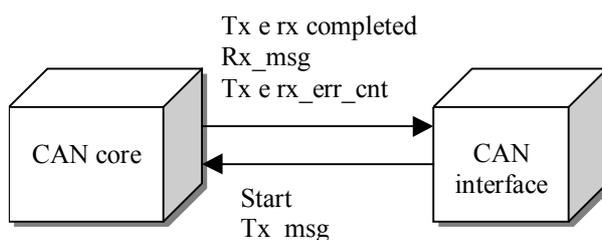


Figura 69 - Interação entre os bloco do *CAN Controller*

Abaixo segue a descrição dos sinais:

- Tx e Rx_completed sinalizam à interface o fim de uma transmissão ou uma recepção;
- Rx_msg é a mensagem recebida;
- Tx e Rx_err_cnt são os contadores de erro de transmissão e recepção;
- Tx_msg é a mensagem a ser transmitida;
- Start é um sinal que indica ao CAN Core que a transmissão deve ser iniciada.

O diagrama de blocos do *CAN Core* é apresentado na Figura 70. Por questão de organização, somente os principais sinais estão representados neste diagrama.

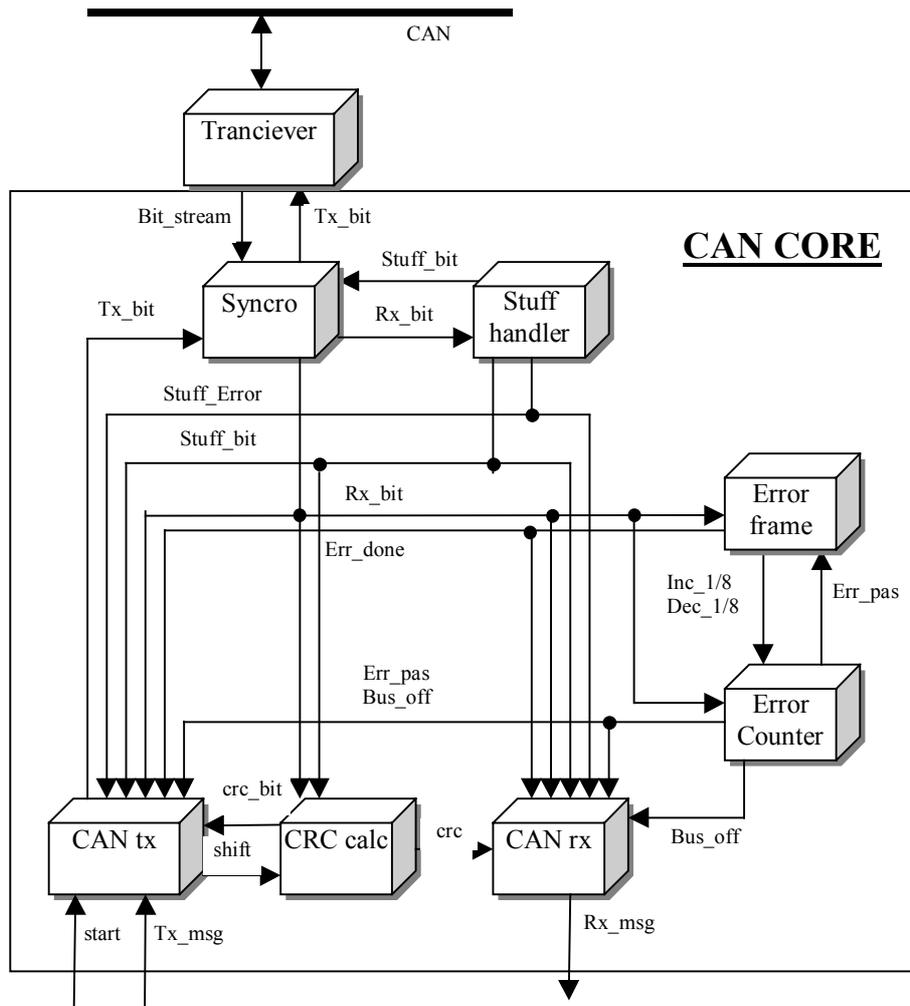


Figura 70 – Interação entre os blocos do CAN Core

Abaixo segue a descrição dos sinais:

- Syncro – Recebe sinal do meio (bit_stream) e distribui ao restante dos módulos (rx_bit). Também recebe o sinal (tx_bit) do CAN tx e transmite para o meio;
- Stuff Handler – Recebe sinal lido e indica se houve violação (stuff_bit e stuff_error);
- Error Counter – Através do valor dos contadores de erro é sinalizado estado passivo ou bus_off (err_passivo e bus_off) ao restante dos módulos. Esse módulo recebe do *error frame* os comandos para mudar os contadores ;
- CRC Calc – Na recepção, esse módulo recebe o dado lido e gera o CRC para o módulo CAN Rx. Na transmissão, esse módulo gera o CRC e transmite-o para CAN Tx serialmente (crc_bit);

- CAN Tx – Depois de recebido da interface CAN os dados que devem ser enviados, este módulo envia o pacote bit-a-bit ao sync. Sinais de controle do *Stuff handler* e Error Frame são usados para decisão de estado interno do módulo;
- CAN Rx – Este módulo recebe do módulo sync o sinal lido do barramento e, assim, vai montando o pacote para enviar os dados à interface CAN. Assim como CAN Tx, sinais de controle de Stuff handler e Error Frame são usados para decisão de estado interno do módulo;

8.1.2 Módulo Controlador

Este é o principal módulo quando se pretende usar o *core* CAN com um processador, pois o mesmo possui uma interface genérica para processadores que facilita a troca de informações. Neste projeto, esse módulo não está sendo utilizado pois não foi usado um processador.

8.1.2.1 Descrição da Entidade

| Sinal | Tipo ²¹ | Descrição | Origem/Destino |
|----------------|--------------------|--|----------------|
| Data | IO Vetor 8 bits | É a entrada e saída do dado que é recebido e transmitido pelo módulo | Can_Interface |
| Cs | I Bit | chip select | Can_Interface |
| Rd | I Bit | Ativa o módulo para leitura (recepção) | Can_Interface |
| Wr | I Bit | Ativa o módulo para escrita (transmissão) | Can_Interface |
| Add | I Vetor 6 bits | endereço do registrador interno que será lido/escrito | Can_Interface |
| Bit_stre am | I Bit | Interface de entrada com o barramento CAN | Can_Core |
| Reset | I Bit | reset do módulo CAN | Can_Core |
| Res2 | I Bit | reset do módulo de interface | Can_Interface |
| Tx | O Bit | Interface de saída com o barramento CAN | Can_Core |
| Err_bit | O Bit | Sinal de saída na transmissão de pacote de erro | Can_Core |
| Ack_bit | O Bit | sinalização de reconhecimento do pacote | Can_Core |
| Enable | O Bit | Instante em que os módulos internos são ativados | Can_Core |

Tabela 11 - Descrição da *port list* do *CAN Controller*

8.1.2.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design '**can_controller**'
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

²¹Codificação de direção: I – input, O – output e IO - inout

```

Command Line   : map -p xc3v300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date    : Fri Aug 18 18:24:18 2000

```

Design Summary

```

-----
Number of errors:      0
Number of warnings:   13
Number of Slices22:           749 out of 3,072 24%
  Slice Flip Flops:    477
  Slice Latches:       99
  4 input LUTs:       1,065 (2 used as a route-thru)
Number of Slices containing
  unrelated logic:    0 out of 749 0%
Number of bonded IOBs: 24 out of 260 9%
Number of GCLKs:      4 out of 4 100%
Number of GCLKIOBs:  1 out of 4 25%
Total equivalent gate count for design: 10,950
Additional JTAG gate count for IOBs: 1,200

```

8.1.3 Módulo de Interface

É o módulo responsável por particionar os pacotes CAN em bytes e pela sinalização à CPU.

8.1.3.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|---------------|------------------|--|----------------|
| Add_i | I vetor 6 bits | endereço do registrador interno que será ativado | Can_Controller |
| Data | IO vetor 8 bits | Interface de entrada e saída de dados | Can_Controller |
| Write | I bit | | Can_Controller |
| Read | I bit | | Can_Controller |
| Chip select | I bit | | Can_Controller |
| Tx_complete_d | I bit | Sinaliza finalização de uma transmissão | Can_Core |
| Rx_complete_d | I bit | Sinaliza finalização de uma recepção | Can_Core |
| Start | I bit | Comando para iniciar uma transmissão | Can_Core |
| Msg | O vetor 101 bits | Mensagem a ser transmitida | Can_Core |
| Rx_msg | I vetor 101 bits | Mensagem a ser recebida | Can_Core |
| Rx_err_cnt | O vetor 8 bits | Contador de erro de recepção | Can_Core |
| Tx_err_cnt | O vetor 8 bits | Contador de erro de transmissão | Can_Core |

Tabela 12 – Descrição da *port list* do CAN Interface

8.1.3.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design 'interface'
 Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

²² Slices da família de dispositivos Virtex é o equivalente a 2 CLBs da família XC4000.

```

-----
Command Line   : map -p xc3v300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date    : Fri Aug 18 16:33:43 2000

```

Design Summary

```

-----
Number of errors:      0
Number of warnings:   12
Number of Slices:      212 out of 3,072    6%
  Slice Flip Flops:    7
  Slice Latches:       99
  4 input LUTs:        382
Number of Slices containing
  unrelated logic:     0 out of 212    0%
Number of bonded IOBs: 239 out of 260    91%
Number of GCLKs:       4 out of 4    100%
Total equivalent gate count for design: 2,867
Additional JTAG gate count for IOBs: 11,472

```

8.1.4 Módulo CAN Core

É o principal módulo, pois a partir dele está descrito o protocolo CAN.

8.1.4.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|-------------|---------------------|--|----------------|
| Bit_stream | I Bit | Interface de entrada com barramento CAN | CAN_Controller |
| Remote frm | I Bit | Sem utilidade | |
| Reset | I Bit | | CAN_Controller |
| Clock | I Bit | | CAN_Controller |
| Tx_msg | I vetor 101 bits | Mensagem a ser transmitida | CAN_Interface |
| Start | I Bit | Comando para iniciar transmissão | CAN_Interface |
| Sample | O Bit | Sem utilidade | |
| Bus off | O Bit | Sem utilidade | |
| Tx_bit | O Bit | Interface de saída com barramento CAN | CAN_Controller |
| Rx_complete | O Bit | Sinaliza fim de recepção | CAN_Interface |
| Tx_complete | O Bit | Sinaliza fim de transmissão | CAN_Interface |
| Rx_msg | O vetor 101 bits | Mensagem recebida | CAN_Interface |
| Err passive | O Bit | Sem utilidade | |
| Rx err cnt | O Bit | Contador de erro de recepção | CAN_Controller |
| Tx err cnt | O Bit | Contador de erro de transmissão | CAN_Controller |
| Err_bit | O Bit | Sinal de saída na transmissão de pacote de erro | CAN_Controller |
| Ack_bit | O Bit | sinalização de reconhecimento do pacote | CAN_Controller |
| Enable | O Bit | Instante em que os módulos internos são ativados | CAN_Controller |

Tabela 13 - Descrição da *port list* do CAN Core

8.1.4.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design '**can_core**'
 Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```

Command Line   : map -p xc3v300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date   : Fri Aug 18 18:14:06 2000

```

Design Summary

```

Number of errors:      0
Number of warnings:   1
Number of Slices:      603 out of 3,072 19%
  Slice Flip Flops:    476
  4 input LUTs:       802 (2 used as a route-thru)
Number of Slices containing
  unrelated logic:    0 out of 603 0%
Number of bonded IOBs: 233 out of 260 89%
Number of GCLKs:      1 out of 4 25%
Number of GCLKIOBs:  1 out of 4 25%
Total equivalent gate count for design: 8,845
Additional JTAG gate count for IOBs: 11,232

```

8.1.5 Módulo de Recebimento

8.1.5.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|------------------|---------------------|--|-----------------------|
| Clock | I Bit | | Can Core |
| Enable | I Bit | | Sync |
| Rx Bit | I Bit | Bit lido do barramento | Sync |
| Reset | I Bit | | Sync |
| Error_Frame_Done | I Bit | Indica fim do pacote de sinalização de erro | Error_Frame |
| Tx Busy | I Bit | CAN Tx está transmitindo | CAN Tx |
| Stuff_Error | I Bit | Sinaliza uma violação da regra de stuff | Stuff_Handler |
| Stuff_Bit | I Bit | Indica a necessidade de um bit de stuff | Stuff_Handler |
| Rx_Completed | O Bit | Sinaliza fim de recepção | Sync, err_cnt, can_tx |
| Busy | O Bit | Indica que uma recepção está em andamento | Sync, can_tx, |
| Acknowledge_N | O Bit | Indica um recebimento de ack (ativo em 0) | sync |
| Msg | O vetor 101 Bits | Mensagem recebida | Can_core |
| Signal Error | O Bit | Sinaliza erro de recepção | Error frame |
| Crc | O vetor 15 Bits | CRC da mensagem recebida | Crc_Calc |
| Bus_Off | I Bit | Indica que o nodo está em bus off | Error_Counter |
| Crc_Stop | O Bit | Indica os momentos onde não se calcula crc | Crc_Calc |
| Crc_Reset | O Bit | Reseta CRC (fim da recepção) | Crc_Calc |
| Stuff_Disabled | O Bit | Indica os momentos onde não se calcula stuff | Stuff_handler |
| Msg_State | O MsgStateType | Indica o estado atual da recepção | Can_tx |
| Rtr_Bit | O Bit | Indica se é um <i>remote frame request</i> | Can_tx |

Tabela 14 - Descrição da *port list* do CAN Rx

8.1.5.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design 'can_rx'
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```
-----
Command Line   : map -p xc3v300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date    : Fri Aug 18 17:21:42 2000
```

Design Summary

```
-----
Number of errors:      0
Number of warnings:   1
Number of Slices:      264 out of 3,072    8%
  Slice Flip Flops:    258
  4 input LUTs:        202
Number of Slices containing
  unrelated logic:     0 out of 264    0%
Number of bonded IOBs: 148 out of 260  56%
Number of GCLKs:      1 out of 4    25%
Number of GCLKIOBs:   1 out of 4    25%
Total equivalent gate count for design: 3,276
Additional JTAG gate count for IOBs: 7,152
```

8.1.6 Módulo de Transmissão

8.1.6.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|----------------|----------------|---|-----------------|
| crc_outbit | I Bit | Crc bit-a-bit | Crc Calc |
| reset | I Bit | | Sync |
| clock | I Bit | | Can Core |
| enable | I Bit | | Sync |
| err_passive | I Bit | Indica que o nodo está em erro passivo | Error_Counter |
| bus_off | I Bit | Indica que o nodo está em <i>bus off</i> | Error_Counter |
| start | I Bit | Comando para iniciar transmissão | Can_Core |
| rx_busy | I Bit | Sinaliza que uma recepção está em andamento | CAN Rx |
| rx_bit | I Bit | Bit recebido | Sync |
| err_frame_done | I Bit | Sinaliza fim de um pacote de erro | Error_Frame |
| stuff_bit | I Bit | Indica a necessidade de um bit de stuff | Stuff_Handler |
| stuff_error | I Bit | Indica violação de regra de stuff | Stuff_Handler |
| msg | I CANMsg | Mensagem a ser transmitida | Can Core |
| rtr_bit_rcved | I Bit | Indica que deve-se responder à requisição | CAN_Rx |
| msg_rcved | I CANMsg | Mensagem recém recebida | CAN Rx |
| rx_completed | I Bit | Sinaliza fim de uma recepção | CAN Rx |
| remote_frm | I Bit | Indica a transmissão de um rtr | Can_Core |
| msg_state | I MsgStateType | Estado do módulo de recepção | CAN_Rx |
| tx_bit | O Bit | Bit a ser transmitido | Sync |
| tx_completed | O Bit | Sinaliza fim de uma | Can rx,err cnt, |

| | | transmissão | |
|------------------|-------|--|---------------------|
| signal_error | O Bit | Indica a existência de erro na transmissão | Error_frame |
| busy | O Bit | Transmissão em andamento | Can_rx, crc_c, sync |
| Crc shift out | O Bit | Requisição de crc | Crc Calc |
| tx_err_cnt_noupd | O Bit | | |

Tabela 15 - Descrição da *port list* do CAN Tx

8.1.6.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design '**CAN_TX**'
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```
-----
Command Line   : map -p xc3v300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date    : Fri Aug 18 17:29:56 2000
```

Design Summary

```
-----
Number of errors:      0
Number of warnings:   1
Number of Slices:      99 out of 3,072    3%
  Slice Flip Flops:    113
  4 input LUTs:        188
Number of Slices containing
  unrelated logic:      0 out of 99    0%
Number of bonded IOBs: 148 out of 260  56%
Number of GCLKs:        1 out of 4    25%
Number of GCLKIOBs:     1 out of 4    25%
Total equivalent gate count for design: 2,032
Additional JTAG gate count for IOBs: 7,152
```

8.1.7 Módulo de Cálculo de CRC

Executa o algoritmo de CRC descrito na Seção 5.3.2.3;

8.1.7.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|--------------|--------------------|-------------------------------------|----------------|
| Clock | I Bit | | Can Core |
| Enable | I Bit | | Sync |
| Reset | I Bit | | Can Rx |
| stuff bit | I Bit | | Stuff Handler |
| crc_stop | I Bit | Comando para parar o cálculo de crc | Can_Rx |
| rx bit | I Bit | Bit recebido | Sync |
| tx_busy | I Bit | Indica transmissão em andamento | Can_Tx |
| crc_shiftout | I Bit | Comando para enviar crc | Can_Tx |
| Outbit | O Bit | Crc bit-a-bit | Can Tx |
| crc | O vetor 15 bits | Crc paralelo | Can_Rx |

Tabela 16 - Descrição da *port list* do CRC Calc

8.1.7.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design 'crc_calc'
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```
-----
Command Line   : map -p xc3v300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date    : Fri Aug 18 17:58:53 2000
```

Design Summary

```
-----
Number of errors:      0
Number of warnings:   1
Number of Slices:      21 out of 3,072    1%
  Slice Flip Flops:    31
  4 input LUTs:        27
Number of Slices containing
  unrelated logic:     0 out of 21    0%
Number of bonded IOBs: 23 out of 260    8%
Number of GCLKs:       1 out of 4    25%
Number of GCLKIOBs:    1 out of 4    25%
Total equivalent gate count for design: 410
Additional JTAG gate count for IOBs: 1,152
```

8.1.8 Módulo de Sincronismo

Executa o sincronismo do módulo CAN descritos na Seção 5.3.5. Gera os sinais de controle interno do módulo CAN e os dados de entrada e saída “sincronizados” .

8.1.8.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|------------|-------|---|----------------|
| Clock | I Bit | | Can Core |
| tx_busy | I Bit | Indica transmissão em andamento | Can_Tx |
| rx_busy | I Bit | Indica recepção em andamento | Can Rx |
| bit_stream | I Bit | Bit recebido do barramento CAN | Can_Core |
| reset | I Bit | | Can Core |
| stuff bit | I Bit | Sinaliza presença de stuff | Stuff Handler |
| tx_bit | I Bit | Bit a ser transmitido para o barramento CAN | Can_Tx |
| err_bit | I Bit | Sinal de sadia na transmissão de pacote de erro | Error_Frame |
| ack_bit | I Bit | Sinaliza um reconhecimento | Can_Rx |
| rx_bit | O Bit | Bit recebido | Todos módulos |
| Can_Reset | O Bit | Reset para restante dos blocos | Todos módulos |
| Enable | O Bit | Enable para restante dos blocos | Todos módulos |
| CANBit | O Bit | Bit transmitido para o barramento CAN | Can_Core |

Tabela 17 - Descrição da *port list* do Sync

8.1.8.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design '**synchronizer**'
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```
-----
Command Line   : map -p xcv300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date    : Fri Aug 18 17:41:19 2000
```

Design Summary

```
-----
Number of errors:      0
Number of warnings:   1
Number of Slices:      31 out of 3,072    1%
  Slice Flip Flops:    22
  4 input LUTs:       56
Number of Slices containing
  unrelated logic:    0 out of 31    0%
Number of bonded IOBs: 12 out of 260 4%
Number of GCLKs:      1 out of 4    25%
Number of GCLKIOBs:   1 out of 4    25%
Total equivalent gate count for design: 512
Additional JTAG gate count for IOBs: 624
```

8.1.9 Módulo de Stuffing

Executa a técnica de *stuffing* descrita na Seção 5.3.2.1.

8.1.9.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|--------------|-------|--------------------------------------|----------------|
| rx bit | I Bit | Bit recebido | Sync |
| clock | I Bit | | Can Core |
| enable | I Bit | | Sync |
| reset | I Bit | | Sync |
| disabled | I Bit | Comando para desabilitar stuff | Can_Rx |
| stuff_bit | O Bit | Indica a necessidade de bit de stuff | Todos módulos |
| Stuff_err or | O Bit | Indica violação a regra de stuff | Can_rx, can_tx |

Tabela 18 - Descrição da *port list* do *Stuff Handler*

8.1.9.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design '**stuffHandler**'
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```
-----
Command Line   : map -p xcv300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
```

Mapper Version : virtex -- C.21
 Mapped Date : Fri Aug 18 17:38:43 2000

Design Summary

```

-----
Number of errors:      0
Number of warnings:   1
Number of Slices:      6 out of 3,072    1%
  Slice Flip Flops:    6
  4 input LUTs:       9
Number of Slices containing
  unrelated logic:    0 out of 6    0%
Number of bonded IOBs: 6 out of 260    2%
Number of GCLKs:      1 out of 4    25%
Number of GCLKIOBs:   1 out of 4    25%
Total equivalent gate count for design: 102
Additional JTAG gate count for IOBs: 336
  
```

8.1.10 Módulo de Controle de Erros

Esse módulo recebe a sinalização de erros dos outros módulos CAN e gera os comandos corretos para o módulo de contadores de erros indicando que o tipo de falha detectado deve fazer com que o contador de erro de recebimento, por exemplo, seja incrementado.

8.1.10.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|------------------|-------|---|----------------|
| rx_signal_error | I Bit | Indica erro na recepção | Can_Rx |
| tx_signal_error | I Bit | Indica erro na transmissão | Can_Tx |
| clock | I Bit | | Can Core |
| enable | I Bit | | Sync |
| rx_bit | I Bit | Bit recebido | Can Rx |
| reset | I Bit | | Sync |
| err_passive | I Bit | Nodo em modo erro passivo | Error Counter |
| done | O Bit | Sinaliza fim do pacote de sinalização de erro | Can_tx,can_rx |
| err_bit | O Bit | Sinal de saída na transmissão de pacote de erro | sync |
| rx_err_cnt_plus8 | O Bit | Comando incrementar contador RX em 8 | Error_Counter |
| rx_err_cnt_plus1 | O Bit | Comando incrementar contador RX em 1 | Error_Counter |
| tx_err_cnt_plus8 | O Bit | Comando incrementar contador TX em 8 | Error_Counter |

Tabela 19 - Descrição da *port list* do *Error Frame*

8.1.10.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design '**err_frame_gen**'
 Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```

-----
Command Line : map -p xcv300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device : xv300
Target Package : bg352
Target Speed : -5
Mapper Version : virtex -- C.21
Mapped Date : Fri Aug 18 17:33:07 2000
  
```

Design Summary

```

-----
Number of errors:      0
Number of warnings:   1
Number of Slices:      24 out of 3,072   1%
  Slice Flip Flops:    16
  4 input LUTs:       39
Number of Slices containing
  unrelated logic:    0 out of 24   0%
Number of bonded IOBs: 11 out of 260   4%
Number of GCLKs:      1 out of 4   25%
Number of GCLKIOBs:   1 out of 4   25%
Total equivalent gate count for design: 362
Additional JTAG gate count for IOBs: 576

```

8.1.11 Módulo de Contadores de Erros

Armazenam os registradores de controle de erros. Esses contadores indicam o modo de operação em que o módulo CAN está (Seção 5.3.2.6).

8.1.11.1 Descrição da Entidade

| Sinal | Tipo | Descrição | Origem/Destino |
|-----------------------|-------------------|---|----------------|
| reset | I Bit | | Can Core |
| clock | I Bit | | Can Core |
| enable | I Bit | | Sync |
| rx_err_cnt_min us1 | I Bit | Comando decrementar contador RX em 1 | CAN_Rx |
| tx_err_cnt_plu s8 | I Bit | Comando incrementar contador TX em 8 | Error_Counter |
| tx_err_cnt_min us1 | I Bit | Comando decrementar contador TX em 1 | CAN_Tx |
| rx_err_cnt_plu s8 | I Bit | Comando incrementar contador RX em 8 | Error_Counter |
| rx_err_cnt_plu s1 | I Bit | Comando incrementar contador RX em 1 | Error_Counter |
| tx_err_cnt_nou pd | I Bit | Comando decrementar contador TX em 1 | CAN_Tx |
| rx bit | I Bit | Bit recebido | Sync |
| err passive | O Bit | Sinaliza nodo em erro passivo | Can tx,err frm |
| bus off | O Bit | Sinaliza nodo em <i>bus off</i> | Can tx,can rx |
| Tx_err_cnt | O vetor 8 bits | Contador de número de erros de transmissão | Can_Core |
| Rx_err_cnt | O vetor 8 bits | Contador de número de erros de recepção | Can_Core |

Tabela 20 - Descrição da *port list* do *Error Counter*

8.1.11.2 Resumo de Informações de Ocupação de Área

Xilinx Mapping Report File for Design '**error_counters**'
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.

Design Information

```

-----
Command Line   : map -p xcv300-5-bg352 -o map.ncd esa.ngd esa.pcf
Target Device  : xv300
Target Package : bg352
Target Speed   : -5
Mapper Version : virtex -- C.21
Mapped Date    : Fri Aug 18 17:36:04 2000

```

Design Summary

```

-----
Number of errors:      0
Number of warnings:   1
Number of Slices:      68 out of 3,072    2%
  Slice Flip Flops:    30
  4 input LUTs:       100 (2 used as a route-thru)
Number of Slices containing
  unrelated logic:    0 out of 68    0%
Number of bonded IOBs: 27 out of 260  10%
Number of GCLKs:      1 out of 4    25%
Number of GCLKIOBs:  1 out of 4    25%
Total equivalent gate count for design: 1,065
Additional JTAG gate count for IOBs: 1,344

```

8.1.12 Resumo de Informações do HurriCANE

Abaixo a Tabela 21 apresenta um resumo de informações, em relação a área, dos módulos que compõem o HurriCANE. Informações de desempenho de tempo do sistema está fora do escopo do trabalho.

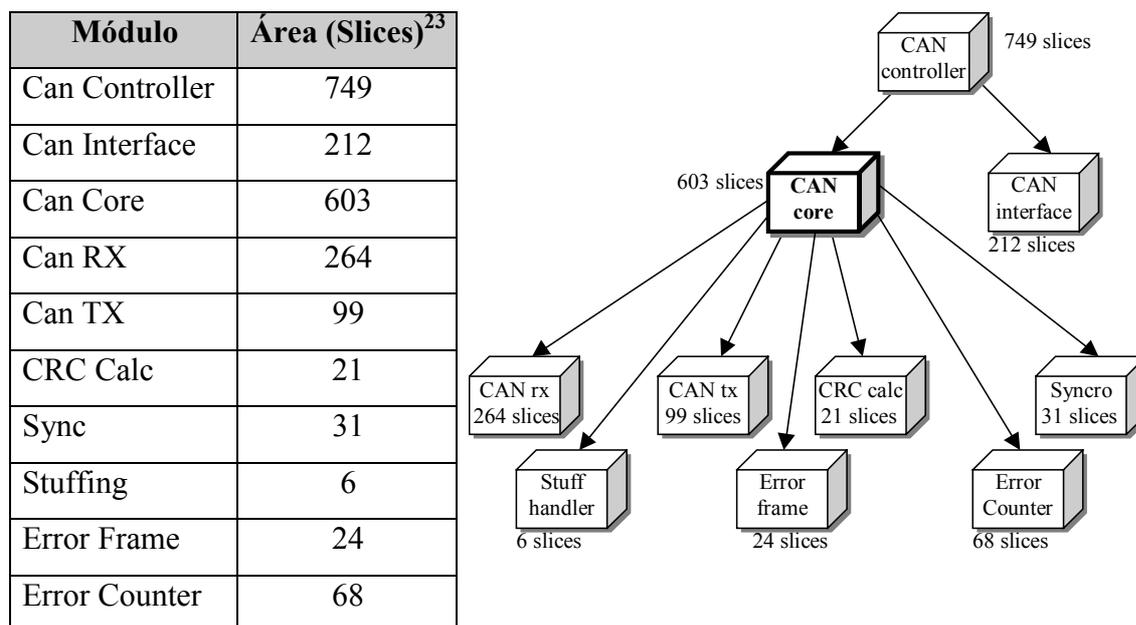


Tabela 21 – Resumo de dados dos módulos HurriCANE

8.2 Arquitetura dos Nodos Escravos

²³ Usando FPGA xcv300-5-bg352 da família Virtex 300 da Xilinx sem esforço máximo de síntese

Os nodos escravos são responsáveis por controlar os periféricos da casa. Poderemos ter, por exemplo, um nodo que controla lâmpadas, sensores de temperatura, entre outras aplicações possíveis.

Os nodos escravos serão, por limitação de recursos, dois módulos **xc4010-3-EPC84** [30] que possuem **400 CLBs**. O ideal seria se tivéssemos disponível pelo menos dois dispositivos com 700 CLBs, não necessitando assim investirmos tempo em adequar o HurriCANE para esse FPGA.

Para que fosse possível carregar o xc4010-3-EPC84 com HurriCANE mais a aplicação do usuário, precisamos retirar certas funcionalidades do protocolo CAN como o bloco de contadores de erro e bloco de controle de erros. Devido a esse motivo, esses módulos não têm implementados sinalização de erros. O módulo de interface também não será usado pelo fato de não estar sendo usado um processador para implementar a aplicação.

A plataforma onde será implementado o nodo escravo é apresentada na Figura 71. Na Figura 72 [30] apresentamos em forma de diagrama de blocos. Essa placa de prototipação é usada na graduação do curso de informática e na pesquisa para desenvolvimento de pequenos projetos. Seus recursos são listados na figura abaixo.

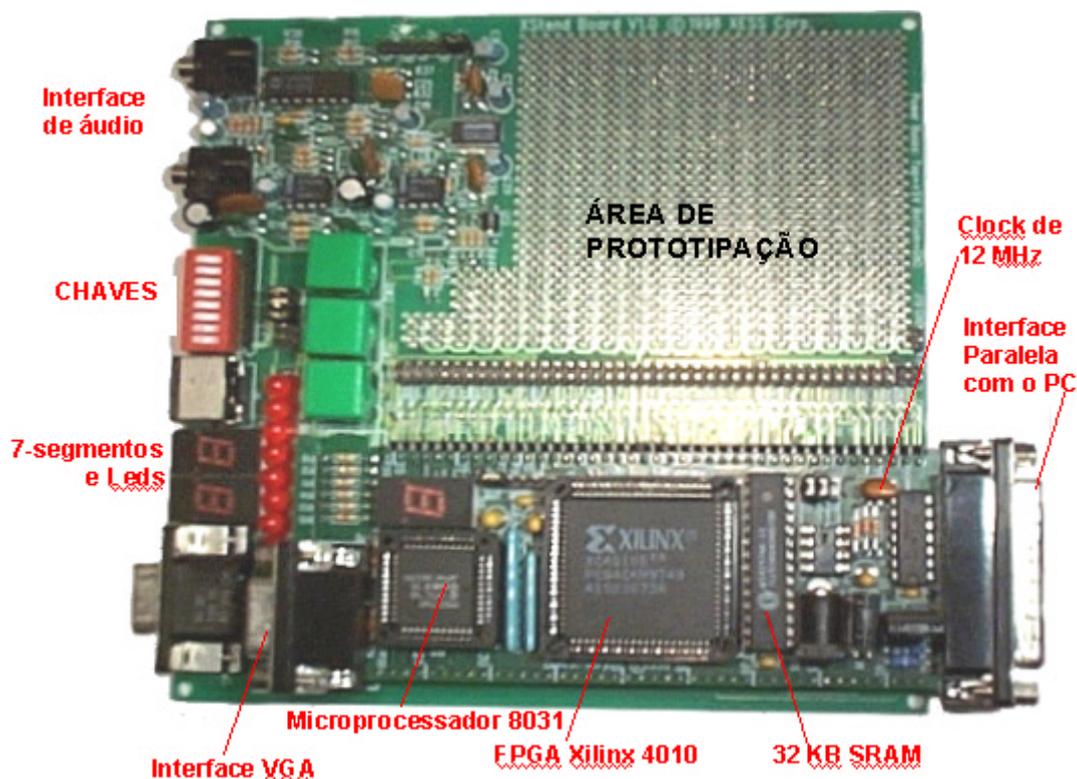


Figura 71 – Plataforma nodo escravo (XS40)

Cada aplicação controlada existente na casa deverá possuir um suporte tanto em *hardware* quanto em *software*. Por exemplo, para se fazer o controle de lâmpadas é necessário existir no sistema módulos (*hardware*) que farão a interface de lâmpadas com o barramento CAN, e código

(software) para gerenciar lâmpadas. Dessa forma, foi implementado para este projeto uma aplicação de lâmpadas, que demonstrará o funcionamento do sistema.

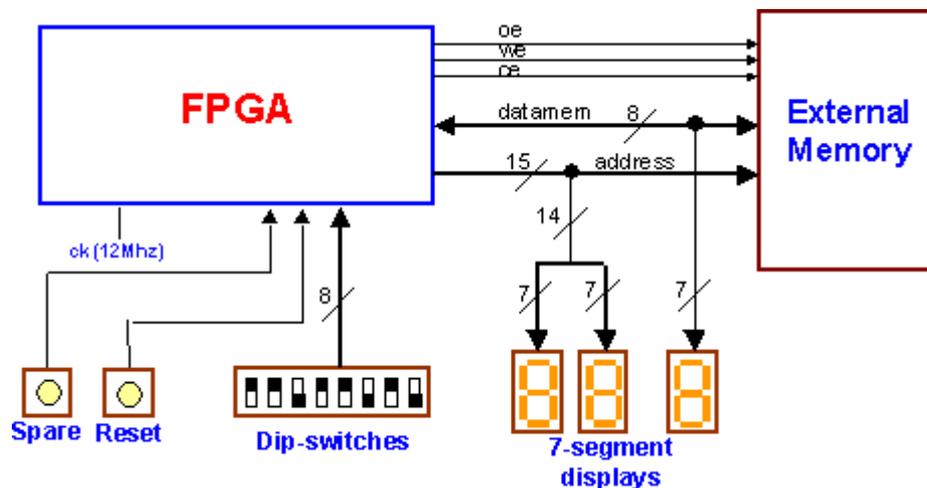


Figura 72 – Esquema XS40

8.2.1 Nodo de Lâmpadas

Estes nodos são responsáveis por gerenciar lâmpadas. Eles suportam até 64 lâmpadas, uma vez que o protocolo CAN possui 8 bytes de dados (8bytes x 8bits) por nodo. Cada bit do campo de dados representa o estado de uma lâmpada (acesa ou apagada).

A arquitetura dos nodos de lâmpadas pode ser visualizada na Figura 73.

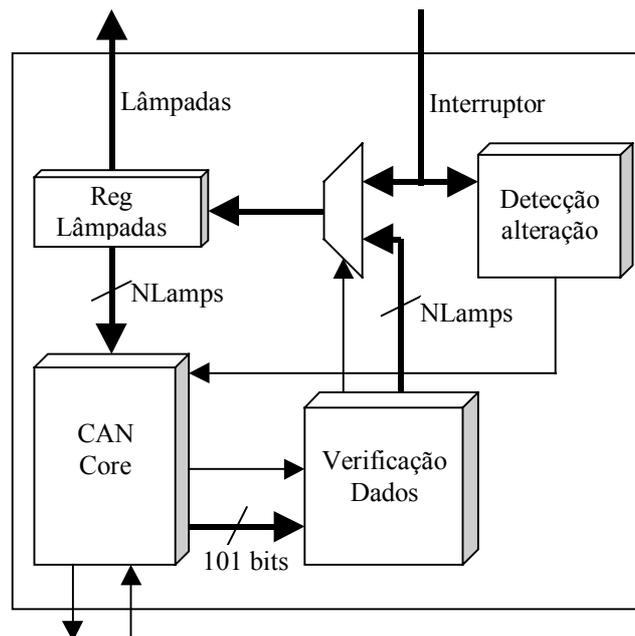


Figura 73 – Arquitetura do nodo de lâmpadas

O módulo *CAN Core* faz a codificação dos pacotes para o protocolo CAN.

O módulo de verificação de dados é ativado quando um pacote é recebido. Ele possui a função de verificar o destinatário do pacote, fazendo a comparação do ID do nodo com o ID recebido no pacote. Se esses forem iguais, será ativado a escrita no registrador de lâmpadas.

O módulo de detecção de alteração é responsável por detectar alguma mudança nos interruptores. Quando uma mudança ocorre, é disparado uma transmissão para o nodo mestre de forma a atualizar o estado das lâmpadas no servidor. Desta forma, existem duas formas de se alterar o estado de uma lâmpada: a manual (efetuada pelo morador normalmente) e a remota (efetuada pelo sistema).

Para fins de apresentação deste projeto usaremos as chaves da plataforma de prototipação como os interruptores das lâmpadas e os leds como lâmpadas.

8.2.2 Resumo de Relatório de Síntese

Essa Seção apresenta um resumo de informações de síntese geradas pela ferramenta Xilinx Foundation²⁴ [39]. O relatório de roteamento destaca a área em número de CLBs em relação a um FPGA **xc4010e-3-pc84** (294 CLBs com 73% de utilização total). O relatório de temporização apresenta uma aproximação da frequência máxima (**9.018MHz**) que essa descrição suporta.

8.2.2.1 Place and Route Report

```

Design Information
-----
Command Line   : mlmap -p xc4010e-3-pc84 -o map.ncd ver1.ngd ver1.pcf
Target Device  : x4010e
Target Package : pc84
Target Speed   : -3
Mapper Version : xc4000e -- C.21

Design Summary
-----
Number of errors:      0
Number of warnings:   6
Number of CLBs:      294 out of 400 73%
  CLB Flip Flops:     365
  4 input LUTs:       474 (2 used as route-throughs)
  3 input LUTs:       105 (49 used as route-throughs)
Number of bonded IOBs: 16 out of 61 26%
  IOB Flops:          0
  IOB Latches:        0
Number of clock IOB pads: 1 out of 8 12%
Number of primary CLks: 1 out of 4 25%
Number of secondary CLks: 3 out of 4 75%
Number of OSC:        1 out of 1 100%
Total equivalent gate count for design: 5316
Additional JTAG gate count for IOBs: 768

```

²⁴ <http://www.xilinx.com>

8.2.2.2 Timing Report

Design statistics:

Minimum period: 110.887ns (Maximum frequency: 9.018MHz)

Maximum net delay: 26.120ns

8.2.3 Formas de Onda do Osciloscópio

Nesta Seção apresentamos algumas formas de onda básicas tiradas do osciloscópio²⁵.

A Figura 74 apresenta a monitoração do barramento durante a transmissão de uma pacote CAN. Esse pacote é formado pelo campo ID 145, dado 0 e DLC 1 . Foi destacado nesta figura o início do pacote (SOF), o momento que o receptor escreve o reconhecimento do pacote (Ack bit) e o fim do pacote (EOF).

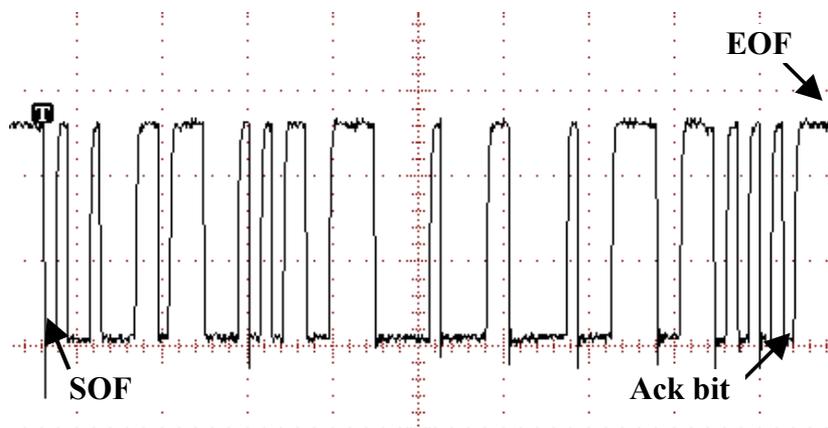


Figura 74 – Monitoração pelo osciloscópio de um pacote CAN completo

Seguindo a mesma monitoração, na Figura 75 temos a visualização dos bits transmitidos pelo transmissor e pelo receptor (ack bit). O E lógico desses sinais formará a figura mostrada anteriormente.

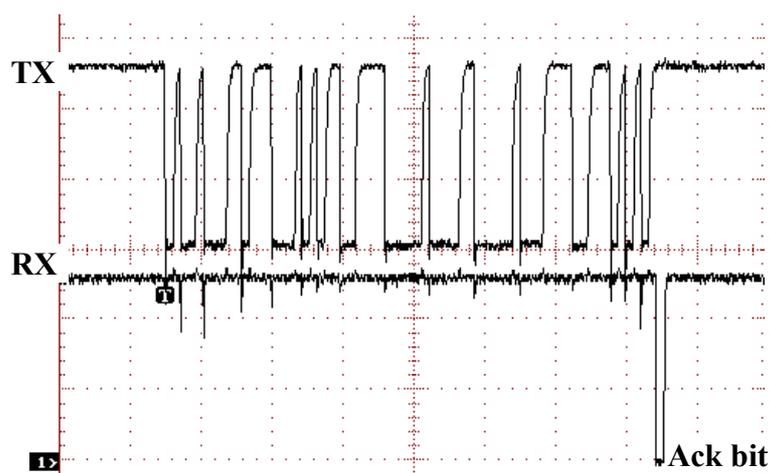


Figura 75 – Transmissor X receptor CAN

²⁵ Tektronix TDS 220

8.2.4 Formas de Onda por Simulação

Nesta Seção apresentamos algumas formas de onda obtidas por simulação que auxiliam na compreensão da implementação interna do módulo.

De forma semelhante à Figura 74, a Figura 76 apresenta o pacote CAN completo para o mesmo dado que na Figura 74, porém simulado na ferramenta QHSIM²⁶.

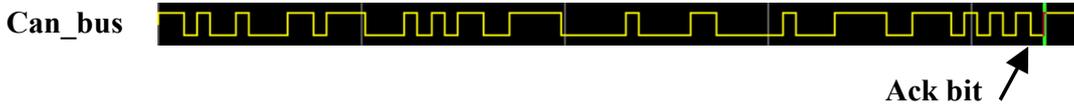


Figura 76 – Simulação de um pacote CAN Completo

Novamente fazendo um comparativo de imagens do osciloscópio e de simulação, mostramos na Figura 77 o sinal *can_bus* que representa o barramento CAN, o sinal *can0/tx_bit1* representando o transmissor e o *can1/tx_bit1* representando o receptor. Note, novamente, que o barramento é formado pelo E lógico entre a sinalização do transmissor e do receptor.

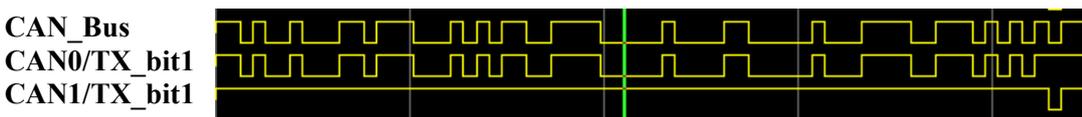


Figura 77 – Comparação entre barramento, bits do transmissor e do receptor

Nas figuras seguintes é detalhado o processo de transmissão e de recepção do módulo.

Em um nodo de lâmpada, uma transmissão é disparada sempre que algum dos interruptores de lâmpadas mudar, isso quer dizer que sempre que uma lâmpada mudar de valor por um comando local uma nova transmissão será iniciada com o motivo de atualizar o Servidor. O sinal *SwitchChangeOr* é um Ou lógico entre todos os interruptores. Quando esse sinal vai para nível lógico 1 é disparada uma transmissão. Nesse instante a mensagem a ser transmitida é carregada no buffer *Msg* e o sinal de *Start* vai para nível lógico 1 onde voltará para 0 somente quando a transmissão terminar. O sinal *Tx_Completed* sinaliza o fim da transmissão.

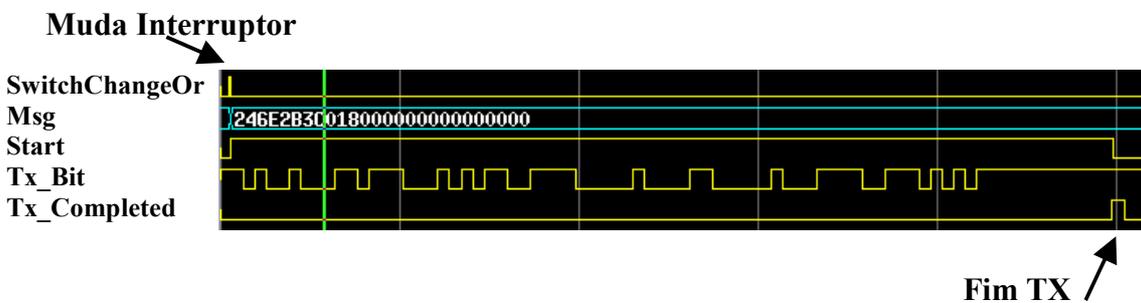


Figura 78 – Processo de envio de pacotes

²⁶ Simulador VHDL da Mentor Graphics (<http://www.mentor.com>)

Quando um nodo detecta o SOF (borda de descida do barramento) eles entram em estado de recepção. Porém somente o nodo destino, indicado no campo ID, irá receber a mensagem até o fim pois a aplicação de lâmpada possui uma consistência que faz a comparação entre o ID recebido e o ID do nodo. Somente se forem iguais é que as lâmpadas serão ligadas/desligadas.

A Figura 79 apresenta o processo de recepção de um pacote CAN. Ao fim de uma recepção, o nodo destino gerará um sinal chamado *RxCompleted*, que indica que uma mensagem foi recebida. Note que o *buffer RxMsg* é atualizado praticamente no meio do *RxCompleted*, por esse motivo um novo sinal *RxCompletedEdgeLow*, que detecta uma borda de descida do *RxCompleted*, foi criado já que neste ponto tem-se a garantia que o *buffer* de recebimento já fora atualizado. Entre o período que ocorre a sinalização do *RxCompletedEdgeLow* e o *MsgReceived* ocorre a fase de comparação do ID da mensagem recebida com o ID do nodo. *MsgReceived* somente será sinalizado se esses IDs forem iguais. Quando *MsgReceived* é ativado ocorre a atualização das lâmpadas, desligando as lâmpadas em que o bit representativo for '0'.

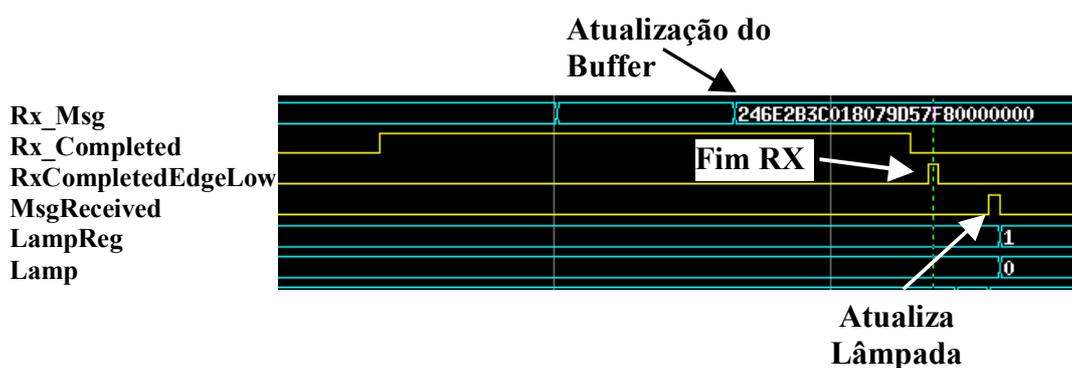


Figura 79 – Processo de recepção de pacotes

8.3 Arquitetura do Nodo Mestre

O Nodo mestre é responsável por fazer a interface entre a rede CAN e o Servidor.

O mestre é um FPGA **xcv300-5-bg352 [28]** que possui **3072 slices**. Neste nodo não houve a necessidade de retirar funcionalidades do protocolo CAN, uma vez que ele comporta essa descrição VHDL.

A arquitetura do nodo mestre pode ser visualizada na Figura 80. Essa é baseada em um sistema de filas pelo fato de as duas interfaces (serial e CAN) terem velocidades de transmissões diferentes. Quando a interface CAN recebe um pacote, o mesmo é dividido em 13 bytes e gravado byte a byte na *fila CAN*. Enquanto a fila CAN não está vazia a interface serial vai transmitindo dados desta fila pela serial. De forma análoga, quando a interface serial recebe algum dado, a

mesma o grava na *fila Serial*. Enquanto essa fila não está vazia, a interface CAN transmite dados pelo barramento CAN para que o nodo destino possa recebê-lo.

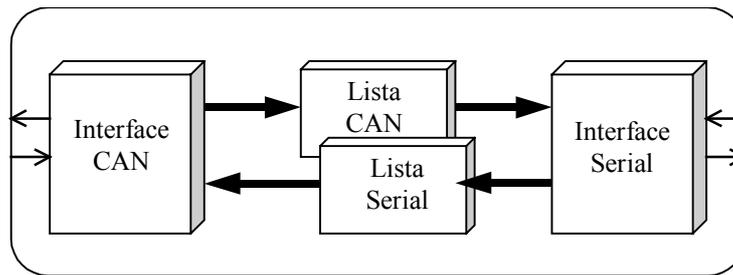


Figura 80 – Arquitetura do nodo mestre

As *filas* da interface CAN e serial são implementadas usando recurso interno ao FPGA chamado *Select Ram*[26]. Nessa implementação o *Select Ram* está configurado para ser uma memória *dual port* (sendo uma porta somente de escrita e outra somente de leitura) de 8 bits de tamanho de palavra, com 512 palavras. A escolha dessa configuração foi motivada pelo fato de que a serial transmite 8 bits, simplificando o sistema, pois o dado lido da fila CAN é diretamente transmitido pela serial.

A *interface CAN* faz a interface do nodo mestre com o barramento CAN. Quando essa interface recebe um pacote, o mesmo é dividido em 13 bytes e gravado na fila CAN. No processo de divisão dos pacotes, algumas modificações nos dados são feitas para que seja gravado na memória exatamente a sintaxe que o servidor entende e que posteriormente será transmitido pela serial. Essas modificações são explicadas na Seção 8.5

A *interface serial* é responsável por enviar/receber dados ao/do servidor. Esse módulo funciona sem paridade, com *stop bit* em 1 e com taxa de transmissão de 19200 bps expansível até 115200 bps.

A plataforma onde será implementado o nodo mestre é apresentado na Figura 81 [29]. Essa placa é usada principalmente para grandes projetos pois possui um FPGA de grande porte e vários recursos adicionais listados na figura abaixo.

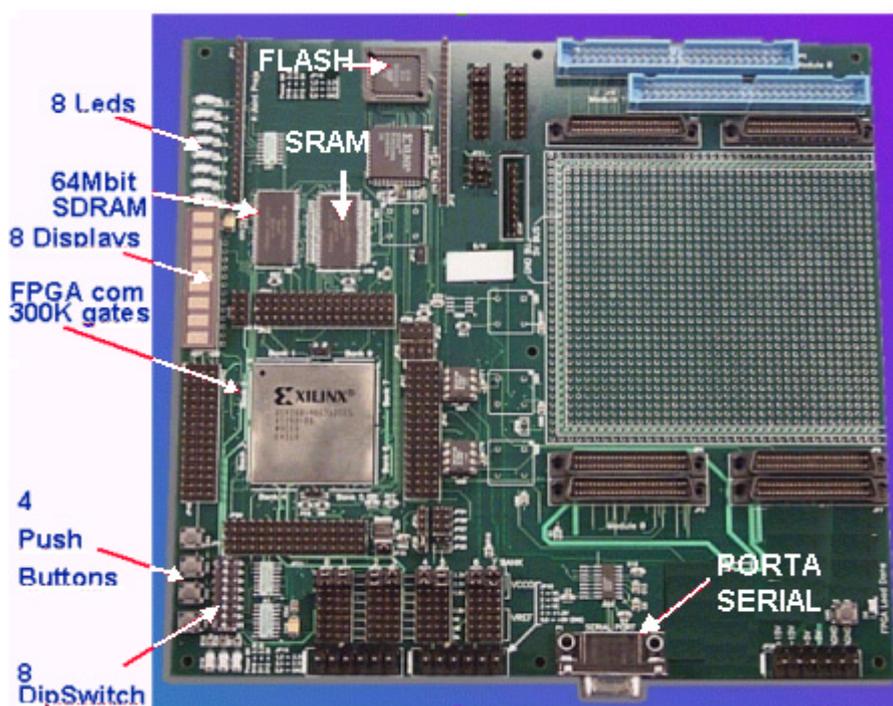


Figura 81 – Plataforma nodo mestre (VW300)

8.3.1 Resumo de Relatório de Síntese

A Tabela 22 apresenta um resumo com as principais informações geradas durante o processo de síntese do nodo mestre pela ferramenta Foundation. Esses dados foram gerados levando-se em consideração a utilização do FPGA xcv300-5-bg352, sem uso de esforço máximo de síntese.

| Módulos | Ocupação (Slices) | Ocupação (%) | Frequência Máxima (MHz) |
|------------------|-------------------|--------------|-------------------------|
| Mestre | 731 | 23 | 26.134 |
| Interface CAN | 512 | 16 | 25.028 |
| Interface Serial | 65 | 2 | 85.434 |
| Fifos | 116 | 3 | 37.448 |

Tabela 22 – Resumo de relatórios de síntese do mestre

8.3.2 Formas de Onda por Simulação

Nesta Seção é apresentado alguns resultados de simulação do módulo de aquisição de dados (mestre). Primeiramente, na Seção 8.3.2.1, é apresentado os resultados das entradas e saídas do mestre. Nas próximas Seções são apresentas simulações de blocos internos do mestre.

Nestas simulações foram usados dois pacotes de teste transmitidos em seqüência. O primeiro pacote possui os valores 55 00 FF 0B 01 0F associados aos campos ID, RTR+DLC e DADOS respectivamente. O segundo possui os valores AA FF 00 15 01 0F.

O teste desta simulação é suportado pelo *testbench* explicado na Seção 8.3.3.

8.3.2.1 Mestre

Na Figura 82 apresentamos alguns sinais significativos em relação a conversão CAN para Serial. Essa conversão ocorre quando o barramento da casa possui um pacote CAN válido. Desta forma esse pacote deve ser convertido para o padrão do Servidor (Seção 8.5).

Devido ao fato que a taxa de transferência CAN ser muito mais rápida que a taxa de transferência serial (na ordem de 15 vezes), a recepção CAN (*CANBitStream*) aparece na Figura 82 como um pequeno bloco. Durante a recepção CAN, o sinal *CANBusy* sinaliza que o módulo CAN está em operação. Esse sinal é importante, pois somente quando o mesmo estiver ocioso é que a fila Serial poderá escrever um novo dado no barramento para esse ser transmitido.

Assim que uma recepção CAN é finalizada o sinal *RX_Completed* gera n pulsos (onde n é o número de bytes do pacote recebido) de 1 ciclo de *clock* que comandam a escrita na fila CAN. Desta forma, nota-se que o sinal *FIFOCANEmpty* vai para nível lógico 0 assim que uma mensagem CAN é recebida e escrita na fila CAN.

Sempre que a fila CAN não está vazia e a transmissão serial não estiver ocupada será iniciada uma transmissão serial. Notamos que logo que o sinal *FIFOCANEmpty* foi para 0, foi iniciada essa transmissão em *SerialTX* do dado lido da fila CAN, indicado pelo sinal *SerialDataIn*.

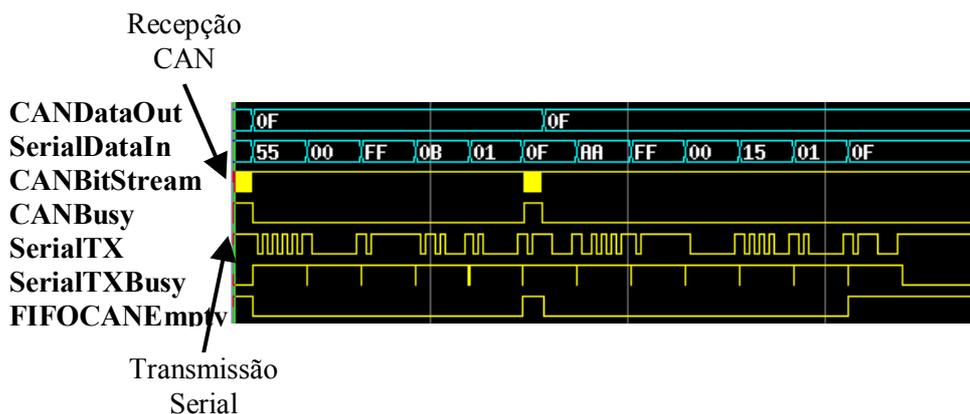


Figura 82 – Conversão CAN para Serial

De forma análoga, a Figura 83 apresenta a recepção de um pacote serial e sua transformação para o protocolo CAN. No sinal *SerialBitStream* observamos a recepção dos dados pela porta serial. O Sinal *SerialDataOut* representa o dado recebido serialmente.

O sinal *FIFOSerialEmpty* indica que, logo que o dado é recebido pela serial, a fila serial é escrita. Então o sinal *CANWr* é gerado (sempre que a fila serial não está vazia), lendo o dado da fila e escrevendo-o no *buffer* interno do módulo CAN. O módulo CAN só transmitirá os dados quando receber todos os bytes pertencentes ao pacote.

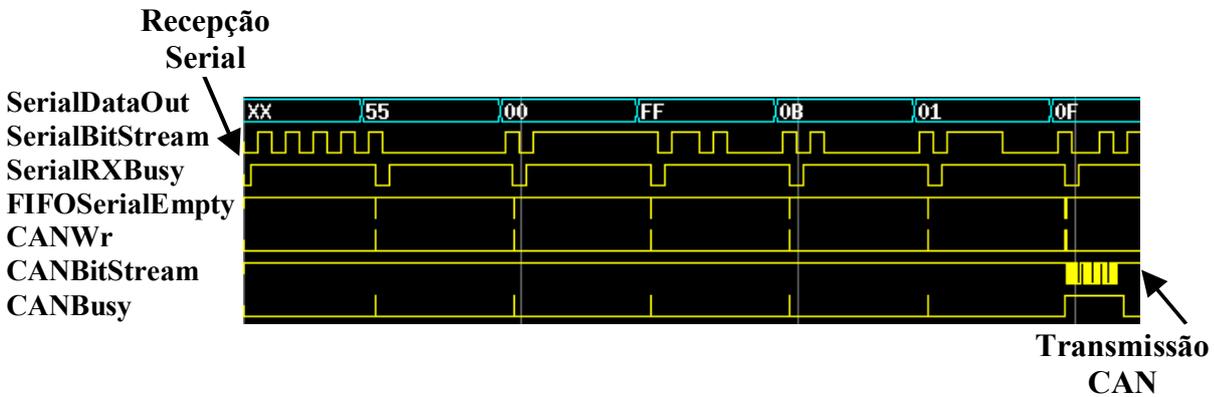


Figura 83 – Conversão Serial para CAN

8.3.2.2 Interface Serial

Sempre que a fila CAN não estiver vazia e o sinal *SerialTXBusy* estiver em nível lógico 0, inicia-se uma transmissão serial. Esse processo é apresentado na Figura 84. O Sinal *SerialStartTX* é gerado para iniciar uma transmissão serial. Sempre que esse sinal é ativado, parte-se do pressuposto que o dado a ser transmitido está na porta *SerialDataIn*. Logo após o sinal *SerialStartTX* ser gerado, o sinal *SerialTXBusy* vai para nível 1, impedindo a geração de novos sinais *SerialStartTX* antes da transmissão ser concluída. A transmissão serial é concluída por completo somente quando a fila CAN estiver vazia.



Figura 84 – Transmissão serial do mestre

8.3.2.3 Interface CAN

Sempre que um módulo de aplicação (escravo) transmitir algo no barramento, o módulo mestre receberá esse dado e sinalizará a correta recepção do pacote enviado um sinal de ACK. Após o término da recepção, é gerada uma seqüência de sinais *CANRXCompleted*. Cada sinal gerado

dispara a gravação de um byte do pacote na fila CAN. O sinal *CANDataOut* apresenta o dado que esta sendo gravado. Logo que a fila CAN recebe o primeiro dado, ela não fica mais vazia. Desta forma, uma transmissão serial é iniciada para transmitir ao servidor os novos dados recebidos.

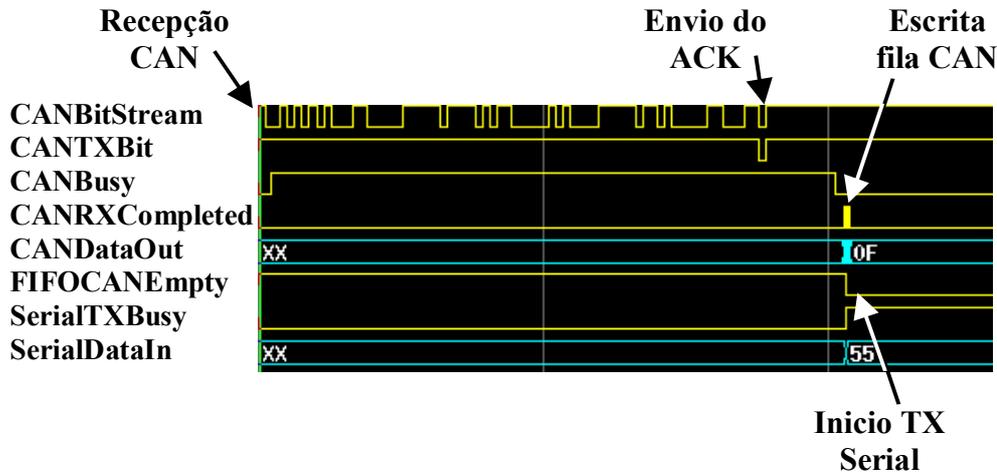


Figura 85 – Recepção CAN no mestre com transmissão serial

8.3.2.4 Filas

As filas internas do mestre funcionam com base no conceito de *MailBoxes*. Cada *mailbox* possui o tamanho de 16 bytes, sendo que o pacote CAN pode ter até 13 bytes. As filas implementadas possuem capacidade de armazenar até 32 *mailboxes* cada uma. Facilmente poderia-se aumentar a capacidade destas filas pois o FPGA usado²⁷ possui mais 16 blocos de *SelectRam* [26](ram interna).

A Figura 86 apresenta a estrutura interna da fila CAN. O sinal *FIFOCANLastMB* é o ponteiro para o último *mailbox* escrito e o sinal *FIFOCANLastByte* é o ponteiro para qual dos 16 bytes do *mailbox* foi feita a última escrita. A concatenação destes dois sinais é o endereço do último byte escrito na fila (*FIFOCANLastAddr*). Exemplo, se *FIFOCANLastMB* for igual a 2 e *FIFOCANLastByte* é igual 0 então (“00010”&”0000”) o valor de *FIFOCANLastAddr* é igual a 32.

De forma semelhante, o ponteiro para o primeiro da fila é dado pelo sinal *FIFOCANFirstAddr*. Esse sinal aponta para o endereço do próximo dado a ser transmitido pela serial.

Os sinais *FIFOCANTamPackWr* e *FIFOCANTamPackRd* auxiliam na identificação do tamanho do pacote que está sendo escrito ou lido respectivamente.

Pode-se observar que quando é feito uma escrita, o sinal *FIFOCANEmpty* indica que fila não está mais vazia. Por outro lado, quando todo conteúdo da fila é transmitido pela serial e a fila se esvazia, esse sinal vai sinalizar o novo estado da fila.

²⁷ xcv300-5-bg352

8.3.3 TestBench

Visando efetuar testes sobre este módulo, desenvolveu-se um *testbench* para simular a recepção serial e CAN. Para facilitar a recepção CAN foi utilizado um módulo CAN auxiliar que tem a função de simular um nodo escravo da rede. Desta forma a estrutura do *testbench* é como o apresentado na Figura 88.

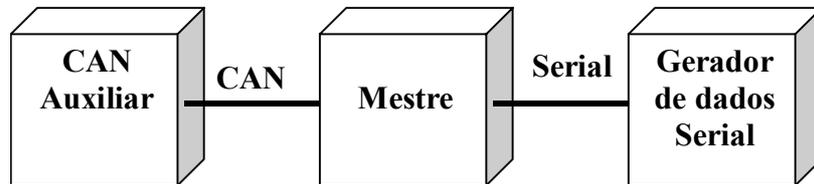


Figura 88 – Estrutura do *testbench* do mestre

O *testbench* também possui um conjunto de *Test Patterns*. Ou seja, um conjunto de dados preestabelecidos que representam os dados que serão inseridos nos sistemas e suas respostas esperadas. Desta forma consegue-se fazer com que o sistema injete dados de entrada e no final podemos comparar as saídas do sistema com as respostas esperadas. Se as mesmas coincidirem, o sistema está correto.

Esta técnica acelerou o processo de desenvolvimento do nodo mestre, pois não era necessário verificar formas de ondas complexas para ver se a última mudança teve o resultado esperado, uma vez que o próprio *testbench* compara a saída do sistema com a resposta esperada e gera um pequeno relatório. Esses relatórios podem ser vistos a baixo na Tabela 23 e na Tabela 24. O primeiro é gerado na transformação de CAN para Serial e o segundo de Serial para CAN. Note que as comparações e conclusões são geradas ao longo da simulação e estão em negrito.

| | |
|---|---|
| <pre># TESTING CAN TO SERIAL CONVERSION # TEST PATTERN1 : # Verify ID1 ... # OK! # Verify ID2 ... # OK! # Verify ID3 ... # OK! # Verify ID4 ... # OK! # Verify RTR and DLC ... # OK! # Verify DATA1 ... # OK! # No Errors found! # TEST PATTERN2 : # Verify ID1 ... # OK! # Verify ID2 ... # OK! # Verify ID3 ... # OK! # Verify ID4 ... # OK! # Verify RTR and DLC ... # OK! # Verify DATA1 ... # OK! # No Errors found!</pre> | <pre># TESTING CONVERSION SERIAL TO CAN # TEST PATTERN1 : # Sending IDs ... # Sending RTR and DLC ... # Sending DATA ... # Sending IDs ... # Reception completed # Verify ID : # Msg sent 1010101000000001111111101011 # Msg recv 1010101000000001111111101011 # OK! # Verify RTR and DLC : # Msg sent 00001 # Msg recv 00001 # OK! # Verify DATA1 : # Msg sent 00001111 # Msg recv 00001111 # OK! # No Errors found!</pre> |
| <p>Tabela 23 – Relatório do <i>testbench</i> do mestre (CAN para Serial)</p> | |

| | |
|--|---|
| | <pre> # TEST PATTERN2 : # Sending IDs ... # Sending RTR and DLC ... # Sending DATA ... # Reception completed # Verify ID : # Msg sent 010101011111111100000000010101 # Msg rcv 010101011111111110000000010101 # OK! # Verify RTR and DLC : # Msg sent 00001 # Msg rcv 00001 # OK! # Verify DATA1 : # Msg sent 00001111 # Msg rcv 00001111 # OK! # No Errors found! </pre> <p style="text-align: center;">Tabela 24– Relatório do <i>testbench</i> do mestre (Serial para CAN)</p> |
|--|---|

A Plataforma de teste de síntese usada para validar sistema de *hardware* é apresentada na Figura 89. Os seguintes itens foram usados:

- um micro-computador (1) que funcionava como o servidor do sistema, comunicando-se com o mestre pela porta serial;
- Osciloscópio (2) que monitorava o sinal no barramento CAN;
- Analisador Lógico (3) que monitorava sinais do mestre usados para depuração;
- Placa de prototipação VW300 (4) onde a descrição VHDL do mestre era executada;
- Placa de prototipação XS40 (5) onde a descrição VHDL da aplicação de lâmpadas era executada;

A Figura 90 mostra em maiores detalhes o cabeamento usado para depuração do sistema de *hardware*.

1. Cabo para carregar a descrição do mestre na VW300;
2. Extensor serial que comunica a VW300 com o servidor;
3. Cabeamento de alimentação da VW300;
4. Cabos dos analisador lógico;
5. Cabo do osciloscópio;
6. Cabo para carregar a descrição da aplicação de lâmpadas na XS40;
7. LEDs que simulam as lâmpadas da casa;
8. Chaves que simulam os interruptores da casa;

9. Fios que simulam o cabeamento necessário na casa para a comunicação entre o mestre e seus periféricos;
10. Placa XS40 com aplicação de lâmpadas;
11. Placa VW300 com aplicação do mestre.

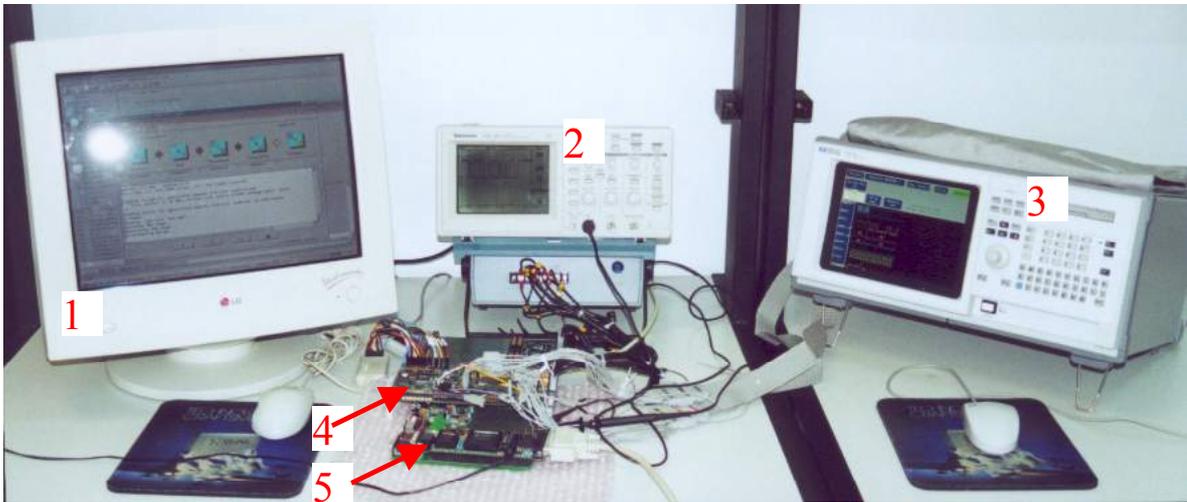


Figura 89 – Bancada de teste do mestre

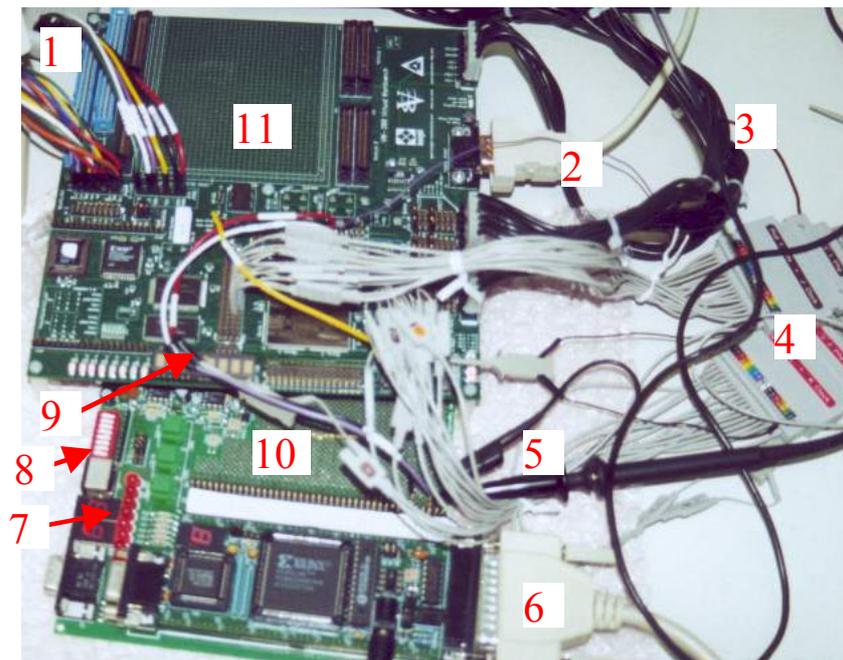


Figura 90 – Detalhes da bancada de teste

8.4 Pacotes de Dados x Pacotes de Controle

Para que fosse possível transmitir dados maiores que 8 bytes foi desenvolvido uma técnica que divide os pacotes CAN em dois grupos: pacotes de dados e de controle.

Vale ressaltar que com essa técnica a compatibilidade com o protocolo CAN foi preservada. Sendo assim, é possível usar componentes CAN comerciais para execução prática deste projeto. As

alterações serão refletidas em um nível de abstração acima do CAN Core, nível este onde são implementadas as aplicações que controlam os periféricos em questão.

Os pacotes de dados, apresentados na Figura 91, são os pacotes normalmente transmitidos pela rede. Esse pacote possui o limite de 8 bytes de dados, sendo especialmente interessante para redes de controle, pois as mesmas possuem a característica de trafegar uma quantidade menor de dados.

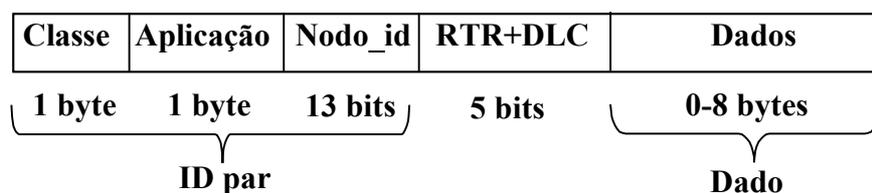


Figura 91 – Formato de pacotes de dados

Os pacotes de controle, apresentados na Figura 92, são serviços especiais do sistema que solicitam ou informam alguma característica específica para determinada aplicação. Por exemplo, uma câmera de vídeo pode enviar um pacote de controle para o servidor informando que irá começar a transmitir uma imagem de 100 Kbytes. Outro possível exemplo seria a transmissão de uma mensagem de controle do servidor para um nodo de lâmpadas solicitando um auto teste do nodo (explicado na Seção 9.4).

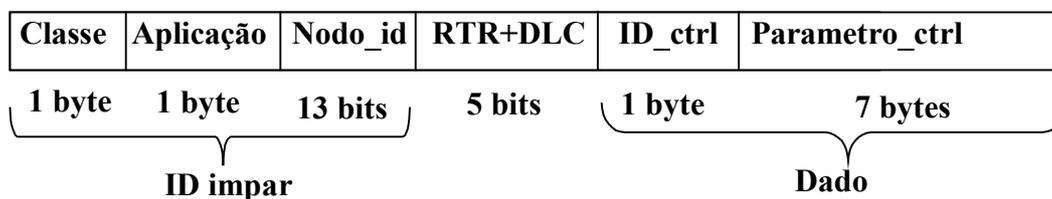


Figura 92 – formato de pacotes de controle

O que diferencia um pacote de dados de um pacote de controle é o ultimo bit do ID. Se esse for 0 (par), o sistema interpreta como um pacote de dados. Caso contrario, o pacote recebido será interpretado como pacote de controle. Uma outra observação é que o campo DLC de pacotes de controle sempre indicam 8 bytes de dados.

O primeiro byte do campo de dados do protocolo CAN representa a identificação do tipo de controle que esta sendo gerado. Os 7 bytes restantes são utilizados como um possível parâmetro para uma mensagem de controle. Por exemplo, um nodo que controla uma câmera possui 3 serviços de controle: envio de pacotes grandes (maiores que 8 bytes), auto teste e reconfiguração. Cada serviço será identificado com um valor para tipo de serviço diferente, previamente estipulado. Caso o nodo da câmera receba um pacote de controle com tipo 2, ele saberá que devera executar o auto

teste. Se receber com tipo 3 será executada reconfiguração. Quando for inicializado uma transmissão de dados com mais de 8 bytes, o nodo gerará para o servidor uma mensagem indicando que começará a transmitir X bytes de dados. Esse valor X vai no campo de parâmetro da mensagem de controle.

Essa abordagem possibilitou uma *grande flexibilidade* ao sistema, pois dá suporte a criação de até 256 serviços para cada aplicação existente no sistema, sendo que cada serviço pode ter até 7 bytes de parâmetros. Para o caso de transmissão de pacotes maiores que 8 bytes, o *hardware* suporta uma transmissão teórica de até 2^{56} (8bits*7bytes) bytes de dados.

Para apresentação desse trabalho, será apresentado a implementação, somente em software, do serviço de envio de pacotes de dados maiores que 8 bytes. Outros serviços são sugeridos para projetos futuros na Seção 9.

8.5 Interface *Hardware/Software*

A interface entre *hardware* e *software* dá-se através de comunicação serial a 19200bps sendo expansível à 115000bps. No futuro pretende-se implementar uma interface PCI (ver Seção 9.1).

O pacote CAN apresentado na Figura 91 e Figura 92 é dividido em 13 pacotes no protocolo serial, seguindo o modelo apresentado na Figura 93.

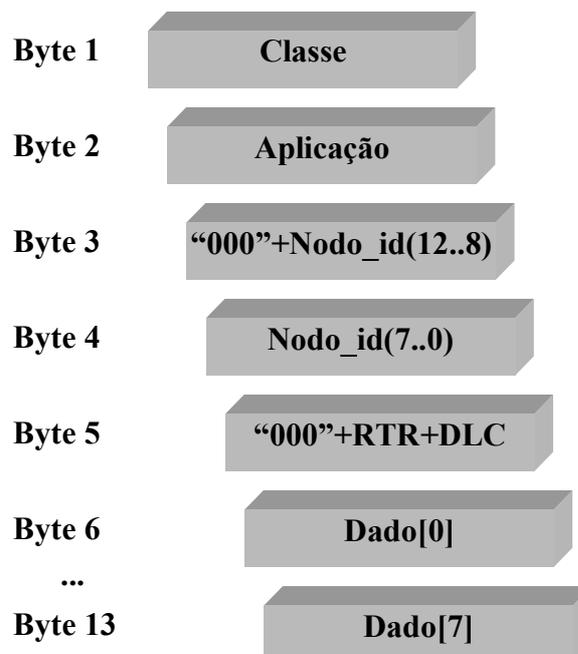


Figura 93 – Interface *Hardware/Software*

A forma adotada usa a inserção de 3 bits em nível lógico 0 nos bits mais significativos do byte 3 e 5.

9 Projetos Futuros

Dentre as propostas para continuação desse projeto sugerimos alguns serviços adicionais para o sistema.

9.1 Interface PCI

Atualmente estamos usando protocolo serial como interface entre *hardware* e *software*. Porém, devido a baixa taxa de transmissão deste protocolo, uma versão com protocolo PCI torna-se recomendada. Aplicações com transferência de imagem, vídeo e som teriam um grande aumento de desempenho com a taxa de 33MHz/66MHz que PCI possui. O uso de PCI removeria o gargalo do sistema, que atualmente é interface *hardware/software* ²⁸.

Um estudo detalhado de performance também seria recomendado para futuros trabalhos.

9.2 Protocolo *Plug and Play*

Tem a intenção de facilitar a instalação de um novo nodo no sistema. Em sistemas de domótica esse é um requisito importante, pois os consumidores não necessariamente tem afinidade com configuração de máquinas, placas, etc.

Podemos seguir a seguinte hipótese: Todo nodo vem de “fábrica” com sua classe e aplicação gravadas em ROM. O `nodo_id`, que distingue de outros nodos com a mesma função, será *especificado dinamicamente*, de acordo com a ocupação na rede em que este nodo for instalado.

O possível processo de instalação deste nodo P&P seria:

- Sempre que um novo nodo for inserido na rede, ele vai enviar para o uma mensagem solicitando um `nodo_id` de volta. Enquanto este nodo não tiver um `nodo_id` próprio ele será identificado com um valor predefinido (todos bits em nível ‘1’) que é impar (controle). O primeiro byte do campo de dados terá o código de uma requisição de `nodo_id` (serviço P&P);
- Chegando a mensagem no servidor, será identificado que trata-se de uma mensagem de controle (ID impar). Será identificado que é um pacote de requisição de `nodo_id`. O servidor vai consultar na sua tabela de nodos e procurar um `nodo_id` não utilizado para aquele par de *classe e aplicação* recebido;

- Ao descobrir um `nodo_id` não utilizado. O servidor envia um pacote para um endereço pré-definido (todos bits do `nodo_id` em nível '1') uma mensagem de controle com o mesmo par classe e aplicação recebido e com o *novo nodo_id no campo de parametro de controle*;
- O servidor deverá acrescentar a sua tabela de nodos esse novo nodo e deixar uma mensagem de log para o usuário que o nodo foi acrescentado com sucesso e que ele deve inserir os dados relativos aos periféricos (localização de cada periférico) controlados por esse novo nodo;
- Quando o nodo receber essa mensagem, ele vai gravar em uma pequena memória *eeeprom* ou *flash* o seu `nodo_id`, e a partir desse momento este nodo só usará esse `nodo_id`.

Deve-se partir do pressuposto que *somente um nodo na rede será acrescentado por vez* para não gerar colisão de ID, pois inicialmente todos os novos nodos usam o mesmo ID padrão.

9.3 Alive?

Esse Serviço funcionaria como um *Echo Request* no protocolo TCPIP. Sua função é fazer com que o servidor, de tempos em tempos, envie uma mensagem de controle “*keep alive?*” para cada nodo do sistema para verificar se todos estão respondendo.

Essa seria uma forma bem simples de manter a consistência do banco de dados da casa com a vantagem da facilidade de implementação mas como a desvantagem de ser um teste muito superficial.

9.4 Testabilidade Remota

Outra possível abordagem para se solucionar o problema de consistência da rede seria a implementação de rotinas de auto teste (BIST²⁹) acionadas esporadicamente.

Esse teste faria a verificação da configuração do FPGA. A técnica descrita para este serviço é detalhada no artigo [27].

Basicamente, em um FPGA extra existe um módulo que funciona hora em modo de contagem lenta de tempo, hora como um LFSR³⁰ compactador de assinatura [7] e [27].

No modo contador de tempo esse LFSR roda em uma frequência muito baixa, para contar tempos longos (exemplo 12 horas). Quando esse contador atingir 12 horas, ele entrará em **modo**

²⁸ medidas de desempenho precisas do sistema não fazem parte do escopo deste trabalho.

²⁹ *Built-In Self Test*

³⁰ *Liner Feedback Shift Register*

compactador de assinatura, compactando a configuração interna do FPGA alvo. Ao terminar de compactar essa assinatura, o nodo envia para o servidor uma mensagem de controle onde o campo do tipo de controle será o valor relacionado ao auto teste e o parâmetro será a assinatura.

O servidor vai receber esse pacote e comparar a assinatura recebida com a que ele tem armazenada no banco de dados. Se coincidirem, é porque o FPGA está funcionando, caso contrário será inicializado um outro serviço chamado reconfiguração remota. Onde o FPGA é reconfigurado e desta forma voltará a ativa.

Esse sistema foi o escolhido pois é aplicado em sistemas onde o *down-time* é baixo, como no caso de domótica.

O *overhead* de área no FPGA extra introduzido por essa técnica é bem reduzido. Aproximadamente um LFSR e um divisor de *clock*.

Essa abordagem ainda não faz um teste completo do nodo, pois testa somente o FPGA e não testa a placa e seus *chips* de periferia. Uma rotina de teste mais alto nível que abordaria todos os componentes da placa seria ainda necessária para garantir o correto funcionamento do sistema.

9.5 Reconfiguração Remota

Esse serviço é útil sobretudo para manutenção do sistema, pois permite fazer upgrades da aplicação do nodo remotamente.

Para isso o servidor envia uma mensagem de controle identificando um comando de reconfiguração. O campo de parâmetro tem o tamanho de dados para uma nova configuração.

Desta forma o nodo alvo sabe que os próximos pacotes a serem recebidos serão parte de sua nova configuração. A medida que a esses pacotes estão sendo recebidos, o módulo grava-os em uma memória *flash* essa nova configuração. Ao término da recepção o sistema é inicializado, carregando assim a sua nova versão de aplicação.

9.6 Mecanismo de TTL – Time To Live

Durante o envio de pacotes grandes (Seção 8.4), pacotes de dados podem ser perdidos durante a transmissão devido a problemas na comunicação serial.

Devido a este problema, algum mecanismo semelhante ao TTL utilizado em redes deve ser implementado. Desta forma, quando o TTL do pacote ultrapassar o limite máximo, sua entrada na lista de pacotes é removida.

9.7 Controle de acesso Web por múltiplas permissões

Atualmente, o controle de acesso a interface Web ocorre a partir de uma única permissão por usuário. Esta permissão indica se o usuário pode ou não ter acesso a interface Web.

O que está sendo sugerido é que para cada usuário, tenha-se o controle sobre quais as aplicações que o mesmo pode interagir, desta forma, podemos restringir determinadas aplicações sobre usuários específicos. Exemplo: podemos definir que as crianças da residência não tem acesso de controle sobre determinados eletrodomésticos.

10 Conclusões

Através da domótica, está se propondo uma grande quebra de paradigma nos costumes da sociedade, onde pessoas que possuem dificuldades de se dedicarem aos seus lares poderão em um futuro breve controlá-los remotamente através da Internet.

De acordo com o que foi proposto no trabalho, é possível que qualquer pessoa autorizada possa controlar sua residência de qualquer lugar do mundo com custos baixos. Para que isto seja possível, utilizou-se o que há de mais fácil acesso nos dias de hoje, um computador conectado à Internet com um navegador *Web* instalado.

Segundo a arquitetura do sistema proposta no capítulo 0, além do computador Cliente, teríamos a necessidade de um computador Servidor, que estaria localizado diretamente dentro da residência, cuja responsabilidade principal é servir de *gateway* entre o navegador *Web* e as aplicações domésticas encontradas na residência.

Para o desenvolvimento da arquitetura descrita no trabalho, muitos estudos foram feitos sobre várias tecnologias capazes de proporcionar que o objetivo do trabalho fosse alcançado. Devido ao trabalho segmentar-se em duas áreas aparentemente distintas, pesquisas de tecnologias foram separadas na área de *software* e *hardware*.

Como pontos relevantes na área de *software*, citamos a escolha do sistema operacional apropriado a ser utilizado no Servidor, assim como a escolha das linguagens de programação apropriadas para a desenvolvimento do trabalho e que possibilitem facilmente a migração do sistema para outras plataformas de maneira simples, proporcionando rapidamente o crescimento do número de residências controladas remotamente. Neste caso, foram escolhidas as linguagens Java e PHP devido a razões apresentadas durante o desenvolvimento do trabalho.

Na área de *hardware*, o protocolo de comunicação escolhido foi o CAN, pois o mesmo é utilizado em maior número de aplicações, amplamente usado no ramo de automação, possui robustez e controle de erros.

Mecanismos de segurança são prioritários no desenvolvimento de aplicações que envolvem domótica, garantindo neste caso a segurança do sistema em relação as invasões normalmente provenientes de *hackers*.

Atualmente necessitamos de um computador conectado a Internet com apenas um navegador *Web* instalado, no entanto, no futuro prevê-se a utilização de um aparelho celular ou PDA no lugar do computador.

A domótica é uma área que está apenas recentemente começando a ser explorada, ao contrário de outras áreas. Sendo assim é importante o desenvolvimento de pesquisas e trabalhos científicos nesta área, garantindo neste caso que a domótica progrida e se popularize como a Internet se popularizou.

Links Interessantes

- [1] INTEL . Stand-Alone Controller Area Network.
[<http://developer.intel.com/design/auto/82527.htm>].
- [2] NATIONAL . The COP888EB ROM based microcontrollers.
[<http://www.national.com/pf/CO/COP888EB.html>].
- [3] PHILIPS. Controller Area Network – CAN. [<http://www.semiconductors.com/can/>].
- [4] Smart Distributed System - SDS. [<http://content.honeywell.com/sensing/prodinfo/sds/>].
- [5] DALLAS. DS80C390 Dual CAN High-Speed Microprocessor.
[<http://www.dalsemi.com/datasheets/pdfs/80c390.pdf>].
- [6] Motorola. 68HC05X32 Microcontroller Technical Data Book. [<http://mot-sps.com/products/>].
- [7] Motorola. 68HC705X32 Microcontroller Technical Data Book.
- [8] Microchip. MCP2510 Stand-Alone CAN Controller. [<http://www.microchip.com>].
- [9] Texas Instruments. Understanding the CAN Controller on the TMS320C24x DSP Controller. [<http://www.ti.com/sc/docs/psheets/abstract/apps/spra500.htm>].
- [10] CAN in Automation (CiA). [<http://www.can-cia.de/>].
- [11] Bosch CAN Homepage. [<http://www.bosch.de/k8/can/>].
- [12] HurriCANE Home Page. [<ftp://ftp.estec.esa.nl/pub/ws/wsd/CAN/can.htm>].
- [13] Mark J. Sebern. Coleção de artigos comparando sistemas operacionais.
[<http://www.msoc.edu/~sebern/courses/cs384/papers97/>].
- [14] The Source for Java(TM) Technology. [<http://www.java.sun.com>].
- [15] Palm, Inc. Palm VII Connected Organizer.
[<http://www.palm.com/products/palmvii/index.html>].

- [16] Home Automation, Inc. HAI Web-Link Software Homepage. [<http://homeauto.com/web-link/index.htm>].
- [17] Keware Technologies. HomeSeer Software. [<http://www.homeseer.com>].
- [18] Home Automated Living. HAL 2000 Software. [<http://www.automatedliving.com/>].
- [19] MisterHouse. Home Automation Software. [<http://www.misterhouse.net/>].
- [20] Omnipotence Software. Event Control System - ECS. [<http://www.usit.com/omnip/home-automation-1.htm>].
- [21] Al's House. [<http://www.alltel.net/~al/alshouse.html>].
- [22] The Home Automation Forum. [<http://www.homeautomationforum.com/>].
- [23] European Home Systems Association – EHSA.
[<http://www.domotics.com/ehs/EhsTechno.htm>].
- [24] G. Pierre. The European Home Systems Protocol - Concepts and Products. SGS-THOMSON Microelectronics. Rousset, France.
[<http://www.domotics.com/eh-sanda/papers/EHSproto.htm>].
- [25] R. Seyer, M, Stege. System Technology for Private Houses – EHS. Daimler-Benz AG Research and Technology.
[<http://www.domotics.com/eh-sanda/papers/daimlerbenz.htm>].
- [26] *Byte Data Link Controller*. Reference Manual - BDLCRM/AD. Motorola.
[<http://www.maneapc.demon.co.uk/motorola/prgman.htm>].
- [27] Ashenden, Peter. “*The VHDL Cookbook*”. [<ftp://ftp.cs.adelaide.edu.au/pub/VHDL>].
- [28] PHP Hypertext Preprocessor. [<http://www.php.net>].
- [29] Microsoft Corporation. [<http://www.microsoft.com>].
- [30] The MD5 Message - Digest Algorithm. [<http://www.faqs.org/rfcs/rfc1321.html>]
- [31] Java(TM) Communications API. [<http://www.java.sun.com/products/javacomm/index.html>]
- [32] Sun Microsystems. [<http://www.sun.com>]

- [33] The Apache Software Foundation. [<http://www.apache.org>]
- [34] Allaire : HomeSite. [<http://www.allaire.com/homesite/>]
- [35] Client Side State - HTTP Cookies. [http://home.netscape.com/newsref/std/cookie_spec.html]
- [36] JavaServer Pages(TM) Technology. Sun Microsystems.
[<http://www.java.sun.com/products/jsp/index.html>]
- [37] JavaScript Guide. [<http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/>]
- [38] HurriCANE Home Page. [<ftp://ftp.estec.esa.nl/pub/ws/wsd/CAN/can.htm>]
- [39] Xilinx Home Page. [<http://www.xilinx.com>]

Referências Bibliográficas

- [1] F. Baumann; B. Jean-Bart; A. Kung, P. Robin. Eletronic Commerce Services for Home Automation. Trialog. [<http://www.trialog.com/emmsec9-97.pdf>].
- [2] A. Kung; B. Raither. Eletronic Commerce Services Expand Home Automation Capabilities. Trialog. [<http://www.trialog.com/emmsec6-99.pdf>]
- [3] IEEE Standard VHDL language : reference manual. IEEE. 150p.
- [4] Mazor, Stanley; Langstraat, Patricia. “*A guide to VHDL*”. Boston : Kluwer Academic, 1996. 250p.
- [5] Ott, Douglas E.; Wilderotter, Thomas J. “*A designer's guide to VHDL synthesis*”. Boston: Kluwer Academic Press, 1996. 306p.
- [6] Brown, Stephen D.; Francis, Robert J. Field-Programmable Gate Arrays. Kluwer Academic. 1992. 206p.
- [7] Terroso, Anderson Royes. Projeto de arquiteturas tolerantes a falha através da linguagem de descrição de Hardware VHDL. Porto Alegre, 1999. 143f. Dissertação (Mestrado em Engenharia Elétrica). Faculdade de Engenharia Elétrica, PUC-RS.
- [8] Angel, Patricia M.; Fraigi, Liliana B. Introduccion a la Domotica Tomo I. VI Escuela Brasileño Argentina de Informatica. 1993. 172p.
- [9] Miyagi, Paulo E.; Barreto, Marcos P. R. Domótica: Controle e Automação Tomo II. VI Escuela Brasileño Argentina de Informatica. 1993. 108p.
- [10] Brandão, Eduardo Scharnberg. Sistema Cliente-Servidor para Supervisão de Processos através da Web. trabalho de conclusão do curso de informática da UFRGS. fevereiro de 1999. 57p.
- [11] Brown, Bruce. Automação Doméstica. PC Magazine Brasil. Janeiro de 1999. p 61.
- [12] ROBERT BOSCH GmbH. CAN Especification Version 2.0. Stuttgart. 1991. [<http://www.bosch.de/k8/can/docu/can2spec.pdf>]. [<http://www.can-cia.de/CAN20B.pdf>].
- [13] Lawrenz, Wolfhard. CAN System Engineering From Theory to Practical Applications. 1997. 468 pp.
- [14] BRAISZ, H.; SCHUHMANN, N.; GLAUERT, W.. A VHDL Model of a Parameterizable CAN Controller as an Example of a Reuse Library Component. International Workshop on Logic and Architecture Synthesis - IWLAS98. Institut National Polytechnique of Grenoble, France. 15-16 December, 1998. [http://www.e-technik.uni-erlangen.de/~braisz/CAN_IWLAS98_slides.ps].

- [15] EKIZ, H.; KUTLU, A.; POWNER, E.T. Design and Implementation of CAN/CAN Bridge. School of Engineering University of Sussex. International Symposium on Parallel Architectures, Algorithms and Networks – I-SPAN’96.
- [16] Tanenbaum, Andrew S. Redes de Computadores. Rio de Janeiro, Editora Campus, 1994.
- [17] Cornell, Gary; HorstMann, Cay S. Core Java Second Edition. Califórnia, SunSoft Press, 1997.
- [18] Kennedy, Randall C.. Windows NT – Como Integrar o Windows NT com seu Ambiente de Trabalho. Rio de Janeiro, Editora Campus, 1994.
- [19] Probst, Stefan; Flaxa, Ralf. Power Linux – International Edition. Alemanha, Springer, 1997.
- [20] Petersen, Richard. Linux : The Complete Reference. Califórnia, Osborne McGraw-Hill, 1996.
- [21] Parker, Tim. Linux System Administrator’s Survival Guide. Indiana, Sams Publishing, 1996.
- [22] Schwartz, Randal L.; Christiansen, Tom. Learning Perl. Califórnia, O’Reilly & Associates Inc., 1997.
- [23] Dewire, Dawna Travis. Client/Server Computing. Califórnia, McGraw-Hill, 1993.
- [24] Thomas, Rebecca; Yates, Jean. Unix Total. McGraw-Hill, 1988.
- [25] Wigglesworth, Joe; Lumby, Paula – Java Programming Advanced Topics. Thomson Learning, 2000.
- [26] Using the Virtex Block SelectRAM+ Features (XAPP130). Virtex Series.
- [27] Eduardo Bezerra, Fabian Vargas, Michael Paul Gough. Merging BIST and Configurable Computing Technology to Improve Availability in Space Applications. 1st IEEE Latin-American Test Workshop. Mar. LATW’2000. 146-151.
- [28] Virtex 2.5V Field Programmable Gate Arrays (DS003). Virtex Series.
- [29] The Virtual WorkBench Guide. Virtual Computer Corporation. [<http://www.vcc.com>].
- [30] XS40 Board V1.4 User Manual . XESS Corporation. [http://www.xess.com/manuals/xs40-manual-v1_4.pdf].

1. INSTALAÇÃO DO SISTEMA DE *HARDWARE*

- Carregar na placa VW300 usando foundation o arquivo master.bit;
- Carregar na placa XS40 usando xsload o arquivo lamp.bit;
- Curto circuitar o terra das placas;
- Curto circuitar os pinos 66 e 69 (J18) da placa XS40 com os pinos 21 e 23 (JP49) da placa VW300;
- Conectar o extensor serial no servidor e na placa VW300;

2. ALTERAÇÃO DO NÚMERO DE LÂMPADAS

- Alterar o arquivo lamp.ucf com o número de lâmpadas requisitado;
- Alterar a linha 187 do arquivo lampApplic.vhd de forma a suportar todas as lâmpadas. (SwitchChangeXor <= SwitchChange(0) ... SwitchChange(x));
- Alterar a linha 159 do arquivo canpckgs.vhd com o novo numero de lampadas. (NRO_LAMPS = x);
- Alterar a linha 161 do arquivo canpckgs.vhd com o número de bytes necessário para representar o novo número de lâmpadas. (CLDC = x);
- Usando o programa Foundation, gere um novo arquivo de configuração (.bit);
- Carregue o arquivo .bit gerado com o programa xsload.

3. INSTALAÇÃO DO SOFTWARE NO SERVIDOR

Passos para a instalação do software em Java:

- Instalação do JDK 1.2.2 ou JRE1.2.2 [32];
- Instalação do pacote CommApi [31];
- Descompactar o banco de dados (database.zip) para o disco rígido;

- Criação de driver ODBC denominado **tc**, apontando para o banco de dados Access;
- Descompactar arquivos do software (serial.zip) para o disco rígido;
- Vide Item 4 para configuração do software em Java;
- Compilar o software : javac *.java
- Executar o software : java Serial

Passos para a instalação da interface cliente :

- Instalação do servidor web Apache 1.3.14 [33];
- Instalação do módulo PHP [28];
- Descompactar arquivos da interface cliente (cliente.zip) para o disco rígido (descompactar para dentro do diretório **htdocs** do servidor web Apache)
- Com o servidor web rodando, acessar a *homepage* da residência utilizando o navegador Internet Explorer 4.0 ou superior. O endereço da *homepage* para acessos é :
http://ENDERECO_IP_DO_SERVIDOR/tc/
- O sistema possui dois usuários padrões:

Usuário : junior

Senha : junior

Usuário : amory

Senha : amory

4. CONFIGURAÇÃO DO SOFTWARE EM JAVA

- Driver de banco de dados

Atualmente o sistema utiliza para comunicação com o sistema de banco de dados o mecanismo JDBC-ODBC, para que este mecanismo seja modificado no futuro, deve-se alterar as duas linhas abaixo que encontram-se no arquivo BancodeDados, linhas 15 e 16.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
conTemp = DriverManager.getConnection("jdbc:odbc:domotica","","");
```

- Monitoração do banco de dados

No software está definido que a leitura da base de dados irá ocorrer de 5 em 5 segundos, conforme pode ser visto na linha 35 (Thread.sleep(5000)) :

```
while(true) {
    monitoraMestre();
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {}
}
```

- Configuração da porta serial

Para que a porta serial seja alterada, deve-se modificar o código fonte do programa em dois lugares :

Arquivo SerialWrite.java

Linha 11 : serialPort = ConectaSerial.getSerial("COM1");

Arquivo SerialRead.java

Linha 38 : serialPort = ConectaSerial.getSerial("COM1");

Configurações extras como por exemplo a velocidade de comunicação, paridade, etc, também devem ser realizadas em dois arquivos :

Arquivo SerialWrite.java

A partir da linha 17 :

```
serialPort.setSerialPortParams(19200,
    SerialPort.DATABITS_8,
    SerialPort.STOPBITS_1,
    SerialPort.PARITY_NONE);
```

Arquivo SerialRead.java

A partir da linha 50 :

```
serialPort.setSerialPortParams(19200,
    SerialPort.DATABITS_8,
    SerialPort.STOPBITS_1,
    SerialPort.PARITY_NONE);
```

- Salvamento de arquivos enviados através da porta serial

Dependendo da aplicação doméstica, muitas vezes se faz necessário que arquivos sejam salvos no disco rígido, como por exemplo uma imagem. Nesta situação deve-se configurar o local onde estes arquivos serão salvos.

Para que isto seja realizado, no arquivo Serial.java, linha 13, configuramos este caminho :

```
public static final String DIRETORIO_DESTINO="c:\\program files\\apache
group\\apache\\htdocs\\domotica\\";
```

5. INSTALAÇÃO DO SOFTWARE DE DEPURAÇÃO

- Instalação do JDK 1.2.2 ou JRE1.2.2 [32];
- Instalação do pacote CommApi [31];
- Vide Item 6 para configuração do software de depuração;
- Compilar o software de depuração : javac *.java
- Executar o software de depuração : java DebugAwt

6. CONFIGURAÇÃO DO SOFTWARE DE DEPURAÇÃO

O depurador está configurado para funcionar utilizando à "COM1", com velocidade de comunicação de 19200 bps, sem paridade, stop bits em 1 e databits em 8.

Para modificar a porta serial a ser utilizada ou modificar demais configurações como por exemplo velocidade, deve-se modificar o depurador (código fonte).

- Abaixo listamos os arquivos que devem ser modificados para mudar a configuração do depurador :

SerialWrite.java → linha 11

SerialRead.java → linha 38

SerialReadDebug.java → linha 185

Os trechos de código a serem alterados são os seguintes :

```
serialPort = ConectaSerial.getSerial("COM1");
```

Logo abaixo da linha acima, existe o seguinte trecho de código nos três arquivos :

```
try {
    serialPort.setSerialPortParams(19200,
        SerialPort.DATABITS_8,
        SerialPort.STOPBITS_1,
        SerialPort.PARITY_NONE);
}
catch (UnsupportedCommOperationException e) {}
```

Neste trecho de código está configurada por exemplo a velocidade de comunicação, entre outras configurações. É importante salientar que se alguma destas linhas sejam alteradas, é necessário que as alterações sejam refletidas nos três arquivos.

Como utilizar o depurador :

ENVIO DE PACOTES

Deve ser digitado os valores para a *classe*, *aplicação*, *nodo id*, selecionar se o RTR deve ou não ser enviado (checkbox). Digitar o *DLC*, e no campo *DADO* colocar os dados propriamente ditos. Clicar no botão ENVIAR quando deseja-se que os dados sejam enviados através da porta serial.

Exemplos de como usar o depurador:

Se quisermos enviar 3 bytes, colocar o valor 3 no campo de *DLC* e no campo *DADO* digitar os valores dos 3 bytes. O valor de cada byte deve ser identificado através de **um espaço em branco** entre os bytes. O byte mais significativo fica à esquerda e o menos significativo fica à direita.

Exemplo → Dado : 0 0 3 → estou enviando a seqüência de bytes 0 0 3.

Exemplo → Dado :1 4 90 → estou enviando a seqüência de bytes 1 4 90.

Para enviar um pacote grande por exemplo :

Colocar o valor 10 no campo *DLC*

No campo *DADO* colocar os seguintes valores por exemplo: 1 2 3 4 5 6 7 8 9 0

Pacotes grandes podem ser enviados a partir da leitura de arquivos, utilizando o botão **arquivo** (característica exclusiva da versão J++ do Depurador).

RECEPÇÃO DE PACOTES :

Todas as informações dos pacotes que chegam através da serial são impressas no componente *TEXTAREA*.