

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE  
CENTRO DE CIÊNCIAS E TECNOLOGIA  
CURSO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

## **DISSERTAÇÃO DE MESTRADO**

COMUNICAÇÃO DE DADOS PARA UM SISTEMA DE  
TELEMETRIA DE BAIXO CUSTO

**Maurício Marinho Formiga**

**Elmar Uwe Kurt Melcher  
Joseana Macêdo Fechine  
(Orientadores)**

Campina Grande – PB  
Maio – 2005

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE  
CENTRO DE CIÊNCIAS E TECNOLOGIA  
CURSO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

COMUNICAÇÃO DE DADOS PARA UM SISTEMA DE  
TELEMETRIA DE BAIXO CUSTO

**Maurício Marinho Formiga**

**Elmar Uwe Kurt Melcher**  
**Joseana Macêdo Fchine**  
**(Orientadores)**

Dissertação submetida à Coordenação do Curso de Pós-graduação em Informática da Universidade Federal de Campina Grande, como parte dos requisitos necessários para obtenção do grau de mestre em Informática.

**Área de concentração:** Ciência da Computação.

Campina Grande – PB  
Maio – 2005

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

F725c Formiga, Maurício Marinho

Comunicação de dados para um sistema de telemetria de baixo custo/Maurício Marinho Formiga. - Campina Grande, 2005.

146f. il.

Inclui bibliografia.

Dissertação (Mestrado em Informática) – Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia.

Orientadores: Elmar Uwe Kurt Melcher e Joseana Macedo Fachine.

1 – Telemetria – Computação 2 – Monitoramento – Computação 3 – Telemetria – Computação de Dados I – Título

CDU 681.326.7

**“COMUNICAÇÃO DE DADOS PARA UM SISTEMA DE TELEMETRIA DE  
BAIXO CUSTO”**

**MAURÍCIO MARINHO FORMIGA**

**DISSERTAÇÃO APROVADA EM 31.05.2005**



**PROF. ELMAR UWE KURT MELCHER, Dr.  
Orientador**



**PROF<sup>a</sup> JOSEANA MACÊDO FECHINE , D.Sc  
Orientadora**



**PROF. WOLFREDO DA COSTA CIRNE FILHO, Ph.D  
Examinador**



**PROF. KEPLER BORGES FRANÇA, Ph.D  
Examinador**



**PROF. LEONARDO VIDAL BATISTA, D.Sc  
Examinador**

**CAMPINA GRANDE – PB**

“A vida é um poço infinito,  
quanto mais se sabe,  
mais se compreende que há muito para se  
aprender”

(Charles Chaplin)

## **Agradecimentos**

A Deus, criador e mantenedor de todas as coisas.

À minha mãe, Anazilda, pelo apoio não só no período da Pós-Graduação, mas também em todos os momentos importantes da minha vida.

Aos meus orientadores, Elmar e Joseana, pelo bom direcionamento do trabalho e por conseguirem conciliar tão bem a exigência de um trabalho de boa qualidade com o reconhecimento do mesmo.

Ao professor Kepler, coordenador do LABDES (onde foi montada a estrutura de desenvolvimento do projeto), pela oportunidade que me foi dada desde o período de Graduação para desenvolver o sistema.

Ao meu parceiro no início do projeto, Osman, aluno de Engenharia Elétrica, pela contribuição em alguns pontos da estrutura do sistema não diretamente relacionados com a minha área, porém fundamentais para a concretização do trabalho.

## Resumo

Muitos tipos de equipamentos podem apresentar problemas de operação devido à falta de monitoração adequada ocasionando altos custos de manutenção. O processo de gerenciamento que minimiza esses problemas pode ser realizado através de uma manutenção preditiva feita a partir de um sistema integrado de telemetria, controle e gestão de alarmes, que possibilite a verificação remota do desempenho das estações monitoradas. Este trabalho é referente ao desenvolvimento de uma tecnologia que utiliza computador embutido (embarcado) com poucos recursos de *hardware*, de forma a viabilizar a monitoração automática de equipamentos, distantes ou não da estação de controle, sem incrementar substancialmente os custos operacionais e de manutenção dos mesmos. O modelo implementado no sistema de telemetria aplica compressão de dados, criptografia e alguns requisitos de tratamento de falhas que tornam possível a utilização de um dispositivo embutido com poucos recursos (microcontrolador) para a coleta, armazenamento e transmissão de dados de uma forma eficiente, segura e correta.

## Abstract

Many types of equipment can present operational problems due to the lack of adequate monitoring, causing high maintenance costs. The management process that reduce those problems can be carried out through a predictive maintenance using an integrated telemetry system capable of sending alarms and monitoring the performance of the remot sites. This work is about the development of a technology that uses embedded computers with few hardware resources to automatically monitor equipment, distant or not from the controlling station, without increasing the operational and maintenance costs substantially. The telemetry system implementation applies data compression, cryptography and failure treatment and uses a low cost microcontroller for collection, storage and transmission of data in an efficient, secure and correct way.



## Índice

<b>1.1</b>	<b>Conceitos .....</b>	<b>16</b>
<b>1.2</b>	<b>Motivação.....</b>	<b>17</b>
<b>1.3</b>	<b>Objetivos .....</b>	<b>18</b>
1.3.1	OBJETIVOS ESPECÍFICOS.....	19
<b>1.4</b>	<b>Relevância .....</b>	<b>20</b>
<b>1.5</b>	<b>Organização da dissertação.....</b>	<b>21</b>
<b>2.</b>	<b>DESCRIÇÃO DO SISTEMA DE TELEMETRIA .....</b>	<b>23</b>
<b>2.1</b>	<b>Contexto da pesquisa .....</b>	<b>23</b>
<b>2.2</b>	<b>Projeto de sistemas de dessalinização.....</b>	<b>25</b>
<b>2.3</b>	<b>Manutenção dos equipamentos.....</b>	<b>28</b>
<b>2.4</b>	<b>Requisitos relacionados ao computador embutido .....</b>	<b>30</b>
<b>2.5</b>	<b>Projeto e implementação do módulo de transferência de informação .....</b>	<b>34</b>
<b>2.6</b>	<b>Considerações finais.....</b>	<b>37</b>
<b>3.</b>	<b>FUNDAMENTAÇÃO TEÓRICA.....</b>	<b>39</b>
<b>3.1</b>	<b>Sistemas embutidos .....</b>	<b>39</b>
3.1.1	REQUISITOS DE PROJETO .....	40
3.1.2	COMPONENTES BÁSICOS .....	41
3.1.3	UCP PARA SISTEMAS EMBUTIDOS .....	44
3.1.4	CISC, RISC OU SISC.....	46
3.1.5	UTILIZAÇÃO DE MICROCONTROLADORES.....	47
<b>3.2</b>	<b>Compressão de dados.....</b>	<b>50</b>
3.2.1	MODELAGEM .....	52
3.2.2	MODELOS DE CONTEXTO FINITO .....	54
3.2.3	ALGORITMOS .....	55
<b>3.3</b>	<b>Criptografia .....</b>	<b>60</b>
3.3.1	ALGORITMOS .....	61
3.3.2	MODELO DE CRIPTOGRAFIA CONVENCIONAL (SIMÉTRICA).....	62
3.3.3	SISTEMAS DE CHAVE PÚBLICA .....	63
3.3.4	ALGORITMOS PADRÃO DE CRIPTOGRAFIA .....	64
<b>3.4</b>	<b>Protocolo de comunicação .....</b>	<b>67</b>
3.4.1	ORGANIZAÇÕES INTERNACIONAIS DE PADRONIZAÇÃO .....	67
3.4.2	O MODELO OSI E SUAS CAMADAS .....	68
3.4.3	DETECÇÃO DE ERROS.....	69
<b>3.5</b>	<b>Considerações finais sobre os algoritmos apresentados .....</b>	<b>71</b>
<b>4.</b>	<b>RESULTADOS OBTIDOS.....</b>	<b>73</b>
<b>4.1</b>	<b>Ambiente de desenvolvimento.....</b>	<b>73</b>
<b>4.2</b>	<b>Modelo de implementação do módulo embutido.....</b>	<b>76</b>

<b>4.3</b>	<b>Computador embutido.....</b>	<b>78</b>
4.3.1	INTERFACES DO MICROCONTROLADOR .....	80
4.3.2	PROGRAMAÇÃO DO PIC EM LINGUAGEM C .....	83
<b>4.4</b>	<b>Compressão.....</b>	<b>84</b>
4.4.1	MODELO 1: ALGORITMO DE HUFFMAN NO SERVIDOR .....	85
4.4.2	MODELO 2: ALGORITMO HÍBRIDO HUFFMAN/RLE DE ZEROS NO SERVIDOR .....	91
4.4.3	ANÁLISE DOS MODELOS IMPLEMENTADOS.....	95
<b>4.5</b>	<b>Criptografia .....</b>	<b>101</b>
<b>4.6</b>	<b>Protocolo de Comunicação .....</b>	<b>105</b>
4.6.1	DESCRIÇÃO DAS ETAPAS .....	106
4.6.2	TRATAMENTO DE FALHAS NA COMUNICAÇÃO .....	110
<b>4.7</b>	<b>Testes finais.....</b>	<b>111</b>
<b>4.8</b>	<b>Orçamento para instalação de um sistema de monitoração remota .....</b>	<b>116</b>
<b>4.9</b>	<b>Considerações finais.....</b>	<b>118</b>
<b>5.</b>	<b>CONCLUSÕES E SUGESTÕES .....</b>	<b>121</b>
5.1	Sugestões para trabalhos futuros.....	123
	<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>125</b>
	<b>APÊNDICE A .....</b>	<b>130</b>
	<b>APÊNDICE B .....</b>	<b>132</b>
	<b>APÊNDICE C .....</b>	<b>137</b>
	<b>APÊNDICE D.....</b>	<b>140</b>

## Lista de Figuras

<b>FIGURA 2.1: ESQUEMÁTICO DE UM DESSALINIZADOR .....</b>	<b>25</b>
<b>FIGURA 2.2: REPRESENTAÇÃO DO PROCESSO DE OSMOSE INVERSA .....</b>	<b>26</b>
<b>FIGURA 2.3: ESQUEMÁTICO DO SISTEMA DE TELEMETRIA .....</b>	<b>29</b>
<b>FIGURA 2.4: ESTRATÉGIA DE MANUTENÇÃO DE SISTEMAS DE DESSALINIZAÇÃO .....</b>	<b>30</b>
<b>FIGURA 2.5: AMOSTRA DE MEDIDAS COLETADAS DE SENSORES CONECTADOS AO EQUIPAMENTO .....</b>	<b>35</b>
<b>FIGURA 2.6: SISTEMA DE DESSALINIZAÇÃO PILOTO (SDP).....</b>	<b>36</b>
<b>FIGURA 2.7: CONDIÇÕES DE ALARME DO SISTEMA DE DESSALINIZAÇÃO .....</b>	<b>37</b>
<b>FIGURA 3.1: COMPUTADOR EMBUTIDO TÍPICO .....</b>	<b>45</b>
<b>FIGURA 3.2: MICROCONTROLADOR PIC16F877 .....</b>	<b>48</b>
<b>FIGURA 3.3: EXEMPLO DE CODIFICAÇÃO DE HUFFMAN .....</b>	<b>57</b>
<b>FIGURA 4.1: ESTAÇÃO DE GERÊNCIA .....</b>	<b>73</b>
<b>FIGURA 4.2: AMBIENTE DE SIMULAÇÃO E PROGRAMAÇÃO DO COMPUTADOR EMBUTIDO.....</b>	<b>74</b>
<b>FIGURA 4.3: TELA DE MONITORAÇÃO DE ENTRADA E SAÍDA DOS DADOS DA PORTA SERIAL .....</b>	<b>75</b>
<b>FIGURA 4.4: AMBIENTE DE DESENVOLVIMENTO DA APLICAÇÃO DO COMPUTADOR EMBUTIDO .....</b>	<b>75</b>
<b>FIGURA 4.5: SISTEMA EMBUTIDO CONECTADO AOS SENSORES DO DESSALINIZADOR.....</b>	<b>76</b>
<b>FIGURA 4.6: DIAGRAMA EM BLOCOS DO MODELO PROPOSTO PARA O COMPUTADOR EMBUTIDO .....</b>	<b>76</b>
<b>FIGURA 4.7: TABELA DE VARIÁVEIS DE MEDIDA .....</b>	<b>78</b>
<b>FIGURA 4.8: ESQUEMA INTERNO DO MICROCONTROLADOR PIC16F877 .....</b>	<b>79</b>
<b>FIGURA 4.9: MODEM DE8661 .....</b>	<b>80</b>
<b>FIGURA 4.10: CONFIGURAÇÃO DOS PINOS DO MICROCONTROLADOR.....</b>	<b>81</b>
<b>FIGURA 4.11: FORMATO DA TABELA DE CODIFICAÇÃO DO MODELO 1 .....</b>	<b>88</b>
<b>FIGURA 4.12: SEQÜÊNCIA DE BITS DA TABELA DE CODIFICAÇÃO.....</b>	<b>89</b>
<b>FIGURA 4.13: FORMATO DA TABELA DE CODIFICAÇÃO DO MODELO 2 .....</b>	<b>93</b>

<b>FIGURA 4.14: ESTATÍSTICA DE COMPILAÇÃO DO MODELO 1 .....</b>	<b>96</b>
<b>FIGURA 4.15: ESTATÍSTICA DE COMPILAÇÃO DO MODELO 2 .....</b>	<b>97</b>
<b>FIGURA 4.16: RELATÓRIO DA IMPLEMENTAÇÃO DA VERSÃO 1 DO RIJNDAEL</b>	<b>102</b>
<b>FIGURA 4.17: COMUNICAÇÃO VIRTUAL ENTRE AS CAMADAS DO PROTOCOLO .....</b>	<b>106</b>
<b>FIGURA 4.18: SIMULAÇÃO DE ALARME.....</b>	<b>111</b>
<b>FIGURA 4.19: PRIMEIRA CONEXÃO ENTRE EMBUTIDO E SERVIDOR .....</b>	<b>112</b>
<b>FIGURA 4.20: ENVIO DA TABELA DE CODIFICAÇÃO.....</b>	<b>113</b>
<b>FIGURA 4.21: COLETA DE AMOSTRA .....</b>	<b>114</b>
<b>FIGURA 4.22: TRANSMISSÃO DE MEDIDAS CODIFICADAS.....</b>	<b>115</b>
<b>FIGURA B.1: MATRIZ DE ESTADO E MATRIZ DE CHAVE PARA <math>NB = 6</math> E <math>NK = 4</math></b>	<b>132</b>
<b>FIGURA B.2: TRANSFORMAÇÃO <i>SHIFTR</i> EM UM ESTADO .....</b>	<b>134</b>
<b>FIGURA B.3: TRANSFORMAÇÃO <i>ADDROUNDKEY</i> .....</b>	<b>135</b>
<b>FIGURA B.4: AS ETAPAS DO RIJNDAEL.....</b>	<b>136</b>
<b>FIGURA C.1: EFICIÊNCIA DO CANAL.....</b>	<b>139</b>

## Lista de Tabelas

<b>TABELA 2.1: VARIÁVEIS DE MEDIDAS DO DESSALINIZADOR .....</b>	<b>36</b>
<b>TABELA 3.1: PRINCIPAIS CARACTERÍSTICAS DE ALGUNS MICROCONTROLADORES PIC .....</b>	<b>50</b>
<b>TABELA 3.2: MEMÓRIA NECESSÁRIA PARA DIVERSAS ORDENS DE MODELOS DE CONTEXTO FINITO .....</b>	<b>55</b>
<b>TABELA 3.3: PROBABILIDADE E FAIXA DOS SÍMBOLOS PARA A CODIFICAÇÃO ARITMÉTICA .....</b>	<b>58</b>
<b>TABELA 3.4: CODIFICAÇÃO ARITMÉTICA.....</b>	<b>58</b>
<b>TABELA 3.5: ALGORITMOS CRIPTOGRÁFICOS DE CHAVE SIMÉTRICA COMUNS .....</b>	<b>63</b>
<b>TABELA 3.6: MEMÓRIA MÍNIMA REQUERIDA PARA IMPLEMENTAÇÃO EM SMART CARDS .....</b>	<b>66</b>
<b>TABELA 4.1: CÓDIGOS RESULTANTES DO MODEM.....</b>	<b>82</b>
<b>TABELA 4.2: CONEXÃO DOS SENSORES COM O MICROCONTROLADOR.....</b>	<b>83</b>
<b>TABELA 4.3: CÓDIGO DE HUFFMAN DO MODELO 1 .....</b>	<b>87</b>
<b>TABELA 4.4: TABELA DE CODIFICAÇÃO DO MODELO 1.....</b>	<b>89</b>
<b>TABELA 4.5: CÓDIGO DAS DIFERENÇAS ARMAZENADAS DO MODELO 1.....</b>	<b>90</b>
<b>TABELA 4.6: CÓDIGO DE HUFFMAN PARA ZEROS DO MODELO 2.....</b>	<b>93</b>
<b>TABELA 4.7: CÓDIGO DE HUFFMAN PARA DIFERENÇAS DO MODELO 2 .....</b>	<b>93</b>
<b>TABELA 4.8: TABELA DE CODIFICAÇÃO DO MODELO 2.....</b>	<b>94</b>
<b>TABELA 4.9: CÓDIGO DAS DIFERENÇAS ARMAZENADAS DO MODELO 2.....</b>	<b>95</b>
<b>TABELA 4.10: UTILIZAÇÃO DOS RECURSOS DO MICROCONTROLADOR .....</b>	<b>98</b>
<b>TABELA 4.11: TRÊS OPÇÕES DE PREÇO DO CLP COMPLETO: CPU, CARTÕES DE ENTRADA E SAÍDA, FONTE, RACK .....</b>	<b>116</b>
<b>TABELA A.1: TABELA COMPARATIVA DAS POSSÍVEIS SOLUÇÕES PARA O SISTEMA DE TELEMETRIA .....</b>	<b>131</b>
<b>TABELA B.1: NÚMERO DE VOLTAS EM FUNÇÃO DO TAMANHO DO BLOCO E DA CHAVE .....</b>	<b>133</b>
<b>TABELA B.2: PARÂMETROS PARA A QUANTIDADE DE BYTES DESLOCADOS NO SHIFTRW.....</b>	<b>134</b>

## Lista de Abreviaturas

AES – *Advanced Encryption Standard*  
ARM – *Advanced RISC Machine*  
BER – *Bit Error Rate*  
BIOS – *Basic Input/Output System*  
CISC – *Complex Instruction Set Computer*  
CLP – *Controlador Lógico Programável*  
CRC – *Cyclic Redundancy Code*  
DES – *Data Encryption Standard*  
DMA – *Direct Memory Access*  
DOS – *Disk Operating System*  
DSPs – *Digital Signal Processors*  
EEPROM – *Electrically Erasable Programmable Read-Only Memory*  
EIDE – *Enhanced Integrated Drive Electronics*  
IRQ – *Interrupt ReQuest*  
ISO – *International Organization for Standardization*  
JPEG – *Join Photographic Expert Group*  
LABDES – *Laboratório de Referência em Dessalinização*  
MPEG – *Moving Picture Experts Group*  
NIST – *National Institute of Standard and Technology*  
NSA – *National Security Agency*  
OSI – *Open Systems Interconnection*  
PDA – *Personal Digital Assistant*  
PC – *Personal Computer*  
PLC – *Power Line Communication*  
PPM – *Prediction by Partial Matching*  
PTT – *Push-to-talk*  
PWM – *Pulse Width Modulation*  
QAM – *Quadrature Amplitude Modulation*  
RAM – *Randomic Access Memory*  
RISC – *Reduced Instructions Set Computer*  
RLE – *Run-Length Encoding*

RM-OSI – *Reference Model Open Systems Interconnection*

ROM – *Read Only Memory*

RTOS – *Real-time Operating System*

SDRAM – *Synchronous Dynamic Random Access Memory*

SISC – *Specific Instruction Set Computer*

SO-DIMM – *Small Outline Dual In-line Memory Module*

UCPs – *Unidades Centrais de Processamento*

USART – *Universal Synchronous Asynchronous Receiver Transmitter*

USB – *Universal Serial Bus*

# Capítulo 1

## 1. Introdução

---

Neste capítulo serão apresentadas as motivações que levaram ao desenvolvimento desse trabalho, introduzindo os temas referentes ao sistema de telemetria de baixo custo. Em seguida, serão apresentados os objetivos gerais e específicos, algumas questões referentes à relevância do trabalho e, por fim, a descrição da estrutura do restante do documento da Dissertação.

### 1.1 Conceitos

Telemetria é a transferência (via rede fixa ou sem fio) e utilização de dados provenientes de múltiplas máquinas remotas, com sensores de captação de dados e/ou medidas, distribuídas em uma área geográfica de forma pré-determinada, para a sua monitoração, medição e controle [Bonde, 00].

As empresas têm um custo significativo para monitorar os seus equipamentos manualmente. Custos adicionais também incidem quando as máquinas e sensores falham. Um sistema de telemetria pode solucionar esses problemas e reduzir substancialmente os custos operacionais. A telemetria também pode proporcionar alarmes parametrizados por ocorrência para tomada de decisão ou prevenção.

A eficiência da solução de telemetria aumenta ainda mais dependendo do tipo de informação coletada e enviada, garantindo qualidade, rastreabilidade e eficácia. A geração de alarmes e a monitoração em intervalos de tempo menores ajudam na operação e manutenção de sistemas e na detecção das razões do funcionamento inadequado de uma determinada máquina permitindo, por exemplo, que sejam enviados os profissionais mais adequados para solucionar os problemas [M2M, 03].

Um sistema embutido (também é utilizado o termo “sistema embarcado”) consiste, normalmente, de um conjunto de componentes de *hardware* e de *software* que interagem para executar um conjunto de operações prescritas. Aplicações embutidas



incluem sistemas aeroespaciais, sistemas de controle de automóveis, sistemas multimídia, telecomunicações (incluindo telefones móveis), controladores industriais, dentre outros. De fato, qualquer equipamento de aplicação específica que requer alguma forma de “inteligência” interna está incluído. Normalmente, os componentes dos sistemas embutidos trabalham juntos para conseguir o comportamento especificado, considerando vários critérios de projeto, incluindo custo, consumo de energia, desempenho, etc. [Ernst, 98].

Arquiteturas típicas para sistemas embutidos complexos consistem de um ou mais processadores, memórias, co-processadores, transdutores de entrada-saída, bem como *software* de aplicação e de sistema [Flynn, 97]. No passado, tais sistemas teriam sido implementados utilizando-se múltiplos circuitos integrados. Hoje, com o incremento da complexidade dos circuitos integrados, é possível construir um sistema embutido completo sobre um único circuito integrado – o chamado “*system on a chip*” (SoC) [Lavagno, 03].

## 1.2 Motivação

Os sistemas de telemetria disponíveis no mercado que utilizam sistemas embutidos são normalmente caros e voltados para aplicações específicas. O custo está normalmente associado à alta precisão requerida para os equipamentos de monitoração e controle e ao fato das tecnologias serem tipicamente importadas (ver Plataformas para sistemas de telemetria no Apêndice A) [Bonde, 00].

Em sistemas de telemetria, considerando-se a relação custo/benefício de várias alternativas relacionadas à transmissão de dados (linha telefônica, rádio, satélite, etc), é possível verificar o alto custo de soluções envolvendo sistemas *on-line* de comunicação em grandes distâncias, nos quais as estações que estão sendo medidas estão conectadas permanentemente com a estação de controle que recebe as medidas de todos os equipamentos. Assim, pode ser necessário o armazenamento das medidas antes de enviá-las em intervalos de tempo pré-determinados. Computadores embutidos utilizados em sistemas de telemetria de baixo custo provavelmente possuem recursos escassos em relação ao tamanho de memória disponível. Isto pode inviabilizar a utilização dos mesmos, dependendo do limite da quantidade de informações possíveis para o armazenamento e das necessidades do projeto. Além disso, dependendo dos requisitos do

sistema, pode ser necessário implementar gestão de alarmes, mecanismos de segurança envolvendo criptografia ou implementações de requisitos para tratamento de falhas no armazenamento e na transmissão de dados. Todos esses possíveis requisitos de um sistema de telemetria tornam, portanto, o uso eficiente da memória no sistema embutido, acoplado à estação monitorada, uma questão que necessita de estudo mais aprofundado.

Em alguns projetos, a viabilidade da implantação de um serviço de telemetria requer que o mesmo não incremente substancialmente os custos de produção dos equipamentos monitorados. É importante destacar também que, em alguns casos, a precisão requerida para os equipamentos de monitoração e controle não é muito alta. É necessário atender uma ampla gama de aplicações para as quais os requisitos de precisão dos equipamentos de monitoração e controle são menos estritos, possibilitando a redução do custo do sistema.

Portanto, torna-se útil o desenvolvimento de tecnologias alternativas que viabilizem a automação dos equipamentos a baixo custo. Dessa forma, o desenvolvimento de sistemas de telemetria de baixo custo, utilizando como solução uma implementação embutida, pode trazer um impacto bastante positivo em diversos setores produtivos que atualmente não usufruem os benefícios da utilização de automação. Portanto, o desenvolvimento desse tipo de sistema se constitui um interessante nicho de mercado, bem como um tema de pesquisa de relevância que abrange aspectos de *hardware* e de *software* com vistas a uma implementação eficiente de baixo custo.

### **1.3 Objetivos**

Este trabalho tem como objetivo central o desenvolvimento da comunicação de dados de um sistema de telemetria com requisitos de baixo custo. Para tanto, será implementada uma solução embutida, a ser aplicada em dessalinizadores utilizados no Projeto Água Doce [Franca, 03], que visa o desenvolvimento, implantação e monitoração desses equipamentos a baixo custo. Este projeto dispõe de centenas de dessalinizadores espalhados, em sua maioria, pelo sertão nordestino.

### 1.3.1 Objetivos Específicos

Os objetivos específicos desse trabalho podem ser descritos, em linhas gerais, da seguinte forma:

- Implementação de método de compressão de dados que proporcione o uso eficiente da memória disponível do computador embutido no armazenamento das informações coletadas.

São utilizados microcontroladores como computadores embutidos responsáveis por coletar, armazenar e enviar medidas dos equipamentos, através da linha telefônica, para a estação de controle. Os microcontroladores, apesar de possuírem praticamente todos os recursos necessários para realizar o processo de monitoração, são limitados no que se refere à quantidade de memória. Para armazenar as medidas em um espaço de memória restrito, é necessário um processo de compressão que possibilite a representação dos dados de forma mais compacta. A análise da eficiência da técnica de compressão está diretamente relacionada com a utilização de memória durante o processo de codificação e também com o resultado obtido após a compressão dos dados.

Com a utilização de algoritmos de compressão, espera-se obter um aproveitamento eficiente dos recursos limitados do microcontrolador para que seja possível atender aos requisitos do sistema de monitoração. Também se espera obter uma redução no tempo de transmissão da informação, minimizando a probabilidade de eventuais erros nos dados e o custo de utilização do meio de transmissão utilizado (linha telefônica).

- Implementação de mecanismo de criptografia para minimizar a possibilidade de alteração ou leitura indevida de informações por intrusos durante a comunicação [Stallings, 03].

O conjunto de atividades relacionadas à criptografia objetiva a autenticidade e a privacidade da informação. A primeira atividade, referente à autenticidade, visa garantir a identidade de *quem enviou* a mensagem; a segunda procura assegurar a identidade de *quem recebeu* a mensagem, ou seja, garantir que a mensagem não será entendida por um intruso que, de alguma forma, consiga lê-la no canal de comunicação [Assis, 03].

O mecanismo de criptografia implementado visa proporcionar um nível de segurança mínimo pré-estabelecido e que satisfaça aos requisitos de memória estrita do computador embutido.

- Implementação de um protocolo de comunicação que possa garantir a transmissão dos dados coletados de uma forma correta e eficiente.

Com a eficiência do protocolo, espera-se obter um tempo de transmissão menor, diminuindo a probabilidade de erros na transmissão [Tanenbaum, 03].

Dentro do contexto da implementação, deve ser considerado o tratamento de falhas, de forma a garantir que os dados coletados sejam armazenados e transmitidos corretamente. Ou seja, o sistema deve estar preparado para se recuperar de possíveis problemas ocorridos na realização da transferência de informações. Esses problemas podem ocorrer, entre outros motivos, devido a quedas ou erros na linha de transmissão.

- Implementação das funcionalidades da telemetria, coletadas a partir de informações de um especialista sobre o sistema monitorado, incluindo mecanismos de gestão de alarmes.

Em conformidade com os requisitos do sistema, deve ser realizada a integração dos módulos definidos anteriormente para a concretização e implantação do sistema de telemetria.

Diante do exposto, este trabalho tem por objetivo, portanto, o desenvolvimento de técnicas de compressão de dados, criptografia e de um protocolo de comunicação, que melhor atendam aos requisitos de eficiência para armazenamento, tratamento e transmissão de medidas através da linha telefônica, em um sistema de telemetria de baixo custo, utilizando um microcontrolador como sistema embutido.

## **1.4 Relevância**

A implementação de um sistema de telemetria em um dispositivo com recursos tão limitados como o microcontrolador permitirá demonstrar a viabilidade da utilização desses dispositivos, como computadores embutidos, em sistemas de telemetria de baixo custo.

A maioria das implementações de algoritmos de compressão ou criptografia não leva em consideração a memória temporária utilizada para obter os resultados finais, já que geralmente são utilizados computadores com memória volátil suficiente para executar os programas. Dessa forma, projetos que utilizam compressão de dados ou criptografia e precisam utilizar eficientemente a memória do dispositivo para executar esses programas serão beneficiados com os resultados das análises feitas neste trabalho.

A partir das necessidades de outros sistemas de telemetria (quantidade de parâmetros de operação do equipamento a serem medidos, intervalos de tempo entre armazenamento de medidas, quantidade de medidas armazenadas antes da transmissão, etc), será possível realizar uma análise comparativa dos mesmos com o sistema desenvolvido neste projeto. Ou seja, a implementação do sistema de telemetria objeto deste trabalho e as análises dos resultados obtidos podem facilitar a escolha da arquitetura mais adequada do dispositivo embutido a ser utilizado em sistemas semelhantes.

Também é importante ressaltar que a implementação de uma técnica de criptografia, juntamente com o tratamento de eventuais falhas na comunicação, possibilita o desenvolvimento do sistema de telemetria com níveis de confiabilidade e robustez aceitáveis apesar das limitações dos recursos.

## **1.5 Organização da dissertação**

**Capítulo 2 – Descrição do Sistema de Telemetria.** Apresentação do documento de requisitos do sistema (incluindo os referentes ao tratamento de falhas); do modelo de dados obtido, que será utilizado como base para a escolha dos algoritmos a serem escolhidos e implementados; e, por fim, apresentação do modelo proposto para o sistema embutido.

**Capítulo 3 – Fundamentação Teórica.** Apresentação de conceitos fundamentais e caracterização dos componentes essenciais do sistema desenvolvido; de técnicas para as possíveis soluções de implementação dos módulos que compõem o sistema embutido (computador embutido, módulo de compressão, módulo de criptografia e módulo de comunicação).

## **Capítulo 4 – Apresentação e Análise dos Resultados**

**Computador embutido:** Características, módulos utilizados e componentes que fazem interface com o microcontrolador utilizado como computador embutido.

**Módulo de compressão:** Apresentação dos resultados obtidos, em especial a eficiência, da implementação do algoritmo de compressão.

**Módulo de criptografia:** Descrição dos resultados alcançados, em especial a eficiência, da implementação do algoritmo de criptografia.

**Módulo de comunicação:** Apresentação dos resultados obtidos relacionados ao protocolo de comunicação implementado.

**Capítulo 5 - Conclusões e Sugestões para Trabalhos Futuros.** Será apresentado, inicialmente, um sumário dos resultados da pesquisa, seguido da descrição das contribuições relevantes do trabalho, destacando as principais contribuições dos módulos que compõem o sistema de telemetria de baixo custo. No final, serão apresentadas algumas sugestões para trabalhos futuros.

# *Capítulo 2*

## **2. Descrição do Sistema de Telemetria**

---

Neste capítulo será descrita a lista dos requisitos funcionais e não-funcionais do sistema desenvolvido. Antes disso, será descrito o contexto sobre o qual o sistema de telemetria desenvolvido está inserido. Em seguida, o modelo da solução de baixo custo é apresentado juntamente com os requisitos necessários para sua implementação.

Também será apresentado o modelo de dados, obtido a partir das medidas dos sensores do equipamento monitorado, que serve de base para a escolha e implementação do algoritmo de compressão. Por último, será apresentado o modelo proposto para o computador embutido do sistema de telemetria.

### **2.1 Contexto da pesquisa**

Têm-se observado que as variações climáticas que a região Nordeste enfrenta ao longo de cada ano são um dos parâmetros responsáveis pela falta de água para o consumo humano, acarretando em um dos maiores problemas sociais e econômicos da região. Em busca de soluções, as águas subterrâneas têm sido mais exploradas pelo homem. Todavia, as águas comumente encontradas são impróprias para o consumo humano devido aos seus altos índices de sais dissolvidos [Franca, 01].

Visando o aumento da disponibilidade de água, em situações específicas, a alternativa técnico-econômica e social mais viável identificada é a da dessalinização de águas salobras objetivando o atendimento da demanda de consumidores de diferentes portes. Estas águas tratadas e com padrões de qualidade compatíveis com as normas nacionais, são utilizadas, prioritariamente, para o consumo humano e, suplementar, tanto para a irrigação em projetos agropecuários pilotos de portes físicos compatíveis com a disponibilidade hídrica, como para uso na indústria e no setor de serviços, quando possível.

Em todo o território nacional, a região selecionada para a implantação dos estudos e a aplicação dos métodos e das técnicas de dessalinização foi o Nordeste e o Norte de Minas Gerais, na região do polígono das secas. Esta região geográfica, de feições econômicas heterogêneas, é caracterizada pela elevada vulnerabilidade às condições geoclimáticas adversas, onde fica fortemente ressaltada a carência hídrica. Na faixa semi-árida, são constantes os períodos de seca, muitas vezes obrigando a população a migrar

para outros centros, por absoluta falta de condições de permanência e de sobrevivência. Para minimizar esta situação e melhor utilizar os volumes disponíveis de água, a Secretaria de Recursos Hídricos do Ministério do Meio Ambiente, através do Programa Água Boa, instalou centenas de equipamentos de dessalinização via osmose inversa, em poços tubulares que apresentam altos índices de sais dissolvidos.

O Laboratório de Referência em Dessalinização do Departamento de Engenharia Química, implantado no âmbito do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande, através da Secretaria de Recursos Hídricos (SRH) do Ministério do Meio Ambiente (MMA), tem coordenado tecnicamente o Programa Nacional de Dessalinização da SRH/MMA. Este foi responsável pelos projetos e implantação de sistemas de dessalinização via osmose inversa para atender a pequenas e médias comunidades de vários estados do Nordeste, além de sistemas de grande porte implantados no Espírito Santo e no arquipélago de Fernando de Noronha. Com a experiência absorvida ao longo de cinco anos, observou-se que os sistemas de dessalinização constituem uma tecnologia viável para evitar ou minimizar as crises por falta de água de boa qualidade, bem como para o aproveitamento do potencial hídrico que se encontra sem condições de uso.

Através do acompanhamento periódico destes sistemas, tem-se detectado os principais problemas técnicos que afetam de forma direta o processo como um todo, a ponto de causar a paralisação da produção de água dessalinizada. Problemas tais como a falta de pré-tratamento da água bruta, manutenção preventiva, monitoração adequada e pessoal qualificado levaram à elaboração do projeto de um sistema de telemetria.

Atualmente existem mais de mil unidades de dessalinização instaladas no Nordeste e nenhuma delas apresenta um sistema de proteção para os componentes do dessalinizador, inclusive para os elementos de membranas. Existindo uma grande quantidade de dessalinizadores sob uma mesma administração, torna-se inviável realizar a gerência desses equipamentos baseada apenas nos dados coletados através de vistorias presenciais realizadas por operadores no campo. Para uma boa monitoração surge, então, a necessidade de automação das vistorias e descentralização dos dados obtidos através das mesmas. Para que os objetivos do Programa de Dessalinização sejam atendidos, é essencial que a automação seja obtida a um baixo custo e necessite de pouco suporte técnico. Dessa forma, é necessário o desenvolvimento de um sistema para transferência



de informações a ser instalado junto com o dessalinizador que habilite a gerência remota do mesmo sem incrementar substancialmente o custo total do sistema.

## 2.2 Projeto de sistemas de dessalinização

O dessalinizador é um equipamento utilizado para a extração de sal da água, tornando-a pura para o consumo. A Figura 2.1 mostra um dessalinizador composto das seguintes unidades: TA - tanque de alimentação (água salobra), TP - tanque do permeado, TC - tanque do concentrado, F - filtro, B - bomba centrífuga, P1, P2, P3, P4 e P5 – manômetros, Q1 e Q2 - rotâmetros, HF- membrana de alta filtração, ULP - membrana de ultra baixa pressão, HR - membrana de alta rejeição [Kleber, 01].

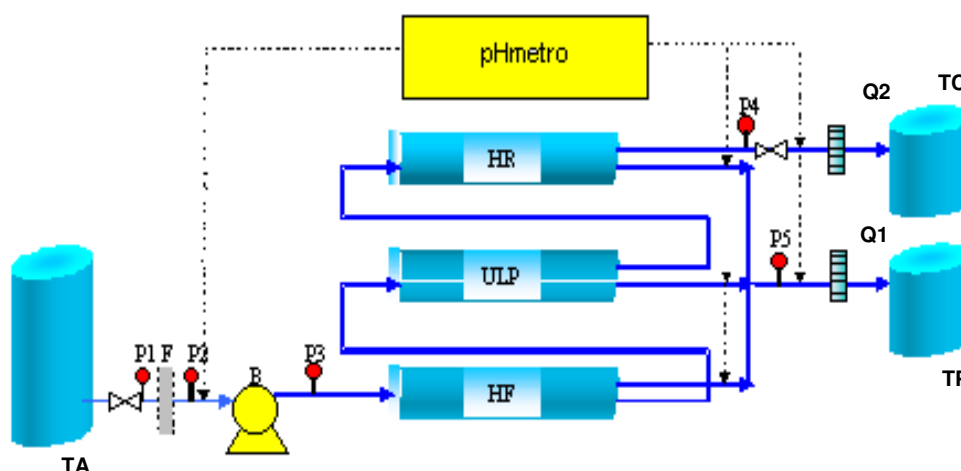


Figura 2.1: Esquemático de um dessalinizador

Do tanque de alimentação, onde se encontra a água salobra que será filtrada, a água passa por um filtro cuja função é retirar impurezas maiores como folhas, lascas de madeira, etc. Ao ser impulsionada pela bomba centrífuga em direção à membrana, a água sofre uma filtração mais refinada, pois a mesma terá que passar por aberturas do tamanho de  $1\text{Å}$  (um angstrom). A água que passa pelas membranas é a água pura e o que não consegue passar é o concentrado com um alto grau de impurezas. A água pura é guardada num tanque de permeado e o rejeito num tanque de concentrado. Os manômetros permitem a visualização das medidas de pressão nas membranas e os rotâmetros informam a quantidade de água pura e de água concentrada que o dessalinizador está fornecendo [Franca, 01].

No dessalinizador, a parte principal é o sistema de membranas onde a água salobra é filtrada. No universo de membranas, os processos para dessalinização de águas salobras e salinas que vêm predominando são os de osmose inversa e eletrodialise. No processo de osmose inversa, a separação dos componentes da água ocorre em função de um gradiente de pressão sobre uma membrana semipermeável; na eletrodialise, a força motriz aplicada ao processo é um potencial elétrico [Amjad, 92].

A osmose natural ocorre quando duas soluções salinas de concentrações diferentes encontram-se separadas por uma membrana semipermeável. Neste caso, a água (solvente) da solução menos concentrada tenderá a passar para o lado da solução de maior salinidade. Com isto, esta solução mais concentrada, ao receber mais solvente, se dilui, num processo impulsionado por uma grandeza chamada *pressão osmótica*, até que as duas soluções atinjam concentrações iguais [Joyce, 01]. Para melhor entender o fenômeno recorre-se à Figura 2.2 [Kerr, 01]. No item I, são apresentadas duas soluções, uma salina e outra sem sal, separadas por uma membrana semipermeável. No item II a água pura dilui a salgada até que seja atingido o equilíbrio osmótico. Por fim, no item III, a aplicação de uma pressão superior à diferença de pressão hidrostática inverte o processo.

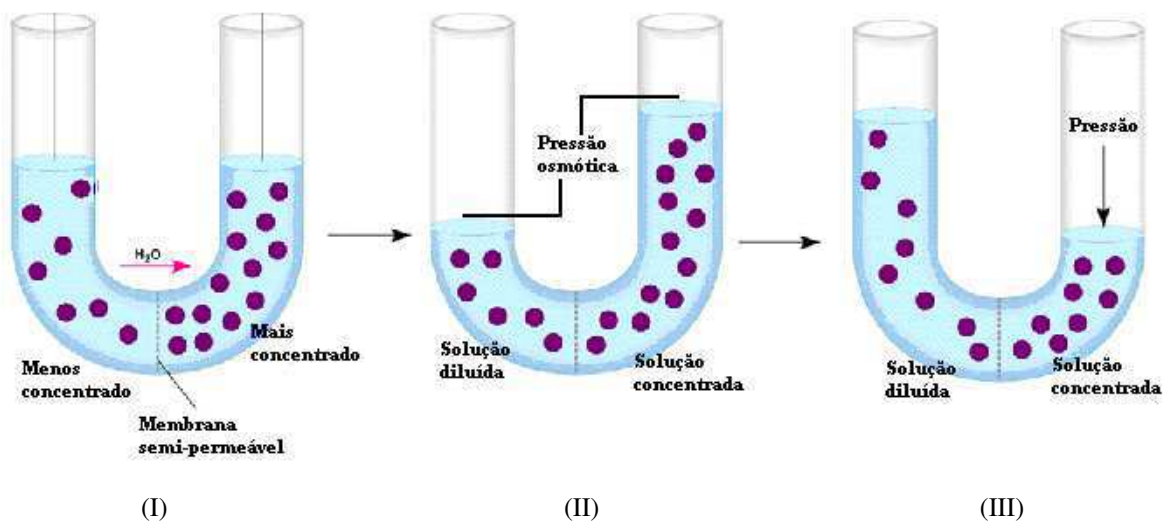


Figura 2.2: Representação do processo de osmose inversa

A osmose inversa é utilizada para dessalinizar águas salinas, salobras e de superfície, utilizando membranas semipermeáveis sintéticas. A pressão aplicada deve superar a pressão osmótica da solução para separar os sais da água. Na prática, a pressão de operação deve superar também a resistência da membrana, a resistência da zona de

polarização de concentração e a resistência interna do equipamento. As pressões de operação reais são, portanto, mais elevadas do que a pressão osmótica da solução. A principal função das membranas é a rejeição de sais, que depende da temperatura, pressão, pH, concentração de sal e rendimento [Schneider, 01].

O processo de osmose inversa possui vantagens em relação aos outros processos de dessalinização, como a destilação a múltiplo efeito, a eletrodialise, a troca iônica, a destilação flash com múltiplo efeito, etc. Apresenta-se como a melhor alternativa existente para certos processos ou regiões e ainda mostra-se mais econômica que caminhões pipa. As principais vantagens são:

- Baixo custo de investimento;
- Baixo consumo de energia;
- Simplicidade de operação e manutenção;
- Aproveitamento dos efluentes;
- Processo contínuo;
- Ocupa área muito reduzida para instalação;
- Flexibilidade para futuras expansões.

Normalmente, os problemas mais encontrados em sistemas de dessalinização são conseqüências de uma falta de manutenção preventiva. Abaixo estão os itens que dão origem aos problemas de pequeno e de médio porte:

- Pré-tratamento da água bruta;
- Dimensionamento e escolha de pré-filtros;
- Tipo e número de elementos de membranas;
- Arranjo de elementos de membranas;
- Limpeza química dos elementos de membranas;
- Metodologia de operação;

Combinando estes fatores, os custos de operação e manutenção podem aumentar substancialmente nos sistemas de dessalinização, podendo tornar o processo como um todo oneroso. Para águas salobras os elementos de membranas podem representar mais de 20% do custo e 30% para os sistemas de água do mar [Amjad, 92].

## 2.3 Manutenção dos equipamentos

A implantação de centenas de dessalinizadores em uma área tão vasta quanto o Nordeste Brasileiro cria um problema não-trivial de manutenção. Por um lado, é essencial que esses equipamentos tenham uma boa manutenção para que a água obtida no processo seja de boa qualidade e para que a vida útil desses equipamentos seja maximizada, conseqüentemente reduzindo o custo da água dessalinizada. Por outro lado, a falta de pessoal qualificado junto aos dessalinizadores e a grande área geográfica coberta pelos mesmos dificultam e encarecem a tão necessária manutenção.

Pretende-se abordar este problema através da realização de atividades de gerência remota dos dessalinizadores, nos quais estas, dentro de intervalos de tempos pré-estabelecidos, coletam e armazenam informações sobre seus componentes e, periodicamente, enviam as informações obtidas nessas coletas para estações de gerência. Os dados coletados dos vários dessalinizadores distribuídos no campo podem ser tratados convenientemente. O *software* executando nas estações gerentes pode, por exemplo, verificar se as medidas atingiram um valor crítico que pudesse gerar um funcionamento inadequado, ou mesmo danificar o dessalinizador, e avisam ao responsável pela administração para que as providências cabíveis possam ser tomadas.

### Detalhamento da solução

Será acoplado a cada dessalinizador instalado um sistema de telemetria composto por um computador embutido, sensores para a medição de pressão, vazão, temperatura e pH, um dispositivo transmissor de dados, além do *software* embutido necessário para adquirir os dados dos sensores, tratá-los e enviá-los a uma ou mais estações de gerência. As estações de gerência executam o *software* para armazenar as informações enviadas pelos sistemas de telemetria em um banco de dados para que essas possam ser consultadas através da *Internet* pelas pessoas envolvidas na manutenção dos dessalinizadores.

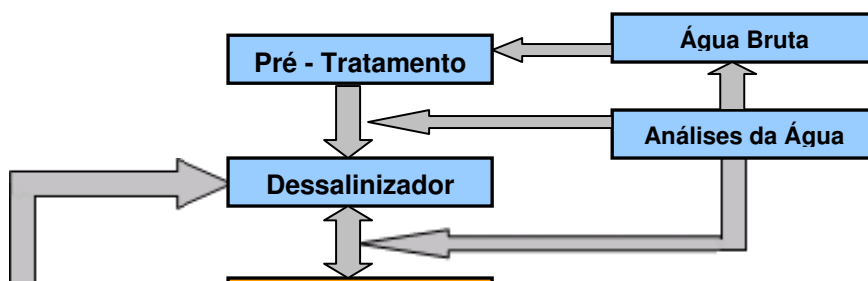
Na base de dados da máquina servidora, além das medidas que são enviadas pelos computadores remotos, também é armazenada uma série de informações acerca do poço, da análise físico-química da água, do equipamento e da comunidade onde o mesmo está instalado. Dessa forma o gerente do sistema pode, além de checar o dessalinizador, ter

informações acerca do ambiente onde se encontra o equipamento. É possível, inclusive, configurar o sistema de telemetria para enviar alarmes em situações consideradas mais graves [Formiga, 03]. Na Figura 2.3 é apresentada uma ilustração dos componentes do sistema.

**Figura 2.3: Esquemático do sistema de telemetria**

Está sendo explorado o desenvolvimento de tecnologias alternativas que viabilizem a automação desse tipo de processo a um custo muito menor. Como um dessalinizador de pequeno porte (esse é o tipo mais comum desse equipamento instalado atualmente) tem um custo na ordem de US\$ 2.500,00, o desafio do projeto é desenvolver um sistema de telemetria com um custo máximo de US\$ 250,00 [Franca, 01].

Esta solução está dentro do projeto completo de estratégia de manutenção dos sistemas de dessalinização. A Figura 2.4 mostra a direção das etapas do projeto visando manter os dessalinizadores funcionando sob uma estratégia de manutenção de baixo custo [Franca, 01].



#### **Figura 2.4: Estratégia de Manutenção de Sistemas de Dessalinização**

Para a obtenção de um sistema de telemetria de baixo custo, serão utilizadas duas estratégias. Primeiro, o desenvolvimento de sensores mais simples e baratos que os comumente encontrados no mercado, aproveitando o fato de que a alta precisão nas medições não é um requisito necessário para o devido acompanhamento dos dessalinizadores. Segundo, o uso de soluções abertas (em contrário a soluções proprietárias) para os componentes de informática e telecomunicações necessários à solução do problema [Franca, 01].

## **2.4 Requisitos relacionados ao computador embutido**

### **Requisitos funcionais**

1. O computador embutido deve coletar e armazenar medidas de oito sensores, realizando quatro medições por dia, durante uma semana.

2. As medidas devem ser transmitidas semanalmente após conexão proveniente do servidor. Após a transmissão, as medidas são apagadas da memória.
3. Para minimizar possíveis erros na obtenção dos valores de medidas dos parâmetros do dessalinizador é necessário aumentar o número de requisições de cada parâmetro e estabelecer um método para descartar valores que se distanciam do valor médio dos mesmos. Isto deve ser realizado antes do armazenamento da medida na memória do computador embutido.
4. Sempre que um estado for coletado, deve ser feita uma checagem desse valor baseado nas condições de alarme especificadas previamente. As condições de alarme apresentam dois limites que estão ordenados e que serão indicados como “alarme amarelo” e “alarme vermelho” no Servidor.
5. Para evitar transmissões desnecessárias ocasionadas por erros de medidas temporários, por exemplo, deve-se verificar a permanência do equipamento em estado de alarme.
6. Em caso de alarme detectado a partir das medidas coletadas, o computador embutido imediatamente inicia uma conexão com o Servidor e envia um sinal para indicação de estado de alerta.
7. Deve ser possível, a partir do servidor, alterar o estado do equipamento de alarme para normal após manutenção no equipamento. Ou seja, as medidas anteriores que ocasionaram um alarme não são mais consideradas para a efetivação de um novo alarme.
8. O sistema deve estar preparado para se recuperar de possíveis problemas ocorridos na realização da transferência de informações. Esses problemas podem ocorrer, entre outros motivos, devido a quedas ou erros na linha de transmissão.
9. Caso a conexão seja interrompida na transmissão dos dados, as medidas deverão permanecer armazenadas.
10. As medidas coletadas antes de um desligamento do computador embutido deverão ser mantidas após o reinício do mesmo. Isto deve acontecer se as medidas coletadas já ocuparam a memória de gravação disponível. Caso contrário, o sistema retorna para o início da coleta de medidas.
11. Em caso de travamento do sistema, o mesmo deverá ser reiniciado.

12. Caso as medidas indiquem um estado crítico de operação do equipamento, será enviado um comando para desligamento do mesmo.
13. O gerente do sistema pode, a qualquer momento, conectar-se com o computador embutido e requisitar o estado atual do equipamento através da leitura de amostra das medidas do mesmo.
14. Deve ser usado um mecanismo de autenticação por criptografia para evitar que uma pessoa que tenha acesso físico ao dessalinizador grave um conjunto de medidas boas e na ocasião da requisição do servidor fique sempre reenviando estes dados gravados, podendo assim sabotar o funcionamento do equipamento sem que isso possa ser percebido pelo sistema de telemetria.

### **Requisitos não funcionais [Franca, 01]**

1. Equipamento microprocessado.
2. Identificador único (número de série) para cada equipamento conforme sua versão de hardware, não modificado por software. Os demais identificadores utilizados para fins de medição, comunicação e informação junto ao dessalinizador deverão ter padronização distinta.
3. Memória de programa.
4. Memória de dados para processamento das informações coletadas, protocolo de comunicação com os concentradores, interfaces de entrada e saída do equipamento.
5. Os dados de medição deverão ser acompanhados de data e hora.
6. Mostrador digital em cristal líquido ou por displays numéricos de sete segmentos (leds). Os mostradores digitais em cristal líquido deverão ser retro-iluminados (*backlight*), para que a medição possa ser realizada em baixas condições de iluminação.
7. O equipamento embutido deve ser alimentado pela rede elétrica e de baixo consumo. A estabilidade na alimentação deverá ser tal a suportar variações de até 15% no sinal da rede.
8. O medidor deverá suportar temperaturas de 0 a 60 °C, com umidade em até 50%.
9. Proteção contra curto circuito e sobre tensão na rede com auto desligamento.



10. O nível de interferência do equipamento no dessalinizador não deverá degradar os equipamentos utilizados por este. Da mesma forma, o equipamento deverá ser imune a interferências geradas pelo dessalinizador durante a utilização de energia (a saber, partida dos motores elétricos das bombas).
11. Deverá ser acondicionado em caixa de proteção, com detector de abertura do mesmo. O equipamento deverá ter grau de proteção mínimo IP-56. A pintura deverá ser do tipo eletrostática caso o equipamento seja de metal.
12. Os conectores utilizados no equipamento, interligando o mesmo ao mundo externo, deverão ser únicos para cada tipo de conexão, imunes a poeira e ao acúmulo de líquido nas suas proximidades.
13. O equipamento deverá prever a utilização de sinais de alarme sonoros, para avisos de falhas, ou para informar ao manipulador sobre seu rendimento diário.
14. Todas as falhas inerentes à medição, vandalismo, comunicação com os concentradores, corte e religamento de energia deverão ser armazenadas em memória não volátil. Os dados das falhas no medidor só poderão ser removidos com comandos especiais originados no centro de processamento e o log de remoção ficará armazenado no equipamento (com data e hora).
15. O equipamento deverá prover uma rotina de auto diagnóstico periódico.
16. Como resultado das medidas dos sensores, tem-se amostras da pressão, vazão, temperatura e pH convertidos em sinais elétricos. De posse desses sinais, é necessário processá-los para a obtenção do valor medido em um determinado intervalo de tempo. Este processamento pode ser inteiramente digital. O processamento digital tem diversas vantagens, tais como (i) a possibilidade de obtenção de vários valores a partir das amostras de pH, pressão e vazão, (ii) a correção de não linearidades através de tabelas de busca e (iii) a calibração através da carga de fatores de correção. Para que o processamento digital seja feito, é necessária a prévia conversão analógico-digital dos sinais de entrada, a ser realizada por um conversor A/D.
17. Deve ser possível estabelecer um canal de comunicação com uma ou mais estações de gerência, que mantêm os dados sobre todos os equipamentos do sistema. Portanto, o computador embutido a ser desenvolvido possuirá um canal de comunicação

bidirecional que permitirá a leitura de valores em horários agendados ou em qualquer tempo, bem como o envio de comandos para o mesmo.

## **2.5 Projeto e implementação do módulo de transferência de informação**

O software a ser implementado no computador embutido faz uma avaliação, a partir dos dados provenientes de sensores, baseada nas condições de alarme pré-definidas e estabelece conexão com o servidor (estação gerente) quando necessário. O módulo de sensores e uma parte do módulo de gerência (banco e dados, interface gráfica e conexão com a Internet) foram considerados em outros projetos.

De acordo com os requisitos funcionais do sistema de telemetria, as medidas são armazenadas durante uma semana no computador embutido. No final da semana, estas medidas serão transmitidas para o servidor, e em caso de alarme serão enviadas imediatamente. As medidas serão armazenadas em um microcontrolador (computador embutido) PIC16F877, que possui os seguintes recursos com relação à memória [Microchip, 01]:

- Memória de Programa FLASH (palavras de 14 bits) → 8K
- Memória de dados temporários RAM R/W (bytes) → 368
- Memória de dados EEPROM (bytes) → 256

Os dois últimos tipos de memória citados acima podem ser utilizados para armazenamento das medidas. O primeiro tipo de memória é utilizado para armazenar o código do programa, não devendo ser aproveitado para outros fins já que o tamanho do código pode ser alterado de acordo com a necessidade de inserção de novos requisitos no sistema.

Algoritmos de compressão são necessários já que devem ser armazenadas, segundo o cliente do projeto, quatro medidas de cada sensor por dia durante a semana. Como são utilizados oito sensores, contabilizam-se sete (número de dias) multiplicado por oito (quantidade de sensores) multiplicado por quatro (número de medidas por dia), resultando em 224 medidas armazenadas durante a semana (224 bytes só para armazenar as medidas). Cada medida é gravada na forma de um número inteiro, que ocupa oito bits no microcontrolador. Considerando-se a utilização de memória EEPROM para outros fins

referentes ao armazenamento de diversos flags para manter a robustez do sistema em caso de falhas e armazenamento de parâmetros de alarmes para cada sensor, verifica-se a necessidade de utilizar algoritmos de compressão de dados que sejam eficientes para a gravação dessas medidas e para a transmissão das mesmas através de linha telefônica.

Uma amostra do modelo de dados coletados de um conversor A/D (Analógico / Digital) de 10 bits (utilizou-se apenas 8 bits – faixa de 0 a 255), para a qual foi realizada a codificação, é apresentada na Figura 2.5. As medidas foram adquiridas a partir de sensores digitais (denominados T, P3 e P4) conectados ao equipamento durante um período de 6 horas. Apesar do intervalo de tempo de medição ser relativamente pequeno com relação ao período de funcionamento do sistema de telemetria no campo, é importante ressaltar que este comportamento se mantém para intervalos de tempo maiores (semanas).

75 140 131 75 140 131 75 140 131 76 141 131 76 141 132 75 140 132 75 140 132 76 141 133 76 141 133 76 141 133 75 140 131 75 140 132 75 140 133 75 140 132 75 140 132 75 140 132
---

**Figura 2.5: Amostra de medidas coletadas de sensores conectados ao equipamento**

As medidas foram adquiridas a partir de um Sistema de Dessalinização Piloto (SDP), mostrado na Figura 2.6, composto dos seguintes componentes: uma bomba de alta pressão de 1,5 CV monofásica, uma bomba dosadora de anti-incrustante, três elementos de filtros de acetato de celulose de 5 µm, três elementos de membranas do tipo BW30-4040, três vasos de alta pressão de 1 metro, dois rotâmetros, dois manômetros, um quadro elétrico de comando, quatro sensores de pressão, dois sensores de vazão, um sensor de temperatura e de pH, um computador embutido e acessórios (uma estrutura metálica, quatro válvulas de esfera de controle de vazão, um tanque de 40 litros para solução anti-incrustante, etc.) [Melo, 04].



**Figura 2.6: Sistema de Dessalinização Piloto (SDP)**

Os manômetros do SDP são usados para a medição da pressão na entrada e na saída dos filtros e das membranas e os rotômetros servem para informar a quantidade de água pura e de rejeito (concentrado) que o dessalinizador irá produzir (Tabela 2.1).

O sistema de monitoração desenvolvido tratará, principalmente, de monitorar os manômetros e os rotômetros, pois, a partir da leitura dos mesmos, será indicada alguma condição de alarme caso o equipamento esteja operando em uma situação crítica. A temperatura e o pH da solução que alimenta o sistema de dessalinização também são monitorados devido à grande influência dessas variáveis no desempenho do SDP.

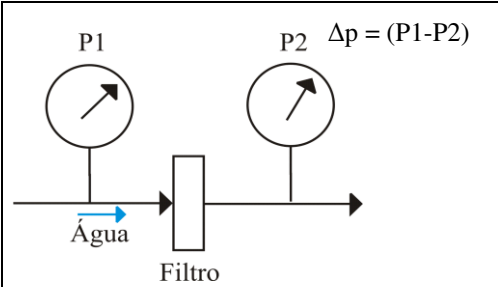
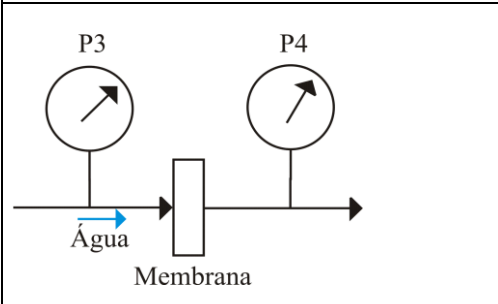
**Tabela2.1: Variáveis de medidas do dessalinizador**

<b>P1 (kgf/cm<sup>2</sup>)</b>	Pressão de entrada dos elementos de filtros
<b>P2 (kgf/cm<sup>2</sup>)</b>	Pressão de saída dos elementos de filtros
<b>P3 (kgf/cm<sup>2</sup>)</b>	Pressão de entrada dos elementos de membranas
<b>P4 (kgf/cm<sup>2</sup>)</b>	Pressão de saída dos elementos de membranas
<b>Q1 (LPM) *</b>	Vazão da água dessalinizada

<b>Q2 (LPM)</b>	Vazão do concentrado (rejeito)
<b>T (°C)</b>	Temperatura de alimentação das membranas
<b>pH</b>	Potencial hidrogeniônico

\* LPM (Litros por minuto)

As condições de alarme a serem adotadas para o monitoramento são apresentadas na Figura 2.7.

	<p>Quando a diferença entre a pressão de entrada (P1) e de saída (P2) dos filtros for maior que 10% da diferença de operação de projeto inicial.</p>
	<p>Quando a diferença entre a pressão de entrada (P3) e de saída (P4) das membranas ultrapassar 15% da diferença de operação de projeto inicial.</p>
<p>Q1, Q2, T</p>	<p>As vazões do permeado e do concentrado também são monitoradas. A última condição de alarme é determinada quando a temperatura da alimentação das membranas atingir 45 °C.</p>

**Figura 2.7: Condições de alarme do sistema de dessalinização**

## 2.6 Considerações finais

Neste capítulo foram apresentados alguns aspectos teóricos do funcionamento do dispositivo a ser monitorado pelo sistema de telemetria, bem como o contexto em que o mesmo está inserido. A partir da necessidade de um gerenciamento mais adequado dos

mesmos, foi elaborado o projeto para a implementação desse sistema de forma a se adequar aos requisitos de baixo custo impostos pela necessidade do cliente do projeto.

Foi realizado um estudo sobre as características e parâmetros de funcionamento do dispositivo. Disso resultou um documento de requisitos a serem implementados no computador embutido e no servidor para a construção do sistema.

## *Capítulo 3*

## 3. Fundamentação Teórica

---

Neste capítulo serão apresentados os conceitos e tecnologias mais importantes utilizados neste trabalho. Na primeira parte serão mostrados alguns dos requisitos normalmente encontrados em um sistema embutido e como o microcontrolador pode ser utilizado em tais sistemas. Na segunda e na terceira partes serão apresentados alguns fundamentos teóricos, juntamente com alguns algoritmos pesquisados para a compressão de dados e criptografia, respectivamente. Na última parte será apresentada, brevemente, uma introdução sobre protocolos de comunicação e, por fim, será descrita a técnica de detecção de erros utilizada.

### 3.1 Sistemas embutidos

Um sistema embutido é um sistema de computação especializado, que faz parte de uma máquina ou sistema mais amplo. Geralmente, é um sistema microprocessado com os programas armazenados em ROM (*Read Only Memory*). Alguns sistemas embutidos podem incluir um sistema operacional, outros são tão especializados que tanto as tarefas a serem executadas quanto às funções específicas de um sistema operacional podem estar implementadas em um único programa.

Atualmente, está cada vez mais difícil classificar um sistema como sendo ou não um sistema embutido, em função dos recentes requisitos de funcionalidade que os mesmos vêm apresentando. Por exemplo, em [Heath, 98] um sistema embutido é designado para um propósito específico, não podendo ser programado pelo usuário da mesma forma que um computador pessoal. Nesse sistema o usuário pode fazer escolhas referentes à funcionalidade, mas não pode mudar a funcionalidade pela adição ou substituição de *software*. Esta afirmação é válida quando aplicada a sistemas embutidos tradicionais, nos quais pouca ou nenhuma alteração em termos de funcionalidade é feita no sistema após a sua conclusão, mas contrasta fortemente com os novos dispositivos embutidos de uso pessoal ou de consumo, cuja funcionalidade pode ser modificada dinamicamente. De acordo com a definição acima, um PDA (*Personal Digital Assistant*) não poderia ser considerado um dispositivo embutido porque possibilita a execução de aplicações diversas da mesma forma que um computador pessoal. De fato, tais dispositivos deveriam se denominados apenas computadores de bolso [Silva, 01].

Vários dispositivos utilizam sistemas embutidos, a saber: telefones celulares, *paggers*, relógios digitais, carros, televisores, videocassetes, cartões inteligentes (*smart cards*), caixas eletrônicos, entre outros.

### **3.1.1 Requisitos de Projeto**

Os sistemas embutidos, normalmente, apresentam grandes restrições em termos de funcionalidade e implementação. Em particular, esses sistemas devem reagir a eventos externos em tempo real, adaptar-se a limites de tamanho e peso, gerenciar consumo de potência, satisfazer a requisitos de confiabilidade e segurança e adequar-se a restrições de orçamento [Silva, 01].

#### **Resposta em Tempo Real**

É comum encontrar em sistemas embutidos requisitos de tempo real. Um sistema de tempo real é um sistema em que as tarefas possuem um tempo máximo para serem realizadas. O tempo que uma tarefa leva para ser executada é chamado tempo de resposta. Em sistemas de tempo real, exceder o tempo de resposta não é aceitável, e pode trazer conseqüências trágicas ao sistema. A gravidade da conseqüência ocorrida quando o limite máximo para o tempo de resposta é excedido oferece uma classificação para os sistemas de tempo real, que pode variar de leve (*soft real-time*) até grave (*hard real-time*).

Sistemas embutidos, como o sistema de freios de um veículo, são sistemas de tempo real, pois um atraso no tempo de resposta não é aceitável. Exceder o tempo de resposta pode fazer vítimas humanas, portanto pode-se classificar esse sistema como *hard real-time*. Um exemplo de sistema de tempo real classificado como *soft real-time* é um sistema destinado à apresentação de vídeos. Nestes sistemas a apresentação dos quadros, que compõem o vídeo, deve obedecer a certa freqüência. Um atraso na apresentação de um ou mais quadros pode ser considerado uma falha de conseqüência leve [Schultz, 99].

#### **Tamanho e custo reduzidos**

Sistemas embutidos geralmente apresentam restrições como tamanho, peso ou custo. Estas restrições são importantes para a definição da arquitetura de um sistema embutido. O tamanho e/ou o peso são restritivos quanto à escolha dos componentes físicos que podem fazer parte do sistema. Da mesma forma, as restrições de custo podem



fazer com que componentes de *hardware/software* que não são os mais poderosos no mercado sejam escolhidos para compor o sistema.

### **Segurança e confiabilidade**

Alguns sistemas embutidos podem trazer riscos aos usuários em caso de falha. É o caso, por exemplo, do controle automático de voo em aeronaves ou sistema anti-bloqueio de freios em carros.

Tradicionalmente, utiliza-se alguma forma de redundância para tornar o sistema tolerante a falhas; seja uma falha de componentes de *software* ou de *hardware*, de informações ou de tempo. No caso dos sistemas embutidos, dadas as suas restrições não só em termos de custo e desempenho, mas também em relação ao volume, peso e consumo de energia, a aplicação de técnicas de tolerância a falhas deve ser mais criteriosa.

Muitos sistemas embutidos operam em ambientes inóspitos ou mesmo inacessíveis, demandando a necessidade de proteção contra choques, vibrações, flutuações na fonte de energia, calor excessivo, fogo, água, corrosão, etc. Em tais ambientes a possibilidade de falhas é elevada, o que demanda um bom mecanismo de tolerância às mesmas [Silva, 01].

### **3.1.2 Componentes Básicos**

Alguns elementos de *hardware/software* estão sempre presentes no desenvolvimento de sistemas embutidos. O *hardware* básico de um dispositivo embutido deve conter um processador, memória e periféricos. Dado que sistemas embutidos são geralmente microprocessados, uma prática comum entre desenvolvedores de sistemas embutidos é escolher um microprocessador específico e montar seu *hardware* em função deste.

#### **Processador**

Embora a capacidade computacional dos microprocessadores tenha aumentado bastante (a capacidade de processamento dos *microchips* dobra aproximadamente a cada 18 meses [Moore, 65]), muitos sistemas embutidos continuam sendo desenvolvidos com o uso de processadores de baixo desempenho, de 8/16 bits (8051 da Intel, 68HC11 da

Motorola, etc). Isto se deve ao fato de haver outros fatores determinantes na escolha do processador além do desempenho, tais como custo, consumo de potência, ferramentas de *software* disponíveis e disponibilidade do componente no mercado, entre outros.

Grandes empresas como IBM, Intel, NEC, etc, licenciam e produzem processadores ARM (*Advanced RISC Machine*) para uso embutido. Estes são baseados em arquitetura RISC (*Reduced Instructions Set Computer*) de 32 bits e podem ser fabricados com baixíssimo consumo ou alto desempenho [ARM, 04].

Os DSPs (*Digital Signal Processors*) são microprocessadores com características próprias que podem ser programados e operam em tempo real, com velocidades muito superiores aos microprocessadores para aplicações genéricas (chegando a 1,1 GHz). Estes níveis de processamento oferecem benefícios adicionais para aplicações em tempo real, como por exemplo, redes de banda larga com ou sem fio e digitalização de imagens. Os DSPs são os principais componentes em 70 por cento dos telefones celulares existentes em todo o mundo [Texas, 04].

## **Memória**

A memória é uma das partes mais importantes do projeto de um sistema embutido e influencia diretamente na forma como o *software* para o sistema é projetado, escrito e desenvolvido. A memória tem duas funções básicas em um sistema embutido:

- Armazenar o *software* do sistema, implementado geralmente em memória não volátil.
- Armazenar dados, tais como variáveis de programa e resultados intermediários, que são armazenados em memória volátil.

Muitos sistemas embutidos apresentam maior quantidade de memória não volátil em relação à volátil, dado o custo maior da última. Como resultado, o *software* para tais sistemas deve ser escrito de forma a minimizar principalmente o uso deste último tipo de memória.

## **Periféricos**

Um sistema embutido comunica-se com o mundo exterior através de periféricos. Dentre os principais periféricos encontrados em um sistema embutido, destacam-se os *displays*, saídas seriais, temporizadores, sensores, etc.

## **Software**

Há várias formas de desenvolvimento de *software* para um sistema embutido, dependendo da complexidade do mesmo, do tempo e da verba disponível. Um desenvolvedor de *software* para sistemas embutidos geralmente precisa se preocupar não só com a funcionalidade que o sistema deve apresentar, mas também com aspectos relacionados à restrição de recursos de memória para executar o programa armazenado, entre outros.

O custo de desenvolvimento do software geralmente representa uma parcela muito significativa do custo geral de desenvolvimento de um sistema embutido. Esta é uma grande motivação para as empresas procurarem um sistema operacional e ferramentas que permitam a redução de custo e tempo para conclusão de projetos. A seguir, será mostrada uma breve análise de algumas opções tradicionais para projeto de sistemas embutidos [Nakanishi, 04]:

- Código direto – Programação em *Assembly*, C/C++ sem sistema operacional. Pode se tornar inviável para projetos complexos e possui alto custo de desenvolvimento.
- DOS (*Disk Operating System*) – Combinado com a opção acima. Trata-se de um ambiente monotarefa e não possui suporte a interfaces gráficas. Possui conectividade de rede limitada.
- Windows (9x, NT, 2000, XP, CE e NT *embedded*). Os Windows 9x, NT, 2000 e XP não são adequados para “embutir”. São grandes, caros e não confiáveis. A Microsoft oferece o Windows CE e NT *embedded* para aplicações “embutidas”, tendo obtido relativo sucesso. A sua vantagem reside no ambiente de desenvolvimento integrado e disponibilidade de *softwares* utilitários. A contrapartida é o alto custo de licenciamento, a limitação na arquitetura de *hardware* suportada e a não disponibilidade de código fonte.
- O Sistema Operacional Linux vem ganhando espaço no mercado de sistemas embutidos. Por ser distribuído livremente e com código fonte

aberto, o Linux possui uma grande variedade de distribuições, destinadas a diferentes nichos na família de sistemas embutidos. Outro fator importante é a grande variedade de plataformas suportadas pelo Linux, tais como x86, RISC e Motorola 68k. O Linux não é um sistema operacional de tempo real (*Real-time Operating System* – RTOS). Para adicionar características de tempo real ao Linux, o núcleo do sistema (*Kernel*) precisa ser modificado. Algumas distribuições possuem suporte a tempo real. Existem várias distribuições Linux para sistemas embutidos, algumas livres (ETLinux, Linux Router Project, etc) e outras comerciais (Tynux, Linux@, etc) [Stival, 03].

- Sistema operacional proprietário (QNX, VRTX, LynxOS, AMX, etc). Nesse caso, há uma falta de padronização, alto custo de desenvolvimento e de licenciamento, mas uma boa adequação do sistema operacional aos requisitos do sistema embutido.

### 3.1.3 UCP para Sistemas Embutidos

Os sistemas embutidos classificados como "simples" lançam mão das UCPs (Unidades Centrais de Processamento) que foram ultrapassadas pela voraz busca por desempenho. Para esse caso, são citadas as UCPs de 8 bits: 8085, Z-80 e 6.800. Esses sistemas também empregam as UCPs de 16 bits: 8086, 8088, 80186, 80188, 80286, 68000, 68010 e são utilizados em controles pequenos como sinais de trânsito, alguns instrumentos médicos, controle de temperatura, etc [Zelenovsky, 99].

Os sistemas embutidos de "complexidade média" usam UCPs compatíveis com a família 386 e 486. Sua arquitetura é muito semelhante ou até idêntica à de um PC (*Personal Computer*). Esses sistemas são empregados em tarefas mais complexas, que podem ser encontradas em instrumentos médicos, computadores de bordo, etc. Os sistemas "sofisticados" lançam mão de UCPs compatíveis com a família Pentium e lembram um moderno computador, sem monitor, teclado e disco. Seu emprego está nos sistemas dedicados multimídia, na automação de centrais telefônicas, no controle de grandes plantas fabris, etc. Na Figura 3.1 é apresentado um computador embutido típico [TWT104, 03] e, em seguida, são apresentadas suas especificações.

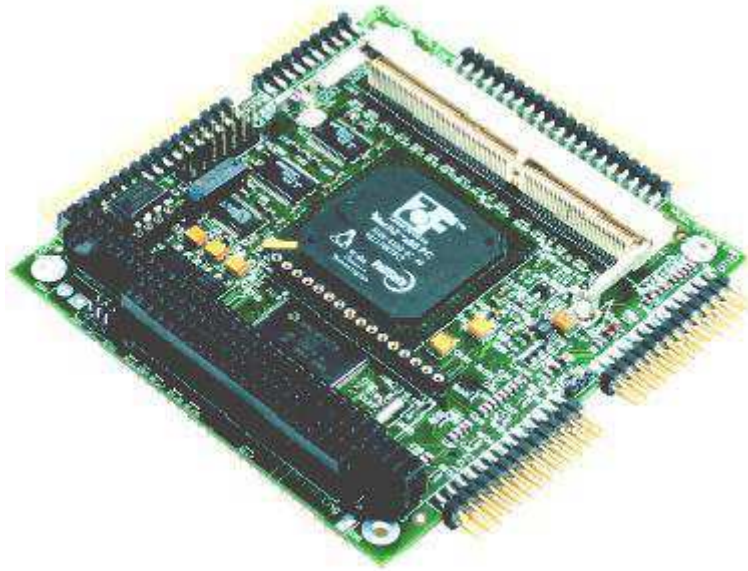


Figura 3.1: Computador embutido típico

### Especificações:

#### **ZFx86 com *PC-on-a-Chip* embutido**

- Core com UCP de 32 bits, com funcionamento entre 33 e 133 MHz;
- Compatibilidade AT plena;
- *Software: Embedded Phoenix PC BIOS (Basic Input/Output System) e Linux image;*
- Suporte para SDRAM (*Synchronous Dynamic Random Access Memory*), com SO-DIMM (*Small Outline Dual In-line Memory Module*) de 8 a 256MB;
- Controlador para unidade de disco flexível.

#### **ROM de *Boot* à prova de falhas (*FailSafe*)**

- Recuperação total do sistema;
- Recupera-se facilmente de corrupção ou perda de dados no *boot* da UCP.

#### **Duplo *Watchdog Timer***

- Fontes programáveis.

#### **Barramento PC/104**

- Compatível com o barramento de expansão PC/104, padrão com IRQ (*Interrupt ReQuest*) e subconjunto de canal DMA (*Direct Memory Access*).

### **Portas Seriais**

- Duas portas seriais RS232 compatíveis com 16550.

### **Portas Paralelas**

- Uma porta bi-direcional para impressora compatível AT.

### **Porta USB (*Universal Serial Bus*)**

- Uma interface USB independente.

### **Interface de *Hard Drive EIDE (Enhanced Integrated Drive Electronics)***

- Conector de 40 pinos padrão;
- Comporta até 2 unidades EIDE (mestre/escravo).

### **Dispositivo de Memória *Flash de Estado Sólido***

- Compatibilidade com sistemas *M-Systems Millennium* e *DiskOnChip 2000*.

### **Compatibilidade de *Software***

- *Phoenix embedded* PC BIOS - compatibilidade x86 total;
- DOS;
- Linux (*Linux Image* com suporte para recursos ZFx86);
- Maioria dos RTOSes compatíveis com PC (*Wind River VxWorks* RTOS com navegador);
- WinCE™, *Windows™ 9x* e *Windows NT™*.

### **Mecânicas/Ambientais**

- Tamanho: 3,6" x 3,8" x 0,9" (91,4mm x 96,5mm x 22,7mm);
- Temperatura de funcionamento:
  - 133MHz: -20°C a +70°C
  - 33MHz, 66MHz, 100MHz: -40 a 85°C (estimativa).

### **3.1.4 CISC, RISC ou SISC**

A maioria dos sistemas embutidos (e também dos microcontroladores) está baseada no conceito CISC – Computador com Conjunto de Instruções Complexo

(*Complex Instruction Set Computer*). Uma UCP CISC, normalmente, possui mais de 100 instruções e muitas são poderosas e específicas para realização de algumas tarefas. O programador é muito exigido, pois cada instrução se porta de uma maneira específica. Algumas operam somente em certos espaços de endereços ou em registradores e outras podem reconhecer somente certo tipo de modo de endereçamento. A vantagem destas UCPs está no uso eficiente da memória para o código do *software*.

Nos últimos tempos, o conceito RISC passou a ser utilizado pelos sistemas dedicados. O termo RISC significa Computador com Conjunto de Instruções Reduzido (*Reduced Instructions Set Computer*). Essas máquinas oferecem poucas instruções (cerca de 35) e, por isso, sua unidade de controle é mais simples permitindo que se logre uma melhor otimização. Com relação aos benefícios do RISC, tem-se: melhor desempenho, CI's menores, menor quantidade de pinos e um menor consumo de energia [Zelenovsky, 99]. A desvantagem do RISC está numa demanda maior de memória de código em relação a um CISC para implementar a mesma funcionalidade.

Com a evolução e desdobramento do mercado de microcontroladores e sistemas embutidos, surgiu o conceito SISC – Computador com Conjunto de Instruções Específico (*Specific Instruction Set Computer*). Sua idéia consiste em limitar ou especializar os recursos da UCP em benefício de outras tarefas como I/O, interrupções e acesso à memória e, além disso, incluir instruções que facilitem a manipulação de bits, de canais de I/O, de temporização, etc. Ou seja, a UCP é “enxuta” e com várias instruções para facilitar as operações de controle [Zelenovsky, 99].

### **3.1.5 Utilização de microcontroladores**

Em poucas palavras, o termo microcontrolador (Figura 3.2) poderia ser definido como um “pequeno” componente eletrônico, dotado de uma “inteligência” programável, utilizado no controle de processos lógicos. Para entender melhor essa definição, é necessário analisá-la por partes [Júnior, 90].



**Figura 3.2: Microcontrolador PIC16F877**

O controle de processos deve ser entendido como o controle de periféricos, tais como: led, botões, *display* de segmentos, *display* de cristal líquido (LCD), resistências, relés, sensores diversos (pressão, temperatura, etc) e muitos outros. São chamados de controles lógicos, pois a operação do sistema baseia-se nas ações lógicas que devem ser executadas, dependendo do estado dos periféricos de entrada e/ou saída [Souza, 00].

O microcontrolador é programável, pois toda a lógica de operação que foi mencionada é estruturada na forma de um programa e gravada dentro do componente. Depois disso, sempre que o microcontrolador for alimentado, o programa interno será executado.

Nessa definição, o microcontrolador ganhou ainda o adjetivo “pequeno”, pois em uma única pastilha de silício encapsulada (popularmente chamada de CI ou CHIP) têm-se todos os componentes necessários ao controle de um processo. Ou seja, o microcontrolador está provido internamente de memória de programa, memória de dados, portas de entrada e/ou saída paralela, *timer*, contadores, comunicação serial, PWM (*Pulse Width Modulation*), conversores analógico-digitais, etc. [Souza, 00].

Os microcontroladores permitem o desenvolvimento de projetos compactos e sofisticados, semelhantes em desempenho aos dos microprocessadores, mas com custos muito menores [Júnior, 90]. Existe na literatura alguma divergência sobre a diferença entre microcontroladores e microprocessadores. Alguns autores classificam o primeiro como um caso especial do segundo [Flynn, 93], enquanto outros classificam como dois componentes eletrônicos distintos, para usos específicos. Mesmo com o advento no mercado de microprocessadores de 32 ou 64 bits, existem aplicações mais simples, específicas e que necessitam de pequenos volumes de códigos para executá-las. Para essas aplicações, microcontroladores de 16 ou 8 bits representam a melhor relação custo/benefício de desenvolvimento. O que torna o microcontrolador um componente



capacitado para estas aplicações especializadas é a incorporação de uma série de blocos dentro de um mesmo encapsulamento, dentre os quais, destacam-se [Vieira, 99]:

- Linha de comunicação serial, que capacita o microcontrolador a trocar informações com outros microcontroladores ou até mesmo com microprocessadores;
- Portas de entrada/saída digitais, que permitem a comunicação do microcontrolador com outros componentes digitais;
- Temporizador/contador, responsável por ações cíclicas, baseadas no tempo;
- Memória de acesso aleatório (RAM R/W), na qual são executados os programas e armazenadas as variáveis em tempo de execução e
- Memória *flash* programável, que simplifica a programação e reprogramação do *chip*.

Existe uma quantidade expressiva de microcontroladores, porém os mais conhecidos são: 8051, 8096, 68HC705, 68HC11 e PIC. Na Tabela 3.1 [Microchip, 01] são apresentadas as principais características de alguns microcontroladores da família PIC.

**Tabela 3.1: Principais características de alguns microcontroladores PIC**

Características de microcontroladores PIC	PIC16F873	PIC16F874	PIC16F876	PIC16F877
<b>Freqüência de operação</b>	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz
<b>Memória de Programa FLASH (palavras de 14 bits)</b>	4K	4K	8K	8K
<b>Memória de Dados (bytes)</b>	192	192	368	368
<b>Memória de Dados EEPROM</b>	128	128	256	256
<b>Interrupções</b>	13	14	13	14
<b>Portas I/O</b>	A, B, C	A, B, C, D, E	A, B, C	A, B, C, D, E
<b>Temporizadores</b>	3	3	3	3
<b>Módulos de Captura/Comparação/PWM</b>	2	2	2	2
<b>Comunicações Seriais</b>	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
<b>Comunicações Paralelas</b>	-	PSP	-	PSP
<b>Módulo Analógico/Digital de 10 bits</b>	5 canais	8 canais	5 canais	8 canais
<b>Número de Instruções de máquina</b>	35	35	35	35

Abaixo segue o significado de algumas siglas utilizadas no quadro acima:

DC → *Direct Current*

MSSP → *Master Synchronous Serial Port*

USART → *Universal Synchronous/Asynchronous Receiver/Transmitter*

PSP → *Parallel Slave Port*

## 3.2 Compressão de dados

A compressão de dados é uma forma de codificar uma mensagem de maneira que o número de bits necessário para sua representação seja menor que o número de bits necessário para a representação original. Um alfabeto é o conjunto completo de símbolos que pode aparecer na mensagem. Um símbolo pode representar qualquer tipo de dado; pode ser um byte, um caractere, ou um pixel, dependendo da aplicação [Assis, 03].

Se a informação, após sua compressão, pode ser exatamente reconstruída, a técnica de compressão é dita sem perdas; caso contrário, a compressão é dita com perdas. Técnicas de compressão com perdas são utilizadas para compressão de áudio, imagens e vídeo, nas quais erros e perdas são toleráveis devido às limitações do ser humano no que se refere à audição de sons ou visualização de certos detalhes de imagens. Exemplos de formatos que utilizam compressão com perdas são JPEG e MPEG, para imagem e vídeo, respectivamente. As técnicas de compressão sem perdas são utilizadas quando uma perda comprometeria a utilidade da mensagem como, por exemplo, na transmissão de documentos de texto. Como exemplo dessas técnicas tem-se: Codificação Aritmética, Codificação de Huffman e Codificação *Run-Length*. Cada um desses algoritmos satisfaz as seguintes condições que são importantes para a aplicação desse trabalho [Assis, 03]:

- Ser *não singular*, ou seja, mensagens distintas devem corresponder a palavras-código distintas. Matematicamente, tem-se:

$$X_i \neq X_j \rightarrow C(X_i) \neq C(X_j)$$

uma aplicação sobrejetora.

- Ser *univocamente decodificável*. Ou seja, não há ambigüidade (função bijetora). Existe uma decodificação única para uma dada seqüência de símbolos concatenados. Exemplo: A (1), B (100), C (000), D (010). Ou seja, a palavra “100” só pode ser decodificada para B e assim por diante.
- *Prefixo*. Na decodificação, sempre há um só símbolo para substituição, pois nenhum grupo de código é duplicado como o início de outro maior. Exemplo: A (1), B (01), C (001).

Os dados (que foram apresentados no Capítulo 2) a serem comprimidos, coletados a partir de sensores ligados ao equipamento monitorado apresentam um comportamento estável (variações pequenas nos seus valores) por um período de tempo longo (aproximadamente semanas). Devido a essa característica, neste trabalho, foi empregada a compressão estatística, que consiste em realizar uma representação otimizada de símbolos ou grupos de símbolos. Símbolos de maior freqüência de ocorrência são representados por códigos binários pequenos e os de menor freqüência são representados por códigos proporcionalmente maiores.

Neste tipo de compressão não é necessário saber qual símbolo vai ser comprimido, mas precisa-se ter o conhecimento da probabilidade de ocorrência de todos

os símbolos sujeitos à compressão. Caso não seja possível a tabulação de todos os símbolos sujeitos à compressão, utiliza-se uma técnica adequada para levantamento estatístico dos dados a comprimir, formando tabelas de probabilidades. As técnicas estatísticas utilizadas neste trabalho têm como propriedades:

1. A fonte de dados é bem caracterizada;
2. Os códigos são associados aos símbolos com base nas suas probabilidades;
3. Possuem duas fases:
  - a. A modelagem da informação que analisa as características estatísticas dos dados a serem comprimidos;
  - b. A codificação dos dados que define o código de cada símbolo do texto fonte.

O esquema de compressão pode ser dividido em três fases: pré-processamento, modelagem e compressão. Pré-processamento é uma transformação reversível que é executada antes da implementação do algoritmo de compressão no processo de codificação e depois no processo de decodificação [Abel, 03]. Esta fase é empregada com o objetivo de diminuir o conjunto de dados a serem modelados, ocasionando, por consequência, uma taxa melhor de compressão dos dados. Como exemplo, o conjunto de dados a serem comprimidos pode ser reduzido através do cálculo da diferença entre seus valores.

### **3.2.1 Modelagem**

A compressão é obtida formando-se modelos dos dados a serem comprimidos. Um modelo provê a distribuição de probabilidade dos símbolos contidos nos dados. Tanto o codificador, como o decodificador, deve usar o mesmo modelo.

O número de bits para codificar um símbolo,  $s$ , está associado com o conteúdo de informação. O conteúdo de informação do símbolo,  $I(s)$ , está diretamente relacionado com a probabilidade do mesmo,  $Pr[s]$ , pela função  $I(s) = -\log Pr[s]$  bits. Se uma fonte produz mensagens igualmente prováveis, cada uma delas conduz a mesma quantidade de informação, neste caso não será possível qualquer redução em relação à quantidade de bits por mensagem [Assis, 03].

No entanto, quando as mensagens têm probabilidades diferentes, pode-se esperar uma codificação mais eficiente para as mensagens da fonte. A maior eficiência se refere aqui como o menor comprimento médio em bits por símbolo gerado pela fonte [Assis, 03].

A quantidade média de informação por símbolo sobre um alfabeto é conhecida como a Entropia da distribuição de probabilidade. A Entropia mede justamente a quantidade de bits média necessária para representar um conjunto de valores  $X$ , com probabilidades associadas e distintas, da maneira mais eficiente possível. A Entropia de uma fonte discreta sem memória  $X$ , com distribuição de probabilidades  $p$  é definida pela Equação 3.1 [Cover, 91]:

$$H(X) = H(p) = \sum_{i=1}^k p(x_i) I(x_i) = \sum_{i=1}^k p(x_i) \log(1/p(x_i)) \text{ bits / símbolo}, \quad (3.1)$$

sendo  $k$  o número de símbolos.

A eficiência do código é definida como a razão entre a Entropia da fonte e o comprimento médio da palavra-código. O comprimento médio  $L(C)$  de um código fonte  $C(X)$  de uma variável aleatória  $X$  é dado pela Equação 3.2:

$$L(C) = \sum_{x \in \mathcal{X}} p(x) l(x) \text{ bits / símbolo}, \quad (3.2)$$

em que  $l(x)$  é o comprimento da palavra-código associada com a saída  $x$  da fonte.

Desde que os símbolos apareçam independentemente e com probabilidades já definidas,  $H$  é um limite inferior para compressão, medido em bits por símbolo. Esse limite inferior foi provado por Claude Shannon [Shannon, 49]. Portanto, um bom algoritmo de compressão deve fornecer um código com comprimento o mais próximo possível do valor da Entropia da fonte. A fórmula acima indica que baixas probabilidades resultam em alta entropia e vice-versa. No caso extremo, quando a probabilidade de um símbolo  $s$  é 1, apenas um símbolo é possível, e  $I(s) = 0$ , indicando que nenhum bit é necessário para transmiti-lo, ou seja, se um símbolo é certo para ocorrer, não precisa ser transmitido. Por outro lado,  $I(s)$  torna-se muito grande quando  $Pr[s]$  aproxima-se de zero, e um símbolo com probabilidade zero não poderia ser codificado [Kohayakawa, 01].

### 3.2.2 Modelos de contexto finito

Para que se possa chegar o mais próximo do valor da Entropia, a probabilidade para o próximo símbolo pode ser tomada levando-se em conta o símbolo anterior. Por exemplo, para a compressão de um texto de língua portuguesa, se o símbolo  $q$  acabou de ser encontrado, a probabilidade do próximo ser  $u$  deve ser bem grande, baseado na frequência em que um  $q$  é seguido por um  $u$ . Os outros símbolos terão probabilidades menores para compensar, e se a predição estiver incorreta, o próximo símbolo será codificado com mais bits.

Modelos que consideram o conjunto dos símbolos imediatamente precedentes para fazer uma predição das probabilidades são chamados modelos de contexto finito de ordem  $m$ , em que  $m$  é o número de símbolos prévios usados para fazer a predição. O termo "finito" é empregado por levar em consideração um número finito de símbolos anteriores ao símbolo corrente [Campos, 00].

O modelo mais simples de contexto-finito é o modelo ordem-0, no qual a probabilidade de cada símbolo independe de qualquer símbolo anterior. Para implementar esse modelo, necessita-se apenas de uma tabela contendo a frequência que cada símbolo possui na fonte. Para um modelo ordem-1, serão geradas  $n$  diferentes tabelas de frequências (considerando  $n$  o tamanho do alfabeto), porque será necessário guardar em separado o conjunto de frequências para cada possível contexto. Modelos com ordem acima de zero são chamados de modelos de Markov (*Markov Information Source*) [Abramson, 73]. A modelagem de Markov de contexto finito, e em particular o método PPM (*Prediction by Partial Matching*), é conhecido por ser um dos métodos mais eficientes para se conseguir altos níveis de compressão. No entanto, requer memória abundante e alta capacidade de processamento [Bloom, 96].

O exemplo da Tabela 3.2 mostra a memória necessária aos modelos, considerando que a fonte possui um alfabeto de  $n = 256$  símbolos diferentes.

**Tabela 3.2: Memória necessária para diversas ordens de modelos de contexto finito**

<b>Ordem do Modelo</b> <i>m</i>	<b>Quantidade de</b> <b>Memória (bytes)</b>	<b>Descrição</b>
0	512	tabela de 256 entradas, com frequências de ocorrência.
1	131072	256 tabelas de 256 entradas cada, com frequências de ocorrência.
2	33554432	65536 tabelas de 256 entradas cada, com frequências de ocorrência.

### 3.2.3 Algoritmos

A seguir serão apresentadas algumas técnicas de compressão de dados cujas características podem se adaptar às necessidades deste trabalho, conforme descrição a seguir.

#### **RLE (*Run-Length Encoding*)**

O RLE comprime/codifica dados baseado em ocorrências sucessivas do mesmo padrão. O resultado da codificação de uma cadeia de bytes é uma palavra composta por 2 bytes: um que indica o padrão de repetição e o outro byte que contém o número de vezes que deverá ser repetido esse padrão [Feijão, 02].

Este método só traz ganhos relevantes se houver grandes agrupamentos de símbolos iguais. Caso contrário, há expansão. Um exemplo típico de uma codificação RLE pode ser representado no seguinte esquema: A seqüência inicial “10 10 10 10 10 10 10 30 30 30 30 95 95 95 95 95” será codificada em “7 10 4 30 5 95”, ou seja, obteve-se uma compressão de 16 bytes para 6 bytes (entre os quais 1 byte é utilizado para armazenar o dado e 1 byte é utilizado para armazenar a quantidade de vezes que o mesmo aparece em seqüência): um fator de compressão de  $16/6 = 2,67$ .

Na compressão JPEG, padrão de compressão de imagens proposto originalmente em 1987 pelo *Join Photographic Expert Group* [JPEG, 04], existem grandes seqüências de zeros a serem codificados, por isso a codificação RLE foi simplificada para apenas contar a ocorrência deste símbolo. Além disso, como apenas os zeros são contados, não é

necessário que este símbolo apareça após o número de ocorrências. Então, a codificação RLE na compressão JPEG conta quantos zeros existem antes de um componente não zero e gera um par “Ocorrências / Valor”, em que o campo “Ocorrências” informa o número de zeros que antecedem o valor não zero e o campo “Valor” é o valor do componente não zero que segue [Agostini, 02].

## Codificação de Huffman

Neste método de compressão, são atribuídos menos bits a símbolos mais freqüentes e mais bits a símbolos menos freqüentes. Assim, o tamanho em bits dos caracteres codificados será diferente [Cover, 91].

A Codificação de Huffman é um exemplo de técnica de codificação estatística que diz respeito ao uso de um código curto para representar símbolos comuns e códigos longos para representar símbolos pouco freqüentes. O algoritmo se fundamenta no seguinte lema: Para uma distribuição qualquer, existe um código ótimo, instantâneo que satisfaz às seguintes propriedades:

1. Se  $p_j > p_k$  então  $l_j < l_k$  ( $p$  é a probabilidade da palavra-código e  $l$  é o comprimento da mesma).
2. As duas palavras-código mais longas têm o mesmo comprimento.
3. As duas palavras-código mais longas diferem apenas no último bit e correspondem aos dois símbolos menos prováveis.

Como exemplo de codificação de Huffman, considere uma fonte  $X$  com símbolos  $x_1, \dots, x_5$  e probabilidades 0.25, 0.25, 0.2, 0.15, 0.15, respectivamente. Pode-se esperar que as palavras-código mais longas sejam as codificações dos símbolos  $x_4$  e  $x_5$ . Os comprimentos de  $C(x_5)$  e  $C(x_4)$  devem ser iguais, caso contrário era possível apagar um bit da palavra mais longa e ainda assim dispor de um código univocamente decodificável com comprimento médio menor. Para um código assim, podem-se combinar os símbolos  $x_4$  e  $x_5$  e passar a tratá-los como um único símbolo com probabilidade 0.30. Este procedimento de agrupar os dois símbolos menos prováveis em um único constitui a idéia chave da codificação de Huffman. A Figura 3.3 descreve o procedimento seguido para a codificação de Huffman neste caso [Assis, 03].



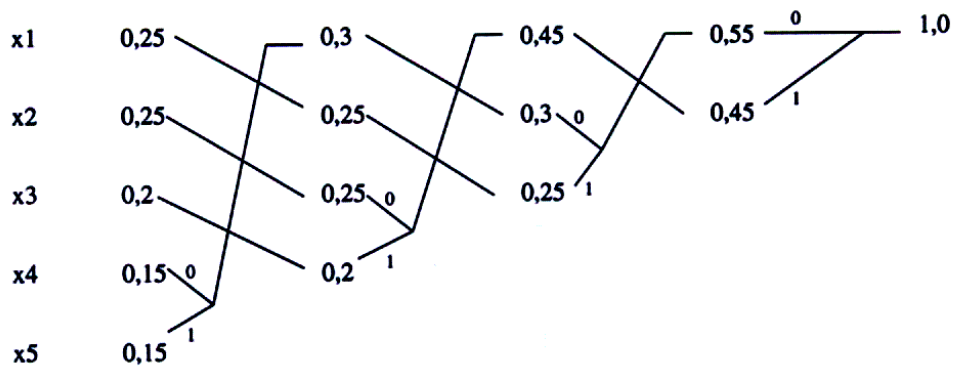


Figura 3.3: Exemplo de Codificação de Huffman

É possível verificar, a partir da figura, que o código resultante é dado por:

$C(x_5) = 001$   
 $C(x_4) = 000$   
 $C(x_3) = 11$   
 $C(x_2) = 10$   
 $C(x_1) = 01$

O comprimento médio desse código é  $l(C) = 12 / 5 = 2.4$  bits.

### Codificador Aritmético

É um método estatístico cujas primeiras implementações práticas datam de 1976 [Rissanen, 76] e [Pasco, 76]. A codificação de Huffman codifica um símbolo com sua entropia aproximada para um número inteiro de bits. A codificação aritmética tem desempenho melhor em taxa de compactação por não existir essa restrição.

A codificação aritmética não utiliza a metodologia de codificação de um símbolo em um código específico; diferentemente, transforma a seqüência completa de símbolos da fonte em um único número real. O processamento da codificação aritmética resulta em um número menor que 1 e maior ou igual a 0, que pode ser decodificado para a seqüência de símbolos original. Para gerar esse número, é atribuído aos símbolos um conjunto de probabilidades. Como exemplo, seja a seqüência "BILL GATES", que tem uma distribuição de probabilidades mostrada na Tabela 3.3. A cada símbolo é, então, atribuída uma subdivisão da faixa de valores entre 0 e 1, correspondendo, em tamanho, à sua probabilidade de ocorrência [Nelson, 91].

**Tabela 3.3: Probabilidade e faixa dos símbolos para a Codificação Aritmética**

<b>Símbolo</b>	<b>Probabilidade</b>	<b>Faixa</b>
Espaço	1 / 10	0.00 – 0.10
A	1 / 10	0.10 - 0.20
B	1 / 10	0.20 - 0.30
E	1 / 10	0.30 - 0.40
G	1 / 10	0.40 - 0.50
I	1 / 10	0.50 - 0.60
L	2 / 10	0.60 - 0.80
S	1 / 10	0.80 - 0.90
T	1 / 10	0.90 - 1.00

Como visto, o segundo símbolo, a letra "A", está contido na faixa 0.10 - 0.20. Com a tabela de frequências concluída, inicia-se a codificação através da construção de duas tabelas, que possuem o número gerado pelo menor valor e pelo maior valor, respectivamente. Os valores mínimo e máximo são colocados nas tabelas de menor e maior valor respectivamente (Tabela 3.4). No início, são atribuídos os valores 0 (zero) e 1 (um) para os campos “Menor valor” e “Maior valor”, respectivamente

**Tabela 3.4: Codificação Aritmética**

<b>Símbolo</b>	<b>Menor valor</b>	<b>Maior valor</b>
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
Espaço	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

A tabela é calculada de acordo com o seguinte algoritmo:

```
menor = 0.0;
maior = 1.0;
enquanto não terminar a seqüência de símbolos faça {
    faixa = maior - menor;
    maior = menor + faixa * maior_da_faixa (símbolo);
    menor = menor + faixa * menor_da_faixa (símbolo);
}
```

em que "maior\_da\_faixa (símbolo)" é uma função que retorna o maior valor da faixa do símbolo, passado como parâmetro, de acordo com a Tabela 3.3, e "menor\_da\_faixa (símbolo)" retorna o menor valor. A cada iteração da estrutura "enquanto", os valores calculados de maior e menor são colocados na Tabela 3.4, em seus símbolos correspondentes. O menor valor calculado para o último símbolo é a codificação final da seqüência, ou seja, 0.2572167752.

### **Codificação LZW**

Esse algoritmo foi proposto por Lempel e Ziv, em 1977. É um método adaptativo de um passo que associa seqüências de tamanho variável de símbolos a códigos de tamanho fixo ou variável. Essas características levam o método a ser chamado de Método Baseado em Dicionário. É também um método dependente do contexto da ocorrência do símbolo na fonte.

O algoritmo consiste em preparar uma tabela que pode conter milhares de itens. Inicialmente são colocados, da posição zero até a posição 255, os usuais 256 caracteres (considerando o alfabeto como o código ASCII). Lêem-se vários símbolos da fonte, e pesquisa-se na tabela pela mais longa correspondência. Supondo que a mais longa correspondência é dada pela seqüência "ABC", coloca-se a posição de "ABC" na tabela. Lê-se o próximo símbolo da fonte. Se o símbolo for "D", coloca-se uma nova seqüência "ABCD" na tabela e reinicia-se o processo com o símbolo "D". Se houver *overflow* da tabela, descarta-se o item mais antigo ou, preferencialmente, o menos usado [Lempel, 78].

### 3.3 Criptografia

Com o incremento do uso de sistemas distribuídos e o uso de redes e facilidades de comunicações para o envio de informações entre computadores, existe uma crescente necessidade de métodos criptográficos mais seguros para proteger os dados durante a transmissão.

Ataques sobre a segurança de um sistema de computação ou de rede são mais bem caracterizados por considerar a função do sistema de computação como provedor de informação. Em geral, há um fluxo de informação de uma fonte, tal como um arquivo ou região de memória principal, para um destinatário, tal como um outro arquivo ou um usuário. A seguir, são mostradas algumas categorias de ataques com relação ao fluxo normal da informação [Stallings, 03]:

- **Interceptação:** Acontece quando houver um acesso não autorizado à informação. Este é um ataque de confidencialidade. A entidade não autorizada pode ser um uma pessoa, um programa ou um computador. Exemplos incluem a cópia ilícita de arquivos ou programas.
- **Modificação:** Uma entidade não autorizada além de ganhar acesso à informação modifica-a. Este é um ataque de integridade. Exemplos incluem a alteração de valores em um arquivo de dados, a modificação de um programa de forma que o mesmo execute diferentemente ou a alteração do conteúdo de mensagens que estão sendo transmitidas em uma rede.
- **Fabricação:** Uma entidade não autorizada insere dados no sistema. Este é um ataque de autenticidade. Exemplos incluem a inserção de mensagens espúrias em uma rede ou a adição de dados em um arquivo.

Sistemas criptográficos são genericamente classificados com relação a três dimensões independentes [Stallings, 99]:

1. **Os tipos de operações usadas para transformar texto plano em texto cifrado.** Todos os algoritmos são baseados em dois princípios gerais: substituição, em que cada elemento no texto plano (bit, letra, grupo de bits ou letras) é mapeado em outro elemento, e transposição, em que elementos no texto plano são colocados em outras posições. O requisito fundamental é que nenhuma informação seja

perdida (isto é, que todas as operações sejam reversíveis). A maioria dos sistemas utiliza múltiplos estágios de substituição e transposição.

2. **O número de chaves usado.** Se ambos emissor e receptor usam a mesma chave, o sistema é referido como simétrico, chave-única, chave-secreta ou criptografia convencional. Se cada um usa uma chave diferente, o sistema é referido como assimétrico, chave-dupla ou criptografia de chave-pública.
3. **A forma na qual o texto plano é processado.** Uma cifra de bloco processa a entrada de um bloco de elementos de uma vez, produzindo um bloco de saída para cada bloco de entrada. Uma cifra de cadeia processa os elementos de entrada continuamente, produzindo na saída um elemento por vez.

A maior parte dos sistemas criptográficos usados na prática não se fundamenta na impossibilidade de quebra, mas ao invés disto, se baseia na dificuldade de quebra do sistema. A segurança contra um inimigo que está sujeito a certa limitação de tempo e recursos computacionais para os seus ataques é denominada **segurança computacional** ou **segurança prática**. Ou seja, um esquema de criptografia é dito ser computacionalmente seguro se obedece a dois critérios:

- O custo de quebrar a cifra excede o valor da informação criptografada;
- O tempo requerido para quebrar a cifra excede o tempo de vida útil da informação.

### *S-boxes*

Uma estrutura de uso freqüente nos algoritmos criptográficos é chamada de *S-Box*. *S-Boxes* são caixas de substituição não lineares que visam emaranhar o texto cifrado para que se torne mais difícil a sua decifragem. As *S-Boxes* variam de tamanho de entrada e de saída e podem ser criadas algoritmicamente ou randomicamente, em outras palavras, o funcionamento de uma *S-Box* se baseia na simples troca de posição dos bits de entrada. Hoje, os algoritmos mais importantes de criptografia usam essa estrutura [Burnett, 02].

### **3.3.1 Algoritmos**

O primeiro relato de um algoritmo de criptografia que se tem é conhecido como algoritmo de César [Weber, 95], usado pelo imperador Júlio César na Roma Antiga. Era

um algoritmo simples que fazia substituições alfabéticas no texto da mensagem. As substituições aconteciam trocando letras por outras, três posições à frente no alfabeto, ou seja, a letra A seria substituída por D, a letra B por E, e assim por diante como mostrado a seguir.

**Texto simples:** Vamos atacar o norte durante a noite.

**Texto cifrado:** Zdprv dxdfdu r qruhx gyudqhx d qxlhx.

Os algoritmos mais modernos usam os mesmos princípios do relatado acima, nos quais o algoritmo é conhecido, mas a chave não. É importante salientar que o tamanho da chave é um fator importantíssimo na segurança do algoritmo; uma chave de dois dígitos possui 100 possibilidades de combinações, uma chave de 6 dígitos possui 1 milhão de possibilidades. Consequentemente, quanto mais possibilidades de combinações, mais tempo um criptoanalista levará para decifrar um texto [Hinz, 00].

Os algoritmos que substituem letras por outras letras, mas mantendo-as na mesma posição são chamados de algoritmos de cifras de substituição. Existem outros tipos de algoritmos que trocam a posição das letras sem trocar o seu valor, que são chamados algoritmos de cifras de transposição.

### 3.3.2 Modelo de Criptografia Convencional (Simétrica)

O processo de cifragem consiste de um algoritmo e uma chave. A chave é um valor independente do texto plano. O algoritmo produzirá uma saída diferente dependendo da chave específica que está sendo usada no momento. Se houver mudança na chave, haverá alteração na saída do algoritmo [Stallings, 99]. Uma vez que o texto cifrado é produzido, o mesmo pode ser transmitido. Na recepção, o texto cifrado pode ser transformado de volta para o texto original através do uso de um algoritmo de decifragem e da mesma chave que foi usada para a cifragem.

A segurança da criptografia convencional depende de alguns fatores. Primeiro, deve ser impraticável decifrar uma mensagem baseando-se apenas no texto cifrado. Além disso, a segurança da criptografia de chave única depende do segredo da chave, não do segredo do algoritmo. Com isto, assume-se que é impraticável decifrar uma mensagem baseando-se no texto cifrado e no conhecimento do algoritmo de cifragem/decifragem.

Em outras palavras, não é necessário manter o algoritmo em segredo; precisa-se manter a chave secreta. Esta característica da criptografia convencional é que faz com que a mesma seja praticável para uso em grande escala. O fato de que o algoritmo não precisa ser mantido em segredo significa que fabricantes podem desenvolver implementações, em chip de baixo custo, de algoritmos de criptografia simétrica. Estes chips estão vastamente disponíveis e incorporados em um grande número de produtos.

Várias cifras de chave simétrica foram criadas e incorporadas a vários produtos. A Tabela 3.5 apresenta algumas dentre as cifras de chave simétrica mais comuns [Tanenbaum, 03].

**Tabela 3.5: Algoritmos criptográficos de chave simétrica comuns**

<b>Cifra</b>	<b>Autor</b>	<b>Comprimento da chave</b>	<b>Comentários</b>
Blowfish	Bruce Schneier	1 a 448 bits	Velho e lento
DES	IBM	56 bits	Muito fraco para usar agora
IDEA	Massey e Xuejia	128 bits	Bom, mas patenteado
RC4	Ronald Rivest	1 a 2048 bits	Algumas chaves são fracas
RC5	Ronald Rivest	128 a 256 bits	Bom, mas patenteado
Rijndael (AES)	Daemen e Rijmen	128 a 256 bits	Melhor escolha
Serpent	Anderson, Biham, Knudsen	128 a 256 bits	Muito forte
DES triplo	IBM	168 bits	Segunda melhor escolha
Twofish	Bruce Schneier	128 a 256 bits	Muito forte; amplamente utilizado

### 3.3.3 Sistemas de Chave Pública

Na década de 1970, pesquisadores criaram a criptografia de chave assimétrica, uma nova maneira para enviar chaves seguramente. Esse esquema utiliza duas chaves diferentes. Mesmo estando relacionadas entre si, são significativamente diferentes. O relacionamento é matemático; o que uma chave encripta a outra chave decripta. Porque

ambas as chaves são necessárias para bloquear e desbloquear os dados, uma delas pode se tornar pública sem pôr a segurança em perigo. Essa chave é conhecida como a chave pública. Sua contraparte é chamada de chave privada [Tanenbaum, 03].

O uso de criptografia assimétrica para comunicação exige um poder computacional maior em ambos os lados da comunicação, devido ao fato das chaves serem maiores que as usadas nos algoritmos simétricos para manter o nível de segurança próximo ao destes, tornando a comunicação mais cara e lenta. Os algoritmos simétricos utilizam, normalmente, chaves de 64, 128, 192 ou 256 *bits*, enquanto que nos algoritmos assimétricos as chaves são de 1024, 2048 ou 4096 *bits*. Por este motivo, os algoritmos assimétricos são utilizados, normalmente, para a troca de chaves de sessão e assinatura digital [Burnett, 02].

### **3.3.4 Algoritmos padrão de criptografia**

A seguir, serão apresentados dados sobre o algoritmo de Criptografia Padrão de Dados – *Data Encryption Standard* (DES) e sobre o Padrão Avançado de Criptografia - *Advanced Encryption Standard* (AES). Para o primeiro (DES) será mostrado um esboço do seu funcionamento; para o segundo (AES) serão mostradas as características que o tornaram o algoritmo padrão atual.

#### ***Data Encryption Standard***

O *Data Encryption Standard* – DES - é um padrão criptográfico criado em 1977 pela IBM através de uma licitação aberta pela antiga Agência Nacional de Segurança americana - *National Security Agency* (NSA) [Schneier, 94]. Após algumas modificações no seu código original, chegou-se ao padrão de 64 bits de leitura, aplicando uma chave com 56 bits à mensagem. Tanto o algoritmo quanto a chave são simétricos, ou seja, são os mesmos utilizados na geração do arquivo criptografado e na sua descryptografia. Em sua forma original, o algoritmo já não é mais seguro; no entanto, em uma forma modificada, ainda é útil.

#### ***Advanced Encryption Standard***

Uma disputa muito acirrada ocorreu no final do milênio passado quanto à criação do novo padrão criptográfico. Durante 4 anos, cientistas da computação e matemáticos do



mundo disputaram pela publicação do seu algoritmo no *Advanced Encryption Standard* (AES). Os algoritmos concorrentes deveriam possuir blocos de leitura de 128 bits com chaves simétricas de 128, 192 ou 256 bits, código publicado para testes e liberação para padronizações do Instituto Nacional de Padrões e Tecnologia – *National Institute of Standard and Technology* (NIST) – caso fosse escolhido como vencedor [Hinz, 00].

O algoritmo Rijndael (comumente pronunciado “raïne-dol”), dos belgas Joan Daemen e Vicente Rijmen foi eleito o vencedor do concurso obtendo maior soma de pontos votados pelos Engenheiros de Sistemas do NIST e público em geral através de cartas e correspondências manuais ou eletrônicas.

### **Complexidade Computacional**

Devido às exigências do NIST para a submissão de algoritmos ao AES, impondo que os mesmos deveriam ser aplicados tanto em *software* quanto em *hardware*, os algoritmos criptográficos não são muito complexos computacionalmente. Por complexidade computacional se entende todos os fatores que caracterizam a implementação de um algoritmo, como por exemplo: memória necessária, velocidade de processamento, capacidade de ser aplicado em diversas plataformas, etc.

### **Requisitos de memória**

A capacidade de criptografar ou descriptografar usando pouca memória foi um requisito bastante valorizado na avaliação do algoritmo, visto que o mesmo pode ser aplicado em máquinas com muita memória ou em pequenos *smart cards* que dispõem de menos de 1KB de memória. Testes para comprovar a quantidade de memória necessária para a utilização dos algoritmos foram feitos [Schneier, 00].

A implementação do algoritmo em *hardware* talvez seja o ponto no qual mais se deseja uma economia de memória. Por exemplo, os *smart cards*, nos quais são implementados algoritmos de criptografia, possuem no máximo 256 bytes de memória RAM [Schneier, 00]. Na Tabela 3.6, conforme [Schneier, 00], é apresentada a quantidade de memória utilizada pelos últimos 5 algoritmos da fase final do concurso realizado pelo NIST, em suas implementações em *smart cards*.

**Tabela 3.6: Memória mínima requerida para implementação em *smart cards***

<b>Algoritmo</b>	<b>Memória Mínima Requerida (<i>bytes</i>)</b>
MARS	100
RC6	210
Rijndael	52
Serpent	50
Twofish	60

Mesmo um *smart card* de 256 *bytes* não comporta todos os tipos de algoritmos aqui apresentados, pois, além do bloco de encriptação (ou descriptação), o algoritmo necessita de mais memória para outras estruturas que são utilizadas durante a aplicação do algoritmo. Teoricamente, todos os algoritmos apresentados seriam implementáveis em *smart cards*, pois o algoritmo que mais necessita de memória é o RC6 com 210 *bytes*, cerca de 46 *bytes* a menos que o máximo utilizável no *smart card*. Entretanto, qualquer algoritmo que utiliza mais que 64 *bytes* não tem sua implementação em *smart card* garantida. Conclui-se, então, que somente o Rijndael, Serpent e Twofish têm sua implementação garantida [Hinz, 00].

### **Velocidade de processamento**

Apesar de segurança ser o requisito mais importante a ser analisado em um algoritmo criptográfico, a velocidade em que se consegue processá-lo é um fator determinante para a sua utilização, pois não adianta a segurança de um algoritmo estar diretamente relacionada a uma quantidade de processamento impraticável na tecnologia utilizada.

A relação segurança-velocidade também passa pelo ponto que envolve flexibilidade, ou seja, um algoritmo necessita ser implementável nas mais diversas plataformas. Uma comparação precisa e relevante dos algoritmos quanto à velocidade com que são processados é uma comparação a partir da qual são realizados testes nas mais diversas plataformas. A partir dos testes realizados em [Schneier, 00], conclui-se que o Rijndael e o Twofish são, na média, os algoritmos mais rápidos, tanto implementados em *Assembly* quanto em C [Hinz, 00].

O Apêndice B apresenta detalhes de funcionamento do Rijndael, que foi o algoritmo escolhido para a implementação no computador embutido. Alguns requisitos para a aprovação do mesmo como algoritmo padrão de criptografia (principalmente os relacionados à capacidade de utilizar pouca memória) coincidem com as restrições impostas por um sistema com pouca disponibilidade de recursos, a exemplo do microcontrolador.

### 3.4 Protocolo de comunicação

A principal função dos protocolos é gerenciar a comunicação entre os dispositivos da rede (equipamentos) para que se processem as trocas de dados de forma segura e ordenada. Os dados devem ter uma seqüência lógica quando transmitidos; sua integridade deve ser protegida contra eventuais erros e devem ser utilizados mecanismos de recuperação desses dados.

Para permitir que todos os equipamentos da rede troquem informações entre si, utiliza-se uma mesma “*linguagem*”, isto se trata do “*protocolo*”. Entre os principais protocolos existentes atualmente estão: TCP/IP, NetNEBUI, X.25, Frame Relay, ATM e o SPX/IPX [Lima, 02].

#### 3.4.1 Organizações Internacionais de Padronização

As organizações internacionais de padronização podem ser classificadas conforme seu enfoque técnico e por sua estrutura geográfica e política. Em relação a redes de computadores, as organizações mais importantes são [ISO, 04]:

- ISO: *International Organization for Standardization*. Lida com padrões que não são abordados pelos demais órgãos, como por exemplo: Mecânica, Química e outros.
- IEC: *International Electrotechnical Commission*;
- ITU-T: *International Telecommunications Union*;
- Em cada país existe um órgão, reconhecido internacionalmente e que é parte da ISO, responsável pela coordenação de seus trabalhos. No Brasil, este órgão é a ABNT (Associação Brasileira de Normas Técnicas) e nos Estados Unidos é a ANSI (*American National Standards Institute*).

### 3.4.2 O Modelo OSI e suas camadas

Para permitir o intercâmbio de informações entre computadores de diversos fabricantes, tornou-se necessário definir uma arquitetura padrão, que fosse aberta e pública para que todos os fabricantes pudessem ter acesso. Com este objetivo, a ISO definiu um modelo chamado de OSI (*Open Systems Interconnection*) também conhecido como RM-OSI (*Reference Model Open Systems Interconnection*).

A camada física desse modelo define as especificações elétricas, mecânicas, funcionais e de procedimentos para ativar, manter e desativar o link físico entre sistemas finais. Características como níveis de voltagem, taxas de dados físicos, distâncias máximas de transmissão, conectores físicos e outros atributos similares são definidos pelas especificações da camada física. Essa camada recebe o quadro da camada superior e o adapta ao meio de transmissão, ou seja, transforma-o em sinais elétricos (0 e/ou 1). Se for fibra óptica, transforma-o em sinais luminosos. O papel desta camada é efetuado pela placa de rede ou pelo Modem, por exemplo.

Há várias alternativas quanto à transmissão de dados na camada física. No contexto desse trabalho, necessita-se de um meio que permita a transmissão de dados a longas distâncias (devido à dispersão geográfica dos equipamentos monitorados). As principais alternativas são [Franca, 01]:

- **Linhas de energia.** A comunicação pelas linhas de energia (PLC - *Power Line Communication*), como o nome indica, utiliza como meio físico para comunicação, os próprios fios de energia que alimentam o equipamento monitorado. A PLC tem sido usada há décadas, com aplicações que vão desde automação residencial e predial ao controle de subestações. O principal problema a ser enfrentado é a interferência causada pela transmissão, bem como a susceptibilidade da própria transmissão a interferências vindas de equipamentos elétricos.
- **Modem sobre linha telefônica.** Para a comunicação, seria necessária a disponibilidade de uma linha telefônica e de um modem para cada comunicante.
- **Rádio PTT (*Push-to-talk*).** Cada equipamento monitorado teria seu próprio transceptor de rádio com capacidade de atingir uma estação central. O custo desta opção é provavelmente proibitivo neste projeto de telemetria dado o grande número de transmissores de alta potência envolvidos.

- **Rede de rádios.** Cada computador embutido poderia ter um pequeno rádio transceptor com alcance suficiente apenas para atingir outra estação monitorada. Formar-se-ia uma rede onde as informações seriam repassadas até atingir o objetivo. O custo desta opção seria relativamente alto e atingir o grau de confiabilidade necessário e a implementação do protocolo seriam as maiores dificuldades.
- **Canais de controle de celulares e satélites.** As células utilizadas na telefonia celular reservam canais para controle. Estes são usados para funções auxiliares como troca de células, medições de parâmetros do sistema, etc. Os protocolos utilizados nestes canais são diferentes dos utilizados pelos canais de assinantes. Determinadas empresas vendem soluções completas que utilizam estes canais para transmitir e receber pequena quantidade de dados em qualquer lugar com a cobertura da operadora de telefonia celular. Utiliza-se um aparelho celular simplificado, vendido pelas empresas que exploram este serviço. Este celular possui o módulo de RF e um microcontrolador programado com os protocolos. Outras empresas empregam técnicas similares com canais de satélite.

Considerando o custo/benefício de cada alternativa, a preferência é por utilizar modem sobre linhas discadas, uma tecnologia muito disseminada e já bastante dominada [Alencar, 98]. Entretanto, pode ocorrer que nem todas as estações monitoradas tenham acesso a uma linha de telefonia fixa. Nestes casos, poderiam ser utilizadas conexões de controle de celulares ou satélites como alternativa.

Alguns aspectos relacionados à eficiência da transmissão de dados, em um meio de transmissão como a linha telefônica, estão apresentados no Apêndice C.

### 3.4.3 Detecção de erros

Um dos métodos de detecção de erros mais utilizado é o código polinomial, também conhecido por código CRC – *Cyclic Redundancy Code*. Os códigos são baseados no processamento dos bits como representações de polinômios com coeficientes 0 e 1. Uma seqüência de  $k$  bits é considerada como uma lista de coeficientes de um polinômio com  $k$  termos entre  $x^{k-1}$  e  $x^0$ . Um polinômio deste tipo diz-se de grau  $k-1$ . O bit de ordem superior (mais à esquerda) é o coeficiente de  $x^{k-1}$ ; o bit seguinte é o coeficiente de  $x^{k-2}$  e assim sucessivamente. Por exemplo, a seqüência “110001” possui 6 bits, representando

por isso um polinômio com seis termos cujos coeficientes são 1, 1, 0, 0, 0 e 1:  $x^5 + x^4 + x^0$  [Rodrigues, 04].

Entre os polinômios, utiliza-se aritmética de módulo 2: tanto a adição como a subtração são idênticas à operação “OU EXCLUSIVO”; a divisão é efetuada como em binário exceto as subtrações que também são efetuadas com “OU EXCLUSIVO”; diz-se que um divisor “cabe” no dividendo se este possuir tantos bits quanto o divisor.

Quando se aplica um método de código polinomial, o emissor e o receptor têm que entrar previamente em acordo sobre um polinômio gerador,  $G(x)$ . Ambos os bits de ordem superior e inferior têm que ser 1. Para calcular o *checksum* de uma seqüência com  $m$  bits, correspondente ao polinômio  $M(x)$ , a seqüência deve ser maior que o polinômio gerador. A idéia é acrescentar um *checksum* no fim da seqüência de forma que o polinômio representado pela seqüência, já com o *checksum*, seja divisível por  $G(x)$ . Quando o receptor recebe a seqüência com o *checksum* incluído, tenta dividi-la por  $G(x)$ . Se no resultado da divisão o resto não for igual a zero, houve um erro de transmissão.

O algoritmo para o cálculo do *checksum* é o seguinte:

- Seja  $r$  o grau de  $G(x)$ . Acrescentam-se  $r$  bits “0” no lado de ordem inferior da seqüência, ficando a mesma com  $m + r$  bits correspondentes ao polinômio  $x^r M(x)$ ;
- Divide-se a seqüência correspondente a  $x^r M(x)$  pela seqüência de  $G(x)$  usando a divisão de módulo 2;
- Subtrai-se o resto (que possui sempre  $r$  ou menos bits) da seqüência de bits correspondentes a  $x^r M(x)$  usando subtração de módulo 2. O resultado é a seqüência, já com o *checksum*, que será transmitida. O seu polinômio é designado por  $T(x)$  que é sempre divisível por  $G(x)$ .

Existem três polinômios geradores que se tornaram normas internacionais:

$$\text{CRC-12} = x^{12} + x^{11} + x^3 + x^2 + x + 1$$

$$\text{CRC-16} = x^{16} + x^{15} + x^2 + 1$$

$$\text{CRC-CCITT} = x^{16} + x^{12} + x^5 + 1$$

O CRC-12 é usado quando a seqüência a ser verificada possui um comprimento de 6 bits. Os outros dois são usados quando as seqüências são de 8 bits. Um *checksum* de 16 bits, como CRC-16 e CRC-CCITT, detecta todos os erros simples e duplos, todos os erros com um número ímpar de bits, todos os erros em rajada (erros em seqüência) de comprimento igual ou inferior a 16, 99.997 % de erros em rajada de 17 bits de comprimento e 99.998 % de erros em rajada de comprimento igual ou superior a 18 bits [Rodrigues, 04].

### **3.5 Considerações finais sobre os algoritmos apresentados**

Este capítulo forneceu os conhecimentos básicos associados aos temas tratados neste projeto. Foram apresentados os requisitos e os componentes básicos para o projeto de um sistema embutido. Em seguida, foi visto que o microcontrolador, apesar de conter recursos limitados, pode ser usado como um computador embutido em um sistema de telemetria de baixo custo. Isto se deve ao fato de que esse dispositivo contém os componentes necessários para a coleta e armazenamento das medidas do equipamento monitorado.

Foram apresentadas técnicas de compressão de dados que podem viabilizar a implementação do sistema na medida em que possibilitam um processo de armazenamento na memória escassa do microcontrolador mais eficiente. É importante ressaltar que existem outras técnicas de compressão de dados que não foram apresentadas, mas que podem ser implementadas tanto no microcontrolador quanto na máquina gerente. No entanto, essas técnicas não foram apresentadas nem comparadas devido ao escopo do modelo escolhido para implementação. Neste modelo, a implementação do algoritmo de descompressão e parte do algoritmo de compressão é realizada na máquina gerente, que possui recursos em abundância. Dessa forma, no capítulo referente aos resultados, será realizada uma análise comparativa do modelo proposto e implementado no computador embutido, apresentando suas vantagens em relação ao método natural de implementação que seria: algoritmo de compressão de dados implementado no computador embutido que armazena os dados coletados do dispositivo monitorado e algoritmo de descompressão de dados implementado na máquina gerente para o processamento das medidas recebidas.

Para atender requisitos de segurança, foi realizado um estudo sobre técnicas de criptografia que poderão ser utilizadas de uma forma eficiente em sua implementação e na sua execução. Foram apresentados os requisitos do algoritmo a ser escolhido para sua adequação e implementação no microcontrolador.

Com relação à transmissão dos dados, foi apresentada a necessidade de implementação de um protocolo de comunicação para comunicação entre a estação monitorada e a estação gerente. A partir das alternativas relacionadas ao canal físico de comunicação a ser utilizado, a linha telefônica se mostrou uma escolha viável para esse sistema. Finalmente, foi apresentada a técnica escolhida para detecção de erros na transferência dos dados.



# Capítulo 4

## 4. Resultados Obtidos

---

Neste capítulo será realizada a descrição dos resultados obtidos referentes ao computador embutido utilizado para o armazenamento das medidas coletadas dos sensores ligados ao equipamento monitorado. Serão apresentados os resultados e análises dos modelos de compressão de dados e criptografia aplicados no armazenamento e transmissão das medidas, utilizando um microcontrolador, de uma forma eficiente e segura. Por último, será apresentado o protocolo de comunicação implementado que incorpora as técnicas de compressão e criptografia, além de considerar requisitos de tratamento de falhas de forma a garantir a integridade dos dados que trafegam no canal de comunicação.

### 4.1 Ambiente de desenvolvimento

Nas Figuras 4.1, 4.2, 4.3, 4.4 e 4.5 são apresentadas as partes que compõem o ambiente de desenvolvimento do projeto, incluindo os equipamentos e *softwares* utilizados.



**Figura 4.1:** Estação de gerência

### Características da estação de gerência:

- Sistema Operacional *Mandrake Linux* 9.0;
- JDK (*Java Development Kit*) 1.4.1\_01;
- Banco de Dados *Interbase* 6.0;
- Servidor *Jakarta Tomcat* 4.0.3;
- Modem interno, linha telefônica e conexão com a Internet.



Figura 4.2: Ambiente de simulação e programação do computador embutido

### Características da estação do computador embutido:

- Sistema Operacional Windows 98;
- Microcomputador conectado ao circuito de gravação via porta serial;
- PCW *Compiler*, IDE *Version* 3.26;
- *Software* de gravação EPIC for Windows, *Version* 2.41.

Na Figura 4.3 é apresentada a tela de monitoração dos dados armazenados no microcontrolador e de entrada e saída da transmissão desses dados através do Modem (uso da porta serial). Na parte superior (escura) estão mostrados os dados de leitura dos sensores e a tabela de codificação armazenada (todos os valores em decimal). Na parte inferior são mostrados os mesmos valores anteriores em seu formato hexadecimal.

Em seguida, na Figura 4.4 são mostrados os componentes do ambiente de desenvolvimento da aplicação do computador embutido.

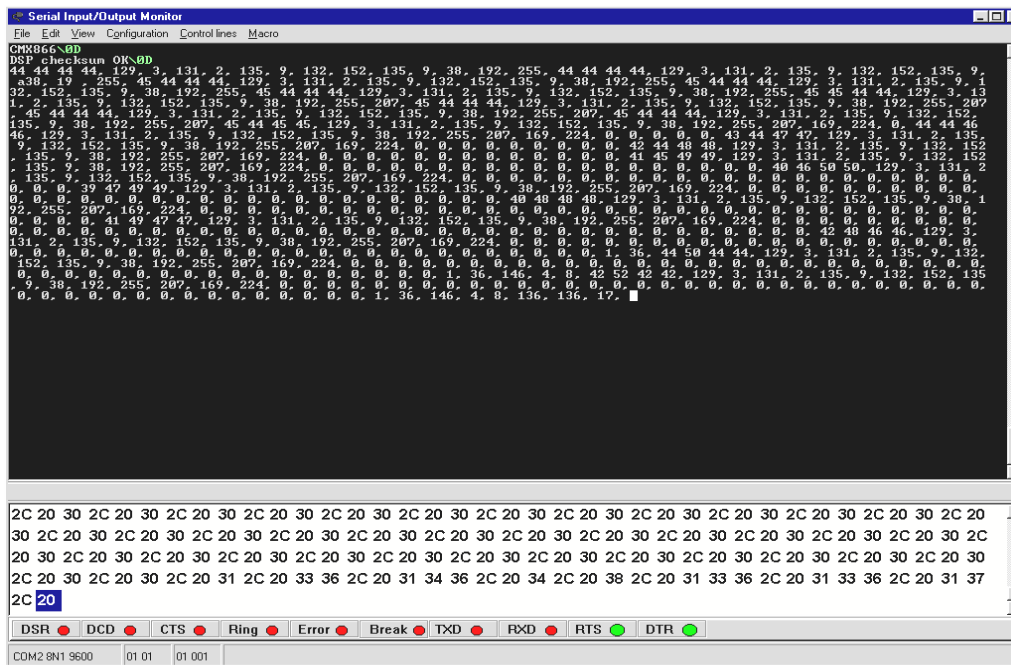


Figura 4.3: Tela de monitoração de entrada e saída dos dados da porta serial

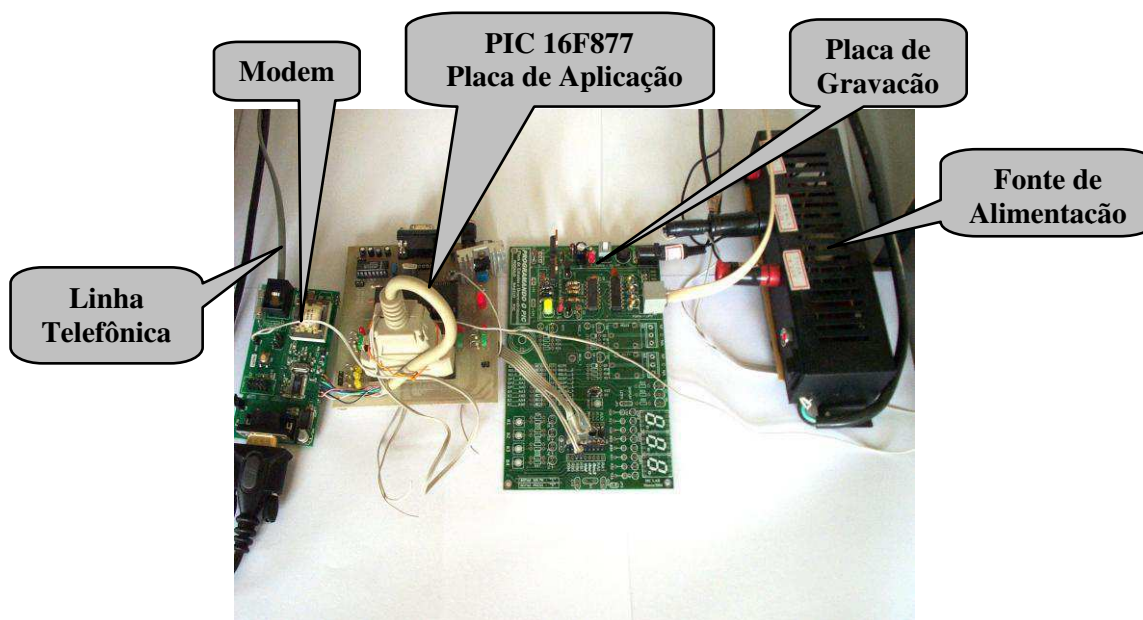


Figura 4.4: Ambiente de desenvolvimento da aplicação do computador embutido

Na Figura 4.5 é apresentada a vista de cima do Sistema de Dessalinização Piloto (SDP) da Figura 2.6 (Capítulo 2).

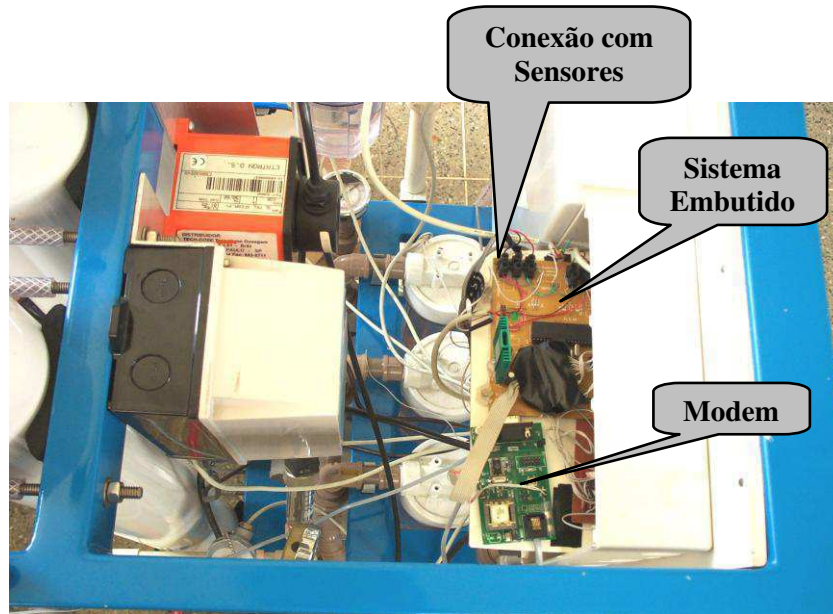


Figura 4.5: Sistema embutido conectado aos sensores do dessalinizador

## 4.2 Modelo de implementação do módulo embutido

Na Figura 4.6 é apresentado um diagrama em blocos com os componentes e respectivas conexões (fluxo de dados) do modelo para a implementação no computador embutido, de acordo com os requisitos apresentados anteriormente.

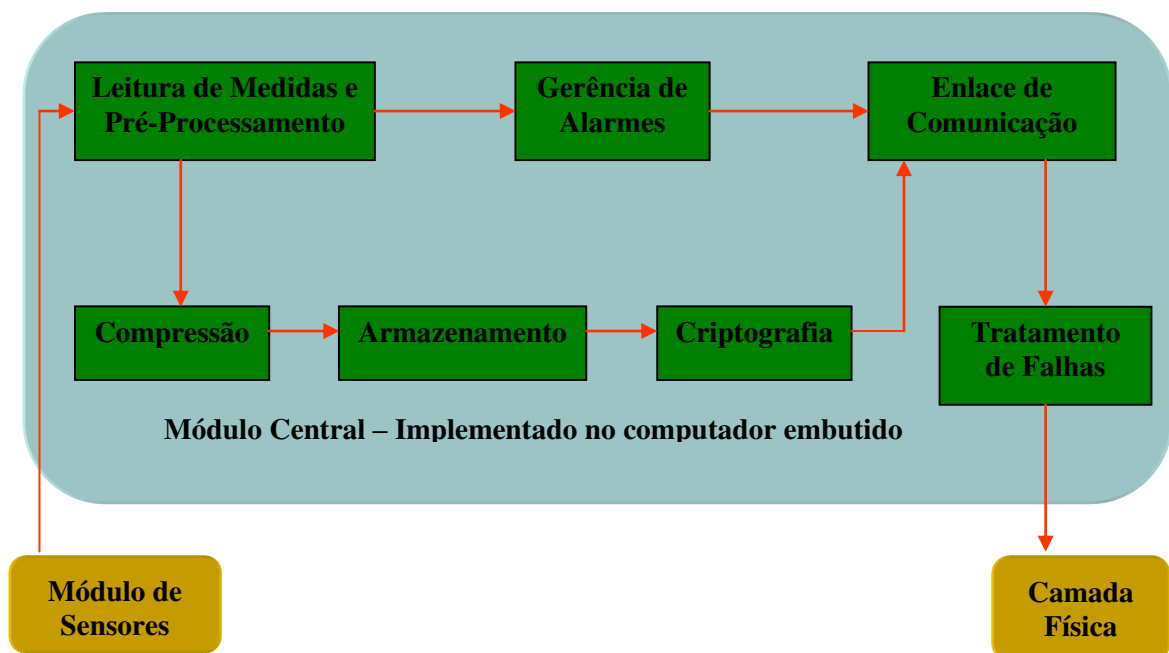


Figura 4.6: Diagrama em blocos do modelo proposto para o computador embutido

A seguir são descritas, brevemente, as funções de cada bloco do sistema:

**Módulo de Sensores** – Realização da medição das grandezas físicas. O sistema físico analisado gera continuamente e espontaneamente sinais que são captados pelo transdutor. O papel do transdutor é transformar esses sinais, com seus formatos originais (pressão, pH, vazão, etc), em grandezas elétricas (tensão, corrente, impedância, frequência, etc). Este módulo não faz parte do contexto deste trabalho e está implementado.

**Leitura de Medidas e Pré-processamento** – Responsável por realizar requisições ao módulo de sensores para a captação das medidas em intervalos de tempo pré-definidos. Nesta etapa, armazena-se temporariamente determinada quantidade de medidas sobre as quais são aplicadas técnicas para a eliminação de erros (cálculo da mediana) e conseqüente efetivação de uma medida.

**Compressão** – Responsável por aplicar uma técnica de compressão de dados sobre as medidas coletadas para compactá-las e enviá-las ao bloco responsável pelo armazenamento.

**Armazenamento** – Implementa a estrutura de dados que é utilizada para armazenar, temporariamente, as medidas antes de enviá-las ao servidor.

**Gerência de Alarmes** – Realiza o processamento necessário para a transmissão das medidas para o servidor em caso de alarme. A efetivação do alarme é baseada nos limites dos parâmetros que indicam as condições de alarme do equipamento que está sendo monitorado. Esses limites são configuráveis a partir de requisição do servidor. O teste de verificação de alarme deve ser realizado para cada medida coletada.

**Criptografia** – Módulo responsável por aplicar uma técnica de criptografia sobre os dados a serem transmitidos para a estação gerente.

**Enlace de Comunicação** – Implementa o protocolo de comunicação com o servidor para a transferência de medidas de uma forma robusta.

**Tratamento de Falhas** – Neste módulo é realizado o tratamento de eventuais falhas (definidas nos requisitos funcionais anteriormente) na transmissão de dados para o servidor.

**Camada Física** – Neste módulo encontra-se o dispositivo responsável por adaptar as características dos sinais de dados para a forma conveniente de transmissão e recepção via linha telefônica. Esta tarefa é realizada pelo Modem.



Na Figura 4.7 é apresentada a tela gerada pelo programa implementado no Servidor (gerente do sistema) que permite o acesso via *Web*, por parte do administrador, aos dados do sistema de monitoração remota. É apresentada a tabela contendo o resultado de consultas remotas ao equipamento (SDP) montado no Laboratório de Referência em Dessalinização. A tabela da figura apresenta os valores de todas as variáveis: P1 (Pressão de entrada dos filtros), P2 (Pressão de saída dos filtros), P3 (Pressão de entrada das membranas), P4 (Pressão de saída das membranas), Q1 (Vazão do permeado), Q2 (Vazão do concentrado), T (Temperatura) e pH (Potencial Hidrogeniônico) para o equipamento desejado e na faixa de datas solicitada.

Localização:  
Município: Campina Grande - PB  
Poço: LABDES  
ID: 40

Exibir gráficos das variáveis

Apagar	Data			(kgf/cm <sup>2</sup> )				(LPM)		T (°C)	pH	TDS (mg/L)
	Dia	Mes	Ano	P1	P2	P3	P4	Q1	Q2			
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,93	16,63	16,83	4,46	25,87	7,06	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,66	16,12	16,83	4,3	27,05	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,57	16,12	16,83	4,7	27,44	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,48	16,01	16,83	4,7	28,22	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,29	15,91	16,23	4,64	29,01	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,29	15,81	16,23	4,7	29,4	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,2	15,71	16,83	4,2	30,18	7,76	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,11	15,61	16,83	4,36	30,58	7,76	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,02	15,5	16,83	4,2	30,97	7,76	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,93	15,5	17,48	4,25	31,75	7,76	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,84	15,4	15,67	4,3	32,14	7,76	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,84	15,4	17,48	4,3	32,54	7,76	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,84	15,4	16,23	4,46	32,93	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,84	15,4	16,83	4,46	33,32	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,84	15,4	15,67	4,2	33,71	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,74	15,3	16,23	4,25	34,1	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	15,65	15,3	16,83	4,3	34,5	7,68	10267,
<input type="checkbox"/>	16	8	2004	1,45	1,5	16,02	15,71	16,83	4,58	31,36	7,68	10267,

Figura 4.7: Tabela de variáveis de medida

### 4.3 Computador embutido

Na fase inicial do trabalho, foi realizada uma pesquisa com o objetivo de relacionar plataformas utilizadas para solução de sistemas de telemetria (Apêndice A). A partir desta pesquisa, foi escolhido o microcontrolador para ser utilizado como computador embutido por atender às restrições de baixo custo impostas pelo sistema a ser

desenvolvido. A Figura 4.8 [Microchip, 01] apresenta o diagrama com os componentes internos do microcontrolador utilizado no projeto (PIC16F877) e as interfaces do mesmo para as conexões externas.

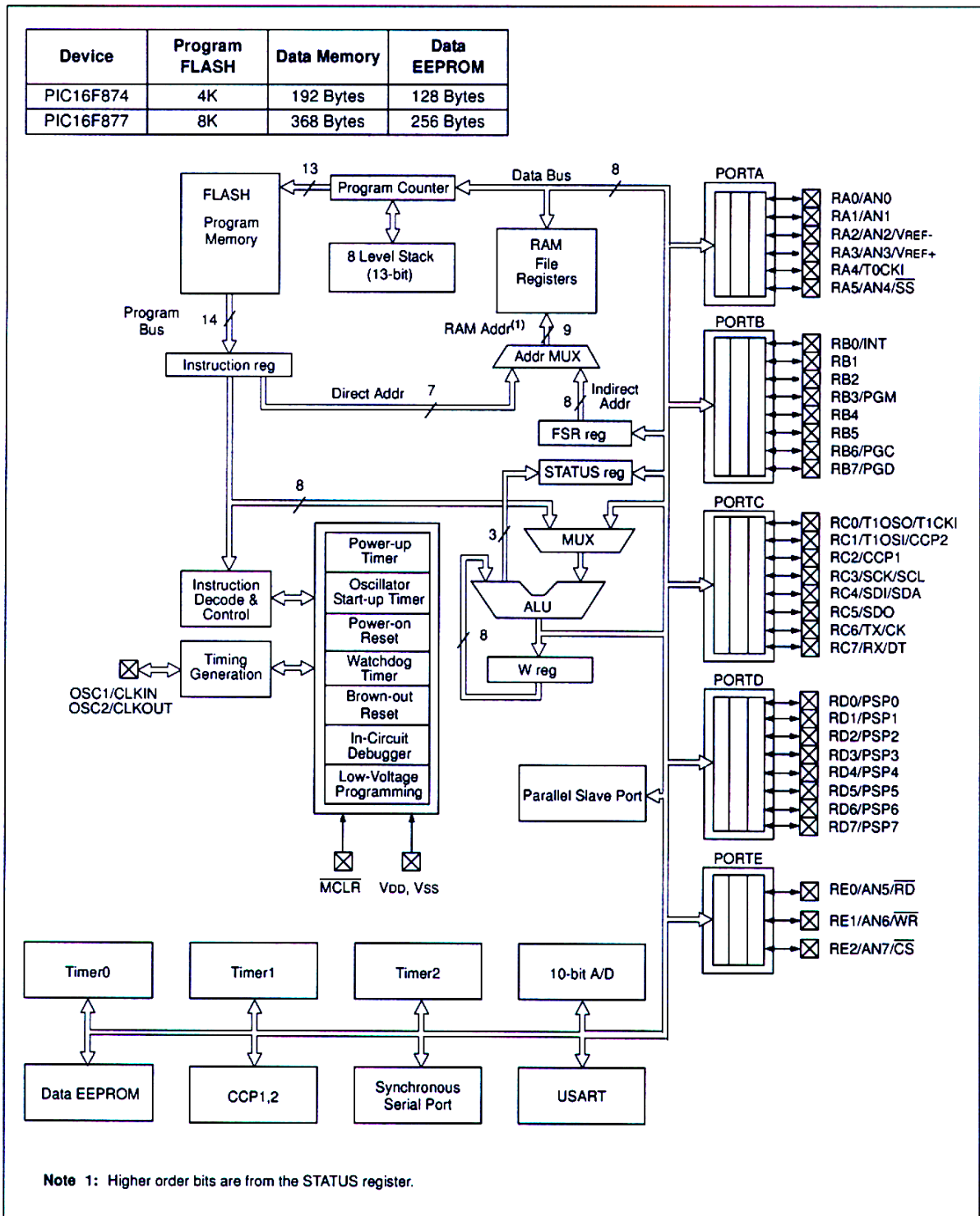


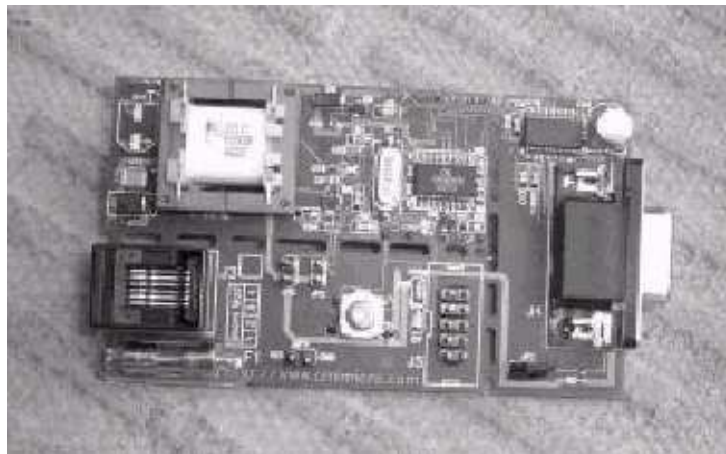
Figura 4.8: Esquema interno do microcontrolador PIC16F877

A seguir serão descritas as conexões do microcontrolador com os principais dispositivos (Modem e sensores) de forma a possibilitar a telemetria.

### 4.3.1 Interfaces do microcontrolador

#### MODEM

A transmissão dos dados, armazenados no microcontrolador, para a estação servidora foi realizada pelo modem DE8661. Este modem é uma placa de demonstração com protocolo *V.22 bis* e suporte a comandos AT, que inclui o projeto do circuito integrado CMX866, apresentado na Figura 4.9 [User, 02].



**Figura 4.9: Modem DE8661**

O modem possui as seguintes características:

- ✓ Interface de linha de 2 fios completamente isolada;
- ✓ Controlado via microcontrolador através de conexão RS232;
- ✓ Conjunto de comandos AT integral com conexão rápida;
- ✓ Operação em 3V ou 5V;
- ✓ Comunicação via DB9 (RS-232) ou via pinos socket modem (TTL).

A interface serial do microcontrolador interno do CMX866, para conexão com o terminal, é assíncrona e opera numa taxa de 9600 bps, com palavras de 8 bits, sem paridade. A taxa de transmissão de dados via linha telefônica é limitada em 2400 bps, conforme descrito no protocolo *V.22 bis*. Fisicamente, a conexão entre o microcontrolador e o modem ocorre através da porta D (porta digital bi-direcional de 8



bits), mostrada na Figura 4.8 (PORTD), e da USART (*Universal Synchronous Asynchronous Receiver Transmitter*) do PIC16F877. Os pinos RXTTL e TXTTL do modem estão conectados aos pinos Rx e Tx da USART e os demais pinos estão conectados à porta D.

A porta D, utilizada para a comunicação serial do Modem com o microcontrolador pode ser configurada como uma porta de entrada/saída bidirecional. As três primeiras linhas da Figura 4.10 definem os pinos indicados pelo comando *input* como entradas. As últimas linhas configuram o pino indicado como uma saída e “setam” o bit de acordo com o comando.

```
#define DSR      input(PIN_D1)
#define CTS      input(PIN_D2)
#define DCD      input(PIN_D3)
#define DTR_1    output_high(PIN_D4)
#define DTR_0    output_low(PIN_D4)
```

**Figura 4.10: Configuração dos pinos do microcontrolador**

- Sinal **DSR** (*Data Set Ready*) – Informação do Modem ao microcontrolador. Indica que o mesmo está pronto para operar, ou seja, que já está devidamente alimentado, que não existe nenhuma tecla de teste acionada e o dispositivo de conversão está conectado à linha telefônica.
- Sinal **CTS** (*Clear to Send*) – Indicação do Modem ao microcontrolador. Informa que o mesmo já está pronto para iniciar a transmissão de dados. Esse sinal foi utilizado para controlar o fluxo de dados de transmissão no sentido microcontrolador – Modem já que a velocidade de transmissão dessa interface é maior que a velocidade de transmissão de dados na linha telefônica.
- Sinal **DCD** (*Data Carrier Detector*) – Enviado pelo Modem ao microcontrolador. Indica que o mesmo está recebendo um sinal na linha com a característica de portadora. Esse sinal foi usado para indicar que o Modem está conectado, apto a transferir e receber dados.

- Sinal **DTR** (*Data Terminal Ready*) – utilizado para indicar ao Modem que o microcontrolador está pronto para operar.

Para a transmissão e recepção de dados foram utilizados, respectivamente, os pinos 6 e 7 da Porta C (porta digital bi-direcional de 8 bits) do microcontrolador, mostrada na Figura 4.8 (PORTC). A diretiva apresentada a seguir ativa o suporte à comunicação serial, especificando a velocidade de comunicação serial, seleciona os pinos utilizados para transmissão e recepção, seleciona a paridade, o número de bits de dados e associa a interface RS232 a um identificador de *stream* de dados.

**#use rs232 (baud=9600, xmit=pin\_c6, rcv=pin\_c7, parity=N, bits=8, stream = padrao)**

A configuração do Modem é realizada através de comandos AT que atribuem valores aos parâmetros determinados pelos registradores internos do mesmo. Para realizar a discagem, por exemplo, é necessário enviar o seguinte comando:

**ATDn** – em que *n* é o número a ser discado.

Através desses comandos, é possível estabelecer o modo de comportamento do Modem em sua comunicação com o microcontrolador e com o Modem remoto (localizado na máquina servidora). Para responder aos comandos AT enviados a partir do programa armazenado no microcontrolador e para indicar seu estado, o Modem utiliza alguns códigos listados na Tabela 4.1.

**Tabela 4.1: Códigos resultantes do Modem**

<b>Resposta Numérica (Decimal)</b>	<b>Correspondente significado alfanumérico</b>
00	OK
01	CONNECT
02	RING (tom de chamada)
03	NO CARRIER (sem portadora)
04	ERROR
05	NO DIALTONE
06	BUSY
07	CONNECT 2400
08	CONNECT 1200

## Sensores

Para a comunicação com os sensores foram utilizadas as entradas analógicas indicadas como AN (0-7) no esquemático da Figura 4.8. Esses são pinos de entrada do conversor A/D (Analógico/Digital) interno e são configurados através da diretiva *setup\_ADC\_ports* (opções), em que *opções* é uma variável ou constante inteira de 8 bits.

O comando *set\_adc\_channel* (valor) seleciona um canal de entrada para o módulo A/D interno e o comando *read\_adc()* efetua uma conversão A/D. Na Tabela 4.2 é apresentada a associação de cada pino do microcontrolador com o sensor do equipamento monitorado.

**Tabela 4.2: Conexão dos sensores com o microcontrolador**

<b>Pino do microcontrolador</b>	<b>Sensor utilizado</b>
AN0	Pressão de entrada dos filtros
AN1	Pressão de saída dos filtros
AN2	Pressão de entrada das membranas
AN3	Pressão de saída das membranas
AN4	Temperatura
AN5	PH

Os pinos RB0 e RB1 indicados no esquema do microcontrolador (Figura 4.8) foram utilizados para coletar medidas dos sensores de vazão do permeado e do concentrado do equipamento. Esses sensores geram uma onda quadrada como sinal de saída cujo período determina o valor da medida.

### 4.3.2 Programação do PIC em Linguagem C

Atualmente, a maioria dos microcontroladores disponíveis no mercado conta com compiladores de linguagem C para o desenvolvimento de *software*. O uso de C permite a construção de programas e aplicações muito mais complexas do que aquelas que seriam viáveis utilizando apenas a Linguagem *Assembly*. Além disso, o desenvolvimento em C permite uma grande velocidade na criação de novos projetos, devido às facilidades de programação oferecidas pela linguagem e também à sua portabilidade, o que permite adaptar programas de um sistema para outro com pouco esforço [Pereira, 2003].

Outro aspecto favorável da utilização da linguagem C é a sua eficiência que, no jargão dos compiladores, é a medida do grau de “inteligência” com que o compilador traduz um programa em C para o código de máquina. Quanto menor e mais rápido o código gerado, maior será a eficiência da linguagem e do compilador [Pereira, 03].

O aspecto eficiência é realmente muito importante quando se trata de microcontroladores cujos recursos são tão limitados como aqueles que fazem parte da família PIC. Afinal, quando se dispõe de apenas 512 palavras de memória de programa e 25 *bytes* de RAM (como no PIC12C508 e 16C54), é imprescindível que se economize memória. Além disso, a utilização de uma linguagem de alto nível como C permite que o programador preocupe-se mais com a programação da aplicação em si, já que o compilador assume para si tarefas como o controle e localização das variáveis, operações matemáticas e lógicas, etc [Pereira, 03].

## 4.4 Compressão

Para a adoção de um mecanismo de compressão de dados eficiente com vistas à sua utilização em um dispositivo com limitação de recursos, foi adotada a seguinte estratégia de modelagem:

1. Implementar o algoritmo de descompressão e parte do algoritmo de compressão de Huffman no Servidor, enviando a tabela de codificação resultante dos dados de medidas acumulados de todos os sensores para o computador embutido.
2. Implementar o algoritmo de descompressão e parte do algoritmo de compressão de Huffman/RLE de Zeros no Servidor, enviando a tabela de codificação resultante dos dados para o computador embutido. Essa tabela é resultante dos dados de medidas acumulados de todos os sensores para o computador embutido e está dividida em:
  - a. Tabela de codificação de Huffman para os valores decorrentes da quantidade de zeros encontrados na seqüência de diferenças de medidas;
  - b. Tabela de codificação de Huffman para as diferenças encontradas (exceto o zero).

Em seguida, é realizada uma análise comparativa das implementações dos modelos mencionados no computador embutido, considerando a eficiência dessas implementações com relação à utilização de memória temporária, de memória de programa e de memória de armazenamento de dados EEPROM (*Electrically Erasable Programmable Read-Only Memory*).

Nos próximos tópicos, os modelos serão descritos com detalhes de implementação e no final será apresentado um quadro comparativo da eficiência dos mesmos, seguido da análise dos resultados. A análise também mostra as desvantagens da implementação do algoritmo de compressão no computador embutido.

#### **4.4.1 Modelo 1: Algoritmo de Huffman no Servidor**

Para a implementação do modelo, foi considerada a captação de medidas de 3 sensores (mostradas na Figura 2.5 do Capítulo 2) para o armazenamento no computador embutido. A implementação é levada a efeito a partir dos seguintes algoritmos:

##### **I – Computador embutido (sem compressão):**

1. Inicialmente, armazena as medidas em seus formatos originais (valores resultantes da conversão A/D de 8 bits);
2. Recebe requisição de envio de medidas por parte do Servidor;
  - a. Envia medidas armazenadas;
  - b. Recebe tabela de codificação;
3. Apaga medidas anteriores e armazena as novas medidas codificadas.

##### **II – Servidor (medidas chegam pela primeira vez):**

1. Armazena primeira amostra;
2. Calcula e armazena diferenças entre amostras de medidas (fase de pré-processamento);
3. A partir dos valores de diferenças, monta tabela de distribuição de probabilidades e aplica o algoritmo de Huffman;
4. Armazena tabela de códigos contendo os seguintes campos: valor da diferença, codificação;
5. Constrói tabela de codificação (apresentada posteriormente) a ser enviada para o microcontrolador;

6. Envia tabela de codificação.

### **III – Computador embutido (com compressão):**

1. Coleta amostra de medidas;
2. Calcula diferença entre a amostra coletada e a última amostra de medidas enviada pela última vez ao Servidor;
3. Verifica na tabela se existe codificação para a diferença calculada;
  - a. Caso exista, o valor codificado da diferença é armazenado;
  - b. Caso não exista, armazena código de escape seguido do valor da diferença não codificado;
4. Recebe requisição de envio de medidas por parte do Servidor;
  - a. Envia medidas armazenadas;
  - b. Recebe tabela de codificação ou *flag* (a partir do segundo envio das medidas codificadas) indicando que não houve alteração na tabela;
5. Apaga medidas anteriores e armazena as novas medidas codificadas.

### **IV – Servidor (recepção de medidas codificadas):**

1. Descomprime os dados recebidos;
2. Calcula e armazena diferenças entre amostras de medidas (fase de pré-processamento);
3. A partir dos valores de diferenças, monta tabela de distribuição de probabilidades e aplica o algoritmo de Huffman;
4. Constrói tabela de codificação novamente;
5. Realiza teste sobre a condição para envio de tabela de codificação (explicado posteriormente):
  - a. Caso o teste seja positivo, envia nova tabela para o embutido e atualiza a tabela no ambiente local;
  - b. Caso contrário, envia *flag* para indicar que não há alteração na tabela.

A maioria dos parâmetros de operação do equipamento monitorado apresenta variações igualmente proporcionais. As medidas apresentam uma relação de dependência, ou seja, uma alteração em uma medida afeta todas as demais [Melo, 04]. Além disso, o

conversor A/D para os sensores analógicos apresenta a mesma resolução de 8 bits (256 possibilidades de medidas).

Essas informações foram determinantes para que fossem aplicados os cálculos das probabilidades considerando todos os valores possíveis de todos os sensores, com o objetivo de diminuir o espaço ocupado pela tabela de codificação na memória não-volátil. Dessa forma, a tabela de codificação é única e é aplicada para todos os sensores.

Em caso de independência entre os parâmetros, essa estratégia ocasionaria uma pior taxa de compressão já que, para um determinado sensor, certo valor seria freqüente e deveria ter uma palavra-código pequena. Contudo, usando uma única tabela, esse valor teria uma palavra-código maior, pois no cálculo geral seria bem menos freqüente. Ou seja, a codificação seria baseada em valores cuja distribuição de probabilidades é quase eqüitativa, diminuindo a Entropia e conseqüentemente piorando a taxa de compressão.

### Processo de Codificação

Seja a seguinte seqüência de diferenças para as variáveis T, P3 e P4, coletadas a partir de sensores conectados ao equipamento, conforme o modelo de dados mostrado na Figura 2.5 do Capítulo 2:

0, 0, 0 | 0, 0, 0 | 1, 1, 0 | 0, 0, 1 | -1, -1, 0 | 0, 0, 0 | 1, 1, 1 | 0, 0, 0 | 0, 0, 0 | -1, -1, -2 | 0, 0, 1 | 0, 0, 0 | 0, 0, -1 | 0, 0, 0 | 0, 0, 0 |

Desse conjunto de valores medidos, obtém-se a distribuição de freqüências e o respectivo código obtido após a aplicação do algoritmo de compressão no Servidor, como mostrado na Tabela 4.3.

**Tabela 4.3: Código de Huffman do Modelo 1**

Valor	Freqüência	Código
0	32	0
1	7	10
-1	5	110
-2	1	1110
-3	1	1111

A primeira coluna da tabela mostra os valores de diferenças encontrados no modelo de dados. A segunda coluna mostra a quantidade de ocorrências desses valores do total de diferenças coletado. Por último, a terceira coluna contém o código resultante da aplicação do algoritmo de Huffman sobre as frequências.

Com o objetivo de manter os valores a serem codificados dentro de um intervalo, o valor utilizado para indicar código de escape, para os casos em que ocorrerem valores de diferenças não codificadas, será o valor imediatamente inferior ao menor valor encontrado no modelo de dados. Sendo assim, como o menor valor encontrado foi  $-2$  de acordo com o modelo de dados, foi atribuído o valor  $-3$  para o código de escape. Como os valores de diferenças não codificadas não são comuns, aplica-se a frequência mínima de valor 1. A tabela de codificação a ser enviada para o computador embutido tem o formato mostrado na Figura 4.11.

**Figura 4.11: Formato da tabela de codificação do Modelo 1**

$X_1$		$X_n$		$L(X_1)$		$C(X_1)$		$L(X_2)$		$C(X_2)$		...		$L(X_n)$		$C(X_n)$	
-------	--	-------	--	----------	--	----------	--	----------	--	----------	--	-----	--	----------	--	----------	--

em que:

$X_1$  – Menor valor de diferença a ser codificada,

$X_n$  – Maior valor de diferença a ser codificada,

$L(X_1)$  – Tamanho do código de  $X_1$ ,

$C(X_1)$  – Código de  $X_1$ .

Os campos  $X_1$  e  $X_n$  denotam os extremos do intervalo dos valores de diferenças possíveis a serem codificadas e ocupam 1 byte cada para o armazenamento. Para a utilização eficiente da memória, os códigos são armazenados sequencialmente independentemente do seu tamanho. Para isso, é necessário saber em que posição do byte da EEPROM se inicia a próxima codificação. Sendo assim, a estratégia adotada consiste em armazenar a quantidade de bits que será necessária para gravar o código (Campo  $L(X_1)$  indicado acima). Esse campo da tabela ocupa um espaço que é determinado pelo tamanho do conjunto de valores das diferenças a serem codificadas.

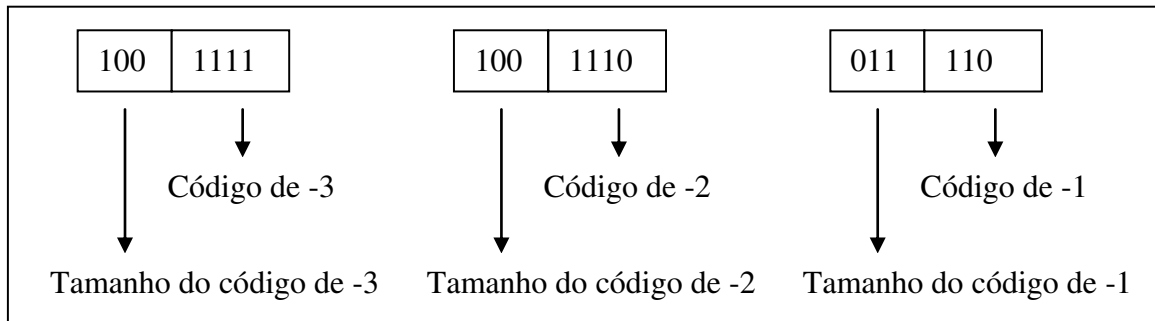
A partir dos resultados mostrados na Tabela 4.3, tem-se os dados da tabela de codificação, a ser enviada para o microcontrolador, mostrados na Tabela 4.4.



**Tabela 4.4: Tabela de codificação do Modelo 1**

Byte	1	2	3	4	5	6	7
Valor	10000011	00000001	10011111	00111001	11100010	01010000	11111111

O primeiro byte é decorrente do valor -3 mostrado na Tabela 4.3. Os números negativos são armazenados somando-se seus módulos ao valor 128, que corresponde à metade da quantidade de valores possíveis de diferenças. Nesse caso, o valor 128 é somado ao módulo de (-3) resultando em 131 (10000011 em binário). A seguir é apresentada uma descrição mais detalhada da estratégia de armazenamento adotada. A Figura 4.12 mostra a seqüência de bits a partir do terceiro byte, mostrado na Tabela 4.4, com o significado da mesma.



**Figura 4.12: Seqüência de bits da tabela de codificação**

A seqüência continua até a codificação do último valor (igual a 1 de acordo com a Tabela 4.3) seguida do delimitador (11111111).

Apenas 3 bits foram necessários para armazenar os tamanhos, pois o maior código encontrado (referente ao -3) possui 4 bits (1111), que é o tamanho máximo da aplicação do algoritmo de Huffman para esse caso. Para se obter o valor da quantidade de bits necessária, verifica-se o tamanho do valor binário correspondente. Dessa forma, o valor 4 (100 em binário) que indica o tamanho da seqüência de bits do código de -3 (1111) possui tamanho 3.

Com relação aos dois primeiros bytes da tabela de codificação, referentes à faixa dos valores possíveis, é necessário que se faça um esclarecimento. Pode haver casos em que alguns valores dentro da faixa não tenham ocorrido. Nesse caso, a estratégia adotada

consiste em atribuir frequência mínima para esses valores (uma ocorrência) e obter os códigos referentes aos mesmos após aplicação de Huffman. No estado normal de funcionamento do equipamento (dessalinizador), o aumento ou diminuição dos valores dos parâmetros de operação do mesmo acontece de forma gradual. Além disso, o sistema analógico de sensores funciona dentro de uma determinada faixa de operação. Sendo assim, não é natural a ocorrência de saltos nos valores. Essa característica, aliada ao fato de se buscar a utilização eficiente da memória do microcontrolador, justifica o método de armazenamento implementado.

Com a tabela armazenada no microcontrolador, as próximas medidas serão gravadas em sua forma codificada. Considerando o mesmo conjunto de medidas anterior, obtém-se o seguinte resultado na gravação, mostrado na Tabela 4.5.

Diferenças calculadas:

0, 0, 0 | 0, 0, 0 | 1, 1, 0 | 0, 0, 1 | -1, -1, 0 | 0, 0, 0 | 1, 1, 1 | 0, 0, 0 | 0, 0,  
0 | -1, -1, -2 | 0, 0, 1 | 0, 0, 0 | 0, 0, -1 | 0, 0, 0 | 0, 0, 0 |

**Tabela 4.5: Código das diferenças armazenadas do Modelo 1**

Byte	1	2	3	4	5	6	7	8	9
Valor	00000010	10000101	10110000	01010100	00000110	11011100	01000000	11000000	00000000

Quando o Servidor recebe, pela primeira vez, as medidas codificadas do computador embutido, este realiza o processo de descompressão, calcula e armazena a taxa de compressão conforme Equação 4.1 e envia a tabela de codificação.

$$T_c = L_m / T_m \quad (4.1)$$

em que:

$T_c$  – Taxa de compressão,

$L_m$  – Total do conjunto de medidas (comprimidas),

$T_m$  – Total de medidas recebidas (descomprimidas).

A partir da segunda remessa de medidas codificadas, após a descompressão no Servidor, é realizado um teste de condição para o envio da nova tabela para o computador

embutido. O teste verifica se a taxa de compressão da remessa atual de medidas é pior que a anterior; em caso afirmativo, a tabela é enviada e atualizada no computador embutido se a diferença entre as taxas compensa o tempo de transmissão da tabela. Ou seja, o valor máximo para essa diferença é um parâmetro a ser definido a partir da relação custo/benefício da transmissão da nova tabela, em que o custo está relacionado com o tempo de transmissão da tabela e o benefício se refere à quantidade de medidas adicionais a serem armazenadas na memória do computador embutido. Essa quantidade de medidas pode ser obtida a partir da diferença entre os valores de quantidades de medidas por byte dos dois últimos conjuntos de medidas codificadas.

A quantidade de medidas adicionais determinante para o envio da nova tabela é um parâmetro que depende dos requisitos do cliente. O exemplo a seguir ilustra um caso em que, provavelmente, seria necessário enviar uma nova tabela.

Exemplo:

Considerar que a memória disponível no microcontrolador, para armazenar as medidas, seja de 200 bytes. Se as medidas fossem armazenadas sem compressão, seria possível gravar 200 medidas (considerando 1 byte por medida). Com uma taxa de compressão de 1:5, seria possível armazenar 1000 medidas, conforme mostra o cálculo a seguir:

$$0.2 = 200/Tm \Rightarrow Tm = 1000 \text{ medidas}$$

Supondo que a taxa de compressão para uma nova remessa de medidas, calculada pelo Servidor, seja 1:2,5, seria possível armazenar apenas 500 medidas. Dessa forma, a diferença entre a quantidade de medidas (500) pode ser um valor considerável para os requisitos do cliente. Dessa forma, seria necessário o envio de uma nova tabela.

#### **4.4.2 Modelo 2: Algoritmo híbrido Huffman/RLE de Zeros no Servidor**

Os algoritmos e considerações referentes ao Modelo 1 são válidos para o Modelo 2. A distinção com relação à codificação e ao formato da tabela será mostrada nesta seção.

A primeira diferença é referente ao item 4 do Algoritmo II da seção anterior. No modelo 2 foram armazenadas 2 tabelas de códigos. A primeira tabela contém os seguintes

campos: quantidade de zeros e codificação. A segunda tabela contém os campos: valor da diferença e codificação. A necessidade desse artifício deve-se ao fato de que no algoritmo RLE de zeros são armazenadas duplas de valores:

- o primeiro indica quantos zeros foram encontrados em seqüência antes de aparecer um valor diferente;
- o segundo indica justamente qual foi o valor diferente.

A seguir são apresentados os valores de diferenças, idênticas aos valores do Modelo 1 para efeitos de comparação a ser realizada na última parte da seção.

Diferenças calculadas:

0, 0, 0 | 0, 0, 0 | 1, 1, 0 | 0, 0, 1 | -1, -1, 0 | 0, 0, 0 | 1, 1, 1 | 0, 0, 0 | 0, 0,  
0 | -1, -1, -2 | 0, 0, 1 | 0, 0, 0 | 0, 0, -1 | 0, 0, 0 | 0, 0, 0 |

Desse conjunto de diferenças medidas, os valores a serem armazenados segundo o modelo 2, usando RLE de zeros, são:

(6, 1), (0, 1), (3,1), (0, -1), (0, -1), (4, 1), (0, 1), (0, 1), (6, -1), (0, -1), (0, -2), (2, 1), (5, -1)  
e (6, 0).

O último valor 0 da dupla (6, 0) não precisa ser codificado, pois o Servidor já sabe a quantidade total de medidas enviadas. Dessa forma, quando completar os 6 últimos zeros, sabe-se que não existem outras medidas.

A partir desses valores, obtém-se a distribuição de freqüências e os respectivos códigos obtidos após a aplicação do algoritmo de Huffman, para os valores de quantidades de zeros encontrados e para os outros valores de diferenças, como mostrado nas Tabelas 4.6 e 4.7.

**Tabela 4.6: Código de Huffman para zeros do Modelo 2**

Número de zeros	Frequência	Código
0	7	1
6	3	000
5	1	0110
4	1	0111
3	1	0100
2	1	0101
1*	1	0010
-1	1	0011

\* valor inserido para preenchimento do intervalo

O valor -1 (obtido a partir do valor imediatamente inferior ao menor valor codificado) é utilizado para inserir o código de escape para os casos em que ocorrerem valores de diferenças não codificadas que estão fora da faixa. A tabela de codificação para os outros valores de diferenças é mostrada a seguir.

**Tabela 4.7: Código de Huffman para diferenças do Modelo 2**

Diferenças	Frequência	Código
1	7	0
-1	5	10
-2	1	110
-3	1	111

Novamente, o código da diferença -3 tem a função de código de escape. A tabela de codificação a ser enviada para o computador embutido tem o formato mostrado na Figura 4.13.

$  \begin{array}{l}  \mathbf{X_1} \mid \mathbf{X_n} \mid \mathbf{Y_1} \mid \mathbf{Y_n} \mid \mathbf{L(X_1)} \mid \mathbf{C(X_1)} \mid \mathbf{L(X_2)} \mid \mathbf{C(X_2)} \mid \dots \mid \mathbf{L(X_n)} \\  \mid \mathbf{C(X_n)} \mid \mathbf{L(Y_1)} \mid \mathbf{C(Y_1)} \mid \mathbf{L(Y_2)} \mid \mathbf{C(Y_2)} \mid \dots \mid \mathbf{L(Y_n)} \mid \mathbf{C(Y_n)}  \end{array}  $
---

**Figura 4.13: Formato da tabela de codificação do Modelo 2**

em que:

$X_1$  – Menor valor de quantidade de zeros a ser codificada,

$X_n$  – Maior valor de quantidade de zeros a ser codificada,

$L(X_1)$  – Tamanho do código de  $X_1$ ,

$C(X_1)$  – Código de  $X_1$ .

$Y_1$  – Menor valor de diferenças a ser codificada,

$Y_n$  – Maior valor de diferenças a ser codificada,

$L(Y_1)$  – Tamanho do código de  $Y_1$ ,

$C(Y_1)$  – Código de  $Y_1$ .

A partir das informações mostradas nas Tabelas 4.6 e 4.7, o Servidor monta uma seqüência de *bytes* correspondentes à tabela de codificação para os valores dos códigos calculados (Tabela 4.8). Essa tabela é enviada para o computador embutido.

**Tabela 4.8: Tabela de codificação do Modelo 2**

<b>Byte</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>
<b>Valor</b>	10000001	00000110	10000011	00000001	10000110	01110000	10100010
<b>Byte</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>
<b>Valor</b>	11000100	10001111	00011011	10000000	01111101	11100101	00010000
<b>Byte</b>	<b>15</b>	<b>16</b>	<b>17</b>	<b>18</b>	-	-	-
<b>Valor</b>	01111101	11100101	00010000	11111111	-	-	-

Os dois primeiros campos denotam os extremos do intervalo dos valores de quantidades de zeros encontrados nas medidas recebidas. Os dois valores seguintes informam o intervalo dos possíveis valores de diferenças encontrados. Nesse intervalo o valor zero não está incluído. O mesmo poderia ser usado se a seqüência de medidas terminasse com zeros. No entanto, no protocolo de comunicação, foi estabelecido que antes de enviar as medidas, é enviado um pacote contendo o total do número de medidas. Dessa forma, o Servidor sabe quando chegou o final da codificação.

Da mesma forma do Modelo 1, com a tabela armazenada no microcontrolador, as próximas medidas serão gravadas em sua forma codificada. A partir do conjunto de diferenças de medidas apresentadas a seguir (idêntico ao conjunto do Modelo 1), obtém-se o resultado na gravação mostrado na Tabela 4.9.

Diferenças calculadas:

0, 0, 0 | 0, 0, 0 | 1, 1, 0 | 0, 0, 1 | -1, -1, 0 | 0, 0, 0 | 1, 1, 1 | 0, 0, 0 | 0, 0,  
0 | -1, -1, -2 | 0, 0, 1 | 0, 0, 0 | 0, 0, -1 | 0, 0, 0 | 0, 0, 0 |

**Tabela 4.9: Código das diferenças armazenadas do Modelo 2**

Byte	1	2	3	4	5	6	7
Valor	00001001	00011011	00111010	10000101	10111001	01001101	00000000

### 4.4.3 Análise dos modelos implementados

Nesta seção será realizada uma análise comparativa entre os modelos implementados, bem como as vantagens desses modelos com relação à estratégia de modelagem referente à implementação do algoritmo de compressão no computador embutido.

As estatísticas de compilação do programa, geradas pela ferramenta PCW C *Compiler* IDE [Pereira, 03], permitem a comparação em termos relativos de utilização de memória de programa e de memória RAM entre as implementações dos dois modelos. Na Figura 4.14 é apresentada a estatística gerada para o Modelo 1 referente à implementação do algoritmo de Huffman no Servidor.

Lines	Stmts	%	Files	
264	196	35	d:\mauricio\embarcado9_huffman_remoto\dessalinizador.c	
37	15	2	d:\mauricio\embarcado9_huffman_remoto\pre-processamento.c	
283	218	28	d:\mauricio\embarcado9_huffman_remoto\enlace.c	
44	24	2	d:\mauricio\embarcado9_huffman_remoto\calculo_crc.c	
111	65	6	d:\mauricio\embarcado9_huffman_remoto\cifrador.c	
46	40	4	d:\mauricio\embarcado9_huffman_remoto\s_box.c	
-----	-----	---	-----	
2074	1116	Total		
Page	ROM	%	RAM	Functions:
0	130	3	7	processa_medidas_temporarias
0	138	4	12	gera_16bit_crc
0	79	2	4	transmitePacote
0	245	6	10	recebe_codigos
0	41	1	1	pedir_amostra
0	94	2	2	enviaTotalDeMedidas
0	82	2	1	delimitador_recebido
0	26	1	1	requisicao
0	394	10	10	get_sbox
1	989	25	2	trata_interrupcao_serial
1	317	8	5	armazenaLeitura
1	82	2	3	gravar
0	91	2	1	zeraMedidas
0	30	1	1	getPosicaoInicio

**Figura 4.14: Estatística de compilação do Modelo 1**

Na Figura 4.15 são apresentados os resultados estatísticos para a implementação do Modelo 2 referente ao algoritmo de Huffman e RLE de Zeros no Servidor.



ROM used: (51%)				
RAM used: (39%) at main() level				
(74%) worst case				
Lines	Stmts	%	Files	
-----	-----	---	-----	
290	244	38	d:\mauricio\embarcado10_rle\dessalinizador.c	
37	15	2	d:\mauricio\embarcado10_rle \pre-processamento.c	
290	246	28	d:\mauricio\embarcado10_rle \enlace.c	
44	24	2	d:\mauricio\embarcado10_rle \calculo_crc.c	
111	65	6	d:\mauricio\embarcado10_rle \cifrador.c	
46	40	4	d:\mauricio\embarcado10_rle \s_box.c	
-----	-----			
2140	1268	Total		
Page	ROM	%	RAM	Functions:
-----	-----	---	-----	-----
0	130	3	7	processa_medidas_temporarias
0	138	3	12	gera_16bit_crc
0	83	2	4	transmitePacote
0	263	6	10	recebe_codigos
0	43	1	1	pedir_amostra
0	102	2	3	enviaTotalDeMedidas
0	82	2	1	delimitador_recebido
0	26	1	1	requisicao
0	394	9	10	get_sbox
0	247	6	11	XorRoundKey
0	145	3	7	ShiftRows
0	68	2	4	SubBytes
1	1015	24	2	trata_interrupcao_serial
1	362	9	1	armazenaLeitura
1	82	2	3	gravar
0	91	2	1	zeraMedidas
0	30	1	1	getPosicaoInicio

**Figura 4.15: Estatística de compilação do Modelo 2**

Dentre as informações disponíveis nas figuras anteriores, destacam-se:

1. Memória FLASH utilizada para armazenamento do programa;
2. Memória RAM mínima e máxima necessária para execução do programa;
3. Quantidade de linhas de código de cada módulo do sistema;
4. Utilização das memórias para cada função implementada.

Considerando a quantidade de recursos disponíveis relacionados ao microcontrolador utilizado, além dos resultados de estatística mostrados anteriormente, do tamanho das tabelas de codificação (Tabelas 4.4 e 4.8) e da quantidade de bytes necessários para armazenar as diferenças (Tabelas 4.5 e 4.9), é possível obter os dados de utilização dos recursos, mostrados na Tabela 4.10.

**Tabela 4.10: Utilização dos recursos do microcontrolador**

Recurso (Memória)	Utilização	
	Modelo 1	Modelo 2
FLASH (ROM)	48 %	51 %
RAM (Melhor Caso)	36 %	39 %
RAM (Pior Caso)	74 %	74 %
EEPROM (Tabela)	2,7 %	7 %
EEPROM (Medidas)	3,5 %	2,7 %

Os dados da tabela anterior relacionados à porcentagem de utilização da quantidade total disponível de memória EEPROM (256 bytes) para os dois modelos foram obtidos da seguinte forma:

**Utilização da Tabela no Modelo 1 (%)** = Tamanho de ocupação da tabela (bytes) / Total de memória disponível (bytes) =  $7 / 256 = 2,7 \%$

**Utilização da Tabela no Modelo 2 (%)** =  $18 / 256 = 7 \%$

**Utilização das medidas no Modelo 1 (%)** = Tamanho de ocupação das medidas codificadas (bytes) / Total de memória disponível (bytes) =  $9 / 256 = 3,5 \%$

**Utilização das medidas no Modelo 2 (%)** =  $7 / 256 = 2,7 \%$

Pode-se observar, pelos dados encontrados, que é necessário mais memória de programa para armazenar o código do Modelo 2 devido à implementação de duas funções responsáveis pela codificação da quantidade de zeros e dos outros valores de diferenças encontrados. Esse código adicional, naturalmente, incrementa a demanda por memória temporária. Com relação à memória não-volátil, observa-se que a tabela do Modelo 1 ocupa menos espaço justamente por conter a codificação apenas dos valores de diferenças

encontrados. Em compensação, consegue-se uma taxa de compressão melhor usando a estratégia de codificar a seqüência de zeros.

As taxas de compressão obtidas após implementação dos modelos foram as seguintes:

$$\text{Taxa Compressão do Modelo 1} = 9 / 45 = 0,2 \text{ (1:5)}$$

$$\text{Taxa Compressão do Modelo 2} = 7 / 45 = 0,15 \text{ (1:6,67)}$$

A Entropia da fonte para o Modelo 1, dada pela Equação 3.1 (Capítulo 3), pode ser calculada da forma:

$$H(X) = (0.695 * 0.5249) + (0.1521 * 2.716) + (0.108 * 3.201) + (0.021 * 5.523) + (0.021 * 5.523)$$

$$H(X) = 0.3648 + 0.4131 + 0.3457 + 0.1159 + 0.1159 = 1.3554$$

A partir dos comprimentos dos códigos mostrados na Tabela 4.3, pode-se calcular a eficiência do código no Modelo 1 dada pela razão entre a Entropia da fonte e o comprimento médio do código, conforme descrição a seguir.

$$\begin{aligned} \text{Eficiência} &= 1.3554 / (1 * 0.659) + (2 * 0.1521) + (3 * 0.108) + (4 * 0.021) + (4 * 0.021) \\ &= 93.14 \% \end{aligned}$$

Os mesmos cálculos aplicados para o Modelo 2 resultam em uma eficiência de código de 99.35 % para a codificação de zeros e 91.81 % para a codificação dos outros valores de diferenças. Pode-se verificar que o algoritmo de Huffman apresenta uma eficiência alta para o modelo de dados obtido. Quanto maior essa eficiência melhor a taxa de compressão, já que o mapeamento dos dados é realizado com códigos menores.

### **Implementação do algoritmo no computador embutido**

A implementação do algoritmo de compressão de Huffman no microcontrolador e conseqüente implementação do algoritmo de descompressão no Servidor apresentam algumas desvantagens. É necessário utilizar o recurso de alocação dinâmica de memória para montar a árvore de Huffman já que não se sabe, inicialmente, o total de valores a ser

codificado. Esse mecanismo, juntamente com a utilização de funções recursivas, não é permitido pelo compilador CCS devido às restrições impostas pela quantidade escassa de recursos de memória disponível e ao pequeno tamanho da pilha. O algoritmo pode ser implementado com a restrição de que teria que ser alocada uma quantidade máxima de memória para essa função. Esse método é bastante ineficiente, já que boa parte da memória temporária estaria comprometida para esse armazenamento. Adicionalmente, na memória EEPROM, além da tabela de frequências, seria necessário armazenar um contador para cada valor de diferença codificado, para que fosse possível atualizar a tabela de frequências após a transmissão em caso de mudança de frequência dos valores das medidas (feita pelo Servidor nos modelos implementados).

Além das questões abordadas anteriormente, a necessidade de pouca memória de programa e memória temporária (mostradas abaixo) para armazenar e executar o código responsável por manipular as tabelas nos dois modelos mostra que essa é uma alternativa eficiente comparando-se com a implementação do algoritmo no computador embutido.

#### **Última versão da aplicação completa no microcontrolador com parte do algoritmo de compressão (Modelo 1):**

ROM used: (87%)

RAM used: (38%) at main() level

(83%) worst case

#### **Última versão da aplicação completa sem o algoritmo de compressão:**

ROM used: (81%)

RAM used: (38%) at main() level

(82%) worst case

A partir dos valores mostrados acima, pode-se calcular a quantidade de memória de programa utilizada pelo algoritmo de compressão:

$((87 - 81) / 100) * 8K = 491$  palavras de memória

Percebe-se que a quantidade de memória temporária permanece praticamente a mesma. Isto deve ao fato de que foi realizada uma otimização através da reutilização de variáveis.

## 4.5 Criptografia

Devido aos requisitos do NIST (*National Institute of Standard and Technology*) para a criação do novo padrão criptográfico, os vários algoritmos propostos foram submetidos a testes de execução. Entre os itens selecionados para comparação estavam:

1. Complexidade computacional (capacidade de ser aplicado em várias plataformas, etc);
2. Requisitos de memória;
3. Velocidade de processamento;
4. Quantidade de funções criptográficas (flexibilidade);
5. Segurança.

O algoritmo escolhido para a implementação no microcontrolador deveria atender às restrições relacionadas à escassez da quantidade de recursos disponíveis. Alguns objetivos a serem alcançados pelo algoritmo escolhido pelo NIST coincidem com os aspectos relacionados ao sistema de telemetria apresentado neste trabalho, tais como: necessidade de pouca memória e alta velocidade de processamento. A programação do dispositivo utilizado para a implementação do algoritmo (microcontrolador) é feita por um usuário através de um gravador conectado ao dispositivo e a um computador que envia os dados do programa compilado. Dessa forma, como a gravação da chave é feita localmente, apenas o programador precisa conhecê-la. Todos os dados criptografados são enviados para uma máquina servidora localizada na estação de controle. Logo, apenas esta máquina recebe as informações. Enfim, como não há necessidade de compartilhamento das informações, não é necessário a utilização de criptografia de chave pública entre computador embutido e o servidor para o envio da chave.

O algoritmo de criptografia simétrico implementado (*Rijndael*) atendeu aos requisitos referentes à escassez de memória disponível e poder de processamento do microcontrolador utilizado. A seguir, serão apresentados os relatórios, gerados pela ferramenta de compilação utilizada, os quais contêm os resultados com relação à quantidade de memória de programa e memória RAM utilizados para a implementação das funções do algoritmo de criptografia.

O relatório da Figura 4.16 mostra os resultados estatísticos de utilização de recursos da implementação do algoritmo Rijndael isolado do contexto da aplicação principal.

Neste momento, o algoritmo foi implementado com o objetivo de realizar testes de desempenho e aquisição de dados sobre a quantidade de recursos utilizados para a sua viabilização. Nessa implementação, as seguintes considerações devem ser feitas:

1. Por causa da quantidade de memória limitada do PIC, não é viável armazenar a chave expandida que o algoritmo *Rijndael* utiliza, de forma que é necessário calcular essa chave quando a mesma for requisitada durante a execução do algoritmo;
2. Pelo mesmo motivo anterior, o valor da tabela de substituição *S-Box* é calculado no momento da utilização da função;
3. A chave (16 *bytes*) foi armazenada na memória não-volátil EEPROM.

ROM used: (12%)				
RAM used: (5%) at main() level				
(15%) worst case				
Lines	Stmts	%	Files	
-----	-----	---	-----	
131	49	55	d:\mauricio\embarcado6\cifrador.c	
252	0	0	C:\ARQUIVOS DE PROGRAMAS\PICC\devices\16f877.h	
46	40	29	d:\mauricio\embarcado6\s_box.c	
Page	ROM	%	RAM	Functions:
----	---	---	---	-----
0	346	34	10	get_sbox
0	27	3	4	FFMul
0	126	13	18	MixColumns
0	304	30	11	XorRoundKey
0	111	11	5	ShiftRows
0	53	5	2	SubBytes

**Figura 4.16: Relatório da implementação da versão 1 do *Rijndael***

A partir dos dados estatísticos e considerando a disponibilidade de memória no microcontrolador utilizado (PIC16F877), foi possível encontrar os seguintes resultados com relação à quantidade de *bytes* usados:

**Disponibilidade:**

RAM → 368 bytes

ROM → 8 K palavras de memória

**Utilizado pelo *Rjindael*:**

$$\text{RAM (pior caso)} = 368 * (15 / 100) = 55 \text{ bytes}$$

$$\text{ROM} = 8192 * (12 / 100) = 983 \text{ palavras de memória}$$

As restrições de implementação dos itens 1 e 2 mostrados anteriormente aumentam o tempo de execução do algoritmo, conseqüentemente incrementam o tempo de resposta. Após a execução do algoritmo sobre o microcontrolador, foi calculado o tempo de 2 segundos para cada 3 *bytes* cifrados. Considerando que o algoritmo deve ser aplicado sobre um bloco de tamanho fixo (16 *bytes*), seria necessário pouco mais de 10 segundos para a execução desse algoritmo sobre cada bloco. Esse tempo de resposta seria inviável em aplicações nas quais fosse necessário enviar e receber dados simultaneamente. No entanto, no sistema de telemetria apresentado, as medidas são armazenadas no microcontrolador antes de serem enviadas em intervalos de tempo pré-definidos que excedem bastante o tempo necessário para cifrar os dados gravados em toda a memória disponível. O algoritmo de criptografia é aplicado aos dados após a conclusão do décimo sexto byte de cada bloco armazenado na EEPROM e antes do estabelecimento da comunicação.

As considerações desta implementação podem ser aplicadas para sistemas de telemetria nos quais a transmissão de dados é simultânea, mas que não possuem restrições de tempo real críticas, em que seria necessária a atuação rápida e garantida.

Adiante, será apresentada parte dos resultados de utilização de recursos da implementação do algoritmo Rijndael incluído na aplicação principal, retirados do relatório gerado pelo compilador. As restrições 1 e 2 mostradas no início da seção foram mantidas pelos mesmos motivos explicados anteriormente. No entanto, houve uma mudança no item 3. Devido à necessidade de economizar memória não-volátil EEPROM, que armazena as medidas coletadas e os parâmetros do sistema, a chave foi gravada na memória RAM. Para a cifragem de cada bloco, a chave precisa ser gravada novamente com seus valores iniciais. Ou seja, o valor original da chave só fica disponível na memória RAM por um curto intervalo de tempo (milissegundos).

**Autenticação**

Outra funcionalidade relativa à segurança dos dados deve ser implementada já que o sistema ainda poderia ser sabotado com as técnicas aplicadas até o momento. O

sabotador poderia repetir os quadros enviados para o Servidor a partir da segunda conexão com o embutido. Esses quadros estariam criptografados e indicariam ao Servidor que o estado de operação do equipamento monitorado continua o mesmo. Para resolver esse problema foi criado um mecanismo de autenticação. Se as próximas medidas armazenadas no sistema forem idênticas às anteriores enviadas ao Servidor, a codificação será a mesma já que a tabela de compressão e a chave de criptografia não mudam. No entanto, foi acrescentada uma informação adicional referente ao contador de conexões. Esse contador é incrementado no computador embutido após o recebimento correto da tabela de codificação do Servidor e é gravado no início da área reservada ao armazenamento das medidas. Dessa forma, mesmo que os valores das medidas sejam iguais, os valores enviados para o computador embutido serão diferentes.

#### **Aplicação principal com o algoritmo de criptografia:**

ROM used: (62%)

RAM used: (28%) at main() level  
(75%) worst case

#### **Aplicação principal sem o algoritmo de criptografia:**

ROM used: (47%)

RAM used: (25%) at main() level  
(50%) worst case

A partir dos resultados mostrados, é possível calcular a quantidade de memória utilizada para a implementação do algoritmo no sistema final.

RAM (pior caso) =  $368 * ((75 - 50) / 100) = 92 \text{ bytes}$

ROM =  $8192 * ((62 - 47) / 100) = 1228 \text{ palavras de memória}$

É possível verificar que houve um acréscimo nesses valores com relação aos obtidos anteriormente a partir da implementação isolada da aplicação principal. Esse incremento é justificado considerando que na implementação atual o armazenamento da chave é feito na memória RAM, além da necessidade de implementação de funções adicionais para a adequação do algoritmo à aplicação principal.



Após a recepção dos dados pelo servidor, o mesmo deve aplicar o algoritmo de decifragem com a conexão estabelecida. Isto é decorrente da necessidade do cálculo da tabela de codificação, mostrada neste capítulo na seção 4.4.1, para posteriormente ser enviada ao computador embutido. Surge, então, a necessidade de minimizar o tempo de resposta da aplicação do algoritmo para redução dos custos de utilização do canal.

A grande disponibilidade de recursos do servidor e a baixa complexidade computacional do algoritmo (requisito necessário para tornar-se algoritmo padrão de criptografia) resultam em um ótimo desempenho da aplicação do *Rijndael* sobre os dados recebidos, obtendo-se um tempo de resposta muito baixo. Esse tempo também é melhorado pelo fato das restrições 1 e 2 não existirem, possibilitando o armazenamento dos valores da chave expandida e dos valores de substituição da *S-Box* antes da utilização dos mesmos.

A chave utilizada para criptografar e descriptografar os dados deve ser aleatória e pode ser criada no servidor através de algum mecanismo de geração de números aleatórios. Também é necessário um mecanismo para a proteção contra leituras indevidas das chaves armazenadas no servidor. No mecanismo adotado, as chaves precisam estar disponíveis a todo instante já que as conexões oriundas dos vários computadores embutidos podem ocorrer a qualquer momento para o envio de dados.

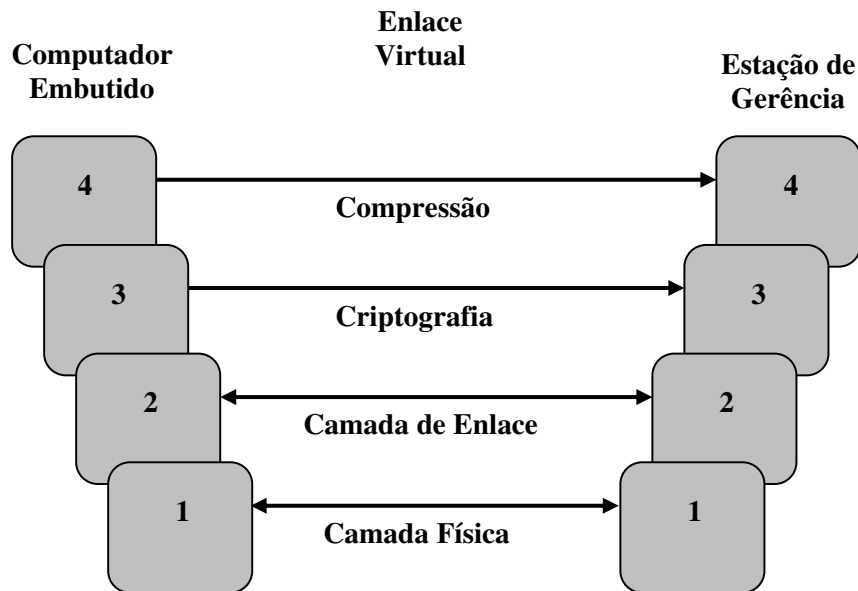
## 4.6 Protocolo de Comunicação

Nesta seção, será apresentado o protocolo de comunicação implementado, que inclui as fases de compressão e criptografia dos dados da aplicação embutida, além de atender aos requisitos de tratamento de falhas para a robustez do mesmo. Serão mostrados os formatos dos campos dos pacotes enviados pela camada de enlace e as questões referentes às temporizações.

O protocolo de comunicação implementado objetivou a transferência de dados de medidas armazenados no computador embutido para a estação de gerência. A transmissão desses dados deve ser realizada de uma forma correta e eficiente. Para atender ao primeiro requisito, foram implementados alguns mecanismos, mostrados posteriormente, para tratamento de possíveis falhas ocorridas durante o envio dos dados. A utilização do método de compressão de dados possibilita a eficiência requerida na medida em que:

1. Diminui a quantidade de dados a serem enviados, reduzindo o tempo de transmissão dos mesmos;
2. Minimiza a possibilidade de erros que ocasionam retransmissões.

Na Figura 4.17 é apresentada a comunicação virtual entre as camadas do protocolo de comunicação implementado.



**Figura 4.17: Comunicação virtual entre as camadas do protocolo**

A camada física é responsável pela transmissão dos dados via linha telefônica e é implementada pelo Modem. As funcionalidades implementadas pelas camadas de compressão e criptografia foram apresentadas em outras seções.

A camada de enlace, entre outras funções, implementa os seguintes requisitos:

1. Fornece interface bem definida para a aplicação principal do sistema;
2. Organiza os dados recebidos da aplicação em quadros (*frames*) a serem transmitidos na rede física;
3. Trata erros na transmissão.

#### **4.6.1 Descrição das etapas**

A seguir será apresentado o protocolo implementado pelo computador embutido e pelo Servidor para a transmissão dos dados, acrescido do formato dos quadros:

1. EMBUTIDO ativa mecanismo de interrupção durante intervalo entre coleta de medidas, podendo atender à requisição de conexão do modem remoto.
2. SERVIDOR estabelece conexão com EMBUTIDO.
3. SERVIDOR requisita envio de medidas:

<b>Delimitador</b>	<b>Requisição</b>	<b>CRC</b>	<b>Delimitador</b>
--------------------	-------------------	------------	--------------------

4. Se EMBUTIDO recebeu a requisição corretamente, vai para o próximo item; caso contrário, vai para o último item dessa seqüência.
5. EMBUTIDO envia quadro contendo dois campos: o primeiro referente ao número total de valores resultantes da coleta das medidas; o segundo se refere ao número total de valores enviados (menor que o valor do primeiro campo se as medidas estiverem comprimidas):

<b>Delimitador</b>	<b>Total de medidas</b>	<b>Total de valores</b>	<b>CRC</b>	<b>Delimitador</b>
--------------------	-------------------------	-------------------------	------------	--------------------

6. EMBUTIDO aguarda confirmação do SERVIDOR:

<b>Delimitador</b>	<b>Flag</b>	<b>Delimitador</b>
--------------------	-------------	--------------------

- a. Se a confirmação for positiva, vai para o próximo item da seqüência principal;
- b. Se for negativa, vai para o último item da seqüência principal.
7. EMBUTIDO envia medidas para o SERVIDOR:

<b>Delimitador</b>	<b>Contador</b>	<b>Versão</b>	<b>X medidas</b>	<b>CRC</b>	<b>Delimitador</b>
--------------------	-----------------	---------------	------------------	------------	--------------------

8. SERVIDOR detecta quadros errados e armazena seus contadores.
9. Enquanto houver quadro errado:
  - a. SERVIDOR pede retransmissão a partir do seu contador;

<b>Delimitador</b>	<b>Contador</b>	<b>Delimitador</b>
--------------------	-----------------	--------------------

- b. EMBUTIDO retransmite quadro pedido (o formato do quadro é o mesmo daquele mostrado no item que indica o envio de medidas para o Servidor).
10. Quando SERVIDOR detecta que todos os quadros estão corretos, envia confirmação através de um quadro contendo um campo que indica o número total de quadros enviados acrescido de uma unidade.

<b>Delimitador</b>	<b>Total de quadros + 1</b>	<b>Delimitador</b>
--------------------	-----------------------------	--------------------

11. EMBUTIDO recebe quadro de confirmação de medidas recebidas e aguarda a tabela de codificação.
12. SERVIDOR descriptografa medidas.
13. Se as medidas estão sendo enviadas pela primeira vez:
  - a. SERVIDOR monta tabela de codificação e a envia para EMBUTIDO:
 

<b>Delimitador</b>	<b>Contador</b>	<b>X valores</b>	<b>CRC</b>	<b>Delimitador</b>
--------------------	-----------------	------------------	------------	--------------------
  - b. EMBUTIDO pede retransmissão dos quadros errados da tabela (mesmo formato do quadro do item que indica o pedido de retransmissão do Servidor quando recebe algum quadro com medidas errado);
  - c. SERVIDOR envia novamente quadros errados.
14. Se as medidas estão sendo enviadas pela segunda vez:
  - a. SERVIDOR calcula tabela de codificação e envia flag para o EMBUTIDO indicando que não há alteração da tabela;
  - b. SERVIDOR aguarda confirmação de EMBUTIDO;
  - c. Vai para o penúltimo item da seqüência.
15. Se nenhuma das duas condições anteriores for satisfeita:
  - a. Servidor realiza teste de autenticação das medidas:
    - i. Se o teste confirmar a autenticidade da mensagem, vai para o item b da seqüência,
    - ii. Em caso de falha no teste, vai para o item que finaliza o protocolo.
  - b. SERVIDOR calcula tabela de codificação e realiza teste de condição (mostrado na seção referente aos resultados da compressão de dados deste capítulo) de envio da mesma;
  - c. Se a condição do item b for satisfeita:
    - i. SERVIDOR envia tabela de codificação;
  - d. Caso a condição do item b não seja satisfeita:
    - i. SERVIDOR envia *flag* indicando que não há alteração da tabela;
    - ii. SERVIDOR aguarda confirmação de EMBUTIDO;
    - iii. Vai para o penúltimo item da seqüência principal.
16. Quando EMBUTIDO detecta que todos os quadros estão corretos, envia *flag* de confirmação.
17. SERVIDOR recebe confirmação do EMBUTIDO.
18. Servidor finaliza protocolo.

O envio do número total de valores permite que o Servidor saiba, a partir do tamanho do quadro, qual será o número máximo referente ao contador de um dos quadros transmitidos. Dessa forma, o não envio deste será detectado pelo Servidor. O campo referente ao número total de medidas informa ao Servidor quantas medidas foram transmitidas. Já que na primeira transmissão, as medidas não estão codificadas, os dois campos anteriores possuem valores iguais. Este segundo campo permite que o algoritmo de descompressão seja executado corretamente, não ultrapassando o número total de medidas comprimidas.

No protocolo, é possível verificar que não existe confirmação para cada quadro enviado. Isto pode ser feito, pois o Servidor sabe a quantidade de quadros que vai receber a partir do número total de medidas recebido inicialmente e do tamanho do quadro previamente armazenado. Este mecanismo evita que dados adicionais de confirmação (*overhead*) que não constituem informação útil (*payload*) trafeguem durante a comunicação, tornando o protocolo mais eficiente.

No **item 7**, o campo referente às medidas pode ter tamanho variável dependendo do tamanho estabelecido para o quadro. O cálculo do CRC considera todos os valores dos campos do quadro exceto os delimitadores. O campo “versão” se refere à versão do protocolo utilizada nesta conexão.

É possível verificar que o quadro do **item 9.a** não possui CRC. Além dos dois *bytes* adicionais necessários para incluir esse campo no quadro a ser transmitido, seria necessário incluir o código para verificação do mesmo no computador embutido. A justificativa para a não inclusão do CRC parte do pressuposto de que a probabilidade de haver erros nesse quadro é bem pequena devido ao seu tamanho. Dessa forma, se houver erro no quadro de pedido de retransmissão do Servidor, o embutido envia um quadro com as medidas e com o contador errados. O quadro será retransmitido de qualquer forma, pois o Servidor detecta que recebeu novamente um quadro errado e faz uma nova requisição.

No início das funções do protocolo do computador embutido, verifica-se a condição do sinal de portadora no Modem. Caso a conexão esteja desfeita, as funções retornam valores que indicam que a tarefa não foi completada. As duas tarefas a serem executadas durante a comunicação são as seguintes: transmissão das medidas no sentido embutido-Servidor e transmissão da tabela de codificação no sentido contrário.

## 4.6.2 Tratamento de falhas na comunicação

Na seção anterior foram mostradas as ações realizadas em casos de problemas nos dados que refletem no CRC. Nesta seção serão consideradas falhas decorrentes de problemas de *time-out* ou queda de linha durante a transmissão das medidas e durante o envio da tabela de codificação. Em qualquer situação, se o Servidor não receber os delimitadores corretamente, desconecta.

A detecção do *time-out* é feita pelo Servidor e é sempre ativada quando é esperado algum dado do computador embutido. Se for detectado *time-out*, o Servidor finaliza a conexão.

Devido às limitações impostas pela pouca disponibilidade de recursos de memória para armazenamento dos dados não temporários, a tabela de codificação é armazenada na medida em que é recebida. Dessa forma, as medidas anteriores ou tabela anterior são sobrepostas pelos novos dados recebidos. Diante disso, é possível verificar duas situações:

1. Problema de *time-out* ou desconexão no momento da transmissão das medidas no sentido embutido-Servidor.
2. Problema de *time-out* ou desconexão no momento do envio da tabela de codificação Servidor-embutido.

Para o primeiro caso, o armazenamento das medidas pode continuar normalmente, pois as medidas recebidas são desconsideradas. Na próxima conexão, Servidor já sabe que o problema ocorreu na primeira tarefa e aguarda o re-envio das medidas.

Para o segundo caso, o armazenamento das medidas é finalizado temporariamente até o recebimento completo da tabela de codificação. Uma parte da tabela já pode ter sido armazenada antes da desconexão e as próximas medidas são baseadas nessa nova tabela. Na próxima conexão o computador embutido e o Servidor já “sabem” que o problema ocorreu no momento da transmissão da tabela e passam para a execução dessa tarefa.

Os casos mais críticos são referentes às confirmações no final de cada tarefa indicadas nos itens 10 e 17. Podem ocorrer as duas situações seguintes:

- **Item 10** – SERVIDOR recebe as medidas corretamente, armazena as mesmas e envia quadro de confirmação para o EMBUTIDO.

- Esse quadro de confirmação se perde por causa de uma desconexão imprevista;
- SERVIDOR conecta novamente para enviar a tabela de codificação, porém o EMBUTIDO quer enviar novamente as medidas.
- **Item 17** – EMBUTIDO recebe a tabela de codificação corretamente, armazena a mesma e envia quadro de confirmação para o SERVIDOR.
  - Esse quadro de confirmação se perde pelo mesmo motivo anterior;
  - SERVIDOR conecta novamente para enviar a tabela de codificação, porém o EMBUTIDO já recebeu a tabela e está armazenando novas medidas.

A solução implementada para os casos anteriores é a seguinte: quando o SERVIDOR conecta, a primeira tarefa é enviar flag indicando o que deseja executar. Nos dois casos anteriores, o EMBUTIDO detecta essa situação e armazena a tabela novamente. Como a nova conexão é realizada em seguida, não existe problema com perda dos dados armazenados.

## 4.7 Testes finais

Esta seção apresenta os testes realizados no sistema (Modelo 1 apresentado neste capítulo na seção 4.4.1) com a inclusão das telas que foram geradas no servidor durante a transmissão dos dados. Falhas são simuladas para verificar a robustez do sistema.

### **Tela 1: Conexão iniciada pelo computador embutido indicando estado de alarme**

```

Porta encontrada: /dev/ttyS3
ATS0=1
OK
---- AGUARDANDO ALARME ----
RING

CONNECT 2400

Identificador Recebido: 0
RECEBEU ALARME!

Desconectado !!

```

**Figura 4.18: Simulação de alarme**

No início desta tela (Figura 4.18) é mostrado o comando enviado ao Modem do Servidor para que o mesmo seja configurado para atender o primeiro sinal de RING (sinal enviado pela central da linha telefônica). Em seguida apresenta a conexão estabelecida na velocidade máxima de 2400, limitada pelo Modem do computador embutido. Por último, o servidor recebe um identificador para saber de qual equipamento procedeu a chamada.

### **Tela 2: Transmissão das medidas pela primeira vez (não codificadas)**

```
BYTES RECEBIDOS: 48, QUANTIDADE DE MEDIDAS: 48
crc1: 49152, crc2: 48

contador: 16
Dados: 87, 232, 109
crc1: 36096, crc2: 160

contador: 15
144, 36, 174
crc1: 62720, crc2: 114

...

contador: 2
Dados: 15, 165, 86
crc1: 688, crc2: 190

contador: 1
Dados: 174, 69, 41
crc1: 15616, crc2: 10

VALORES CRIPTOGRAFADOS RECEBIDOS (DECIMAL): 87 232 109 144 36 174 175
36 49 66 163 223 43 210 29 222 78 27 195 4 120 164 14 13 237 12 203 13 138 90 213 231
63 69 92 109 82 226 69 149 230 247 15 165 86 174 69 41

VALORES RECEBIDOS: 44 44 44 44 44 44 44 44 44 44 45 45 44 45 45 45 44 44 45 44 44
45 45 45 46 45 45 46 45 45 46 44 44 44 44 44 45 44 44 45 44 44 44 44 44 44 44 44 44

TABELA DE CODIFICACAO: 131 1 159 57 226 80 255

Requisitando novamente: 17

Enviando Tabela.....
Dados: 131, 1, 159
crc1: 255, crc2: 65416

Dados: 57, 226, 80
crc1: 70, crc2: 17702

Dados: 254
crc1: 116, crc2: 29841

ERRO NO ENVIO DA TABELA !!!

Desconectado !!
```

**Figura 4.19: Primeira conexão entre embutido e servidor**



A partir da figura anterior pode-se verificar que, inicialmente, o computador embutido transmite a quantidade de medidas coletadas a serem enviadas e a quantidade de bytes utilizados para a transmissão das mesmas. Se as medidas estivessem comprimidas seriam necessários menos bytes para transmiti-las. Como são necessários 48 *bytes* para transmitir as medidas e o sistema está sendo testado com um tamanho da carga útil igual a 3 medidas por *frame*, são necessários 16 *frames* para transmitir todas as medidas. Para cada *frame* está indicado o número de seu contador, os valores das medidas (criptografadas) e o CRC dividido em dois *bytes*.

Durante a recepção, o servidor calcula o CRC dos valores recebidos e compara com o CRC enviado pelo computador embutido para a checagem de erros. Em seguida, como não ocorreu nenhum erro de CRC nessa transmissão, o servidor realiza o processo de descryptografia, armazena as medidas, calcula a tabela de codificação e envia uma confirmação de recebimento das medidas através do envio de um quadro contendo o número do total de frames recebidos (16 *frames*) acrescido de uma unidade (como foi apresentado neste capítulo na seção referente ao protocolo de comunicação). Durante o envio da tabela pode-se verificar que houve um erro. Não está indicado qual o tipo de erro: *time-out*, desconexão, etc. Nesse caso, o servidor deve iniciar uma nova conexão para enviar a tabela.

### Tela 3: Conexão iniciada para a transmissão da tabela

```
Enviando Tabela.....
Dados: 131, 1, 159
crc1: 255, crc2: 65416

Dados: 57, 226, 80
crc1: 70, crc2: 17702

Dados: 254
crc1: 116, crc2: 29841

frame requisitado: 2
Dados: 57, 226, 80
crc1: 69, crc2: 17702

frame requisitado: 4
Desconectado !!
```

Figura 4.20: Envio da tabela de codificação

Pose-se observar que, nessa transmissão (Figura 4.20), houve um erro de CRC indicado pela requisição de retransmissão do *frame 2* (que contém os dados: 57, 226 e 80). Por último, o computador embutido envia a indicação de recebimento correto dos quadros (número de quadros recebidos acrescido de uma unidade).

#### **Tela 4: Conexão para coleta de amostra**

```
BYTES RECEBIDOS: 3
QUANTIDADE DE MEDIDAS: 3
crc1: 3072, crc2: 3

AMOSTRA RECEBIDA: 44, 44, 45
crc1: 50176, crc2: 151

Desconectado !!
```

**Figura 4.21: Coleta de amostra**

Esta tela apresenta uma conexão originada do servidor para a requisição de uma amostra. No momento da requisição, o microcontrolador ainda não havia finalizado a requisição de todas as amostras possíveis para a ocupação da memória disponível.

## Tela 5: Transmissão das medidas codificadas e envio do flag da tabela

```
BYTES RECEBIDOS: 9, QUANTIDADE DE MEDIDAS: 45
crc1: 60928, crc2: 43

contador: 6
Dados: 68, 26, 15
crc1: 26368, crc2: 11

...

contador: 1
Dados: 88
crc1: 57088, crc2: 252

Contador de autenticação recebido: 1

VALORES CRIPTOGRAFADOS RECEBIDOS (DECIMAL): 68 26 15 105 40 145 212 86
10 109 211 254 62 51 131 88

VALORES RECEBIDOS: 2 133 176 84 6 220 64 192 0

VARIAVEIS RECEBIDAS: 0 0 0 0 0 0 1 1 0 0 0 1 -1 -1 0 0 0 0 1 1 1 0 0 0 0 0 0 -1 -1 -2 0
0 1 0 0 0 0 0 -1 0 0 0 0 0 0

TABELA DE CODIFICACAO: 131 1 159 57 226 80 255
TAXA DE COMPRESSAO RECEBIDA: 0.2

Requisitando novamente: 7

Enviando Tabela.....
Enviando flag

Desconectado !!
```

Figura 4.22: Transmissão de medidas codificadas

Nesta conexão (Figura 4.22), é possível verificar a diferença entre a quantidade de medidas a serem enviadas e a quantidade de bytes utilizados para transmiti-las devido ao processo de compressão. Ou seja, foram necessários 9 *bytes* para armazenar 45 medidas. Como está sendo utilizado uma cifra de bloco de 16 *bytes*, foram acrescentados mais 7 *bytes* para completar esse valor. Dessa forma, são necessários 6 *frames* para transmitir as medidas. A taxa de compressão calculada no servidor para essa nova remessa de medidas é a mesma já que foram utilizadas as mesmas medidas. Portanto, um *flag* é transmitido indicando que não há alteração na tabela de codificação.

## 4.8 Orçamento para instalação de um sistema de monitoração remota

Os custos apresentados na Tabela 4.11 a seguir são referentes ao orçamento realizado no Laboratório de Referência em Dessalinização (LABDES) para a instalação de um sistema de monitoração remota, através de um Controlador Lógico Programável (CLP), em um dessalinizador a ser instalado na região amazônica.

**Tabela 4.11: Três opções de preço do CLP completo: CPU, cartões de entrada e saída, fonte, rack**

Quant.	Descrição	Código	Fabricante	Preço Unit.	Preço Total
1	VersaMax CPU 64 kB Ethernet	IC200CPUE05	GE Fanuc	3.339,47	3.339,47
2	24Vdc Power Supply	IC200PWR002	GE Fanuc	153,07	306,14
1	Power Supply Booster Carrier	IC200PWB001	GE Fanuc	153,07	153,07
1	Cartão 16 ED	IC200MDL640	GE Fanuc	347,88	347,88
1	Cartão 32 ED	IC200MDL650	GE Fanuc	717,98	717,98
1	Cartão 32 SD	IC200MDL750	GE Fanuc	773,55	773,55
1	Cartão 4 SA (4-20mA)	IC200ALG320	GE Fanuc	1.057,49	1.057,49
4	Cartão 8 EA (4-20mA)	IC200ALG260	GE Fanuc	1.614,05	6.456,20
8	Versamax I/O Carrier, Box Style, Field Interface Wiring	IC200CHS022	GE Fanuc	153,07	1.224,56

Quant.	Descrição	Código	Fabricante	Preço Unit.	Preço Total
2	Cartão 16 Entradas Digitais	1769-IQ16	Allen-Bradley	876,19	1.752,39
1	Cartão 16 Saídas Digitais	1769-OB16	Allen-Bradley	1.155,15	1.155,15
1	Cartão 2 Saídas Analógicas	1769-OF2	Allen-Bradley	1.792,94	1.792,94
4	Cartão 8 EA	1769-IF8	Allen-Bradley	2.651,00	10.604,00
1	Unidade Base (16 ED, 12 SD - 6 Relé, 6 FET transistor)	1764-28BxB	Allen-Bradley	1.987,10	1.987,10
1	Módulo Processador 7,65 KB	1764-LSP	Allen-Bradley	903,23	903,23
1	Tampa Final (End Cap)	1769-ECR	Allen-Bradley	119,20	119,20
1	Interface de Comunicação NET-ENI	1761-NET-ENI	Allen-Bradley	2.608,91	2.608,91
1	Cabo: Microcomputador (9 pinos DIN) 1761-NETAIC 45cm	1761-CBL-AC00	Allen-Bradley	140,10	140,10

Quant.	Descrição	Código	Fabricante	Preço Unit.	Preço Total
1	CPU 48 kB Premium	TSX P57 203M	Modicom	2.460,81	2.460,81
1	Fonte de Alimentação 24Vcc	TSX PSY 1610M	Modicom	559,69	559,69
1	Rack Não Expansível 12 slots	TSX RKY 12	Modicom	750,03	750,03
2	Cartão 16 ED 24Vcc Lógica Positiva	TSX DEY 16D2	Modicom	484,91	969,82
1	Cartão 16 SD 24Vcc Estado Sólido	TSX DSY 16T2	Modicom	587,84	587,84
2	Bloco de Terminais para Cartão de I/O	TSX BLY 01	Modicom	58,79	117,58
2	Cartão 16 EA 4-20mA/0-10Vcc	TSX AEY 1600	Modicom	2.021,21	4.042,42
2	Base de Terminais Simples para Entradas Analógicas	ABE 7CPA02	Modicom	175,87	351,74
2	Cabo de Conexão para Base de Terminais EA	TSX CAP 030	Modicom	172,73	345,46
1	Cartão 4 SA 4-20mA/0-10Vcc	TSX ASY 410	Modicom	1.727,34	1.727,34

A estimativa de preço para a UFCG, calculada a partir dos valores informados anteriormente, por fabricante, é:

GE Fanuc → R\$ 23.002,14

Allen-Bradley → R\$ 21.063,01

Modicom → R\$ 19.060,37

Para realizar uma comparação do custo necessário para implementar o sistema de monitoração utilizando a solução anterior (CLP) com a solução adotada neste trabalho, será apresentada, a seguir, uma estimativa de custos do sistema embutido utilizado.

#### **Alimentação:**

- 1 regulador de tensão LM7805 – R\$ 1,50
- 1 diodo retificador IN4148 / (1/2)W – R\$ 0,50
- 2 capacitores cerâmicos 100nF e 2 capacitores eletrolíticos 100µF/68V – R\$ 2,00
- 1 chave liga-desliga (tipo H ou gangorra) – R\$ 2,00

#### **Conectores:**

- 23 conectores KRE3 de 2 pinos – R\$ 23,00
- 5 conectores KRE3 de 3 pinos – R\$ 7,50

- 1 conector KRE3 de 8 pinos – R\$ 4,00
- 1 conector de alimentação – R\$ 2,00

**Placa de aplicação:**

- 16 resistores de 10 K $\Omega$  / (1/8)W – R\$ 2,00
- 1 microcontrolador PIC16F877 – R\$ 40,00
- 1 cristal 4000 KHz – R\$ 2,00
- 2 capacitores de 33 pF cerâmicos – R\$ 0,50
- Soquete de 40 pinos torneado – R\$ 4,00
- Conector do tipo barra de pinos (300 unidades) – R\$ 20,00
- Caixa + blindagem eletromagnética – R\$ 50,00
- Filtro de linha (com supressor de transiente) para o Modem – R\$ 30,00
- Fonte de alimentação externa de 24 V (2 ou 3A com proteção) contra curto-circuito – R\$ 100 a 250,00
- Placa de circuito impresso (dupla face) – R\$ 50,00

De acordo com as informações anteriores, o custo aproximado do computador embutido é de R\$ 491,00.

O local de operação do equipamento a ser instalado na região amazônica apresenta condições adversas como, por exemplo, o difícil acesso para a manutenção em caso de problemas. Dessa forma, o equipamento de monitoração a ser instalado precisa ser robusto, justificando o custo mais alto que o descrito neste trabalho.

## **4.9 Considerações finais**

Inicialmente, nas seções 4.1, foi apresentado o ambiente de desenvolvimento do sistema com os principais componentes referentes à estação de gerência e ao computador embutido. Em seguida, na seção 4.2, foram apresentados os módulos implementados no computador embutido para atender aos requisitos do sistema. Foi mostrado o diagrama que contém os componentes do microcontrolador bem como as conexões com os dispositivos que fazem interface com o mesmo, que são: sensores e Modem. Esses dispositivos tornam possível a aquisição e transmissão das medidas dos parâmetros do equipamento monitorado, respectivamente. Na seção 4.3 final foram descritos alguns

aspectos relevantes sobre a ferramenta de construção do programa armazenado no microcontrolador.

Na seção 4.4 foram apresentados os dois modelos propostos e implementados para a compressão dos dados. No primeiro modelo, o algoritmo de Huffman foi implementado no Servidor, sendo sua execução iniciada sempre que uma nova remessa de medidas chega na estação gerente. Após a geração do código resultante do processo de compressão dessas medidas, uma tabela de codificação é gerada e enviada para o microcontrolador. Em seguida, cada medida coletada será substituída pelo valor correspondente do código contido na tabela de codificação. Dessa forma, não é necessária a implementação e a execução do algoritmo de compressão no computador embutido. O segundo modelo apresenta uma técnica híbrida Huffman/RLE de Zeros com resultados de eficiência próximos aos do primeiro modelo. No final desta seção foi realizada uma análise comparativa entre os modelos mostrando vantagens e desvantagens entre os mesmos bem como as vantagens destes em relação à implementação do algoritmo no microcontrolador.

Na seção 4.5 deste capítulo foram apresentados os resultados da implementação do algoritmo de criptografia escolhido (Rijndael). Alguns aspectos foram considerados na implementação para que a viabilidade do mesmo fosse garantida.

Na seção 4.6 foram apresentadas as etapas do protocolo de comunicação implementado, com a descrição dos campos dos quadros trafegados durante a comunicação. Em seguida, foram relacionadas as questões referentes ao tratamento de falhas na comunicação dos dados.

Finalmente, na seção 4.7, foram apresentados, através das telas capturadas na máquina servidora, os resultados do processamento dos itens anteriores. Esses testes finais mostraram a seqüência de passos realizados pelo protocolo de comunicação com a inclusão dos resultados da aplicação dos algoritmos de compressão e criptografia dos dados durante todo o processo.

Por último, na seção 4.8, foi apresentada uma descrição dos custos da implementação de um sistema de monitoração remota em um ambiente que apresenta condições adversas utilizando CLP e da implementação do sistema mostrado neste trabalho utilizando um microcontrolador. A partir das descrições apresentadas, foi

possível verificar uma grande diferença nos custos devido aos vários recursos adicionais encontrados em um CLP, principalmente aos que são referentes ao tratamento de falhas.



# Capítulo 5

## 5. Conclusões e Sugestões

---

Este trabalho abordou o processo de especificação e implementação de um mecanismo de comunicação de dados para um sistema de telemetria de baixo custo a ser implantado no Programa Água Doce para a monitoração remota de dessalinizadores. Para a concretização do processo, utilizou-se um computador embutido (microcontrolador), instalado junto a um equipamento monitorado (protótipo), com disponibilidade de recursos limitada para a coleta, armazenamento e transmissão das medidas provenientes de sensores conectados ao equipamento.

Devido às restrições detectadas, foi proposto e implementado um modelo de compressão de dados para aumentar a eficiência das tarefas atribuídas ao computador embutido, viabilizando a utilização do mesmo em um sistema de telemetria. Ou seja, foi reforçada a aplicabilidade do microcontrolador em projetos de automação apesar de suas limitações no que diz respeito aos recursos de memória e capacidade de processamento disponível.

O modelo de compressão implementado se mostrou eficiente em seu processo de aplicação e nos resultados sobre os dados a serem armazenados, produzindo taxas significativas compressão e, conseqüentemente, aumentando a quantidade de medidas possíveis de serem armazenadas na memória disponível. A estratégia adotada para a implementação do algoritmo de compressão na máquina gerente do sistema pode ser uma alternativa mais eficiente para a utilização do mesmo reduzindo, portanto, a quantidade de utilização de memória de programa e a memória volátil do dispositivo embutido. Finalmente, com a compressão, foi possível verificar um incremento na quantidade média de medidas por *byte* transmitidas, diminuindo o tempo médio de transmissão e, em decorrência disso, reduzindo os custos de utilização do canal de transmissão (linha telefônica).

A utilização de um mecanismo de segurança de dados com criptografia proporciona a confiabilidade almejada sobre as informações trafegadas no sistema desenvolvido. Com a estratégia de implementação adotada no modelo de criptografia, foi possível implementá-lo juntamente com os outros módulos do sistema. Além disso, foi implementado o algoritmo de criptografia de dados padrão mais recente (*Rijndael*), reafirmando o bom desempenho do mesmo em sistemas com recursos limitados.

Diante da diversidade de situações que podem ser atribuídas ao computador embutido instalado no campo, torna-se necessário considerar estados de operação adversos do mesmo. Para isso, foram realizados testes simulando erros ocorridos durante a comunicação com a estação gerente bem como na realização da coleta de medidas dos sensores. Os testes realizados com o protocolo de comunicação mostraram a robustez do sistema em suportar variadas cargas de erros possíveis (*CRC*, *time-out*, queda de linha, etc) em uma comunicação via linha telefônica. Com relação à coleta de medidas, foram realizados testes simulando o desligamento parcial do sistema, coleta de medidas com erros espúrios, medidas que indicam alarme do sistema, etc. Os resultados destes testes mostraram a estabilidade do sistema em suportar essas situações.

Por fim, vale ressaltar os benefícios dos resultados práticos deste trabalho para as comunidades usuárias dos equipamentos monitorados (dessalinizadores). Devido ao baixo custo do sistema implementado, torna-se viável sua implantação no campo em grande escala já que o tempo de vida útil dos componentes dos equipamentos é incrementado e, conseqüentemente, há uma diminuição dos custos relativos à manutenção dos mesmos. Além disso, reforça a garantia da qualidade do serviço oferecido (neste caso, a água) estar dentro dos padrões mínimos requeridos para seu consumo, na medida em que a qualidade da água está sendo monitorada a todo instante.

Resumidamente, as principais contribuições deste trabalho foram:

1. Implementação e resultados das análises de dois modelos de compressão de dados para utilização em sistemas com pouca disponibilidade de recursos.
2. Implementação e resultados das análises de um modelo de criptografia que utiliza o algoritmo padrão de criptografia de dados mais recente (*Rijndael*) em um microcontrolador (*PIC16F877*), reforçando a aplicabilidade do mesmo em sistemas com recursos de memória e poder de processamento limitados.

3. Implementação e resultados das análises de um protocolo de comunicação de dados eficiente e robusto para um sistema de telemetria de baixo custo que utiliza linha telefônica como canal de comunicação.

## 5.1 Sugestões para trabalhos futuros

Após a finalização desse trabalho, são apresentadas algumas sugestões para a continuidade do trabalho ora apresentado, conforme descrição a seguir.

### Compressão de Dados

- Os modelos de compressão implementados foram:
  1. Huffman (uma tabela para os valores de medidas coletadas de todos os sensores);
  2. Huffman + RLEZ (uma tabela para os valores de medidas coletadas de todos os sensores e uma tabela para todos os valores de quantidades de diferenças de zeros de todos os sensores).
- Sugestões para futuras implementações:
  3. Huffman (uma tabela para cada conjunto de valores de medidas coletadas de cada sensor);
  4. Huffman + RLEZ (uma tabela para cada conjunto de valores de medidas coletadas de cada sensor e uma tabela para todos os valores de quantidades de diferenças de zeros de todos os sensores).
  5. Huffman + RLEZ (uma tabela para cada conjunto de valores de medidas coletadas de cada sensor e uma tabela para cada conjunto de valores de quantidade de zeros coletadas de cada sensor).

Ainda com relação à compressão, tem-se as seguintes sugestões para trabalhos futuros: a implementação no microcontrolador de outros algoritmos de compressão de dados para análise comparativa da eficiência dos mesmos com relação ao processo de compressão e aos resultados obtidos após a aplicação do algoritmo.

Podem ocorrer casos em que a faixa de valores para as medidas dos sensores seja grande por motivos de heterogeneidade entre os mesmos, por exemplo. Dessa forma, aplicar a solução do modelo de compressão de dados adotada neste trabalho acarretaria um aumento considerável na tabela de codificação a ser armazenada no computador embutido. Como sugestão, a tabela de codificação poderia ser implementada armazenando a codificação apenas dos valores coletados do último conjunto de medidas coletado. Ou seja, não seria armazenada a codificação para todos os valores da faixa. Se for detectado na leitura de uma medida que esta não está codificada, o armazenamento segue como no modelo implementado com a gravação do código de escape seguido do valor literal da medida.

## Referências Bibliográficas

- [Abel , 03] Abel, Jurjen; Teahan, Willian. *Text Preprocessing for Data Compression*, 2003. Disponível em: <http://www.data-compression.info/Algorithms/Preprocessing/>
- [Abramsom, 73] Abramsom, Norman. *Information Theory and Coding*. McGraw-Hill Eletronic Science Series, 1973.
- [Agostini, 02] Agostini, L.; Silva, I.; Bampi, S. Projeto de Arquitetura de Codificador de Entropia para Compressão JPEG de Imagens em Tons de Cinza. In: VIII WORKSHOP IBERCHIP, 2002, Guadalajara, México.
- [Alencar, 98] Alencar, Marcelo Sampaio de. *Telefonia Digital*, São Paulo: Érica, 1998.
- [Amjad, 92] Amjad, Z., *Reverse Osmosis, Membrane Technology, Water Chemistry, and Industrial Applications*, Ed. Van Nostrand Reinhold, 1992.
- [Analog, 02] Analog Devices, Inc. (NYSE: ADI). *World-leading semiconductor company specializing in high-performance analog, mixed-signal and digital signal processing (DSP) integrated circuits (ICs)*. Disponível em <http://www.analog.com/>.
- [ARM, 04] ARM, *The Architecture for the Digital World*, 2004. Disponível em <http://www.arm.com/>
- [Assis, 03] Assis, Francisco Marcos de. *Aplicações da Teoria da Informação: Códigos para Transmissão, Compressão e Criptografia*, UFCG/DEE, 2003.
- [Bloom, 96] Bloom, Charles. "LZP: A new Data Compression Algorithm" submitted to DCC96, IEEE, 1996.
- [Bonde, 00] Bonde, Ian. *O Mercado Brasileiro de Telemetria*. Itelogy Partners, 2000. Disponível em: <http://www.teleco.com.br>
- [Burnett, 02] Burnett, S. e Paine, S.. *Criptografia e Segurança - O guia oficial RSA*. Editora Campus, 2002.
- [Campos, 00] Campos, Arturo San Emeterio. *The introduction to Data Compression*, 2000. Disponível em: <http://www.arturocampos.com>
- [Cover, 91] Cover, Thomas M.; Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New York, USA, 1991.

- [Daemem, 99] Daemem, J.; Rijmem, V.. *AES Proposal: Rijndael*, 1999.
- [Daemem, 02] Daemem, J.; Rijmem, V.. *A Specification for Rijndael, the AES Algorithm*, 2002.
- [EMJ, 02] EMJ Embedded Systems. *Embedded products and services*. Site: <http://www.emjembedded.com/products/products.html>.
- [Ernst, 98] Ernst, R. *Codesign of Embedded Systems: Status e Trends*, IEEE Design and Test of Computers, 1998.
- [Feijão, 02] Feijão, Pedro Manuel. *Projeto de Sistemas Digitais*. Departamento de Engenharia Eletrotécnica da Universidade de Coimbra, 2002. Disponível em <http://alumni.deec.uc.pt/~pfeijao/pagina4.html>.
- [Flynn, 93] Flynn, Anita M. *Mobile robots: inspiration to implementation*. Wellesley: A K Peters, 1993.
- [Flynn, 97] Flynn, M. *What's Ahead in Computer Design?*, Euromicro 97 Proceedings, September 1997.
- [Formiga, 03] Formiga, M. M.; Franca, K. B.; Ribeiro, D.; Brasileiro, F.V.; Melcher, E.U.K.; Cirne, W.C. *Desenvolvimento de um Sistema de Monitoração Remota de Dessalinizadores*. IV CITEM, Florianópolis-SC, 2003.
- [Franca, 01] França, K. B., *Aumento da Vida Útil de Sistemas de Dessalinização no Campo: Análise e Manutenção Remota*. Projeto do CT-HIDRO 01/2001 – CNPq.
- [Franca, 03] França, K. B.. Programa Água Doce. Ministério do Meio Ambiente/Secretaria de Recursos Hídricos/Agência Nacional de Águas. Laboratório de Referência em Dessalinização, 2003.
- [Heath, 98] Heath, S. *Embedded Systems Design*. Reading: Butterworth-Heinemann, 1998. 350p.
- [Hinz, 00] Hinz, Marco Antônio Mielke. *Um estudo descritivo de novos algoritmos de criptografia*, Monografia apresentada ao Curso de Bacharelado em Informática do Instituto de Física e Matemática da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Informática, 2000.
- [HTH, 03] High Tech Horizon's On-Line Catalog. *Development of products that can be used in the control systems*. Disponível no endereço <http://www.hth.com/snap/>.
- [ISO, 04] ISO – International Organization for Standardization, 2004. Disponível em: <http://www.iso.org/iso/en/ISOOnline.frontpage>

- [Júnior, 90] Júnior, Vidal Pereira da Silva. *Microcontroladores*. São Paulo: Editora Érica, 1990.
- [JPEG, 04] *Home site of the JPEG and JBIG committees*, 2004. Disponível em <http://www.jpeg.org>
- [Joyce, 01] Joyce, A.; Loureiro, D.; Rodrigues, C.; Castro, S. *Small reverse osmosis units Pv systems for water purification in rural places*. *Desalination*, vol. 137, pp. 39-44, 2001.
- [Kerr, 01] Kerr, T. J.; McHale. B. B. *Applications in general microbiology: A laboratory manual*. 6th ed., Hunter Textbooks Inc., Winston-Salem, 2001
- [Kleber, 01] Kleber, Sidnei. *Montagem e avaliação de um dessalinizador via osmose inversa para águas salobras de poços tubulares*, UFPB/PIBIC-CNPq, 2001.
- [Kohayakawa, 01] Kohayakawa, Yoshiharu; Eduard, Carlos; Hayashida, Ulisses Kendi. MAC 499 – trabalho de formatura supervisionado, monografia, Instituto de Matemática e estatística – IME – Universidade de São Paulo – USP, 2001.
- [Lavagno, 03] Lavagno, Luciano; Martin, Grant; Selic, Bran. *UML for REAL. Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, Boston, 2003.
- [Lempel, 78] Lempel, A. & ZIV, J. *A Compression of Individual Sequences via Variable-Rate Coding*. *IEEE Transactions on Information Theory* IT-24(5), Setembro 1978.
- [Lima, 02] Lima , Cristiano Cachapuz e. *Protocolos*. URCAMP - CCEI - Curso de Informática, 2002. Site: <http://www.urcamp.tche.br/~ccl/redes1/>
- [Melo, 04] Melo, Eloi Duarte de. *Normalização de um sistema de dessalinização piloto utilizando um módulo de gerência distribuída com capacidade de monitoração remota*. Dissertação (Mestrado em Engenharia Química), Centro de Ciências e Tecnologia, Universidade Federal de Campina Grande, 2004.
- [Microchip, 01] *PIC16F87X Data Sheet, Microcontrollers*, Microchip Technology Inc, 2001. Disponível em <http://www.microchip.com>
- [Microsys, 02] *Micro/Sys. Single board computers for industrial and OEM applications*, 2002. Site: <http://www.embeddedsys.com/subpages/>
- [Moore, 65] Moore, G.E. "*Cramming More Components Onto Integrated Circuits*". *Electronics Magazine*, Vol. 38, 1965.
- [M2M, 03] *M2M Telemetria: Rede de Serviços Integrados. O que é Telemetria*, 2003. <http://www.m2mtelemetria.com.br/telemetria.htm>.

- [Nakanishi, 04] Nakanishi, Seido. Cercado de Linux. Revista do Linux, 2004. Site: <http://www.revistadolinux.com.br/ed/026/assinantes/>
- [Nelson, 91] Nelson, Mark. *Arithmetic Coding + Statistical Modeling = Data Compression*, Dr. Dobb's Journal, 1991.
- [NIST, 00] NIST, National Institute of Standards and Technology. *Advanced Encryption Standard*. Gaithersburg, 2000. Disponível em: <http://www.nist.gov/aes.htm>.
- [Oppenheimer, 99] Oppenheimer, P., Projeto de Redes *Top-Down*, Campus, 1999.
- [Pasco, 76] Pasco, R. *Source Coding Algorithms for Fast Data Compression*. Ph. D. Dissertation, Dept. of Electrical Engineering, Stanford Univ., Stanford, Calif, 1976.
- [Pereira, 03] Pereira, Fábio. *Microcontroladores PIC: Programação em C*. São Paulo, Érica, 2003.
- [Rissanen, 76] Rissanen, J. J. *Generalized Kraft Inequality and Arithmetic Coding*. *IBM J. Res. Dev.* 20 (1976), 198-203.
- [Rodrigues, 04] Rodrigues, Teles; Ferreira, Manuel. Bit stuffing, CRC e IP. Curso de Engenharia da Informática, Redes de Computadores, 2004. Disponível em <http://ltodi.est.ips.pt/redescomp/>
- [Schneier, 94] Schneier, B.. *Applied Cryptography – Protocols, Algorithms and Source code in C*. 2nd ed. New York: John Wiley, 1994. 624 p. il.
- [Schneier, 00] Schneier, B.; Whiting, D.; *A Performance Comparison of the Five AES Finalists*, 2000. Disponível em <http://www.nist.gov/aes>
- [Schneider, 01] Schneider, R. P.; Tsutiya, M. T. Membranas filtrantes para o tratamento de água, esgoto e água de reuso. ABES, 1ª ed., São Paulo, 2001.
- [Schultz, 99] Schultz, T.W. C and the 8051-*Building efficient applications*. Vol. 2. Prentice Hall PTR, 1999. 458p. p22-25.
- [Shannon, 49] Shannon, Claude E., Weaver, Warren. *The Mathematical Theory of Communication*. Univ. of Illinois Pres, 1949.
- [Silva, 01] Silva, Wellington João da. Tecnologia Java para Sistemas Embarcados. Trabalho de Graduação em Ciência da Computação, UFPE, Centro de Informática, Recife, 2001.
- [Souza, 00] Souza, David José de. *Desbravando o PIC*. São Paulo, Editora Érica, 2000.
- [Stallings, 03] Stallings, William. *Cryptography and Network Security, Principle and Practice*. Prentice Hall, New Jersey, 1999.



- [Stival, 03] Stival, Marcelo Loyola. Linux Embarcado. Trabalho de Graduação, apresentado ao Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2003.
- [Tanenbaum, 03] Tanenbaum, Andrew S. Redes de Computadores, Quarta edição, Editora Campus, 2003.
- [Texas, 04] Texas Instruments. *DSP Overview*, 2004. Disponível em <http://www.ti.com/sc/brasil/mundo/dsp.htm>
- [TWT104, 03] TWT104: Computador PC/104 com ZF86, TransWorld Technology, 2003. Disponível em <http://www.twtembedded.com.br>
- [User, 02] *User Manual for a "socket modem" V22bis DE8661*. CML Microcircuits, Communication Semiconductors; Agosto, 2002.
- [Vieira, 99] Vieira, Rodrigo de Souza. *Protótipo de Um Sistema de Monitoramento Remoto Inteligente*. Dissertação apresentada ao Curso de Pós-graduação em Engenharia de Produção da Universidade Federal de Santa Catarina, para obtenção do título de Mestre em Engenharia, 1999.
- [Weber, 95] Weber, R. F.. *Criptografia Contemporânea*. Congresso Brasileiro de Computação. Canela, 1995.
- [Zelenovsky, 99] Zelenovsky, Ricardo; Mendonça, Alexandre. *Introdução aos sistemas embutidos*, 1999. Disponível em: <http://www.mzeditora.com.br/artigos/embut.htm>

# Apêndice A

## Plataformas para sistemas de telemetria

A base utilizada para a pesquisa foi a capacidade de atendimento aos seguintes requisitos:

- Realizar medição dos sensores do equipamento monitorado (pressão, vazão, temperatura, etc);
- Atuar sobre o equipamento monitorado;
- Estabelecer conexão com o administrador remoto.

As plataformas distinguem-se basicamente pelo conceito. A lista e a descrição das mesmas (em seus modelos de menor custo) são apresentadas a seguir. Os conceitos mais utilizados, comumente encontrados, foram [EMJ, 2002; Microsys, 2002; Analog, 2002]:

- PC (Computador embutido/embarcado),
- $\mu$ PC (Computador dedicado),
- $\mu$ WEB SERVER (Microcontrolador + TCP/IP),
- $\mu$ C (Microcontrolador + SNAP).

SNAP (*Scaleable Node Address Protocol*) é um protocolo de rede aberto baseado em controle de sistemas [HTH, 2003].

Breve descrição de cada um dos modelos apresentados:

**PC** – São computadores (386, 486 ou 586) com memória suficiente para executar DOS ou LINUX (modo texto). Possuem todas as portas que um PC possui (serial, paralela, vídeo) numa placa de dimensões reduzidas.

**$\mu$ PC** – São computadores que visam uma solução específica, normalmente só possuem comunicação serial e/ou ethernet. Possuem sistema operacional próprio e suporte à JAVA.

**$\mu$ WEB SERVER** – São microcontroladores (programação em C ou *Assembly*) com capacidade de emulação TCP/IP.

$\mu$ C – São microcontroladores (programação em C ou *Assembly*) nos quais é implementado um protocolo qualquer de comunicação.

Na Tabela A.1 são apresentados os custos de cada uma dessas soluções. Nesta tabela não foram contabilizados os preços do encapsulamento, fios e conectores que podem compor o sistema [EMJ, 2002; Microsys, 2002; Analog, 2002].

**Tabela A.1: Tabela comparativa das possíveis soluções para o sistema de telemetria**

Solução	Suporte	Memória		Periféricos		Custo Unitário (US\$)
		Interna	Externa	Internos	Externos	
PC	Linux Dos Java C	2M/RAM 2M/FLASH	-	Serial LPT Ethernet Video	A/D Modem	110
$\mu$ PC	Java	2M/RAM 2M/FLASH	-	Serial Ethernet	A/D Modem	70
$\mu$ WEB SERVER	C Assembly	356/RAM 8K/FLASH 254/EEPROM	1M/ EEPROM	Serial LPT I2C A/D	Modem TCP/IPchip	21
$\mu$ C	C Assembly	356/RAM 8K/FLASH 254/EEPROM	-	Serial LPT I2C A/D	Modem	13

# Apêndice B

## Algoritmo padrão de criptografia (Rijndael)

O Rijndael é um algoritmo de blocos iterativos com tamanho de bloco e de chave variáveis, que podem ser especificados independentemente com os tamanhos 128, 192 ou 256 bits.

Este algoritmo diferencia-se da maioria dos outros que são usados atualmente, pois não utiliza uma estrutura do tipo *Feistel* na sua fase de rotação. Numa estrutura *Feistel*, os bits de um estado intermediário são transpostos em uma outra posição sem serem alterados; no Rijndael, a fase de rotação é composta de transformações uniformes inversíveis chamadas de *layers* [Hinz, 00].

O resultado das diferentes operações realizadas no estágio intermediário é chamado de Estado. Um Estado pode ser denotado por uma matriz retangular de bytes; esta matriz possui 4 linhas e o número de colunas (denotado por  $Nb$ ) é igual ao tamanho do bloco dividido por 32, como apresentado na Figura B.1.

A cifra da chave também pode ser denotada por uma matriz retangular de bytes, também com 4 linhas e o número de colunas (denotado por  $Nk$ ) sendo igual ao tamanho da chave dividido por 32.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$k_{0,0}$	$K_{0,1}$	$k_{0,2}$	$k_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$k_{1,0}$	$K_{1,1}$	$k_{1,2}$	$k_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$k_{2,0}$	$K_{2,1}$	$k_{2,2}$	$k_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$k_{3,0}$	$K_{3,1}$	$k_{3,2}$	$k_{3,3}$

**Figura B.1: Matriz de Estado e matriz de chave para  $Nb = 6$  e  $Nk = 4$**

Em alguns casos, é necessário demonstrar estas matrizes como vetores de 4 bytes, cujo comprimento se refere ao tamanho da linha da matriz retangular. Como a quantidade de colunas da matriz é obtida dividindo-se o tamanho do bloco (128, 192 ou 256) por 32, as colunas da matriz podem assumir o tamanho de 4, 6 ou 8.

Por convenção, quando for necessário especificar cada byte dentro do vetor, serão usadas as letras  $a, b, c$  e  $d$  como índice. Tanto a entrada como a saída do Rijndael é tratada como um vetor de bytes numerados até  $4 * Nb - 1$  (estes blocos podem ser de 16, 24 ou 32 bytes). A cifra da chave também é tratada da mesma forma com a diferença de que o vetor é numerado até  $4 * Nk - 1$ .

O número de voltas é denotado por  $Nr$  e depende tanto de  $Nb$  como de  $Nk$ . Na Tabela B.1 [Daemem, 99] é apresentado o número de voltas para  $Nb = Nk = 4, 6, 8$ .

**Tabela B.1: Número de voltas em função do tamanho do bloco e da chave**

<b>Nr</b>	<b>Nb = 4</b>	<b>Nb = 6</b>	<b>Nb = 8</b>
<b>Nk = 4</b>	10	12	14
<b>Nk = 6</b>	12	12	14
<b>Nk = 8</b>	14	14	14

O algoritmo transforma os dados através do número de voltas usando quatro “funções” diferentes, a saber: *ByteSub*, *ShiftRow*, *MixColumn* e *AddRoundKey*. A última volta é um pouco diferente, e apresenta apenas as funções: *ByteSub*, *ShiftRow* e *AddRoundKey* [Daemem, 02] . A seguir, estão descritas as quatro transformações que ocorrem a cada volta do algoritmo.

### ***ByteSub***

A transformação *ByteSub* é uma substituição não linear que opera em cada Estado independentemente, a *S-Box* usada nesta transformação é inversível e é construída pela composição de duas transformações:

- Executando uma multiplicação inversa em  $GF(2^8)$ ;
- Aplicando uma transformação definida por:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

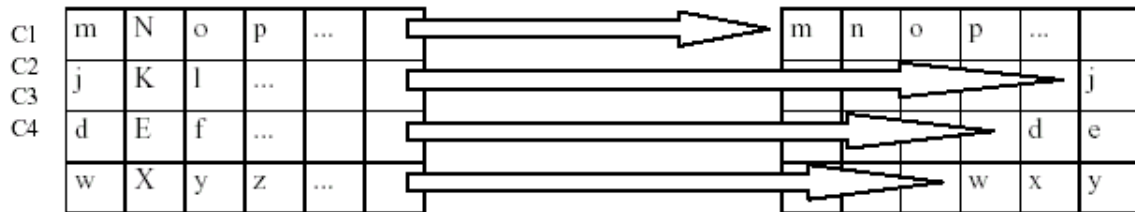
### *ShiftRow*

Nesta transformação, as linhas dos Estados são alteradas ciclicamente, a linha 1 não é alterada, a linha 2 é alterada em  $C1$  bytes, a linha 3 é alterada em  $C2$  bytes e a linha 4 é alterada em  $C3$  bytes. Estes parâmetros  $C1$ ,  $C2$  e  $C3$  dependem de  $Nb$ . Estes valores estão especificados na Tabela B.2 [Daemem, 99].

**Tabela B.2: Parâmetros para a quantidade de bytes deslocados no *ShiftRow***

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

A Figura B.2 mostra o efeito da transformação *ShiftRow* em um Estado.



**Figura B.2: Transformação *ShiftRow* em um Estado**

### *MixColumn*

As colunas de um Estado são consideradas polinômios do tipo  $GF(2^8)$  denotadas por  $a(x)$  e são multiplicadas em módulo  $x^4 + 1$  por um polinômio fixo

$$c(x) = '03'x^3 + '01'x^2 + '01'x + '02'$$

O resultado da multiplicação pode ser denotado por  $b(x)$  e a multiplicação por

$$b(x) = c(x) \text{ XOR } a(x)$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 01 & 02 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

### AddRoundKey

Esta transformação consiste somente em aplicar uma operação XOR num Estado usando uma matriz proveniente do algoritmo da chave. O tamanho da matriz da chave é igual ao tamanho da matriz de Estado. A Figura B.3 [Daemem, 99] mostra a transformação *AddRoundKey*.

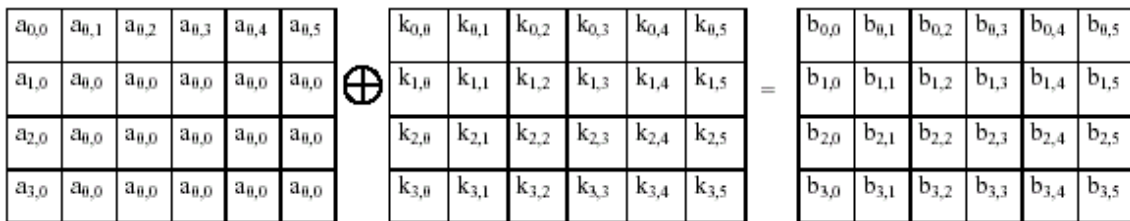


Figura B.3: Transformação *AddRoundKey*

### Tratamento da Chave

As inserções da chave no meio do algoritmo são feitas através da *RoundKeys* que são provenientes da cifra da chave. Esta cifra da chave é tratada por um escalonamento que possui dois componentes: Expansão da chave e Seleção da *RoundKey*. O parágrafo seguinte descreve os princípios seguidos por este processo.

O número total de *RoundKeys* necessários é igual ao tamanho do bloco multiplicado pelo número de voltas mais um. Por exemplo: um bloco de 128 bits e 10 voltas necessita de 1048 *RoundKeys*.

A cifra da chave é expandida dentro de uma *ExpandedKey*. As *RoundKeys* são selecionadas da *ExpandedKey* para serem usadas no algoritmo da seguinte maneira: a primeira *RoundKey* consiste das primeiras *Nb* palavras da *ExpandedKey*, a segunda *RoundKey* consiste das seguintes *Nb* palavras da *ExpandedKey* e assim sucessivamente [Hinz, 00].

## Expansão da chave

A *ExpandedKey* é um vetor linear de palavras de 4 *bytes* e é denotado por  $W[Nb*(Nr+1)]$ . As primeiras  $Nk$  palavras possuem a cifra da chave enquanto as outras palavras são definidas recursivamente em palavras com índices menores. A função que calcula a *ExpandedKey* depende de  $Nk$ , existindo uma versão da função para  $Nk < 6$  e outras para  $Nk > 6$  [Hinz, 00].

## O Algoritmo Rijndael

O algoritmo de criptografia Rijndael é uma associação das funções descritas acima. Na Figura B.4 são mostradas as etapas do algoritmo, que se baseia em três passos: uma adição inicial de uma *RoundKey*,  $Nr - 1$  voltas (*rounds*) e uma volta final.

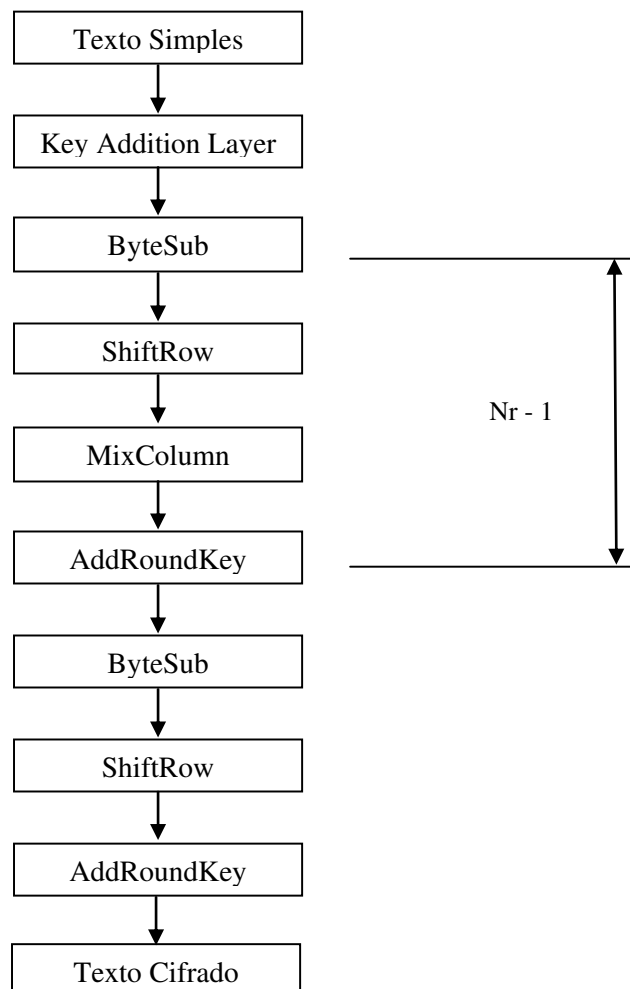


Figura B.4: As etapas do Rijndael



# Apêndice C

## Eficiência do protocolo de comunicação

A definição de um protocolo de comunicação eficiente pode minimizar o tempo de transmissão dos dados reduzindo os custos de utilização do canal. Além disso, a escalabilidade do sistema pode ser melhorada já que o acréscimo de novos pontos de monitoração acarretará uma demanda de tráfego mais alta com a estação de controle.

Se o canal de comunicação fosse livre de erros, seria possível, em uma conexão, obter a eficiência máxima transmitindo todos os dados sem qualquer informação adicional (*overhead*). Diante dessa situação improvável, é necessário enviar os dados em estruturas menores chamadas de quadros ou *frames*. Estruturas grandes utilizam largura de banda de modo mais eficiente do que as estruturas pequenas visto que, para enviar todos os dados, são necessárias menos informações de controle. Além disso, utilizam menos intervalos entre quadros (*gap*). No entanto, quanto maior o tamanho do quadro maior a probabilidade deste ser atingido por um erro e maior o *overhead* na retransmissão. Portanto, a definição da quantidade de informação útil (*payload*) a ser enviada em cada quadro é um compromisso com a taxa de erros do canal (*Bit Error Rate* - BER) [Oppenheimer, 99].

Um tipo comum de modem utiliza a técnica de modulação QAM (*Quadrature Amplitude Modulation*) para a transmissão dos bits. Para esse tipo de modulação, a probabilidade de erro de bit (PE) pode ser calculada a partir das equações [Assis, 03]:

$$PE = (1 / \log_2(N)) \operatorname{erf}(\alpha) , \quad (1)$$

$$\alpha = \operatorname{seno}(\pi / N) (\log_2 N)^{1/2} (E_b / N_0) , \quad (2)$$

$$\operatorname{erf}(\alpha) = 1 - 2 / \sqrt{\pi} \int_0^\alpha e^{-t^2} dt , \quad (3)$$

em que  $N$  é o número de fases,  $\operatorname{erf}$  é a função erro,  $E_b$  é a potência do sinal e  $N_0$  é a potência do ruído.

A eficiência do canal é dada por:

$$E = X / ((X + L_c)(1 + P_q) + (P_q \cdot L_{Nack})), \quad (4)$$

em que:

$X$  - tamanho da carga útil do quadro em bytes,

$L_c$  - quantidade de bytes de controle do quadro,

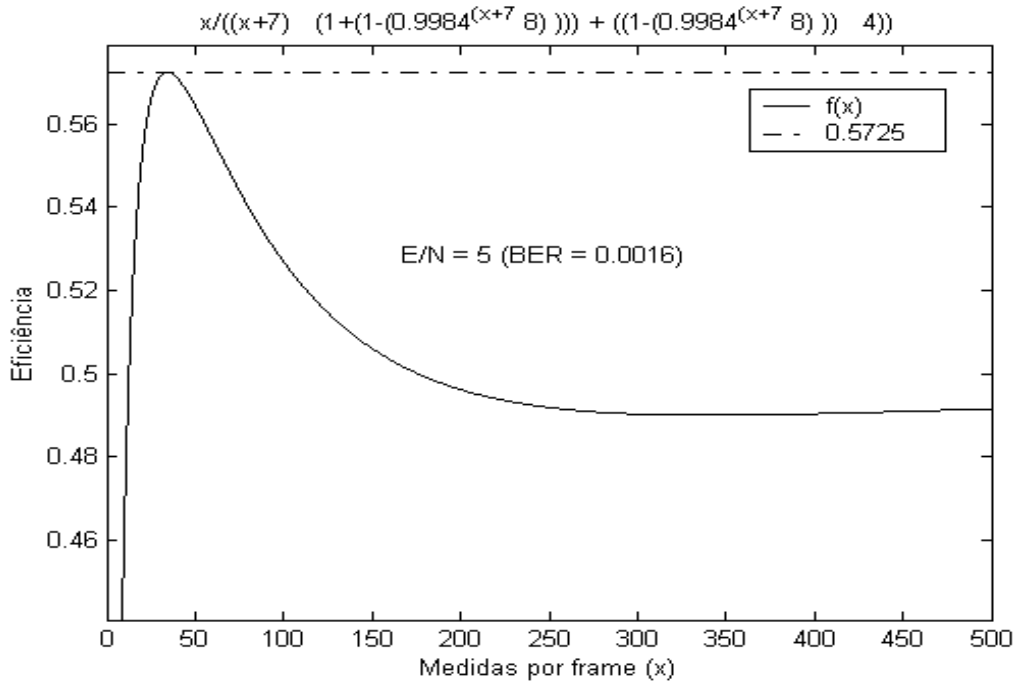
$P_q$  - probabilidade de erro do quadro,

$L_{Nack}$  - quantidade de bytes do quadro de pedido de retransmissão.

A probabilidade de erro do quadro é dada por

$$P_q = 1 - (1 - PE)^{(X + L_c) \cdot 8} \quad (5)$$

A taxa de erros (BER) é dependente de fatores que podem mudar a cada estabelecimento de conexão por linha telefônica. No caso do ruído, pode haver alterações no tipo e na intensidade [Oppenheimer, 99]. Não é possível estabelecer uma taxa de erros fixa devido a esse comportamento do canal físico. Para os testes iniciais do sistema, foi definida uma taxa de erros resultante de uma conexão com relação sinal-ruído de 5 dB. Esse valor foi utilizado para simular uma situação crítica (ruído elevado). Para este valor, a taxa de erro pode ser calculada pelas Equações 1, 2 e 3 mostradas anteriormente. Dado que na modulação QAM são utilizadas 4 fases ( $N$  é igual a 4), obtém-se uma taxa de erros de 0.0016. Aplicando esse resultado na Equação 4 e, considerando vários tamanhos de quadro, é possível encontrar a curva mostrada no gráfico da Figura C.1.



**Figura C.1: Eficiência do canal**

O tamanho da carga útil do quadro que resulta na maior eficiência é o valor da abscissa, que representa o ponto máximo da curva. A partir desse valor, pode-se encontrar que o tamanho que representa essa eficiência é aproximadamente igual a 33 medidas por *frame*.

A partir de um determinado número de transmissões e de possíveis retransmissões, será possível calcular a probabilidade de erro de *frame*. A partir das Equações 4 e 5, é possível calcular o tamanho do *frame* que permitirá a comunicação mais eficiente. Dessa forma, o tamanho do *frame* poderá ser atualizado de acordo com as condições do meio de transmissão em um determinado período.

# Apêndice D

Serão mostradas a seguir as interfaces das funções com os comentários sobre seu funcionamento para cada módulo utilizado no Modelo 1 (Huffman) e no Modelo 2 (Huffman com RLE de Zeros).

## 1. Computador embutido

```
/*
 * @(#) dessalinizador.c
 *
 * Title:    Projeto Água Doce - Monitoração Remota de Dessalinizadores
 * Copyright: Copyright (c) 2004
 * LABDES/DEQ/CCT/UFPB
 *
 * Este é o módulo central do projeto. É responsável por ativar as funções
 * referentes à
 * coleta, armazenamento com ou sem compressão, criptografia e transmissão dos
 * dados. Também é implementado o código referente à compressão de dados.
 *
 * @version    1.0
 * @author    Maurício Marinho
 *
void main ();

/*
Função de tratamento de interrupção a partir da recepção de dados da USART. Recebe a
requisição do Servidor após o estabelecimento da conexão. Em seguida, chama função
para enviar quadro contendo o total de medidas a serem transmitidas. São ativadas as
funções responsáveis por enviar as medidas e receber a tabela de codificação. Aqui,
também é implementada a lógica responsável por tratar as falhas referentes à perda de
confirmação no final dessas tarefas.
*/
#int_rda
void trata_interrupcao_serial();

/*
Esta função é responsável por atualizar a posição de gravação das medidas após receber a
tabela de codificação. Esta posição é imediatamente após o último valor da tabela.
*/
int getPosicaoInicio();

/*
Esta função é chamada após a coleta de uma medida para o seu armazenamento
comprimido. É realizada uma busca para verificar se o valor da medida está codificado na
tabela. Aqui é calculado o valor do tamanho dos códigos armazenados a partir da faixa de
valores possíveis. Finalmente, é ativada a função responsável por buscar o código.
*/
```

**void armazenaLeitura();**

/\*

A partir da posição de início da busca, da ordem do código e do tamanho, essa função varre, sequencialmente, os valores codificados até encontrar a medida codificada. Aqui é ativada a função responsável por acrescentar a medida codificada.

\*/

**void buscaCodigo(int posicao, int ordem\_codigo);**

/\*

Esta função ativa a função de gravação para a medida coletada e atualiza a posição de gravação. Além disso, ativa a função que grava o valor literal após o armazenamento do código de escape caso não haja codificação para a medida coletada.

\*/

**void acrescentaMedida(int posicao, int n, int tamanho\_codigo);**

/\*

Esta função copia bit a bit o valor de codificação para a medida coletada.

\*/

**void gravar(int flag);**

/\*

Esta função armazena o valor literal da diferença entre medidas que não está codificada.

\*/

**void gravaDiferencaNaoCodificada();**

/\*

Esta função atualiza as variáveis do sistema após a recepção correta da tabela de codificação pelo Servidor.

\*/

**void inicializaVariaveis();**

/\*

Esta função remove os dados anteriormente armazenados, habilitando a memória para a gravação de novos dados.

\*/

**void zeraMedidas();**

/\*

\* **@(#) enlace.c**

\* **Este módulo é responsável pelo estabelecimento de conexão com o servidor e**

\* **transmitir medidas ou alarmes.**

\*

\* **@version           1.0**

\* **@author     Maurício Marinho**

\*

\*/

```
/*
Esta função envia os valores armazenados no intervalo indicado. Ativa as funções
responsáveis por calcular o CRC e transmissão do quadro. Além disso, aguarda
requisição do Servidor de quadros errados. Finaliza quando recebe quadro de
confirmação de recebimento correto.
```

```
*/
```

```
int1 envia_medidas(int posicaoInicial, int posicaoFinal);
```

```
/*
```

```
Esta função transmite os campos do quadro.
```

```
*/
```

```
void transmitePacote (int i, int1 inicioDeFrame, int1 fimDeFrame);
```

```
/*
```

```
Esta função é responsável por receber os quadros referentes à tabela de codificação.
Verifica se recebeu um flag indicando que a tabela não deve ser alterada. Calcula CRC
dos quadros para verificação de erros.
```

```
*/
```

```
void recebe_quadro(int inicio_da_gravacao);
```

```
/*
```

```
Esta função é chamada pela aplicação principal para ativar a função de recebimento de
tabela e requisição dos quadros errados. Aqui é enviada a confirmação para o servidor
caso a recepção da tabela foi correta.
```

```
*/
```

```
int1 recebeu_tabela(int inicio_da_gravacao);
```

```
/*
```

```
Esta função envia quadros para o Servidor requisitando os quadros errados da tabela.
```

```
*/
```

```
int1 requisita_quadros_errados(int inicio, int fim);
```

```
/*
```

```
Transmite número da amostra requisitada.
```

```
*/
```

```
void pedir_amostra (int amostra);
```

```
/*
```

```
Envia os campos referentes ao total de valores a serem enviados e o número total de
medidas embutidas nestes valores.
```

```
*/
```

```
void enviaTotalDeMedidas(int total, int quantidadeDeMedidas);
```

```
/*
```

```
Enviar delimitador.
```

```
*/
```

```
void envia_delimitador();
```

```
/*
```

Verifica se um delimitador foi recebido. Também verifica se o flag de continuação da tabela foi recebido.

\*/

**int1 delimitador\_recebido();**

/\* Esta função é executada após a desconexão com o servidor que faz com que o buffer de recepção seja preenchido com caracteres (sujeira) aleatórios gerando interrupções indesejáveis.

\*/

**void limpaBufferRecepcao();**

/\*\*\*\*\*\*

\* **@(#) cifrador.c**

\* **Este programa corresponde ao módulo de criptografia**

\*

\* **@version 1.0**

\* **@author Maurício Marinho**

\*

\*\*\*\*\*/

// Esta função é responsável por ativar as outras funções do algoritmo.

**void cifra ();**

/\* Esta função é responsável por adicionar o número de bytes necessários para completar um bloco de 16 bytes requerido pelo algoritmo de criptografia.

\*/

**int cifraMensagem(int posicaoInicial, int posicaoFinal);**

/\*

As funções seguintes estão explicadas no Apêndice B (Funcionamento do Rjindael)

\*/

**void MixColumns();**

**void XorRoundKey(int chaveTemp);**

**void ShiftRows();**

**void SubBytes();**

/\*\*\*\*\*\*

\* **@(#) calculo\_crc.c**

\* **Este programa é responsável por aplicar o algoritmo do cálculo do CRC.**

\* **@version 1.0**

\* **@author Maurício Marinho**

\*

\*\*\*\*\*/

```

/*
A partir dos dois primeiros valores e em seguida dos valores compreendidos pela faixa,
calcula o valor do CRC.
*/
int gera_16bit_crc( int posicaoLimite, int posicaoFinal, int primeiro, int segundo );

```

```

/*****
* @(#) pre-processamento.c
* Este programa é responsável por aplicar o algoritmo da mediana
* ao receber as medidas temporárias antes do armazenamento da
* medida.
* @version 1.0
* @author Maurício Marinho
*
*****/

```

```

/*
Varre a memória buscando as medidas gravadas temporariamente e calcula a mediana
*/
int processa_medidas_temporarias( int numero_medidas_temp );

```

O módulo referente ao Modelo 2 (Huffman com RLE de Zeros) apresenta diferença apenas no código relacionado à compressão de dados. Neste modelo, além da função responsável por adicionar o código da quantidade de zeros, é acrescentada uma função responsável por atribuir o valor do código para a diferença encontrada que seja diferente de zero na seqüência de leitura das medidas.

## 2. Servidor

```

/**
* Class Discador.java
*
* Classe responsável por implementar o protocolo de comunicação do lado servidor.
*
* @author
* @version 1.10, 08/04/00
*/

```

```

/*
Método responsável por receber as variáveis de medida do computador embutido.
Detecta quais medidas foram recebidas com erro. Chama a função que requisita ao
microcontrolador essas medidas. Chama a função que decifra os valores recebidos,

```



remove os elementos acrescentados para completar o tamanho determinado da cifra de bloco e retorna as variáveis recebidas em sua forma original (comprimidas ou não).

\*/

**public Vector getVariaveis( );**

/\*

Método responsável por chamar a função por calcular os valores das chaves intermediárias e por ativar a função de descritografia.

\*/

**public void decifragem( );**

/\*

Este método é responsável por receber uma amostra das últimas medidas coletadas do microcontrolador.

\*/

**public void recebeAmostra( );**

/\*

Este é o primeiro método ativado após o estabelecimento da conexão. Aqui são recebidos a quantidade de bytes recebidos e a quantidade de medidas embutidas nos bytes recebidos. Se os dados estiverem sendo recebidos pela primeira vez, os dois valores são iguais.

\*/

**public int recebeQuantidadeDeMedidas( );**

/\*

Método responsável por enviar a tabela de codificação que foi calculada a partir dos valores recebidos. No final, ativa o método que aguarda a requisição dos quadros errados recebidos pelo computador embutido. A função retorna a condição "verdade" se foi recebido o quadro de confirmação de recebimento de todas as medidas.

\*/

**public boolean enviaTabela(Vector tabela, boolean tabela\_enviada\_com\_sucesso);**

/\*

Método responsável por aguardar requisições do microcontrolador dos quadros errados da tabela de codificação e enviá-los novamente.

\*/

**public boolean aguardaRequisicoes( Vector tabela );**

/\*

Método responsável por verificar os quadros das medidas com erro e requisitá-los novamente ao computador embutido. Chama a função que requisita o quadro e também a função que recebe o quadro correspondente.

\*/

**public void requisitaMedidasComErro( );**

/\*

Método responsável por receber o quadro enviado novamente pelo microcontrolador após detecção de erro no mesmo e pedido de envio novamente por parte do Servidor.

\*/

**public void pegaVariavel(int amostra);**

/\*

Método responsável por requisitar quadro recebido com erros.

\*/

**public void pedirAmostra(int numeroDaAmostra);**

/\*

Método responsável comparar o CRC do quadro recebido com o que foi recebido como um dos campos do quadro e retornar "verdade" de forem iguais, indicando que o quadro foi recebido sem erros.

\*/

**private boolean checaCRC(char[] medidas, int CRCDados);**

/\*

Método responsável por enviar a informação para o microcontrolador indicando que o mesmo pode manter a mesma tabela enviada pela última vez.

\*/

**public boolean enviaFlag(boolean tabela\_enviada\_com\_sucesso);**

/\*

Método responsável por verificar se o delimitador foi recebido corretamente.

\*/

**public boolean delimitadorRecebido();**

/\*

Método responsável receber os dados iniciais enviados como lixo durante o estabelecimento da conexão (handshake)

\*/

**public void recebeStream();**

/\*

Método responsável por enviar um quadro para o microcontrolador indicando que recebeu as medidas corretamente.

\*/

**public void desconecta();**

/\*

Método responsável por descartar medidas acrescentadas com o objetivo de completar o tamanho do bloco da criptografia.

\*/

**public void removeElementosAdicionais();**

/\*

Método responsável por enviar o delimitador para o microcontrolador.

\*/

**public void enviaDelimitador();**

```

/**
 * Class TesteHuffman.java
 *
 * Classe responsável por implementar as funções referentes à compressão e
 * descompressão dos dados recebidos.
 * @version 1.10, 08/04/00
 */

/*
Método principal desse módulo. Recebe as medidas, aciona métodos responsáveis por:
adicionar valores não codificados, comprime medidas, gerar código, gerar tabela de
codificação, calcular de compressão, verificar se a condição de transmissão da tabela
referente à diferença entre as taxas de compressão estabelecida foi satisfeita.
*/
public static void inicia( Vector variaveisDeMedida );

/*
Converter valores negativos recebidos (129, 130, etc) para valores negativos reais (-1,-2,
etc)
*/
public static void converteParaNegativo( );

/*
Acrescenta às variáveis recebidas os buracos dentro da faixa de medidas.
*/
public static void adicionaValorNaoCodificado( );

/*
Método responsável por descomprimir as medidas codificadas recebidas
*/
public static void descomprime( Vector tabela );

/*
Método responsável por identificar, a partir do código, a medida recebida.
*/
public static void procuraCodigo( int bitTabela, int posicao_bit );

/*
Função responsável por comprimir as variáveis recebidas. Aciona as funções de aplicação
do algoritmo de Huffman e da montagem da tabela de codificação.
*/
public static void comprime( Vector variaveis );

/*
Monta tabela de codificação a partir do código calculado e da faixa de valores
encontrada.
*/
public static void armazenaTabela( );

```

```

/**
 * Class Huffman.java
 *
Esta classe é responsável por implementar o algoritmo de Huffman e algumas funções extras relacionadas à montagem da tabela de codificação como, por exemplo, incluir o tamanho do código encontrado.
 * @version 1.10
 */

/**
 * Class Alarmes.java
 *
Esta classe é responsável por iniciar um processo (Thread) que aguarda a chegada de um alarme.
 * @version 1.10
 */

/**
 * Class CRC16CCITT.java
 *
Esta classe é responsável por implementar um algoritmo de CRC de 16 bits (CCITT) para uma quantidade variável de dados recebidos.
 * @version 1.10
 */

/**
 * Class StreamDeEntrada.java
 *
Esta classe é responsável por implementar as funções relacionadas à chegada e tratamento de dados recebidos pela interface serial (Modem).
 * @author
 * @version 1.10
 */

/**
 * Class StreamDeSaida.java
 *
Esta classe é responsável por implementar as funções relacionadas à chegada e tratamento de dados recebidos pela interface serial (Modem).
 * @version 1.10
 */

/**
 * Class Rijndael.java
 *
Esta classe é responsável por implementar o algoritmo de criptografia utilizado no sistema.
 * @version 1.10

```