



Universidade do Porto
Faculdade de Engenharia

FEUP



Mário Rui Tavares Borges Barreto

Bluetooth em Aplicações Industriais

MEEC
Porto, 2003

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

BLUETOOTH EM APLICAÇÕES INDUSTRIAIS

Mário Rui Tavares Borges Barreto

Licenciado em Engenharia Electrónica Industrial pela Universidade do
Minho

Dissertação submetida para satisfação parcial dos
requisitos do grau de mestre
em

Engenharia Electrotécnica e Computadores
(Área de especialização de Sistemas Digitais e Informática Industrial)

Dissertação realizada sob a supervisão de
Professor Doutor Augusto Oliveira da Silva Gomes,
do Departamento de Engenharia Electrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

Porto, Novembro de 2003

UNIVERSIDADE DO PORTO	
Faculdade de Engenharia	
BIBLIOTECA M	
N.º	<u>79718</u>
CDU	<u>621.3(043)</u>
Data	<u>17, 5, 2004</u>

Resumo

A presente tese aborda o projecto e desenvolvimento de um terminal industrial denominado *BlueTerm*. Este terminal de uso genérico dispõe, em particular, de uma interface de comunicação do tipo *Bluetooth* e foi projectado tendo em conta a sua possível integração numa infra-estrutura já existente chamada *Infracom*. Esta estrutura, composta por módulos com diversas funcionalidades (entradas digitais, saídas digitais, etc.), tem como base a tecnologia de comunicações CAN (*Controller Area Network*). A integração na *Infracom* de terminais com possibilidade de comunicação sem fios e elevada capacidade de processamento, permitirá alargar e melhorar as soluções anteriormente disponibilizadas e diminuir significativamente as barreiras na ligação entre as linhas de produção e o centro de operações em ambiente industrial.

Foi realizado um protótipo do terminal tendo por base um microprocessador de 16 bits recentemente disponibilizado pela *Infineon*, o *XC161*, que dispõe de diversos tipos de interfaces: série assíncrona/síncrona (*ASC*), série síncrona de alta velocidade (*SSC*), *IIC* e *CAN*.

Para permitir a um possível utilizador, desenvolver aplicações específicas para o *BlueTerm*, foi criado um módulo de funções que disponibiliza todas as funcionalidades do terminal de forma transparente relativamente ao *hardware*.

Abstract

The use of wireless communications experienced a great expansion in the last few years, partly due to the extended freedom it gives to its users in accessing information. However, its penetration in the industry sector has been restricted. In this work, it is described a new terminal, named *BlueTerm*, for industrial applications, with a Bluetooth-based wireless communication connection.

The *BlueTerm* provides several digital I/Os, analogue inputs and relay drive outputs, which, in conjunction with local processing capabilities, enable its autonomous operation, despite being usually linked through the wireless connection to a main computer server, usually responsible for its operation management.

Additionally, the *BlueTerm* presents a CAN (Controller Area Network) interface that enables its integration in pre-existent industrial networks. In this case, it operates as a gateway linking CAN and Bluetooth networks, as well as a manager of the communications between the computer server and the modules connected to the CAN network.

A user-friendly library has also been developed to facilitate the build on of user specific applications for the *BlueTerm*.

Prefácio

Este trabalho foi desenvolvido com o objectivo de proporcionar o elemento central para a infra-estrutura existente (*Infracom*) de modo a melhorar o desempenho e a incrementar as potencialidades existentes.

Através da interface de comunicação *Bluetooth* pretende-se facilitar o acesso do centro de controlo à infra-estrutura montada na linha de produção. Até aqui o acesso era feito através da interface *CAN*, sendo necessário colocar no computador do centro de operações uma placa própria para efectuar a comunicação. Esta solução implica um custo elevado, quer de material quer de montagem e manutenção. Com uma infra-estrutura sem fios, fiável e tornando-se um *standard* de facto consegue-se baixar o custo da solução assim como tornar a infra-estrutura muito mais aberta a possíveis evoluções.

Índice

Resumo	2
Abstract	3
Prefácio	4
Índice	5
Índice de figuras	8
Índice de tabelas	10
Abreviaturas	11
1 Introdução	12
2 Especificação do <i>BlueTerm</i>	14
2.1 Condicionais – a Infra-estrutura existente Infracom	14
2.1.1 Terminais de uso genérico <i>CANit</i> e <i>CANit-RF</i>	15
2.1.2 Módulos de entradas e saídas de uso genérico e <i>interfaces</i> de comunicação	16
2.2 Especificação do Hardware	18
2.2.1 Microprocessador de 16 bits (<i>CPU</i>)	19
2.2.2 Memória de programa e de dados	19
2.2.3 Memória de configuração	20
2.2.4 Relógio de Tempo Real	20
2.2.5 Visor e teclado	21
2.2.6 Interfaces de comunicação	22
2.2.7 Interface de comunicação <i>USB</i>	23
2.2.8 Interface de comunicação <i>Bluetooth</i>	24
2.2.9 Interface para leitor de <i>transponders</i>	24
2.2.10 Interface para cartão de memória do tipo <i>flash</i>	24
2.2.11 Entradas digitais/analógicas	25
2.2.12 Saídas digitais	26
2.2.13 Fonte de alimentação	26
2.3 Especificação do Software	26
2.3.1 Memória de configuração	27
2.3.2 Relógio de Tempo Real	27
2.3.3 Visor	27
2.3.4 Teclado	27
2.3.5 Interfaces de comunicação	27
2.3.6 Interface de comunicação <i>USB</i>	28
2.3.7 Interface de comunicação <i>Bluetooth</i>	28
2.3.8 Interface para leitor de <i>transponders</i>	28
2.3.9 Interface para cartão de memória do tipo <i>flash</i>	28
2.3.10 Entradas digitais/analógicas	28
2.3.11 Saídas digitais	28
2.3.12 Genéricos	29
3 Projecto de hardware do <i>BlueTerm</i>	29
3.1 Pesquisa de material	29
3.1.1 Microprocessadores	29
3.1.2 Memórias	33
3.1.3 Relógio de Tempo Real	33
3.1.4 Visor e teclado	34
3.1.5 Interfaces de comunicação	34

3.1.6	Interface de comunicação <i>USB</i>	34
3.1.7	Interface de comunicação <i>Bluetooth</i>	34
3.1.8	Interface para leitor de <i>transponders</i>	35
3.1.9	Interface para cartão de memória do tipo <i>flash</i>	35
3.1.10	Entradas digitais/analógicas e Saídas digitais	35
3.1.11	Fonte de alimentação	35
3.2	Reformulação e ajuste das especificações	35
3.3	Desenho do esquemático	36
3.4	Implementação da lista de material	36
3.5	Verificações do esquemático	38
3.6	Desenho do PCB	38
3.7	Verificações do PCB	38
3.8	Montagem do PCB	39
3.9	Testes eléctricos	39
4	Projecto de <i>software</i>	40
4.1	Módulo “can”	42
4.2	Módulo “coms”	42
4.3	Módulo “display”	44
4.4	Módulo “duart”	46
4.5	Módulo “e2prom”	47
4.6	Módulo “flash”	48
4.7	Módulo “i2c”	49
4.8	Módulo “io”	49
4.9	Módulo “kernel”	51
4.10	Módulo “keyboard”	52
4.11	Módulo “mmcard”	54
4.12	Módulo “mystring”	55
4.13	Módulo “rtc_ext”	57
4.14	Módulo “serial”	58
4.15	Módulo “spi”	59
5	Programa de teste e configuração	60
5.1	Especificação	60
5.2	Implementação	61
6	<i>Bluetooth</i> no <i>BlueTerm</i>	63
6.1	Resumo	63
6.2	História	63
6.3	O SIG	63
6.4	Interoperabilidade	64
6.5	Especificação	64
6.6	Topologia de rede	65
6.7	Segurança	66
6.8	Tecnologias complementares	67
6.9	Pesquisa de controladores de Bluetooth	67
6.10	Teste funcional do BlueWAVE	68
6.11	Valor acrescentado pelo Bluetooth à Infracom	70
7	Aplicação demonstrativa das potencialidades do <i>BlueTerm</i>	71
7.1	Especificação	71
7.2	Implementação	73
8	Conclusões e trabalho futuro	81
	Referências	83

Bibliografia	84
Anexo A – Protótipos das funções de baixo nível	85
A.1 Módulo “can”	86
A.2 Módulo “coms”	87
A.3 Módulo “display”	90
A.4 Módulo “duart”	95
A.5 Módulo “e2prom”	96
A.6 Módulo “flash”	97
A.7 Módulo “i2c”	98
A.8 Módulo “io”	99
A.9 Módulo “kernel”	100
A.10 Módulo “keyboard”	102
A.11 Módulo “mmcard”	105
A.12 Módulo “mystring”	108
A.13 Módulo “rtc_ext”	110
A.14 Módulo “serial”	111
A.15 Módulo “spi”	112
Anexo B – Desenho do esquemático	114

Índice de figuras

Figura 1- Infra-estrutura <i>Infracom</i>	14
Figura 2 - Diagrama de blocos do terminal <i>CANit</i>	15
Figura 3 - Diagrama de blocos do terminal <i>CANit-RF</i>	15
Figura 4 - Diagrama de blocos do terminal <i>CANio-MIX</i>	16
Figura 5 - Diagrama de blocos do terminal <i>CANio-16-IN</i>	16
Figura 6 - Diagrama de blocos do terminal <i>CANio-8-OUT</i>	17
Figura 7 - Diagrama de blocos do terminal <i>CANio-8-IN</i>	17
Figura 8 - Diagrama de blocos do terminal <i>CANio-4-OUT</i>	17
Figura 9 - Diagrama de blocos do terminal <i>CANio-Repetidor/Gateway</i>	18
Figura 10- Diagrama de blocos do <i>hardware</i>	18
Figura 11 – Ligação das memórias ao microprocessador	19
Figura 12 – Ligação dos dispositivos <i>iic</i> ao microprocessador	20
Figura 13 – Visor do terminal <i>CANit</i>	21
Figura 14 – Ligação do visor ao microprocessador	21
Figura 15 - Teclado do terminal <i>CANit</i>	21
Figura 16 – Ligação do teclado ao microprocessador	22
Figura 17 – Implementação das interfaces série RS232/RS422	22
Figura 18 – Implementação das interfaces <i>CAN</i>	23
Figura 19 – Implementação da interface <i>USB</i>	23
Figura 20 – Implementação das interfaces <i>Bluetooth</i>	24
Figura 21 – Implementação da interface para ligação de um <i>MMC</i> ou <i>SD</i> Card	25
Figura 22 – Implementação das entradas digitais/analógicas	25
Figura 23 – Implementação das saídas digitais	26
Figura 24 - Diagrama de blocos do microprocessador XC161 (figura retirada do software DAVe da <i>Infineon</i>)	33
Figura 25 - Vista geral do hardware desenvolvido	39
Figura 26 - Módulos de <i>software</i>	41
Figura 27 - Funções de recepção do módulo " <i>coms</i> "	42
Figura 28 - Funções de transmissão do módulo " <i>coms</i> "	43
Figura 29 - Funções genéricas do módulo " <i>coms</i> "	44
Figura 30 - Funções genéricas do módulo " <i>display</i> "	45
Figura 31 - Funções de escrita do módulo " <i>display</i> "	46
Figura 32 - Funções genéricas do módulo " <i>duart</i> "	47
Figura 33 - Função de leitura do módulo " <i>e2prom</i> "	48
Figura 34 - Função de escrita do módulo " <i>e2prom</i> "	48
Figura 35 - Funções genéricas do módulo " <i>flash</i> "	48
Figura 36 - Funções genéricas do módulo " <i>i2c</i> "	49
Figura 37 - Funções genéricas do módulo " <i>io</i> "	51
Figura 38 - Funções genéricas do módulo " <i>kernel</i> "	52
Figura 39 - Funções genéricas do módulo " <i>keyboard</i> "	53
Figura 40 - Funções de leitura do módulo " <i>keyboard</i> "	54
Figura 41 - Funções de leitura do módulo " <i>mmcard</i> "	54
Figura 42 - Funções de escrita do módulo " <i>mmcard</i> "	55
Figura 43 - Funções de conversão de tipos do módulo " <i>mystring</i> "	56
Figura 44 - Funções genéricas do módulo " <i>mystring</i> "	56
Figura 45 - Funções de leitura do módulo " <i>rtc_ext</i> "	57
Figura 46 - Funções de escrita do módulo " <i>rtc_ext</i> "	58
Figura 47 - Funções genéricas do módulo " <i>rtc_ext</i> "	58

Figura 48 - Funções genéricas do módulo "serial"	59
Figura 49 - Funções genéricas do módulo "spi"	60
Figura 50 - Menu inicial do programa de <i>setup</i>	61
Figura 51 - Menu inicial do programa de configuração	61
Figura 52 - Opções do menu de configuração	61
Figura 53 - Menu inicial do programa de teste	62
Figura 54 - Opções do menu de teste manual	62
Figura 55 - <i>Piconets</i> com uma única ligação <i>Master-Slave</i> (a), uma ligação <i>multi-slave</i> (b), e uma ligação <i>scatternet</i> (c) (figura retirada de www.ericsson.com).....	66
Figura 56 - Dongle USB-Bluetooth	69
Figura 57 - Módulo RS232-Bluetooth	69
Figura 58 - Teste de distância	70
Figura 59 - Infra-estrutura <i>Infracom</i> com o terminal <i>BlueTerm</i> (1)	70
Figura 60 - Infra-estrutura <i>Infracom</i> com o terminal <i>BlueTerm</i> (2)	71
Figura 61 - Aplicação demonstrativa	73
Figura 62 - Aplicação demonstrativa - menu <i>File</i>	74
Figura 63 - Aplicação demonstrativa - menu <i>Config - Com Port</i>	74
Figura 64 - Aplicação demonstrativa - menu <i>Config - Com BaudRate</i>	75
Figura 65 - Aplicação demonstrativa - menu <i>Config - Password</i>	75
Figura 66 - Aplicação demonstrativa - menu <i>Equipment</i>	76
Figura 67 - Aplicação demonstrativa - menu <i>Help</i>	76
Figura 68 - Aplicação demonstrativa - menu <i>Help - About</i>	77
Figura 69 - Aplicação demonstrativa - menu <i>Equipment - BlueTerm</i>	77
Figura 70 - Aplicação demonstrativa - menu <i>Equipment - Canio-MIX</i>	78
Figura 71 - Aplicação demonstrativa - menu <i>Equipment - CANio-16-IN</i>	79
Figura 72 - Aplicação demonstrativa - menu <i>Equipment - CANio-8-OUT</i>	79
Figura 73 - Aplicação demonstrativa - menu <i>Equipment - CANio-8-IN</i>	80
Figura 74 - Aplicação demonstrativa - menu <i>Equipment - CANio-4-OUT</i>	80

Índice de tabelas

Tabela 1 - Alguns microprocessadores de 16 bits existentes no mercado.....	29
Tabela 2 - Exemplo de lista de material.....	37
Tabela 3 - Valores devolvidos pelas teclas 0-9 em função do estado das teclas AZ e FN	52
Tabela 4 - Módulos de <i>bluetooth</i> integrados	68
Tabela 5 - Alguns comandos e respostas para escrita/leitura remota do terminal	72
Tabela 6-A - Definições para o módulo de comunicações	87
Tabela 7-A - Definições do módulo display.....	90
Tabela 8-A - Formato da escrita de um número	92
Tabela 9-A - Formato da escrita de um número em virgula flutuante.....	93
Tabela 10-A - Formatos de apresentação da hora.....	94
Tabela 11-A - Formatos de apresentação da data	94
Tabela 12-A - Definições do módulo Duart	95
Tabela 13-A - Códigos de erro do kernel.....	102
Tabela 14-A - Definições do módulo keyled.....	102
Tabela 15-A - Matriz do teclado.....	103
Tabela 16-A - Mapa de registos do relógio	111
Tabela 17-A - Velocidades disponíveis nas comunicações RS232	111

Abreviaturas

ADC	: <u>A</u> nalog-to- <u>D</u> igital <u>C</u> onverter
ASCII	: <u>A</u> merican <u>S</u> tandard <u>C</u> ode for <u>I</u> nformation <u>I</u> nterchange
ASC	: <u>A</u> synchronous <u>S</u> erial <u>C</u> ommunications
BlueTerm	: <u>B</u> luetooth <u>T</u> erminal
CAN	: <u>C</u> ontroller <u>A</u> rea <u>N</u> etwork
CANit	: <u>C</u> AN industrial terminal
CPU	: <u>C</u> entral <u>P</u> rocessing <u>U</u> nit
DUART	: <u>D</u> ual <u>A</u> synchronous <u>R</u> eceiver/ <u>T</u> ransmitter
EEPROM	: <u>E</u> lectrical <u>E</u> rasable <u>P</u> rogrammable <u>R</u> ead <u>O</u> nly <u>M</u> emory
EPROM	: <u>E</u> lectrical <u>P</u> rogrammable <u>R</u> ead <u>O</u> nly <u>M</u> emory
IIC	: <u>I</u> nter <u>I</u> ntegrated <u>C</u> ircuits
MMC	: <u>M</u> ultiMedia <u>C</u> ard
PCB	: <u>P</u> rinted <u>C</u> ircuit <u>B</u> oard
RAM	: <u>R</u> andom <u>A</u> ccess <u>M</u> emory
ROM	: <u>R</u> ead <u>O</u> nly <u>M</u> emory
RTC	: <u>R</u> ead <u>T</u> ime <u>C</u> lock
SDCard	: <u>S</u> ecure <u>D</u> igital memory <u>C</u> ard
SPI	: synchronous <u>S</u> erial <u>P</u> eripheral <u>I</u> nterface
SCI	: asynchronous <u>S</u> erial <u>C</u> ommunications <u>I</u> nterface
SSC	: High <u>S</u> peed <u>S</u> ynchronous serial interface
USB	: <u>U</u> niversal <u>S</u> erial <u>B</u> us

1 Introdução

A tecnologia sem fios *Bluetooth* é bastante recente e diversos fabricantes de *hardware* estão a desenvolver produtos para utilização mais facilitada desta interface, o que torna aliciente desenvolver um produto que use essa tecnologia promissora perspectivando uma melhoria e mais valia nas funcionalidades disponibilizadas.

Esta tecnologia começou a ser utilizada para comunicação entre telefones portáteis e computadores (para descarregar programas e actualizações de *software*) expandindo-se depois para a comunicação computador - computador, computador - impressora, etc.

Presentemente a tecnologia sem fios *Bluetooth* encontra-se principalmente em produtos de uso doméstico, como por exemplo, telefones portáteis, computadores, impressoras, sistemas de som, etc.. Alguns equipamentos na área da saúde começam a utilizar a tecnologia mas de uma forma muito cautelosa, talvez devido a ser muito recente e as interfaces existentes no mercado não permitirem a integração desta tecnologia em muitos equipamentos sem um grande esforço económico.

A ideia de utilização de tecnologia sem fios *Bluetooth* em aplicações de automação industrial começa a despontar, não tendo sido possível encontrar, apesar de extensa pesquisa bibliográfica realizada, referencias específicas a qualquer uma. Parece por isso indicado, começar agora a trabalhar com esta tecnologia nessa área, de modo a conseguir a necessária familiarização e a perspectivar as melhores aplicações.

O principal objectivo deste trabalho é a introdução de um terminal de uso genérico com tecnologia *Bluetooth* na industria, mais concretamente na área de automação industrial. O nosso país possui um tecido industrial caracterizado em muitos casos por carências na automação o que se traduz muitas vezes em baixa produção, custos elevados e menor qualidade. Sendo um desígnio nacional muitas vezes reiterado o aumento da qualidade e da produtividade industriais e tendo a automação o papel decisivo que se lhe reconhece, perspectiva-se um grande interesse deste equipamento no mercado. Para isso teria que conseguir satisfazer grande parte das necessidades dos industriais a um custo que possibilite a amortização da instalação de uma infra-estrutura deste tipo a curto e médio prazo com evidentes melhorias na produção.

De forma a alargar os horizontes de aplicabilidade do equipamento, pretendeu-se a criação de um terminal para uso diversificado (industria têxtil, metalúrgica, vidreira, etc.) mudando apenas a aplicação de *software* gravada no terminal. A flexibilidade é uma das características mais marcantes neste trabalho, desde a fase de especificação até à fase de execução.

Outra das características marcantes deste trabalho é a possibilidade de integração do *BlueTerm* numa infra-estrutura existente (*Infracom*). Esta infra-estrutura é composta principalmente por módulos de entradas e saídas e de comunicações.

Esta tese foi organizada em oito capítulos, sendo o capítulo 1 uma introdução ao trabalho realizado.

No capítulo 2 é feita a especificação do *BlueTerm*. Na especificação do *hardware* são enunciadas todas as potencialidades pretendidas tendo em atenção as dimensões da placa, a caixa de suporte (a mesma do terminal *CANit* da infra-estrutura *Infracom*) e as suas interfaces. A especificação do *software*, dependente do *hardware* teve como principal objectivo a criação de funções de baixo nível que proporcionassem um acesso transparente ao *hardware* por parte do programador de aplicações.

O capítulo 3 descreve os passos envolvidos no projecto e realização do *hardware* do *BlueTerm*. Inicia-se com a pesquisa de material para a realização do projecto que constituiu uma das fases mais morosas. Encontrar material adequado para a execução do projecto, de acordo com as especificações idealizadas, tornou-se uma tarefa árdua e por vezes complicada, principalmente no que diz respeito à pesquisa de interfaces *Bluetooth* (controlador e rádio). Esta tecnologia é muito recente e existem poucos produtos no mercado a baixo custo e sem grande necessidade de desenvolvimento extra para obtenção das funcionalidades da tecnologia. Neste capítulo aborda-se ainda o projecto e desenho do esquema eléctrico e da respectiva placa, que teve em atenção todas as boas normas de desenho de *hardware*, especialmente em casos onde existirão sinais de frequências muito elevadas.

As etapas envolvidas no desenvolvimento do *software* são o tema do capítulo 4. Esta fase ocupou a maior percentagem de tempo do projecto, como é natural na maioria dos projectos que envolvam as componentes de *hardware* e *software*. Foi desenvolvido um conjunto de funções organizadas em módulos para acesso aos periféricos do terminal (porta série, iic, etc.). Estas funções foram testadas individualmente e posteriormente foi realizada uma aplicação demonstrativa das capacidades do terminal que permitiu o seu teste em conjunto.

No capítulo seguinte é descrito um programa para configuração e teste do terminal com o objectivo de realizar o *setup* do equipamento.

O estudo da tecnologia sem fios *Bluetooth* e da sua aplicação no *BlueTerm* é feita no capítulo 6, sendo uma das fases mais interessantes do projecto, visto ser aquela que permitiu o primeiro contacto com uma tecnologia recente e com um conceito bastante inovador.

No capítulo 7 descreve-se o desenvolvimento de *software* de aplicação, em linguagem *C* para o *BlueTerm* e em *Visual Basic* para um computador de controlo. O principal objectivo foi exemplificar a futuros programadores de aplicações para este terminal, a forma de utilização do *software* de baixo nível para acesso a todos os periféricos. A criação desta aplicação permitiu ainda a realização de um teste rápido e fiável do equipamento de modo a determinar o seu estado após a montagem.

No ultimo capítulo são apresentados os resultados e as conclusões do trabalho.

2 Especificação do BlueTerm

2.1 Condicionais – a Infra-estrutura existente Infracom

O IDITE-Minho (Instituto de Desenvolvimento e Inovação Tecnológica do Minho) desenvolveu uma infra-estrutura denominada *Infracom* com o objectivo de disponibilizar um conjunto de módulos de *hardware* e *software* que permitisse adquirir informação e actuar em inúmeros equipamentos em meio industrial, como por exemplo, na realização do controlo integrado da produção numa fábrica têxtil. Os diversos módulos desta estrutura comunicam entre si e com um eventual controlador central através de uma rede *CAN* (Figura 1).

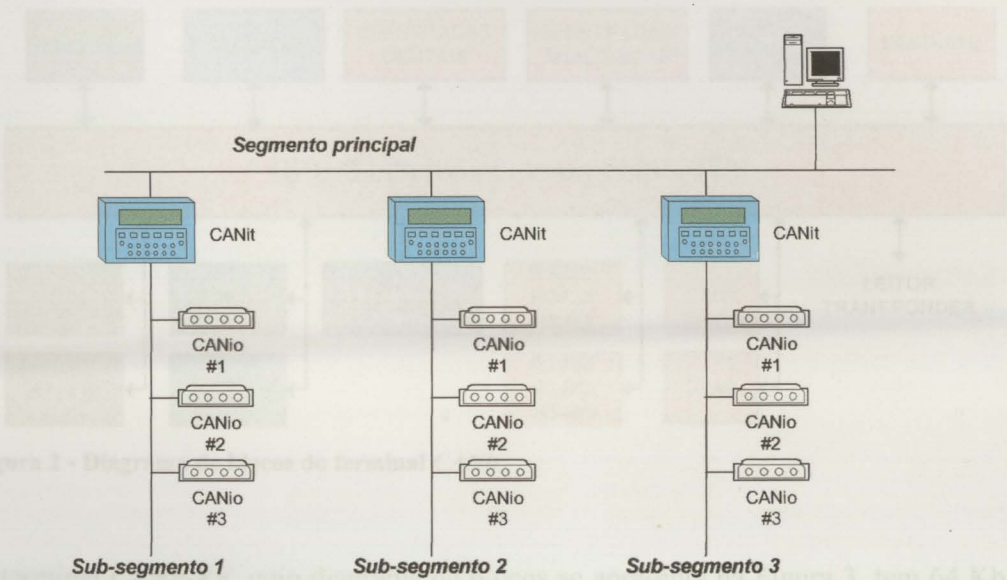


Figura 1- Infra-estrutura Infracom

Sendo um dos objectivos a ter em conta na especificação do *BlueTerm* a sua compatibilidade com a estrutura *Infracom* vamos começar por descrever sucintamente os módulos que a compõe.

2.1.1 Terminais de uso genérico *CANit* e *CANit-RF*

A infra-estrutura *Infracom* disponibiliza dois tipos de terminais de uso genérico denominados *CANit* e *CANit-RF*, ambos baseados em processadores de 8 bits mas com capacidades diferentes.

O terminal *CANit*, cujo diagrama de blocos se apresenta na Figura 2, tem 128 Kb de memória de programa externa do tipo *flash*, 128 Kb de memória de dados não volátil e memória de configuração do tipo *eeeprom*. Possui ainda um visor de quatro linhas por vinte colunas, um teclado em membrana com vinte teclas, relógio de tempo real, interfaces de comunicação *RS232* ou *RS422* ou *RS485* e *CAN*, uma interface que possibilita a colocação de um leitor de *transponders*, quatro entradas digitais, quatro entradas analógicas, quatro saídas em relé e cartão de memória *flash* (*multimedia card*).

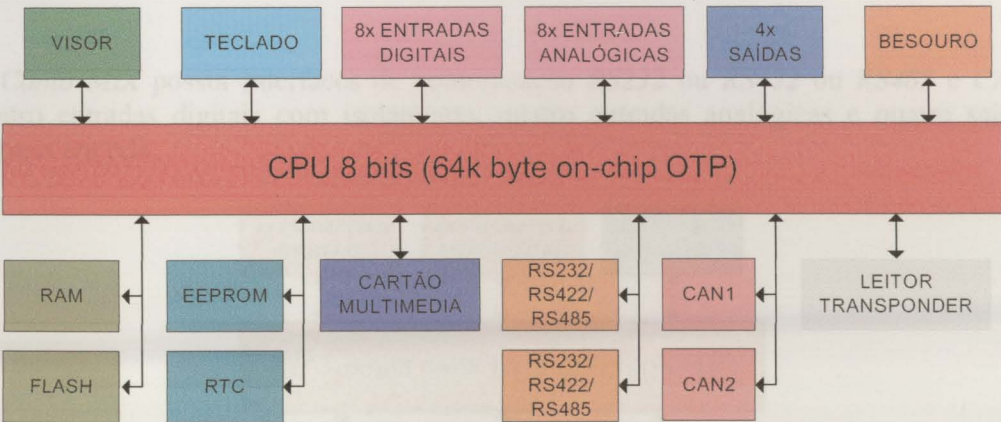


Figura 2 - Diagrama de blocos do terminal *CANit*

O terminal *CANit-RF*, cujo diagrama de blocos se apresenta na Figura 3, tem 64 Kb de memória de programa do tipo *flash*, 128 Kb de memória de dados não volátil, memória de configuração do tipo *eeeprom*, visor de quatro linhas por vinte colunas, um teclado com vinte teclas, relógio de tempo real, interfaces de comunicação *RS232*, *RF* (433 MHz) e *CAN*.

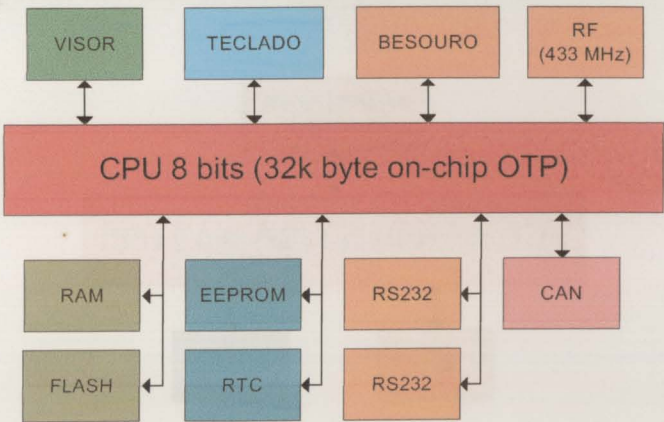


Figura 3 - Diagrama de blocos do terminal *CANit-RF*

2.1.2 Módulos de entradas e saídas de uso genérico e interfaces de comunicação

A *Infracom* tem módulos que permitem realizar diferentes tipos de operações de entrada e saída de dados digitais ou analógicos. Todos estes módulos dispõem de capacidade de comunicação com a rede *CAN* e alguns deles possibilitam comunicação com outro tipo de protocolos série como o *RS232*, *RS422* e *RS485*.

2.1.2.1 Módulos de entradas e saídas

Todos os módulos de entradas e saídas são para uso genérico e configuráveis em função da aplicação.

O *Canio MIX* possui interfaces de comunicação *RS232* ou *RS422* ou *RS485* e *CAN*, quatro entradas digitais com isolamento, quatro entradas analógicas e quatro saídas digitais em relé.

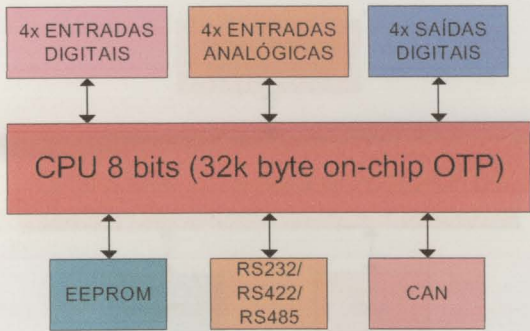


Figura 4 - Diagrama de blocos do terminal *CANio-MIX*

O *Canio 16-IN* possui interface de comunicação *CAN* e dezasseis entradas digitais com isolamento.

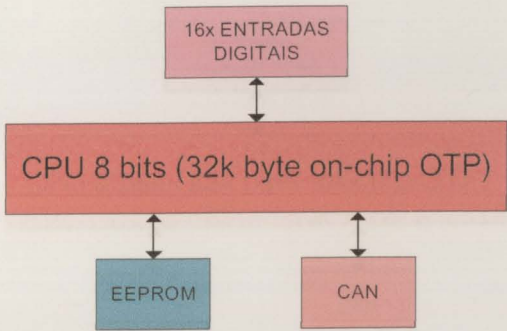


Figura 5 - Diagrama de blocos do terminal *CANio-16-IN*

O *Canio 8-OUT* possui interface de comunicação *CAN* e oito saídas digitais em relé.

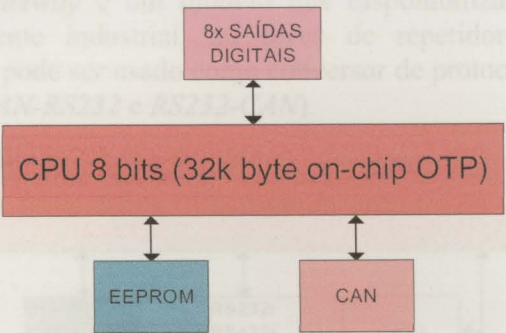


Figura 6 - Diagrama de blocos do terminal *CANio-8-OUT*

O *Canio 8-IN* possui interface de comunicação *CAN* e oito entradas analógicas (4-20 mA ou 0-5V ou 0-12V ou 0-24V). Este módulo encontra-se ainda em fase de desenvolvimento.

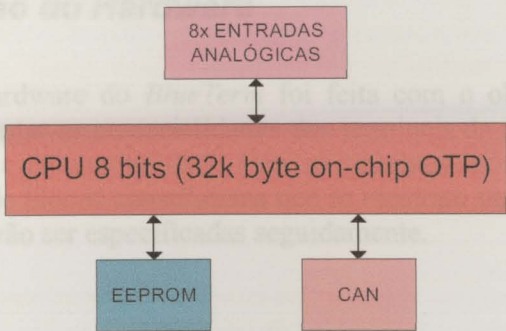


Figura 7 - Diagrama de blocos do terminal *CANio-8-IN*

O *Canio 4-OUT* possui interface de comunicação *CAN* e quatro saídas analógicas em corrente ou tensão. Este módulo encontra-se ainda em fase de desenvolvimento.

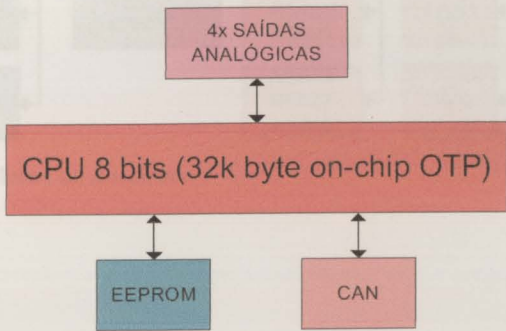


Figura 8 - Diagrama de blocos do terminal *CANio-4-OUT*

2.1.2.2 Módulos interface de comunicações

O *Canio Repetidor/Gateway* é um módulo que disponibiliza duas funcionalidades, necessárias em ambiente industrial, ou serve de repetidor para a interface de comunicações *CAN* ou pode ser usado como conversor de protocolos para a interface de comunicações *CAN* (*CAN-RS232* e *RS232-CAN*).

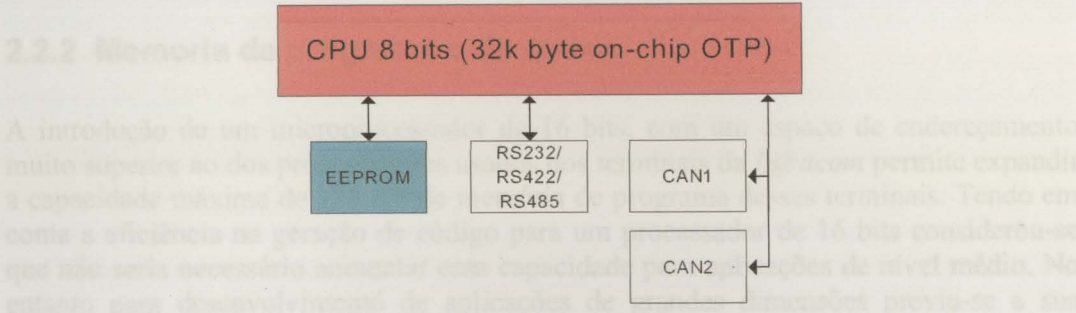


Figura 9 - Diagrama de blocos do terminal *CANio-Repetidor/Gateway*

2.2 Especificação do Hardware

A especificação de hardware do *BlueTerm* foi feita com o objectivo de melhorar o desempenho e incrementar as potencialidades dos terminais da infra-estrutura *Infracom* mantendo no entanto a compatibilidade com os restantes módulos. Chegou-se desta forma a um conjunto de blocos constituintes que se mostram no diagrama da Figura 10 e cujas características vão ser especificadas seguidamente.

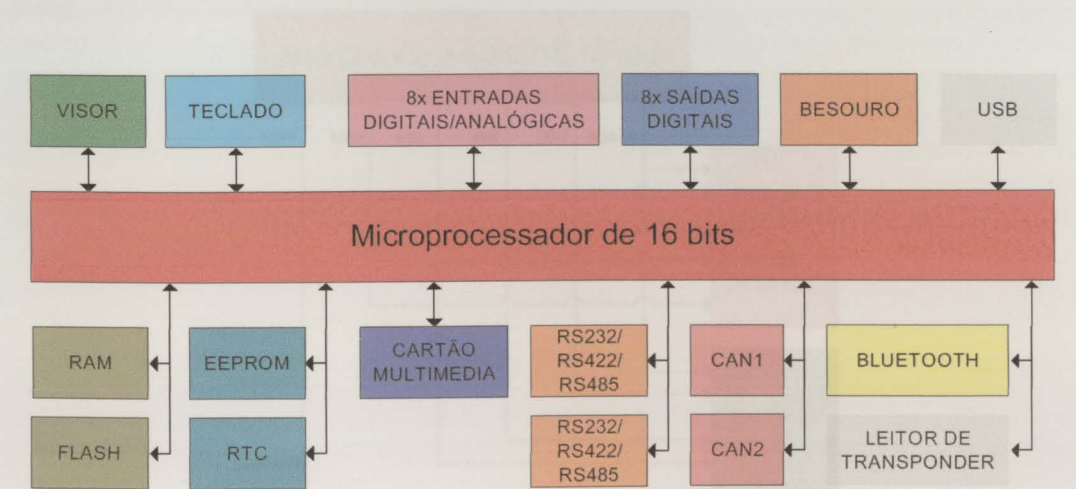


Figura 10- Diagrama de blocos do hardware

2.2.1 Microprocessador de 16 bits (CPU)

Com um processador deste tipo espera-se melhorar a capacidade de processamento relativamente aos microprocessadores de 8 bits usados nos terminais *CANit* e *CANit-RF* e suprimir algumas das lacunas reveladas em aplicações de média/alta complexidade.

2.2.2 Memória de programa e de dados

A introdução de um microprocessador de 16 bits, com um espaço de endereçamento muito superior ao dos processadores usados nos terminais da *Infracom* permite expandir a capacidade máxima de 128 Kb de memória de programa desses terminais. Tendo em conta a eficiência na geração de código para um processador de 16 bits considerou-se que não seria necessário aumentar essa capacidade para aplicações de nível médio. No entanto para desenvolvimento de aplicações de grandes dimensões previu-se a sua expansão até 384 Kb.

A memória deverá ser do tipo *flash* permitindo a gravação do programa (o número de vezes que for necessário) através da interface de comunicações *RS232*.

Manteve-se uma memória de dados de 128 Kb dado que a experiência tem demonstrado que essa capacidade tem sido suficiente em todas as aplicações até agora desenvolvidas para o terminal *CANit*, tendo havido necessidade de utilizar toda a sua capacidade apenas em alguns casos muito específicos como o da criação de *buffers* para facilitar a troca de dados com alguns dispositivos relativamente lentos, como por exemplo o cartão multimédia.

2.2.4 Relógio de Tempo Real

Um praticamente todo o sistema de comunicação industrial necessita de um relógio de tempo real de modo a ser possível associar a cada mensagem transmitida um tempo (por exemplo, a uma transição de uma entrada digital para o estado de alto ou de baixo) e, para ser conveniente usar uma de maior precisão, é necessário que o relógio esteja integrado no sistema com capacidade de manter o seu funcionamento a qualquer custo.

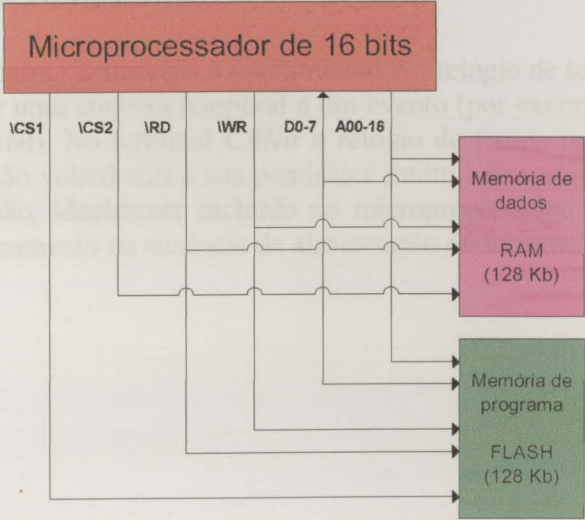


Figura 11 – Ligação das memórias ao microprocessador

2.2.3 Memória de configuração

A necessidade de guardar parâmetros de configuração do terminal levou à introdução de uma memória do tipo *eeeprom*. Existem no mercado memórias deste tipo com várias capacidades (128 byte, 256 byte, 512 byte, 1024 byte ou 2048 byte) podendo utilizar-se qualquer uma em função das necessidades, dada a sua compatibilidade. Por questões de custo/disponibilidade seleccionou-se a memória de maior capacidade, até porque no terminal *CANit* foi constatado que esta capacidade é suficiente em todas as aplicações desenvolvidas.

Note-se que este tipo de memórias utiliza um protocolo de comunicações *iic* e por esse facto será conveniente que o microprocessador de 16 bits a seleccionar incorpore um controlador para esse protocolo.

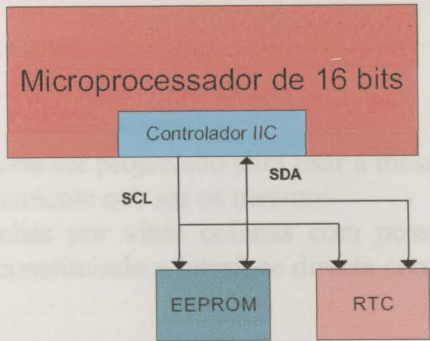


Figura 12 – Ligação dos dispositivos *iic* ao microprocessador

2.2.4 Relógio de Tempo Real

Em praticamente todas as aplicações é fundamental um relógio de tempo real de modo a ser possível associar uma etiqueta temporal a um evento (por exemplo, a uma transição de uma entrada digital). No terminal *CANit* o relógio de tempo real está integrado na memória de dados não volátil mas a sua precisão é muito baixa, sendo conveniente usar um de maior precisão, idealmente incluído no microprocessador, com capacidade de manter o seu funcionamento na ausência de alimentação do terminal.

2.2.5 Visor e teclado

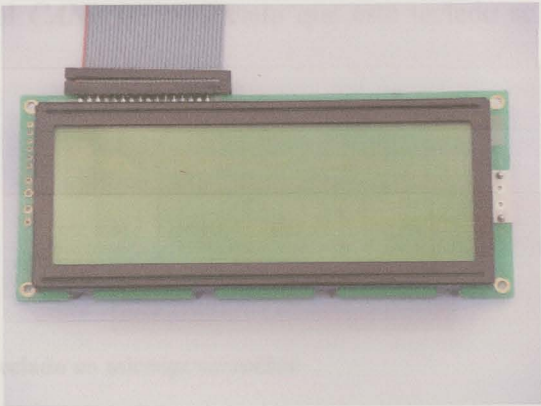


Figura 13 – Visor do terminal CANit

2.2.6 Interfaces de comunicação

Devido ao terminal *BlueTerm* ser projectado para usar a mesma caixa do *CANit*, o visor e o teclado terão obrigatoriamente que ser os mesmos. O visor possui quatro linhas por vinte colunas com possibilidade de regulação de luminosidade e contraste constituindo a interface directa com o utilizador para afixação de mensagens.

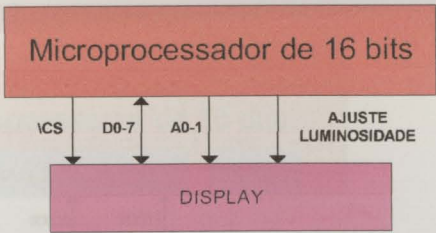


Figura 14 – Ligação do visor ao microprocessador



Figura 15 - Teclado do terminal CANit

O teclado de membrana tem vinte teclas que podem ser derivadas em n funções por meio de uma tecla que permite a comutação entre em modo normal e modo função (F1, F2, ..., F20) e de uma outra para comutação entre modo numérico e modo alfa numérico. No terminal *CANit* foi verificado que este teclado se adequa às aplicações nele desenvolvidas.

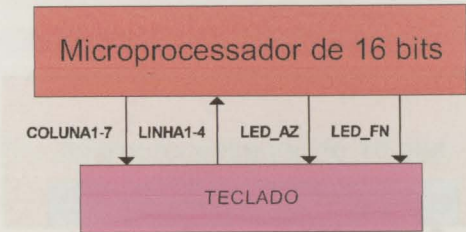


Figura 16 – Ligação do teclado ao microprocessador

2.2.6 Interfaces de comunicação

A experiência de utilização do terminal *CANit* mostrou que se deveriam manter as interfaces de comunicação *RS232/RS422/RS485*, dado que cobrem quase a totalidade das aplicações existentes no mercado. O tipo de interface a utilizar, diferencial ou não, deverá ser escolhido por *software*, sendo a selecção entre diferenciais feita através de ligadores.

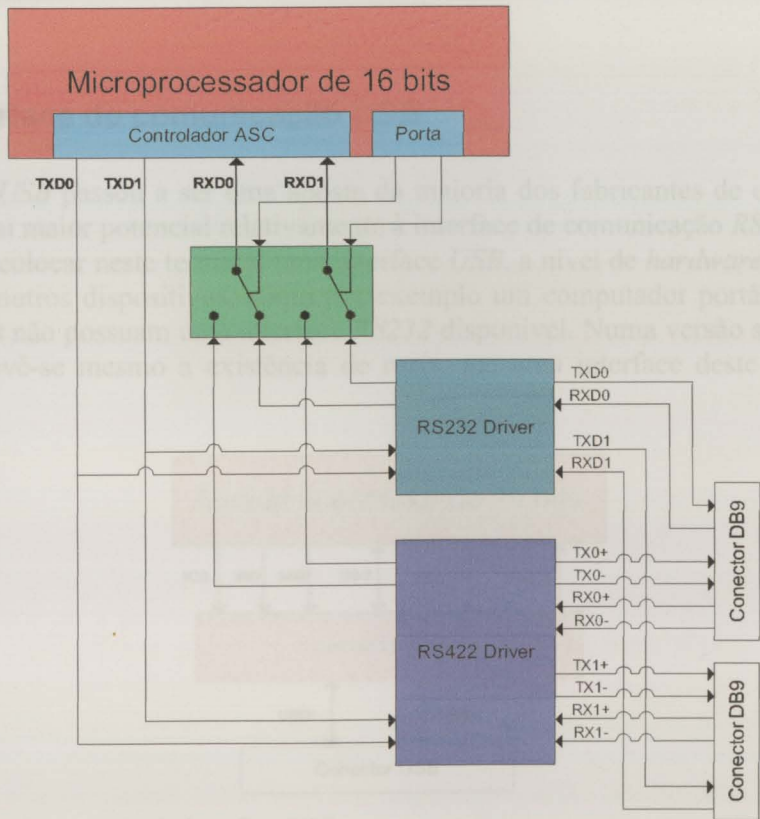


Figura 17 – Implementação das interfaces série RS232/RS422

A compatibilidade do *BlueTerm* com a *Infracom* obriga a introdução de interfaces de comunicação *CAN* que idealmente deveriam ser duas e incorporadas no microprocessador. Foi verificado que o isolamento na interface *CAN* se torna fundamental em grande parte das aplicações apesar do acréscimo de custo inerente. O terminal *CANit* possui um canal com isolamento e outro sem. No *BlueTerm* pretende-se introduzir a possibilidade de colocar ou não isolamento em ambos os canais.

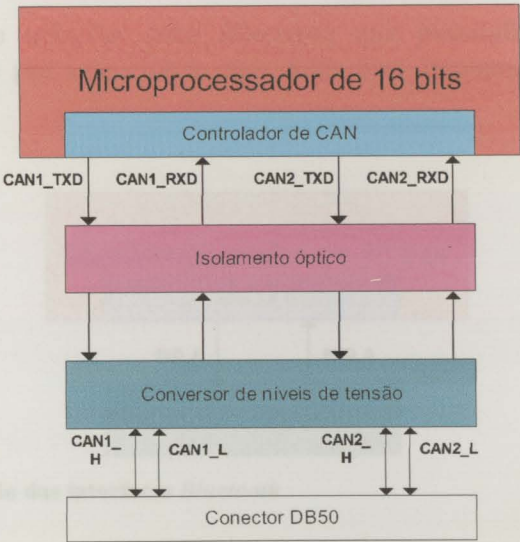


Figura 18 – Implementação das interfaces *CAN*

2.2.7 Interface de comunicação *USB*

A interface *USB* passou a ser uma aposta da maioria dos fabricantes de computadores devido ao seu maior potencial relativamente à interface de comunicação *RS232*. Pretende-se colocar neste terminal uma interface *USB*, a nível de *hardware*, que permita a ligação a outros dispositivos, como por exemplo um computador portátil, nos casos em que estes não possuam uma interface *RS232* disponível. Numa versão seguinte deste terminal prevê-se mesmo a existência de mais que uma interface deste tipo caso se justifique.

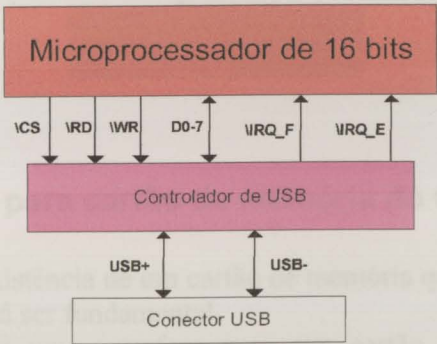


Figura 19 – Implementação da interface *USB*

2.2.8 Interface de comunicação Bluetooth

A interface de comunicação sem fios *Bluetooth* apresenta-se como a inovação que dará uma das maiores valias a este terminal permitindo a sua ligação a outros equipamentos de mercado do modo mais simples que as soluções tradicionais com cabo. Como já foi referido anteriormente prevê-se que esta interface elimine algumas barreiras existentes permitindo assim alargar o leque de aplicações a que este terminal poderá dar resposta. Existindo no mercado soluções para *Bluetooth* que possibilitam a sua ligação a controladores optou-se por seleccionar uma delas para implementar esta interface de comunicação.

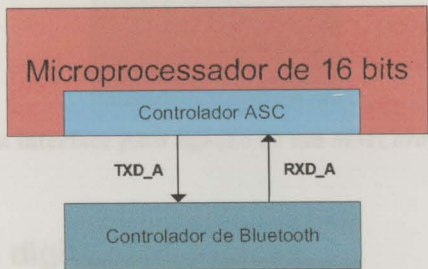
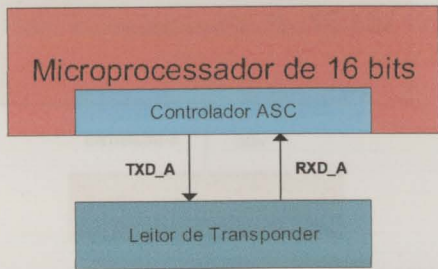


Figura 20 – Implementação das interfaces Bluetooth

2.2.9 Interface para leitor de transponders

Manteve-se a interface para leitor de *transponders*, existente no terminal *CANit*, dada a sua utilidade em determinadas aplicações, como por exemplo o controlo de ponto. Esta interface usa uma ligação série do tipo assíncrona.



2.2.10 Interface para cartão de memória do tipo flash

Em muitas aplicações a existência de um cartão de memória que possibilite a recolha ou troca de informação poderá ser fundamental. O terminal *CANit* possui uma interface para um cartão multimédia, mas tem-se constatado que devido a uma questão de capacidade/custo/aplicação um cartão do tipo

SDCard tem vantagens. Pretende-se no *BlueTerm* a colocação de uma interface que permita o uso dos dois tipos de cartões de memória *flash*. A ligação a este tipo de dispositivos requer um controlador de *spi* que idealmente deverá estar incorporado no microprocessador.

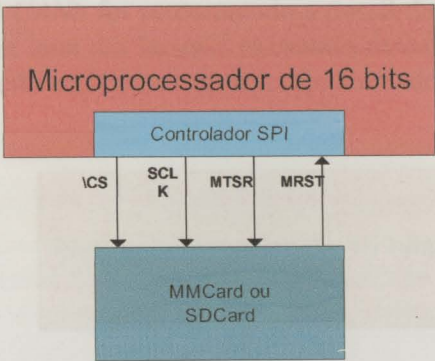


Figura 21 – Implementação da interface para ligação de um *MMC* ou *SD Card*

2.2.11 Entradas digitais/analógicas

O *BlueTerm* deverá ter a capacidade de monitorizar sinais de tipo analógico ou digital provenientes de outros equipamentos. Foram previstas oito entradas configuráveis como digitais ('0' ou '1') ou analógicas (4-20 mA, 0-5V, 0-12V, 0-24 V). A configuração da funcionalidade deverá ser feita por *software* enquanto que a opção do comportamento das entradas analógicas deverá ser feito por alteração de ligações no *hardware*. Para isso o microprocessador deverá possuir portas com possibilidade de configuração como entradas digitais ou analógicas.

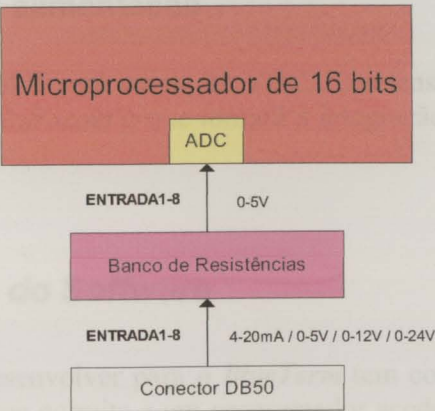


Figura 22 – Implementação das entradas digitais/analógicas

2.2.12 Saídas digitais

O *BlueTerm* deverá ter a capacidade de actuar sobre outros equipamentos através de saídas digitais. Foram previstas oito saídas do tipo relé (normalmente aberto (NA) ou normalmente fechado (NF)) ou em colector aberto. Relativamente ao terminal *CANit* foi acrescentado a possibilidade de saídas em colector aberto suprimindo-se assim uma das lacunas existentes nesse terminal. É conveniente a existência de portas de saída suficientes no microprocessador de modo a implementar esta funcionalidade.

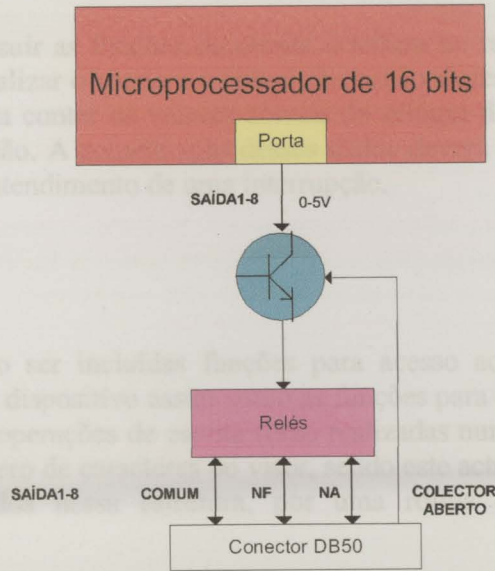


Figura 23 – Implementação das saídas digitais

2.2.13 Fonte de alimentação

A alimentação do terminal deverá ser de 24 VDC. Esta tensão de alimentação é usada por todos os módulos da *Infracom* o que tornará a integração do *BlueTerm* nesta infraestrutura mais facilitada.

2.3 Especificação do Software

O *software* que se vai desenvolver para o *BlueTerm* tem como objectivo primordial a criação de uma camada que permita a um programador aceder aos recursos do terminal de uma forma simples e transparente relativamente ao *hardware*. Vai ser organizado em módulos directamente relacionados com os blocos constituintes do terminal que devem realizar as funcionalidades identificadas a seguir.

2.3.1 Memória de configuração

Este módulo deve possuir as funções de escrita e leitura na memória de configuração. Dado que o protocolo de comunicação com este dispositivo é o *iic* será necessário criar funções que implementam este protocolo.

O programador deverá utilizar as funções deste módulo quando pretender escrever ou ler qualquer configuração neste dispositivo.

2.3.2 Relógio de Tempo Real

Este módulo deve possuir as funções de escrita e leitura no relógio de tempo real, de modo a ser possível realizar o acerto e a leitura dos seus valores. Deverá ser criada uma estrutura de dados para conter os valores actuais do relógio de modo ao programador aceder a esta informação. A actualização desses dados deverá ser feita periodicamente usando uma rotina de atendimento de uma interrupção.

2.3.3 Visor

Neste módulo deverão ser incluídas funções para acesso ao visor. Será necessário efectuar a iniciação do dispositivo assim como as funções para escrita de vários tipos de informação. Todas as operações de escrita serão realizadas numa estrutura de dados de tamanho igual ao número de caracteres do visor, sendo este actualizado periodicamente, com os dados contidos nessa estrutura, por uma rotina de atendimento a uma interrupção.

2.3.4 Teclado

Este módulo deve conter funções que permitam o uso do teclado de uma forma simples de modo a permitir operações básicas como a navegação em menus e a introdução de dados.

2.3.5 Interfaces de comunicação

Este módulo deverá ter funções que possibilitem o acesso aos canais de comunicação *RS232/RS422/RS485*, nomeadamente para envio e recepção de informação ao nível do carácter e ao nível da mensagem. Deverão ser realizadas funções de alto nível que permitam o envio e recepção independentemente do tipo de canal de comunicação.

Para manter a compatibilidade com a *Infracom* deverão ser desenvolvidas funções de baixo nível que permitam a realização futura de uma camada de *software*, que recorrendo às funções de baixo nível, possibilite a troca de informação através da interface CAN com os vários módulos da *Infracom*.

2.3.6 Interface de comunicação *USB*

Nesta fase não está contemplado o desenvolvimento de funções para esta interface. O *hardware* ficará disponível e como trabalho futuro será desenvolvido um módulo de *software* com as rotinas necessárias ao funcionamento desta interface.

2.3.7 Interface de comunicação *Bluetooth*

Como foi referido no capítulo 2.2.8, pretende-se utilizar esta interface recorrendo a um controlador com interface série assíncrona. Logo as funções para troca de informação através do *Bluetooth* recorrerão às funções de baixo nível do módulo de comunicações.

2.3.8 Interface para leitor de *transponders*

Nesta fase não está contemplado o desenvolvimento de funções para esta interface. O *hardware* ficará disponível e como trabalho futuro será desenvolvido um módulo de *software* com as rotinas necessárias ao funcionamento desta interface.

2.3.9 Interface para cartão de memória do tipo *flash*

Este módulo deverá possuir funções para escrita e leitura no cartão de memória *flash*.

Dado que o protocolo de comunicação com este dispositivo é o *spi* será necessário criar funções que implementam este protocolo.

O programador deverá utilizar as funções deste módulo quando pretender escrever ou ler neste dispositivo.

2.3.10 Entradas digitais/analógicas

Este módulo deverá possuir funções para configuração do comportamento das entradas – digitais ou analógicas. Em função do comportamento deverá ser possível efectuar a leitura do estado digital ou do valor analógico. Os valores analógicos deverão ser alvo de filtragem de modo a serem fornecidos valores estáveis.

2.3.11 Saídas digitais

Este módulo deverá ser constituído por funções para actuação das saídas. Deverão ser desenvolvidas funções para actuação imediata e actuação programada em função da data e hora.

2.3.12 Genéricos

Este módulo deverá possuir as funções genéricas que não se enquadram nos módulos atrás descritos. Aqui podem ser incluídas funções de configuração e iniciação dos periféricos. As tarefas periódicas necessárias ao bom funcionamento de outros módulos devem ser implementadas aqui, através do recurso a chamadas de interrupção.

Funções genéricas para conversão de tipos de dados e filtragem de informação podem ser incluídas neste módulo ou, no caso de serem em quantidade significativa, integrar um módulo próprio.

3 Projecto de hardware do *BlueTerm*

3.1 Pesquisa de material

3.1.1 Microprocessadores

Tendo em atenção a aplicação pretendida a pesquisa foi orientada para microprocessadores de 16 bits que possuísem memória de programa re-programável, memória de dados, interface *RS232*, interface *CAN*, conversor analógico-digital, interface *spi* e interface *iic*. Além destas características pretendeu-se um componente para o qual existisse suporte a nível técnico e a nível de ferramentas disponíveis para desenvolvimento (por exemplo compiladores).

Na Tabela 1 apresenta-se uma síntese dos resultados da pesquisa efectuada para microprocessadores da *Intel*, *Motorola*, *ST*, *Philips* e *Infineon*.

Tabela 1 - Alguns microprocessadores de 16 bits existentes no mercado

Frequência de operação máxima (MHz)
Memória de programa Flash (Kb)
Memória de programa EPROM (Kb)
Memória de programa OTP (Kb)
Memória de dados RAM/XRAM (Kb)
Espaço de endereçamento para dados e código(Kb)
Linhas de entradas/saídas
Número de canais/Resolução do ADC (bit)
Temporizadores/Contadores
Unidades de Captura/comparação (PWM)
PWM
Vectores de interrupção
Interfaces série (SCI/USART)
Interfaces série (SSI/SPI)
Interfaces série (IIC)
Rlogio de Tempo Real
CAN interface 2.0B
On-chip Bootstrap Loader
Encapsulamento

ST		Philips				Motorola						Intel	
	ST10F168		PXAG49	PXAG39	PXAG30	PXAC37	MC9S12D512	MC9S12D256	MC9S12D128	MC9S12D64	MC9S12D32	87C196CB	87C196CA
40	25	30	30			32	25	25	25	25	25	16	16
256	256	64	32				512	256	128	64	32		
				32		32						56	32
12	8	2	1	0.5	1	1	14	12	8	4	2	1.5	1
16 MB	16 MB	1024	1024	1024			1024	1024	1024	1024		1024	64
111	111	32	32	32	32	32	91	91	91	91	59	56	38
16/10	16/10						16/10	16/10	16/10	16/10	8/10	8/10	6/10
5	5	3	3	3	3	3	4	4	4	4	4	2	2
2	2						8	8	8	8	8	10	10
4	4						8	8	8	7			
56	56			38		42						37	37
1	1	2	2	2	1	1	2	2	2	2	2	1	1
1	1				1 x SPI	1 x SPI	3 x SPI	3 x SPI	2 x SPI	1 x SPI	1 x SPI	1 x SPI	1 x SPI
							1	1	1	1			
1													
2	1				1	1	5	3	3	1	1	1	1
Sim	Sim	1	1										
PQFP144	PQFP144	PLCC44 / LQFP44	PLCC44	PLCC44 / LQFP44			LQFP112	LQFP112	LQFP112	LQFP112	QFP80	PLCC84	PLCC68

Infineon				
XC167	XC161	XC164	ST10F280	
40	40	40	40	
128	128	128	512	
8	8	6	18	
16 MB	16 MB	16 MB	16 MB	
103	99	79	143	
16/10	14/10	14/10	2x16/10	
5	5	5	5	
2	2	2	2	
7		7	8	
77	74	75	56	
2	2	2	1	
2	2	2	1	
3	3			
1	1	1		
2	2	2	2	
Sim	Sim	Sim	Sim	
TQFP144	TQFP144	TQFP100	PBGA208	

Inicialmente a escolha pendeu para o microprocessador *XC164* devido a este preencher quase a totalidade dos requisitos, mas após um estudo mais detalhado verificou-se que o consumo do relógio de tempo real com o microprocessador em modo *sleep* ou *power-down* é demasiado elevado o que implicaria a colocação de uma pilha de grande capacidade com os inconvenientes inerentes. Este problema deve-se ao facto de o microprocessador apenas possuir entrada para ligação de um cristal que fornece a frequência para o processamento e para os periféricos. Quanto maior a frequência do cristal maior o consumo, logo se pretendemos uma velocidade de processamento elevada teremos que ligar um cristal de frequência elevada o que condiciona o funcionamento do relógio de tempo real em modo *sleep* e *power-down*.

Devido à constatação do problema descrito, foi tido em consideração o número de cristais que se podem ligar ao microprocessador de modo a resolver o problema anterior, pois se existir a possibilidade de ligação de um cristal unicamente para fornecimento de frequência ao relógio de tempo real, este poderá ser de baixo valor (tipicamente 32 KHz) o que implicará um consumo reduzido deste periférico nos modos de *sleep* e *power-down*.

O microprocessador *XC167* surgiu como alternativa pois este permite a ligação de um cristal dedicado ao fornecimento da frequência de oscilação ao relógio de tempo real. Além desta importante característica este microprocessador possui interface iic o que facilita a comunicação com a memória de configuração eeprom. O único inconveniente no uso deste microprocessador é o custo. Como não existiam outras alternativas o projecto do desenho do circuito eléctrico foi iniciado com a inclusão do microprocessador *XC167* da *Infineon*.

Devido ao acompanhamento constante dos desenvolvimentos ocorridos na *Infineon*, no mês de Janeiro deste ano, foi lançado um novo microprocessador de 16 bits com a ordem dos pinos (*pinout*) compatível com o *XC167* mas a um custo bem mais baixo. A única diferença entre os dois microprocessadores centra-se na unidade de PWM projectada para o controlo de motores que o *XC167* possui e o *XC161* não. Como esta

funcionalidade não é crítica para a aplicação pretendida a escolha ficou definitivamente no microprocessador XC161.

Características do processador XC161 da Infineon:

1. Processador de 16 bits com uma pipeline de 5 estágios
2. Tempo de ciclo de 25 ns com uma frequência de oscilador de 40 MHz
3. Espaço de endereçamento de programa e dados até 16 mega byte
4. Largura do barramento de endereços programável (16 a 24 bits)
5. barramento de dados de 8 ou 16 bits
6. chip selects programáveis
7. Módulos de memória interna:
 - a. 2 Kb de dual port ram
 - b. 4 Kb de data sram
 - c. 2 Kb de programa/dados sram
 - d. 128 Kb de memória de programa (*flash*)
8. Módulos de periféricos internos:
 - a. Um conversor analógico digital de 12 canais com resolução programável (10 bits ou 8 bits)
 - b. Duas unidades de comparação/captura de 16 canais para uso genérico
 - c. Cinco temporizadores para uso genérico
 - d. Dois canais série síncrono assíncrono (USART)
 - e. Dois canais síncronos de alta velocidade (SSC)
 - f. Dois canais de CAN (versão 2.0B) com 32 objectos
 - g. Interface iic com 3 canais multiplexados
 - h. RTC com cristal próprio
9. *Watchdog* programável
10. Até 99 linhas de entradas/saídas para uso genérico
11. *Bootstrap loader*
12. Suporte de varias ferramentas de *software* (compiladores) e hardware (placas de desenvolvimento)
13. 144 pinos TQFP com espaçamento de 0.5 mm

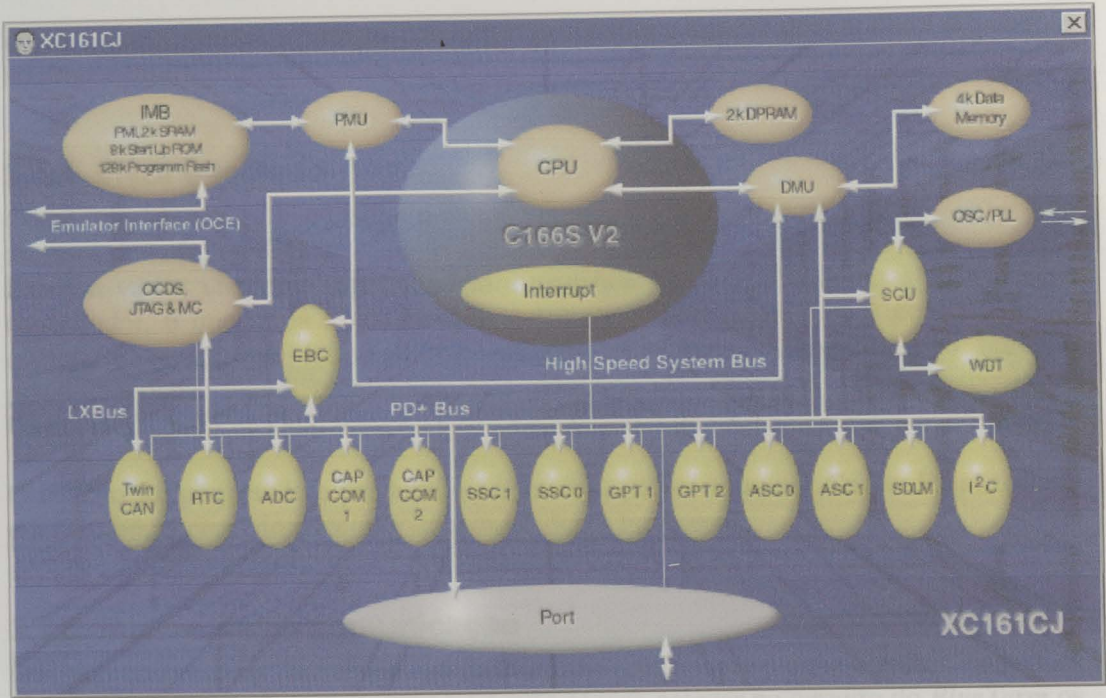


Figura 24 - Diagrama de blocos do microprocessador XC161 (figura retirada do software DAVe da Infineon)

3.1.2 Memórias

A memória de programa externa escolhida foi a AT29C010 da *Atmel*. É uma memória flash de 128 Kb organizada em 1024 sectores de 128 bytes cada. A escolha desta memória deveu-se ao facto de necessitar apenas de uma alimentação, para o seu funcionamento e para a programação.

A memória de dados escolhida foi a M5M51008KV da *Mitsubishi*. Trata-se de uma memória RAM de 128 Kb organizada em 131072 palavras de 8 bits.

A memória de configuração escolhida foi a EEPROM M24C16 da *STMicroelectronics*. Esta referência é de uso comum em aplicações com microprocessadores e a sua comunicação é efectuada através de uma interface IIC. O microprocessador possui um controlador interno que disponibiliza quatro canais multiplexados de interface IIC, deste modo a ligação desta memória ao processador é feita de modo directo.

Teve influência na escolha das memórias de programa e dados a existência de experiências anteriores bem sucedidas com o seu uso.

3.1.3 Relógio de Tempo Real

Apesar de o microprocessador possuir relógio de tempo real interno, não foi possível obter em tempo útil a informação necessária sobre o seu total funcionamento o que levou à escolha de um relógio externo para colmatar essa lacuna. O relógio escolhido foi o MK41T56 da *STMicroelectronics* devido a experiências anteriores bem sucedidas com o seu uso. A comunicação com este relógio efectua-se através de uma interface IIC,

o que, como referido no parágrafo anterior, possibilita a ligação directa deste dispositivo ao processador.

3.1.4 Visor e teclado

Como referido em 2.2.5 o visor e o teclado foram condicionados pela caixa, logo são os mesmos do terminal *CANit*.

Existem vários visores compatíveis em dimensões, disposição dos pinos de ligação e controlador, o que torna a substituição entre eles transparente e sem necessidade de qualquer alteração de hardware ou software. Por questões inerentes ao binómio preço/qualidade foram escolhidos dois modelos de entre os vários disponíveis: CM2041S1LY-J6 da *Data Image* e CMC420L03YBN da *CCT*.

O teclado foi projectado e produzido por medida para o terminal *CANit*. É um teclado em membrana com vinte teclas e dois indicadores luminosos.

3.1.5 Interfaces de comunicação

O microprocessador escolhido já possui um controlador que disponibiliza duas interfaces série assíncrona/síncrona bastando apenas colocar no circuito do *BlueTerm* os conversores de nível de tensão para as interfaces *RS232/RS422/RS485*. O conversor de nível de tensão escolhido para o RS232 foi o MAX202 e para o RS422/RS485 o MAX489, ambos da *Maxim*, devido ao facto de serem comuns para este tipo de interfaces.

Quanto às interfaces CAN, o microprocessador possui um controlador com duas interfaces CAN sendo necessário colocar apenas o conversor de nível de tensão; o 82C250 da *Philips* foi o escolhido devido à existência de conhecimento prévio na sua aplicação.

3.1.6 Interface de comunicação USB

Para disponibilização desta interface foi encontrado um circuito integrado de referência FT245AM da *FTDI* que possibilita a integração de uma forma simples e rápida de uma interface USB numa plataforma com microprocessador. A ligação deste dispositivo ao microprocessador é efectuada através do barramento paralelo de dados (8 bits).

3.1.7 Interface de comunicação Bluetooth

A interface de comunicação sem fio *bluetooth* será disponibilizada com recurso a um módulo que realiza a conversão de RS232 para *bluetooth*. Este módulo da *Wireless Futures* possui a referência BlueWAVE PCB OEM RS232 DCE Terminal. Devido à necessidade de uma interface série assíncrona/síncrona para ligação ao módulo de *bluetooth* e as existentes no microprocessador estarem destinadas a outras funcionalidades, foi necessário escolher um controlador externo ao processador para disponibilizar esta interface. Como a interface do leitor de *transponders* necessita igualmente de uma interface série deste tipo, optou-se por escolher uma DUART da *Philips* de referência SC26C92 que possui dois canais.

3.1.8 Interface para leitor de *transponders*

Esta interface será disponibilizada pelo módulo MicroEngine da *Datatronic*. A escolha deveu-se ao facto de ser a mesma interface pensada para o terminal *CANit*. O módulo possui uma interface de comunicação série assíncrona/síncrona e será implementada com recurso a um controlador externo ao processador como explicado em 3.1.7.

3.1.9 Interface para cartão de memória do tipo *flash*

No circuito será colocado um suporte que permite a utilização de cartões de memória flash multimédia (*MMCard*) e *SDCard*. A interface de comunicação com este tipo de cartões é a SPI, disponibilizando o microprocessador um controlador interno com dois canais. Estes cartões de memória possuem uma alimentação de 3.3 V obrigando à introdução de uma fonte de tensão dedicada e conversor de níveis de tensão de modo a ser possível conectar o cartão ao controlador, uma vez que toda a alimentação da lógica do circuito será de 5 V.

3.1.10 Entradas digitais/analógicas e Saídas digitais

O microprocessador possui um conversor analógico/digital de 10 bits com oito canais. Os pinos do microprocessador que disponibilizam os canais do conversor para o exterior podem ser igualmente configurados como entradas digitais, logo por software será possível escolher o comportamento das oito entradas. No circuito deverá ser colocado uma montagem com resistências associada a cada entrada, de modo a ser possível configurar por *hardware* o tipo de entrada analógica 0-5V, 0-12V, 0-24V ou 4-20 mA.

Quanto às saídas digitais optou-se pela colocação dos relés da *Omron* G5V-1-24. Serão usados pinos de saída do microprocessador ligados a transístores e estes aos relés, de modo a ser possível disponibilizar além das saídas em relé, saídas em colector aberto.

3.1.11 Fonte de alimentação

Para fornecer a alimentação principal do terminal, de 5 volts contínuos, a partir de uma entrada de 24 volts contínuos foi escolhida uma fonte de tensão comutada da *National Semiconductors* de referência LM2825. Esta fonte é a mesma do terminal *CANit* tendo neste demonstrado um bom comportamento.

3.2 Reformulação e ajuste das especificações

A escolha dos componentes para integrar o terminal *BlueTerm*, forçou pequenas reformulações da especificação inicial do *hardware*.

Apesar do microprocessador XC161 possuir relógio de tempo real interno, optou-se pela colocação de um externo (com comunicação série - *iic*) que numa versão mais antiga do terminal *CANit* mostrou muito bom comportamento; uma das razões principais que levou à colocação de outro relógio externo foi a indefinição na forma de

manter o funcionamento do relógio interno do microprocessador após o desligar da alimentação. Apesar das várias diligências para resolução deste problema junto do fabricante do processador não foi possível obter em tempo útil uma informação que nos permitisse a utilização fiável do relógio de tempo real interno.

Além desta introduziram-se ainda as seguintes alterações:

- Limitação da memória de programa *flash* em 128 Kb - esta alteração deve-se ao facto de a utilização de apenas a memória flash interna do microprocessador facilitar muitíssimo a gravação do programa, ao contrário do que seria a utilização de memória interna e externa. Esta alteração não influencia as intenções previamente demonstradas pois foi constatado que uma mesma aplicação usada num microprocessador de 8 bits e num de 16 bits ocupa menos espaço neste ultimo (devido a vários factores incluindo o compilador utilizado) o que faz com que as aplicações que se tornavam grandes para o terminal *CANit* possam ser aqui utilizadas. No entanto não fica posta de parte a intenção de numa futura versão deste terminal aumentar a capacidade da memória de programa.
- Inclusão de uma memória *flash* com capacidade de 128 Kb para dados - a memória *flash* externa que inicialmente iria ser usada para programa ficou disponível para utilização como um disco para armazenamento de informação.
- Redução do número de saídas para quatro - devido ao *BlueTerm* usar a caixa do *CANit*, a ficha de interface para o exterior terá que ser a mesma. A ficha usada para ligação do terminal ao exterior possui cinquenta pinos e devido ao seu número limitado não foi possível colocar as oito saídas como era pretendido inicialmente pois não existia forma de disponibilizar os seus contactos no exterior da caixa.

3.3 Desenho do esquemático

Esta tarefa foi desempenhada recorrendo ao uso do *software* de desenho *PowerLogic* 5.0.1 da *Mentor Graphics*.

Foi necessário começar por desenhar vários modelos de componentes que por serem recentes não existiam no *PowerLogic*. Tendo todos os modelos dos componentes disponíveis para uso, foi feito o desenho do esquemático tendo em atenção os conselhos fornecidos pelos fabricantes dos componentes especialmente do microprocessador.

3.4 Implementação da lista de material

Com a primeira versão do desenho do esquemático concluída foi necessário gerar a lista de material de modo a verificar todos os encapsulamentos (*packages*) escolhidos para os diversos componentes e a realização da pesquisa de mercado com vista à sua obtenção a médio e longo prazo.

No desenho dos modelos dos componentes foram acrescentados atributos para inclusão dos códigos do fornecedor associados a cada componente de modo a facilitar posteriormente a realização de encomendas.

Para geração da lista de material, o *software* de desenho do esquemático recorre ao uso de macros realizadas na linguagem *basic*; de modo à obtenção de uma lista de material personalizada foi necessário recorrer a alterações na macro existente. Os campos colocados na lista de material foram os seguintes:

1. Qty. - quantidade
2. Ref. - referência (atribuída no esquemático)
3. Part Name - nome da peça (atribuída no desenho do modelo)
4. Value - valor; este campo é especialmente usado em componentes como resistências e condensadores
5. Voltage - tensão
6. Tolerance - tolerância
7. Package - encapsulamento, ou seja, o formato que o componente possui (indispensável para o desenho do PCB)
8. Farnell Code - código do fornecedor
9. Farnell Price - preço do componente para grandes quantidades: este campo serve para ser possível realizar uma estimativa de custo de produção do terminal para quantidades significativas
10. Farnell Price (Unit) - preço unitário
11. Price 1 - preço do componente em outro fornecedor que não a Farnell

Tabela 2 - Exemplo de lista de material

Qty.	Ref.	Part Name	Value	Voltage	Tolerance	Package	Farnell Code	Farnell Price	Farnell Price (Unit)	Price1
1	B1	BATERIA 1/2AA		3V		1/2AA	463310	100+; € 4,726	1+; € 5,551	
.....										
.....										
.....										
1	Y2	XTAL- HC49-V	6MHz			HC49-V	3325271	500+; € 0,333	1+; € 0,736	

Para a realização de protótipos, a obtenção de material torna-se complicada pois os representantes dos fabricantes normalmente possuem quantidades mínimas e valor mínimo de factura para a comercialização dos componentes, o que nesta fase dificulta a obtenção do material devido aos custos envolvidos. No entanto existem dois grandes fornecedores de material eléctrico e electrónico (Farnell e RS-Amidata) que oferecem um vasto número de componentes na maior parte dos casos sem exigência de quantidade mínima. O problema que se coloca na obtenção dos componentes a estes fornecedores é o preço elevado (normalmente mais de 100 % acima do preço praticado pelo representante), mas que é aceitável nesta fase de produção.

3.5 Verificações do esquemático

Esta fase corresponde à verificação de todas as ligações entre os modelos dos componentes assim como os atributos de cada componente. Tentou-se reduzir o número de componentes com valor diferente dentro de uma família. Por exemplo, se são usadas dez resistências de 1K e uma de 1K2, deve-se averiguar a possibilidade de substituição da resistência de 1K2 por uma de um 1K de modo a diminuir o número de componentes dispares a utilizar. Verificou-se ainda se o encapsulamento de componentes iguais utilizados no esquema era o mesmo; por exemplo, não é conveniente possuir cinco resistências com o encapsulamento 1206 e cinco com o 0603.

3.6 Desenho do PCB

Esta tarefa foi desempenhada recorrendo ao uso do *software* de desenho *PowerPCB* 5.0.1 da *Mentor Graphics*.

Foi necessário começar por atribuir e desenhar os encapsulamentos dos novos modelos de componentes criados anteriormente no desenho do esquemático de acordo com as medidas fornecidas pelos respectivos fabricantes.

Dispondo de todos os componentes e conhecidas as dimensões da placa, foi realizado o posicionamento dos componentes tendo em especial atenção as conexões entre eles, o acesso exterior que alguns necessitam possuir (por exemplo conectores) e o alinhamento dependente do método de soldadura que se prevê utilizar para montagem das placas.

Nesta fase existe um trabalho que muitas vezes é ignorado por aqueles que observam o produto final. Trata-se da configuração da espessura de pistas, vias, etc., ou seja a especificação de um conjunto de regras importantíssimo para um bom traçado das pistas (*routing*) da placa; aqui como na lista de material, é conveniente verificar se, por exemplo, estamos a usar a maioria das vias com uma dimensão e uma ou duas com uma dimensão distinta sem necessidade, o que para além de indiciar uma menor qualidade do trabalho, aumenta o custo de produção das placas.

3.7 Verificações do PCB

Esta é a fase mais meticulosa das verificações. É importante verificar se os encapsulamentos atribuídos aos componentes são aqueles que o fabricante disponibiliza com maior facilidade, se a equivalência está correcta entre a ordem dos pinos do modelo para o desenho, se as referências estão colocadas juntos ao respectivo componente e de forma visível, se as inscrições realizadas não possuem erros, se a colocação dos componentes é a melhor tendo em atenção o tipo de montagem e método de soldadura a realizar.

Quando todas as verificações foram feitas, ou pelo menos as possíveis, é altura de proceder à produção do *PCB*. Chegada esta altura foi feito um pedido de cotação a diversos fabricantes e escolhido aquele que melhores condições proporcionou quer a nível de custos de protótipo quer a nível de custos de reprodução.

3.8 Montagem do PCB

Após a existência de placas e os respectivos componentes, chegou a fase da montagem, sendo agora possível fazer a verificação da placa com mais exactidão. Caso exista possibilidade, como foi o caso, é aconselhável a montagem e soldadura manual do primeiro protótipo pelo projectista de modo a este conseguir detectar alguma anomalia não prevista o mais cedo possível. Devido à crescente redução do tamanho dos componentes torna-se cada vez mais difícil e até em alguns casos impossível a montagem da placa pelo projectista, devendo neste caso ser realizada uma verificação exaustiva da placa após montagem automática desta, de modo a minimizar tempo de projecto do *software* em que quando surge um problema se podem levantar suspeitas do hardware e do *software*. Quando se possui um hardware verificado minuciosamente e com baixas probabilidades de erros, poupa-se bastante tempo no desenvolvimento do *software*.

Na Figura 25 é possível observar o resultado da montagem da placa protótipo do terminal *BlueTerm*.

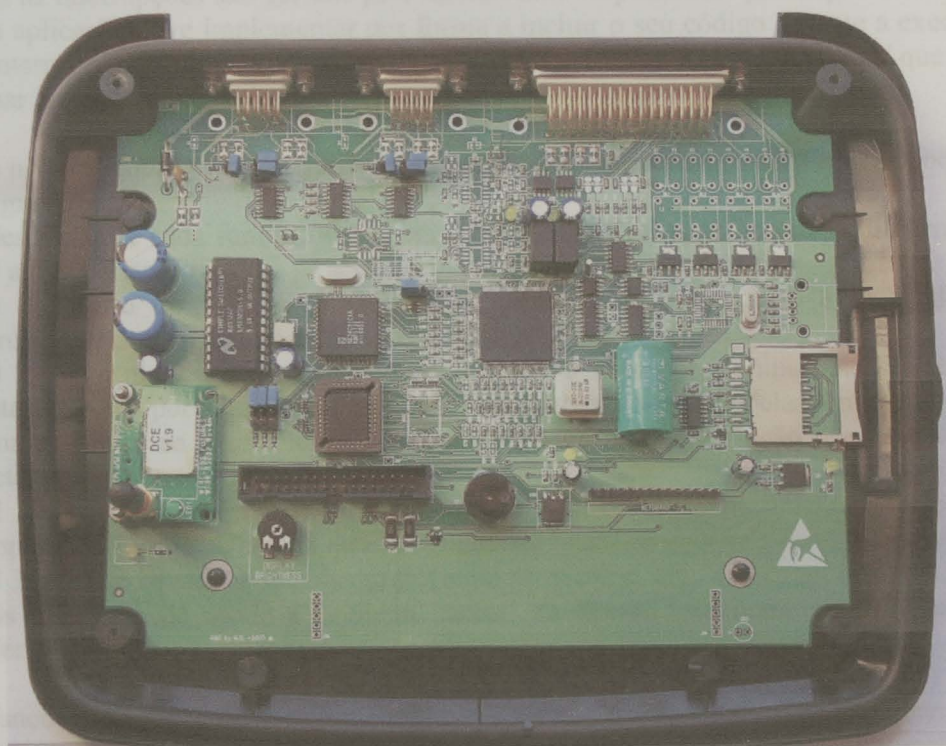


Figura 25 - Vista geral do hardware desenvolvido

3.9 Testes eléctricos

Após montagem da placa foram realizados testes preliminares para a verificação do comportamento eléctrico e funcional do circuito.

Antes de alimentar a placa foi feita uma verificação da existência de curtos-circuitos, principalmente ao nível da alimentação, dado que estes poderiam desde logo causar a destruição parcial ou total de componentes ou da própria placa. Feita essa verificação a placa foi alimentada e visualizados e medidos sinais em diversas partes pré-seleccionadas como a alimentação, as saídas de osciladores e/ou cristais e os barramentos de dados, endereços e controlo do microprocessador.

4 Projecto de software

No projecto de *software* foram usados o editor de texto *µvision2 V2.30* e o compilador para *C 166 V4.22* da *keil Software Inc.*

O *software* foi dividido em dois níveis: o *kernel* e a aplicação. O *kernel* é constituído por código de baixo nível, tais como *drivers*, muitas vezes comum a outro tipo de módulos. A aplicação assenta sobre o *kernel* e contém o código necessário ao funcionamento do terminal.

Todas as interrupções são geridas pelo *kernel*. Este disponibiliza protótipos de funções que a aplicação deve implementar por forma a incluir o seu código durante a execução das interrupções. A aplicação por sua vez, dispõe de várias funções do *kernel* que pode chamar em qualquer altura.

Cada nível de código está dividido em vários módulos de *software*, tal como se pode ver na Figura 26.

Na descrição de cada função dos diversos módulos é realizado um enquadramento do nível a que a função se situa na seguinte hierarquia:

nível 0: acesso directo aos periféricos - normalmente este acesso é realizado em interrupções do microprocessador ou nas iniciações das unidades.

nível 1: neste nível o acesso aos periféricos não é feito por manipulação directa dos registos mas sim por uso de uma estrutura mais elaborada (criada pelas funções do nível 0) por exemplo, na escrita/leitura da memória *eeeprom*, uma função que escreve um carácter pertence a este nível.

nível 2: as funções deste nível usam as funções do nível imediatamente inferior, por exemplo uma função de escrita de *n* caracteres na memória *eeeprom*.

nível 3: aqui se situam as funções gestoras de protocolos, etc., que usam as funções dos níveis inferiores de modo a implementarem um acesso totalmente transparente aos periféricos.

As funções que são utilizadas por mais que um processo foram alvo de implementação de protecção de acesso por um mecanismo de semáforos. Por exemplo, as funções de comunicação *iic* são utilizadas pelas funções de escrita e leitura na *eeeprom* e no *rtc*, logo antes de se aceder ao *iic* é necessário verificar se o recurso está livre e em caso afirmativo este passa logo para o estado de ocupado de modo a mais nenhum processo aceder ao recurso até que o processo que o ocupa o liberte.

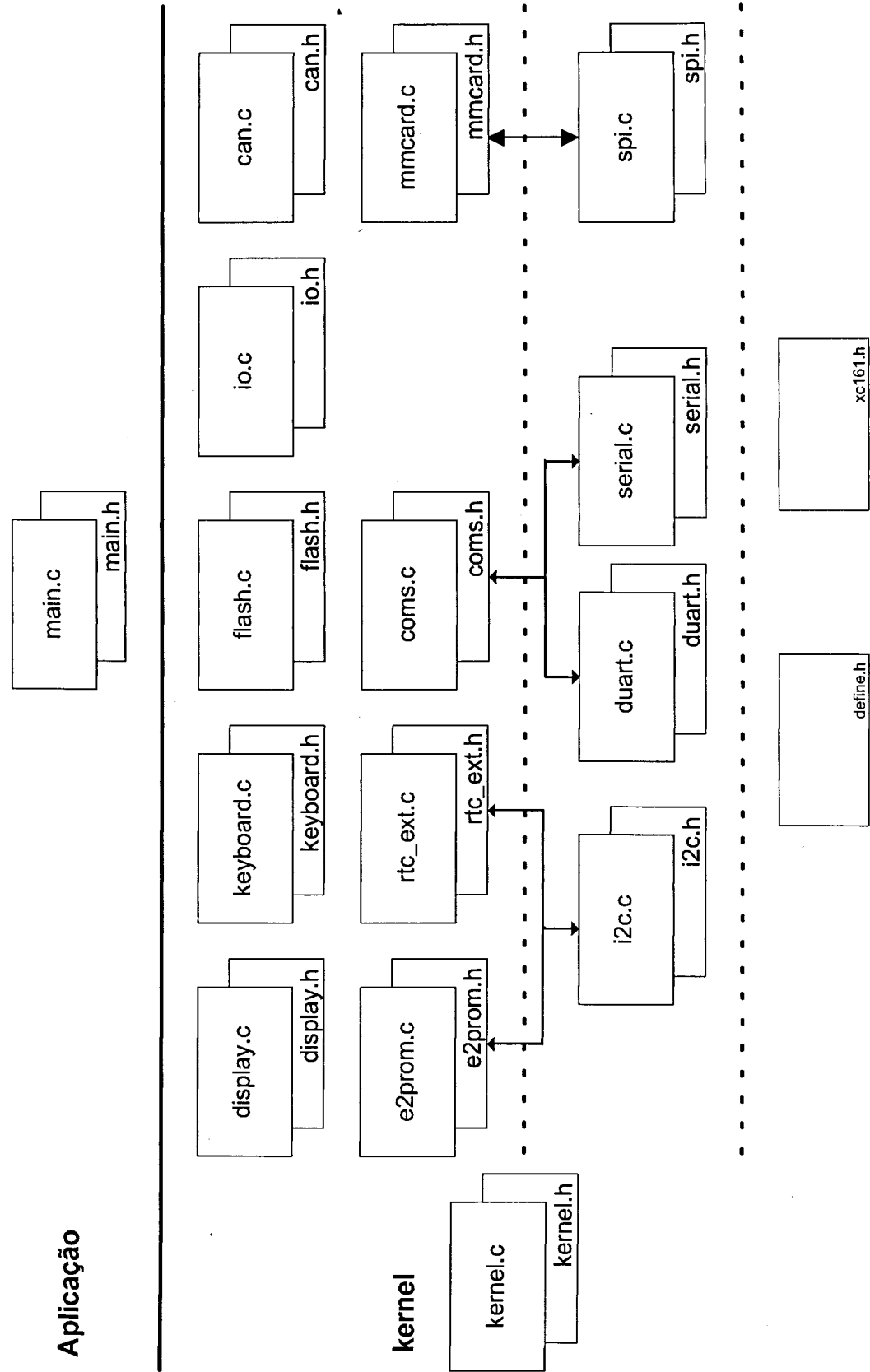


Figura 26 - Módulos de software

4.1 Módulo “can”

Este módulo contém as funções para acesso aos canais de comunicação *CAN*. Existem dois canais *CAN_A* e *CAN_B*. Neste módulo foram apenas implementadas funções básicas para recepção e envio de caracteres. Faz parte do trabalho futuro a estruturação destas funções e a implementação de um acesso transparente a esta interface.

4.2 Módulo “coms”

Este módulo contém as funções para acesso aos canais de comunicação série *RS232/RS422/RS485*. Existem quatro canais: as duas portas série do microprocessador (*ASC0* e *ASC1*) e as duas portas adicionais da *DUART* (*DUART_A* e *DUART_B*). As portas série de base são para uso genérico enquanto que a porta A da *DUART* está dedicada às comunicações com o módulo de *Bluetooth* e a porta B às comunicações com o módulo de leitura de *transponders*.

Foram criados *buffers* para armazenamento dos caracteres recebidos e a transmitir pelos canais de comunicação.

Os *buffers* são os seguintes:

Recepção - *BYTE bComBufferIn[4][COM_BUFFER_SIZE]*

Transmissão - *BYTE bComBufferOut[4][COM_BUFFER_SIZE]*

Foram definidos quatro *buffers* de recepção e quatro de transmissão do tipo FIFO sendo um para cada canal série. A dimensão dos *buffers* é definida pela constante *COM_BUFFER_SIZE* e pode ser alterada no ficheiro *coms.h*, tendo em conta a memória disponível.

O programador para usar os canais de comunicação não necessita de aceder directamente aos *buffers* descritos pois foram criadas funções para possibilitar o envio e recepção de dados através dos canais série de modo transparente.

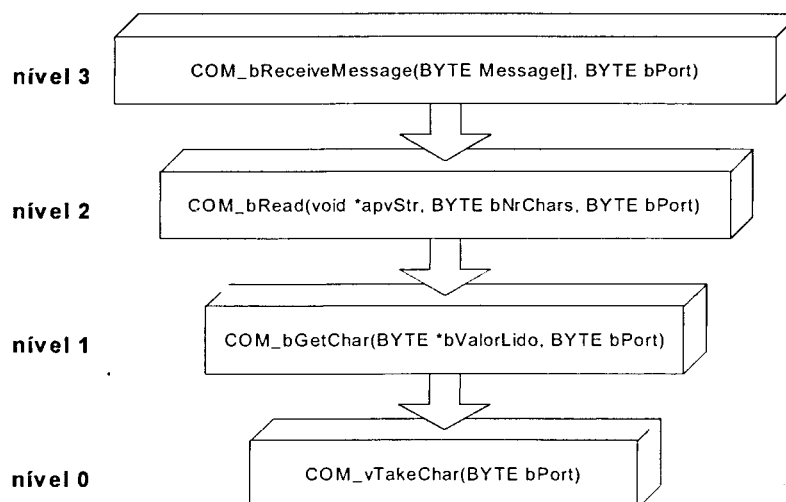


Figura 27 - Funções de recepção do módulo "coms"

Existem várias funções que implementam a recepção (Figura 27). Quando um carácter é recebido numa das portas série ocorre uma interrupção no microprocessador, aí existe uma função (*COM_vTakeChar*) que recolhe os caracteres recebidos e coloca-os no respectivo *buffer*. Esta função faz parte do “sistema operativo” e não é para uso do programador. Nos níveis 1, 2 e 3 da Figura 27 podemos observar as funções disponíveis para o programador. No nível 1 existe uma função que permite a recepção carácter a carácter, no nível 2 existe uma função que permite a recepção de um conjunto de caracteres e no nível 3 já existe uma função que permite a recepção de uma mensagem segundo um protocolo pré-definido (ver 7).

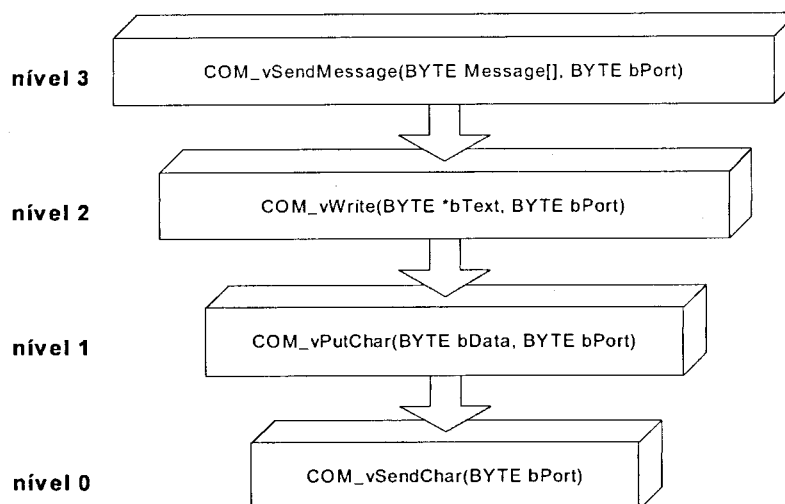


Figura 28 - Funções de transmissão do módulo "coms"

Existem várias funções que implementam a transmissão (Figura 28). Quando se envia um carácter por uma das portas série ocorre uma interrupção no microprocessador, aí existe uma função (*COM_vSendChar*) que retira os caracteres a enviar do respectivo *buffer* e envia-os. Esta função faz parte do “sistema operativo” e não é para uso do programador. Nos níveis 1, 2 e 3 da Figura 28 podemos observar as funções disponíveis para o programador. No nível 1 existe uma função para realizar o envio carácter a carácter, no nível 2 existe uma função para realizar o envio de um conjunto de caracteres e no nível 3 já existe uma função para realizar o envio de uma mensagem segundo um protocolo pré-definido (ver 7).

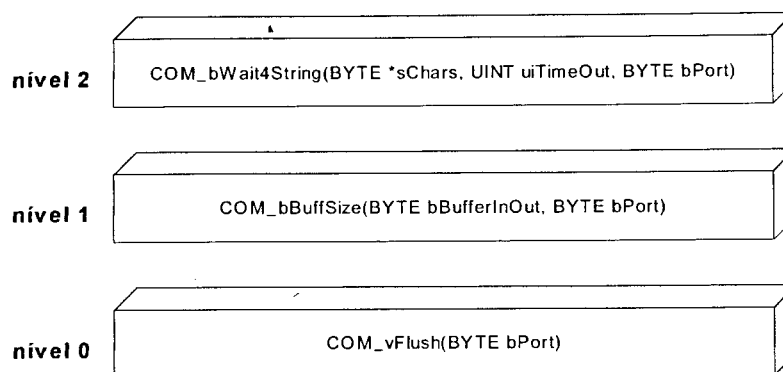


Figura 29 - Funções genéricas do módulo "coms"

Foram implementadas três funções que realizam funções distintas. Foi criada uma função de nível 0 (ver Figura 29) que permite apagar o conteúdo dos *buffers* de recepção e transmissão, uma função de nível 1 que permite determinar a quantidade de caracteres existentes no *buffer* pretendido e por fim uma função de nível 2 que permite a espera da recepção de uma mensagem pretendida.

4.3 Módulo "display"

O visor existente no terminal possui 80 caracteres, distribuídos em 4 linhas de 20 caracteres cada. A tabela *ASCII* dos caracteres a apresentar é *standard*, apresentando os caracteres com códigos de 32 a 127. Os códigos 0 a 7 são reservados a caracteres configurados pelo utilizador, alterando os valores do *array DefinedChars[]*.

Os dados escritos no visor são actualizados à frequência de 4,8 Hz.

No parâmetro *bOptions* das funções *vPrint* pode ser colocado uma combinação de definições bastando para isso colocar o carácter '+' entre elas; por exemplo, se pretendemos escrever no visor um carácter sem intermitência e alinhado à esquerda deveremos colocar `BLINK_OFF+ALIGN_LEFT`.

Para iniciar este periférico é necessário a utilização da função *DISPLAY_vReset*.

A função *DISPLAY_vRefresh* é responsável pela actualização da informação do visor e é chamada numa interrupção de um temporizador com a periodicidade de 26,2 ms.

Para apagar informação foram criadas duas funções, *DISPLAY_vClearScreen* apaga a informação de todas as linhas e *DISPLAY_vClearLine* apaga apenas a linha pretendida.

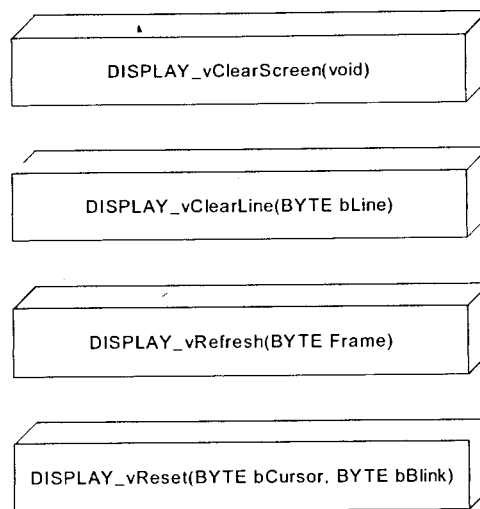


Figura 30 - Funções genéricas do módulo "display"

Para escrita no visor existe um conjunto de funções de forma a permitir a escrita de informação de vários tipos (inteiros, longos, etc.). Existem funções para escrita de caracteres (byte), de inteiros com ou sem sinal, de longos com ou sem sinal, de virgula flutuante (*float*) e de conjuntos de caracteres (*string*). Existe uma função para a escrita da hora, minutos e segundos e outra para escrita da data do relógio de tempo real. Como se pode observar na Figura 31, todas as funções de escrita usam como base `DISPLAY_vPrintString`.

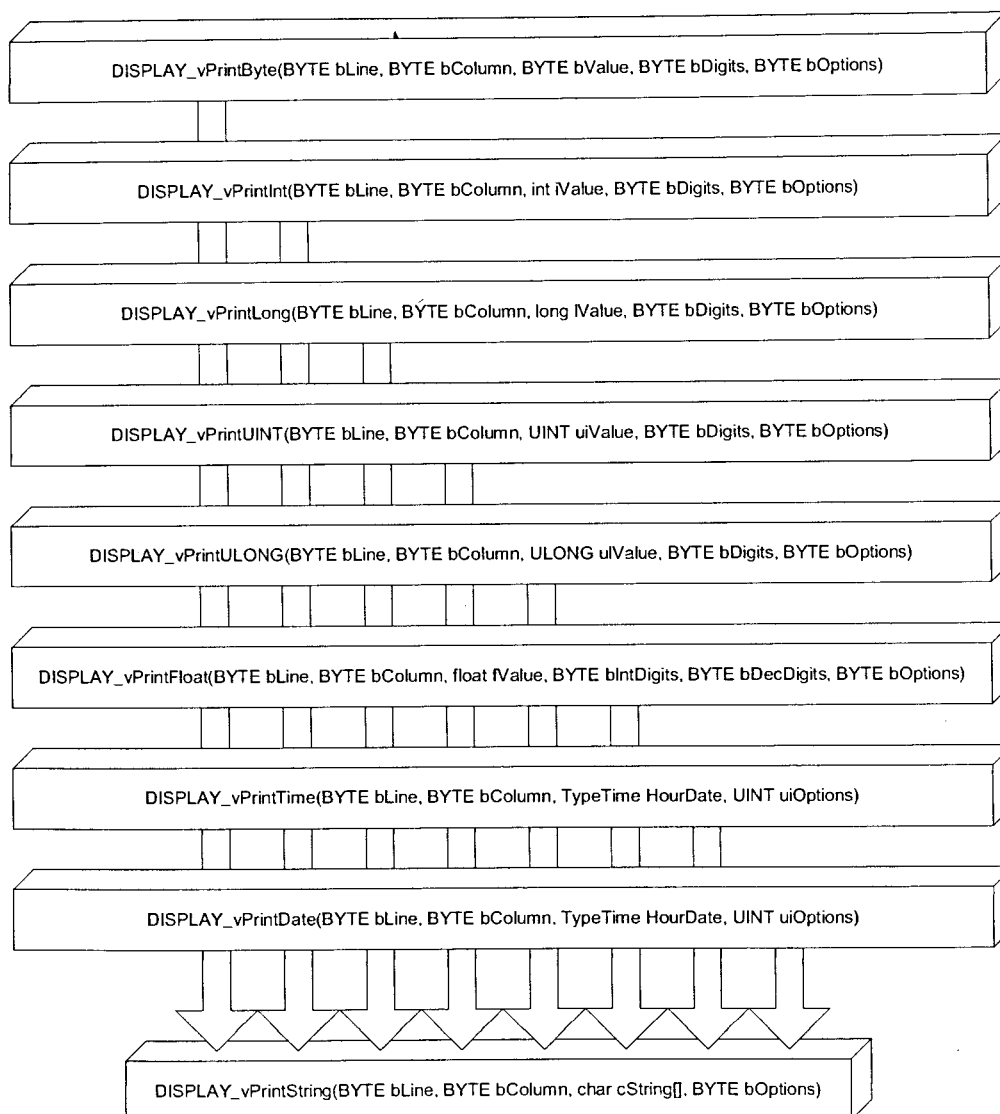


Figura 31 - Funções de escrita do módulo "display"

4.4 Módulo "duart"

A *DUART* (*Dual Universal Asynchronous Receiver/Transmitter*) existente no terminal é suportada por um circuito integrado SCN2681 ou SC26C92 da Philips. Através dos seus parâmetros de configuração esta *DUART* poderá ser programada com os seguintes parâmetros: 7 ou 8 bits de dados; paridade ímpar, par ou sem paridade; um ou dois bits de stop e várias opções de velocidades (*baud rate*).

Este módulo serve de suporte ao "coms" na medida em que o nível mais baixo para acesso aos canais *DUART_A* e *DUART_B* é efectuado aqui. A gestão dos *buffers* de recepção e transmissão dos canais A e B é da responsabilidade deste módulo, ou seja, caso chegue um carácter por uma das portas, este deverá ser transferido para o *buffer* de recepção respectivo. No caso dos *buffers* de transmissão possuírem dados estes deverão ser enviados pelo respectivo canal.

Para a utilização da DUART é necessário a iniciação desta e pode ser efectuada através da utilização da função *DUART_vStart*. Antes de iniciar este dispositivo é conveniente verificar se este foi montado na placa e se está funcional, para isso basta utilizar a função *DUART_vDetection*.

Sempre que é enviado ou recebido um carácter por este dispositivo ocorre uma interrupção (*INT_viEx0*) e nesta é feita a transferência de informação entre *buffers*.

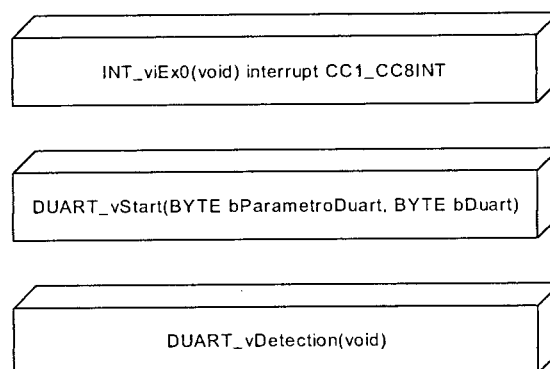


Figura 32 - Funções genéricas do módulo "duart"

4.5 Módulo "e2prom"

O terminal possui uma memória do tipo EEPROM. Este tipo de memórias possui uma interface série com uma linha para o sinal de oscilação (*clock*) e outra linha para os dados (funciona nos dois sentidos); para comunicar através desta interface é necessário recorrer ao protocolo *iic*.

O terminal pode suportar EEPROM de 256 bytes (24C02) até 2048 bytes (24C16). Como referência, utilizam-se as 24CXX da *Microchip* ou da *National Semiconductors*. O tamanho da EEPROM utilizada pelo módulo está definido em *e2prom.h* através de **E2PROM_SIZE**. Esta constante é utilizada pelas funções de escrita e leitura.

Na sua configuração típica, o terminal possui uma EEPROM de 2048 bytes.

Caso se pretenda ler um ou mais caracteres deste dispositivo (Figura 33) deverá se utilizar a função *E2PROM_bRead*, fazendo esta, o acesso ao dispositivo através da função *I2C_bAccess* do módulo "i2c".

Caso se pretenda escrever um ou mais caracteres neste dispositivo (Figura 34) deverá se utilizar a função *E2PROM_bWrite*, fazendo esta, o acesso ao dispositivo através da função *I2C_bAccess* do módulo "i2c".

Visto o *iic* ser um recurso partilhado é necessário o uso de semáforos para assegurar a ausência de conflitos no acesso. Antes da utilização das funções de leitura e escrita é necessário assegurar se o dispositivo se encontra disponível caso contrário deverá se esperar pela obtenção de disponibilidade.

A função *I2C_bEnable* verifica se o dispositivo está ocupado ou livre e neste ultimo caso, comuta o seu estado para ocupado. Desta forma, se primeiramente for chamada a função *I2C_bEnable* e o seu retorno for igual a verdadeiro (*true*), poderá se utilizar as

funções de leitura ou escrita, caso contrário deverá se esperar a obtenção da resposta igual a verdadeiro. Após a chamada das funções de escrita ou leitura deverá ser invocada a função *I2C_vDisable* de modo a libertar o recurso para os outros processos, caso contrário jamais será possível efectuar o acesso aos dispositivos iic.

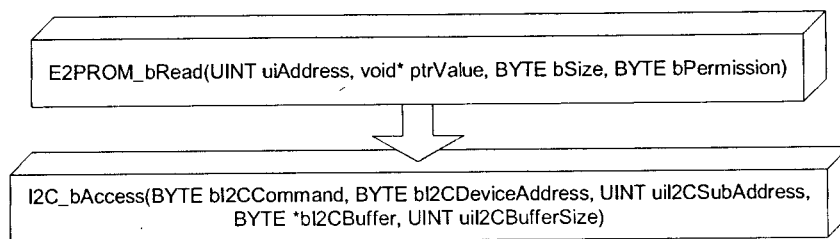


Figura 33 - Função de leitura do módulo "e2prom"

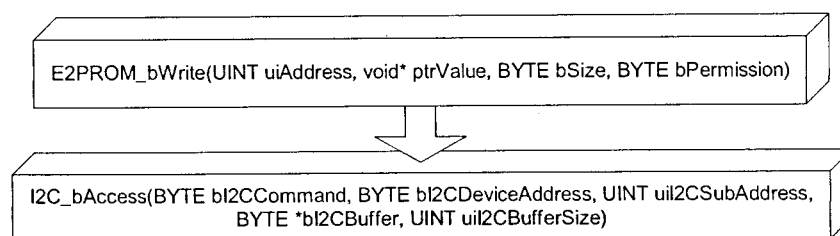


Figura 34 - Função de escrita do módulo "e2prom"

4.6 Módulo "flash"

O terminal possui uma memória do tipo *flash* com 128 Kb (128K x 8 bits) de capacidade. Este tipo de memória tem a vantagem de reter a informação após o corte de alimentação sem necessidade de recurso a bateria. Poderá ser usada para armazenamento de grande quantidade de informação e neste módulo são fornecidas funções para a realização de escrita e leitura, ficando a realização de um conjunto de função de nível mais elevado para trabalho futuro. A *flash* está organizada em sectores de 256 bytes cada.

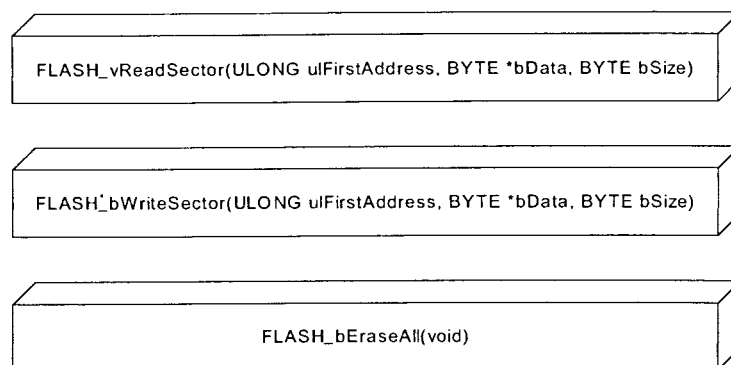


Figura 35 - Funções genéricas do módulo "flash"

4.7 Módulo “i2c”

Este módulo serve de suporte aos módulos “e2prom” e “rtc_ext” fornecendo as funções de baixo nível para acesso à interface iic.

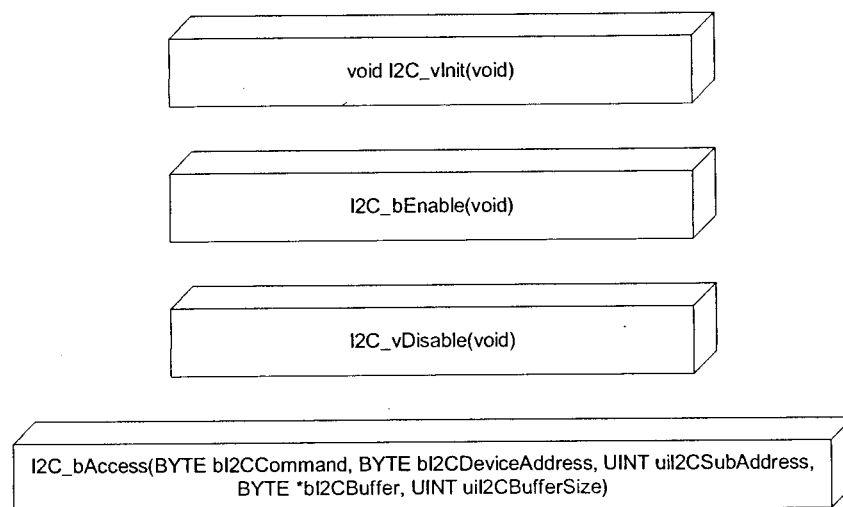


Figura 36 - Funções genéricas do módulo “i2c”

Antes de ser utilizado qualquer dispositivo iic (*eeeprom* ou *rtc*) é necessário iniciar o controlador de iic interno ao microprocessador. A iniciação é feita através da chamada da função *I2C_vInit* na rotina *KERNEL_vStart* do módulo “*kernel*”. Para aceder ao barramento propriamente dito, existe uma função de nível 0, *I2C_bAccess*, que permite às funções de níveis superiores dos módulos “*e2prom*” e “*rtc_ext*”, o acesso para escrita ou leitura em dispositivos iic.

4.8 Módulo “io”

Existem oito entradas e quatro saídas no terminal. As entradas podem ser analógicas ou digitais. As saídas são por relé ou colector aberto.

O ADC (Analog Digital Converter) usado nas entradas analógicas permite uma resolução de 10 bits nos 8 canais implementados. O valor lido está portanto compreendido entre 0 e 1023. Cada canal é lido com uma frequência de amostragem de 4,8 Hz (208 ms). O valor lido do ADC é filtrado por um filtro de alisamento exponencial, sendo aplicado este algoritmo em cada amostra recolhida.

$$V_n = \alpha \times V_{I_n} + (1 - \alpha) \times V_{(n-1)} \xrightarrow{\beta = 64\alpha} V_n = \frac{\beta \times V_{I_n} + (64 - \beta) \times V_{(n-1)}}{64}$$

Equação 1 – Filtro aplicado às entradas analógicas

$V_{(n)}$: Valor filtrado acessível pelo utilizador

$V_{I(n)}$: Valor lido directamente do ADC

α : Constante que determina a frequência de corte do filtro (e respectiva velocidade de resposta)

Por facilidade de cálculo, este filtro foi implementado conforme a segunda formula, em que β varia de 0 a 64:

O valor de β é definido para cada entrada, na variável **bADCFilter[Canal]**, do ficheiro **IO.C** e pode ser alterado pelo programador.

Entradas/Saídas Digitais:

As entradas caracterizadas como digitais são completamente configuráveis (por hardware) relativamente aos valores de tensão na entrada que são considerados '1' ou '0'.

As saídas por relé utilizam contactos normalmente aberto (NA) e normalmente fechado (NF).

Existe uma função para configurar o comportamento das entradas como digitais ou analógicas denominada *Input_bBehaviourConfig*.

Se utilizarmos alguma entrada como analógica é necessário a iniciação do conversor analógico para digital nas iniciações da aplicação, recorrendo à função *ADC_vInit*.

A função *ADC_vKernelRead* é responsável pela leitura para um *buffer* do valor de todas as entradas analógicas e é chamada numa interrupção de um temporizador com a periodicidade de 26,2 ms; a cada interrupção é actualizado o valor de uma entrada sendo este um processo sequencial. O programador deve utilizar a função *ADC_uiRead* para obter o valor analógico das entradas, sendo necessário a obtenção de permissão de acesso a este processo através do uso da rotina *ADC_bEnable*. Aqui existe uma gestão de acessos por semáforo idêntica à existente no acesso ao iic atrás descrito.

Para leitura do estado das entradas digitais existe a função *InputDigital_bRead* e para escrita nas saídas digitais a *OutputDigital_bWrite*.

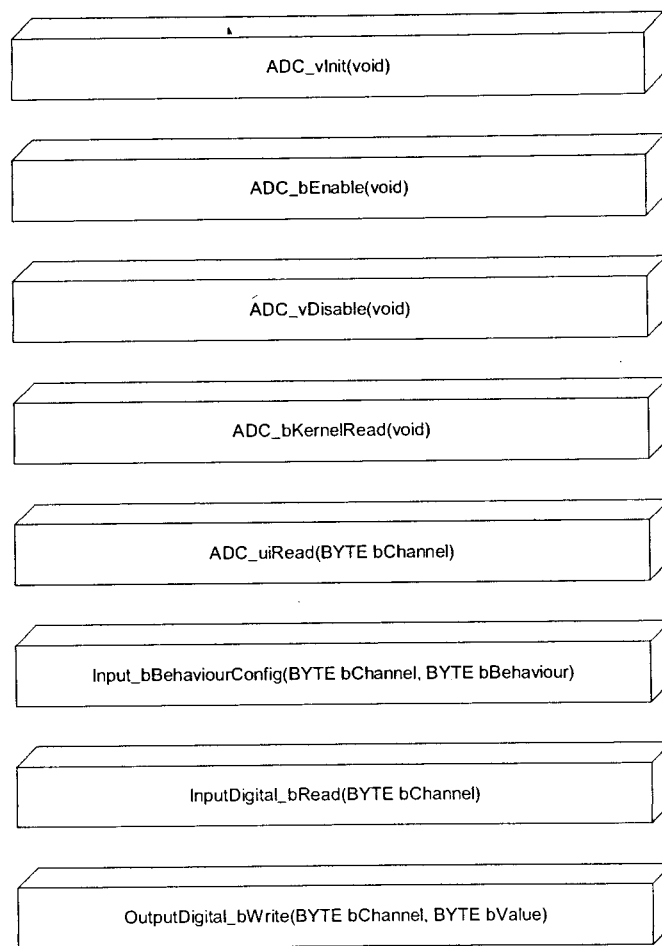


Figura 37 - Funções genéricas do módulo "io"

4.9 Módulo "kernel"

Este módulo contém funções de baixo nível mais genéricas que não se enquadram nos outros módulos, como por exemplo a iniciação de todos os recursos, a gestão periódica de processos, etc.

Na função `KERNEL_vStart` são invocadas todas as funções de iniciação dos vários periféricos e potencialidades do terminal. A realização dessas iniciações está condicionada às definições activas no ficheiro *define.h*. Por exemplo, se não for utilizada pela aplicação o relógio de tempo real dever-se-á colocar em comentário a inclusão deste existente no ficheiro *define.h*.

Existe neste módulo uma função para activação do bescouro por um período de tempo pré-definido (*vBeep*), uma para provocar uma pausa em posições do programa pretendidas pelo programador (*vDelay*).

A função *vError* é utilizada por várias funções ao nível do *kernel* com o objectivo de recolher os possíveis erros ocorridos durante o processamento.

Neste módulo existe ainda uma função (*GPT1_viTmr3*), que é invocada com a periodicidade de 26.2 ms em resultado da programação de um temporizador, onde são

invocados processos como por exemplo o de refrescamento da informação do visor, a leitura do relógio de tempo real, entre outros.

A função *GPT1_viTmr2* é invocada com a periodicidade de 1,5 ms em resultado da programação de um temporizador, e serve de base à função *vDelay* para criação de atrasos temporais o mais aproximado possível aos pretendidos.

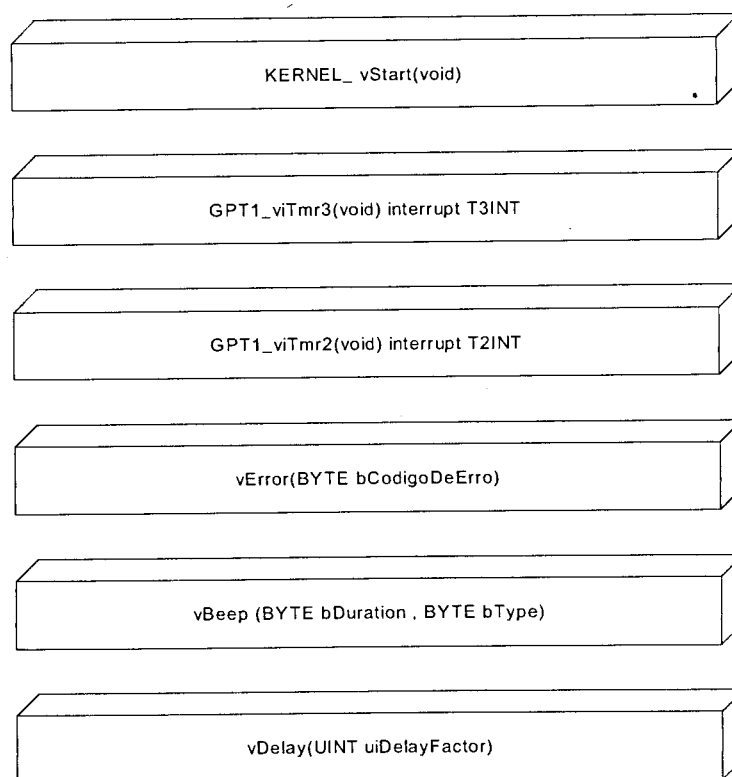


Figura 38 - Funções genéricas do módulo "kernel"

4.10 Módulo "keyboard"

O terminal possui um teclado em membrana com vinte teclas e dois indicadores luminosos (*leds*) - ao estado do indicador vermelho corresponde ao estado da tecla AZ, ao estado do indicador verde corresponde ao estado da tecla FN. Na Tabela 3 pode-se observar qual ou quais os valores devolvidos pelas teclas 0-9 em função do estado das teclas AZ e FN. Desta forma com um teclado de apenas vinte teclas é possível obter um número mais alargado de opções.

Tabela 3 - Valores devolvidos pelas teclas 0-9 em função do estado das teclas AZ e FN

AZ	FN	0	1	2	3	4	5	6	7	8	9
inactiva	inactiva	0	1	2	3	4	5	6	7	8	9
inactiva	activa	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
activa	activa	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20
activa	inactiva	0	1+/-x	2abc	3def	4ghi	5jkl	6mno	7pqrs	8tuv	9wxyz

O varrimento do teclado e respectiva filtragem é efectuado periodicamente à semelhança do refrescamento da informação do visor, ou seja, na função *GPT1_yiTmr3* do modulo “*kernel*”, é invocada a rotina *KEYBOARD_ulRead* onde é realizado o varrimento a todas as teclas e a respectiva filtragem. Como resultado do varrimento existe um longo sem sinal em que a cada bit corresponde o estado de uma tecla. A função *KEYBOARD_bHit* realizada a descodificação do longo sem sinal entrando em consideração com o estado das teclas AZ e FN; o resultado desta função é a tecla premida ou o valor 255 em caso de não existência de tecla(s) premidas. A função *KEYBOARD_bGetKey* aguardar o premir de uma tecla e tem como base a *KEYBOARD_bHit*.

A função *KEYBOARD_vWaitRelease* tem como função a espera de libertação de qualquer tecla que esteja premida.

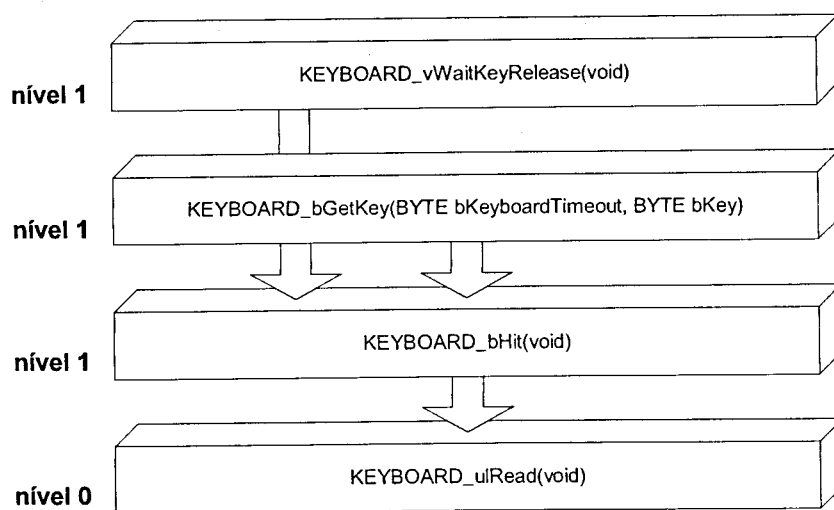


Figura 39 - Funções genéricas do módulo "keyboard"

Para leitura do teclado existe um conjunto de funções de forma a permitir a leitura de informação de vários tipos (inteiros, longos, etc.). Existem funções para leitura de caracteres (byte), de inteiros sem sinal, de longos sem sinal e de conjuntos de caracteres (*string*). Como se pode observar na Figura 40, todas as funções de leitura usam como base *KEYBOARD_bReadString*. Todas estas funções aguardam a introdução do número de dígitos pretendidos e retornam após o premir da tecla de escape (ESC) ou da tecla de confirmação (OK).

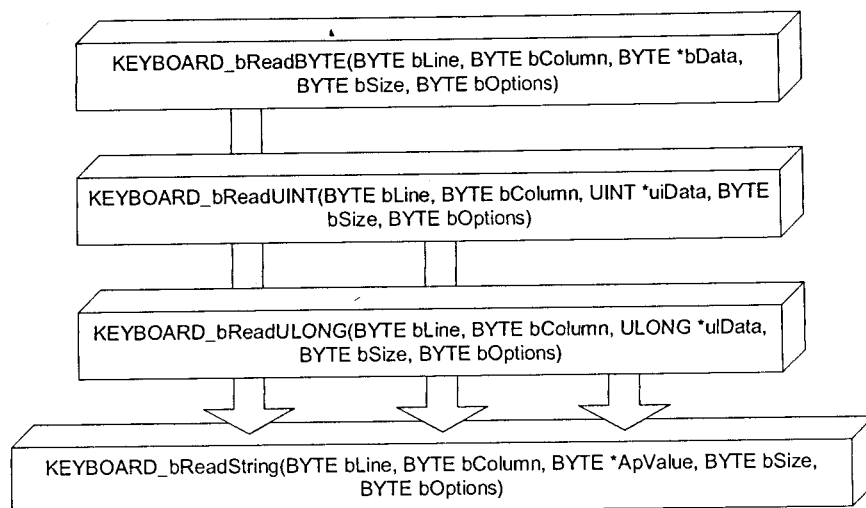


Figura 40 - Funções de leitura do módulo "keyboard"

4.11 Módulo "mmccard"

No terminal existe a possibilidade de uso de um cartão multimídia (*Multimedia Card*) que se trata de uma memória do tipo *flash*, para recolha ou introdução de informação. Actualmente os *Multimedia Cards* de maior capacidade possuem 64 Mbytes, organizados em sectores de 512 bytes.

Neste módulo está definido um conjunto de funções que permite a iniciação e uso do cartão de forma simplificada. As funções deste módulo sobrepõem ao módulo *spi* para acesso às comunicações.

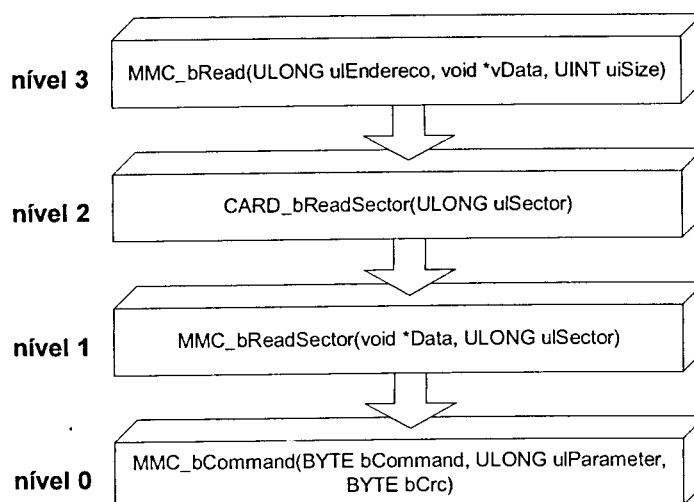


Figura 41 - Funções de leitura do módulo "mmccard"

Foi desenvolvida uma função de nível 0 para envio de comandos para o cartão. Esta função, *MMC_bCommand*, é a base das funções de escrita e leitura no cartão. A função *MMC_bReadSector* permite efectuar a leitura de um sector directamente do cartão enquanto que a *CARD_bReadSector* efectua a leitura do sector indicado para a memória *RAM* de modo a ser criado uma *cache* do cartão. Através do uso da função *MMC_bRead* é possível efectuar a leitura de *n* caracteres, pois caso estes não existam em *cache* é realizada a sua copia do cartão para a *RAM*.

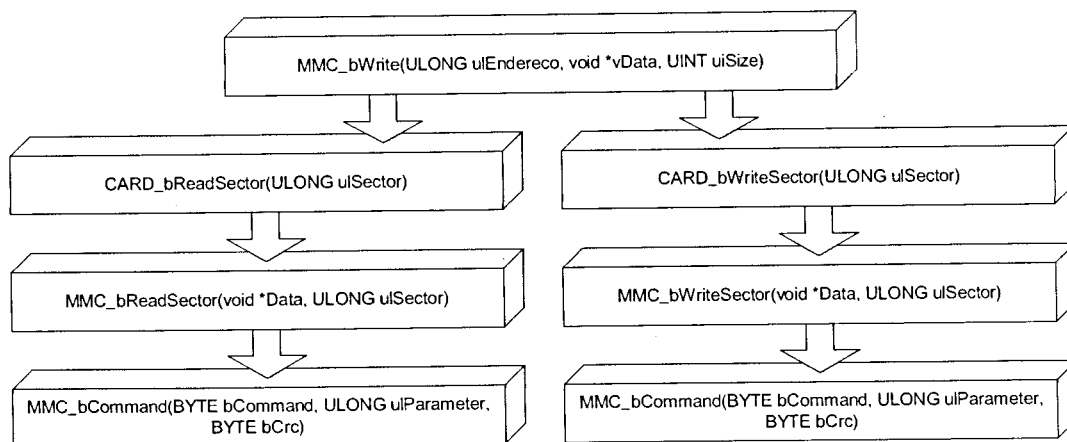


Figura 42 - Funções de escrita do módulo "mmc"

O processo de escrita no cartão de memória até ao nível 2 é semelhante ao de leitura, a função de nível 3, *MMC_bWrite*, necessita utilizar as funções de leitura de modo a conseguir alterar apenas os caracteres pretendidos.

4.12 Módulo "mystring"

Este módulo contém funções de baixo nível para realizar a conversão de variáveis num formato diferente do original; por exemplo converter uma variável do tipo carácter (*byte*) numa *string* - *btoa*. As funções *uito* e *ultoa* são idênticas à *btoa* diferindo apenas o tipo de carácter a converter para *string* - inteiro sem sinal e longo sem sinal, respectivamente. A função *ZeroClean* permite colocar ou não zeros à esquerda do número após a conversão para *string*.

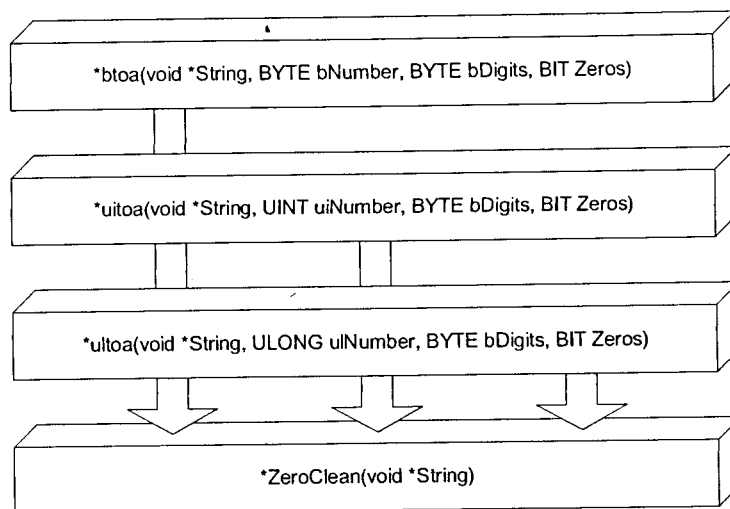


Figura 43 - Funções de conversão de tipos do módulo "mystring"

Neste módulo foram incluídas outras rotinas com funções diversificadas. A função `vAddULONG2HexString` permite adicionar um número do tipo longo sem sinal numa *string*, em formato hexadecimal.

A função `bDec2Bin` faz a conversão de um número de decimal para binário e a `bStringIsDigit` permite determinar se uma *string* é apenas constituída por caracteres de 0 a 9.

A função `bStringExtract` permite extrair uma *string* que esteja delimitada por determinada sequência de caracteres.

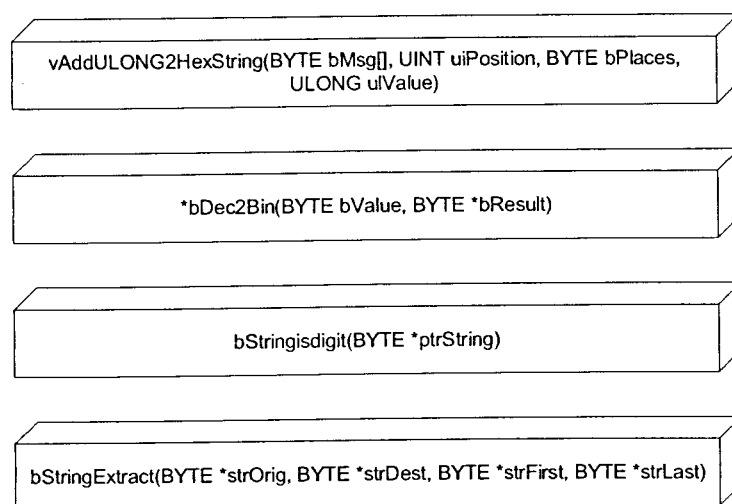


Figura 44 - Funções genéricas do módulo "mystring"

4.13 Módulo "rtc_ext"

O relógio - calendário existente no terminal é suportado por um MK41T56 da SGS-Thomson. Possui 64 bytes, dois quais 8 são utilizados para aceder aos registos do relógio - calendário.

Quando o terminal não utiliza o relógio, o programa pode ser recompilado sem as funções deste, bastando para tal incluir a definição **#define WITHOUT_RTC_EXT** no ficheiro **define.h**.

Quando a aplicação não necessita de usar alguma das funções, no ficheiro **define.h** pode-se retirar o **#define** correspondente de modo a economizar-se espaço de memória de código.

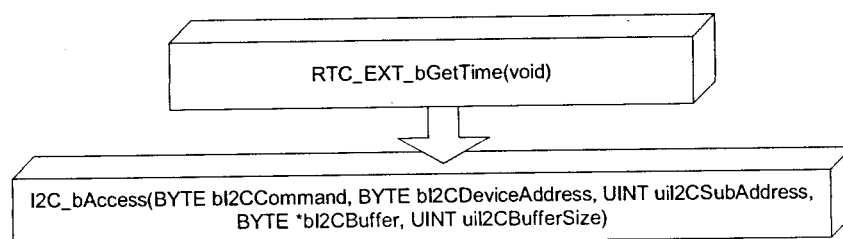


Figura 45 - Funções de leitura do módulo "rtc_ext"

Caso se pretenda ler a data e hora do *rtc* (Figura 45) deverá se utilizar a função **RTC_EXT_bGetTime**, fazendo esta o acesso ao dispositivo através da função **I2C_bAccess** do módulo "*i2c*".

Caso se pretenda acertar o *rtc* (Figura 46) deverá se utilizar a função **RTC_EXT_bPutTime**, fazendo esta o acesso ao dispositivo através da função **I2C_bAccess** do módulo "*i2c*".

Visto o iic ser um recurso partilhado é necessário o uso de semáforos para assegurar a ausência de conflitos no acesso. Antes da utilização das funções de leitura e escrita é necessário assegurar se o dispositivo se encontra disponível caso contrário deverá se esperar pela obtenção de disponibilidade.

A função **I2C_bEnable** verifica se o dispositivo está ocupado ou livre e neste ultimo caso, comuta o seu estado para ocupado. Desta forma, se primeiramente for chamada a função **I2C_bEnable** e o seu retorno for igual a verdadeiro (*true*), poderá se utilizar as funções de leitura ou escrita, caso contrário deverá se esperar a obtenção da resposta igual a verdadeiro. Após a chamada das funções de escrita ou leitura deverá ser invocada a função **I2C_vDisable** de modo a libertar o recurso para os outros processos, caso contrário jamais será possível efectuar o acesso aos dispositivos iic.

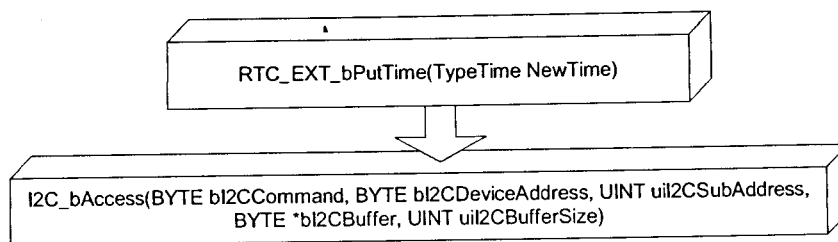


Figura 46 - Funções de escrita do módulo "rtc_ext"

Foram criadas duas funções para facilitar o desenvolvimento de aplicações que utilizem este dispositivo. A função *RTC_EXT_vAdjust* com recurso ao visor e teclado do terminal, permite a realização do acerto do *rtc* por parte do utilizador. A função *bCalcWeekDay* calcula o dia da semana (Segunda, Terça,...., Domingo) em função da data.

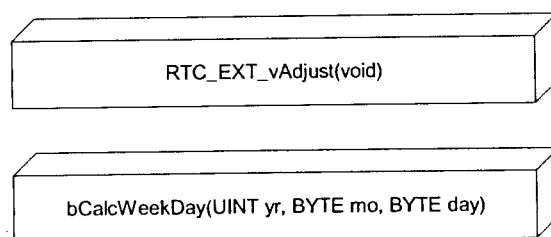


Figura 47 - Funções genéricas do módulo "rtc_ext"

4.14 Módulo "serial"

Este módulo contém as funções de baixo nível para a realização de comunicações através das duas portas de comunicação assíncronas incluídas no processador do terminal.

As funções *ASC0_vInit* e *ASC1_vInit* servem para realizar a iniciação das portas com a velocidade pretendida; por defeito o número de bits de dados é de oito, o número de stop bits é um e a comunicação é efectuada sem paridade.

As rotinas *ASC0_Receive* e *ASC1_Receive* são as funções de serviço às interrupções de recepção às portas 0 e 1 respectivamente.

As rotinas *ASC0_Transmit* e *ASC1_Transmit* são as funções de serviço às interrupções de transmissão às portas 0 e 1 respectivamente.

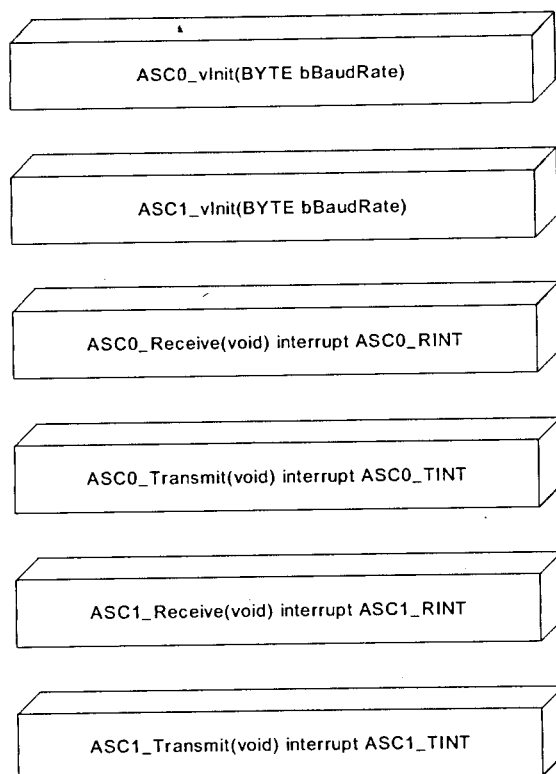


Figura 48 - Funções genéricas do módulo "serial"

4.15 Módulo "spi"

Este módulo serve de suporte ao módulo "mmc card" fornecendo as funções de baixo nível para acesso à interface *spi*.

Antes de ser utilizado o dispositivo *spi* (*mmc card*) é necessário iniciar o controlador de *spi* interno ao microprocessador. A iniciação é feita através da chamada da função *SSC0_vInit* na rotina *KERNEL_vStart* do módulo "kernel". Para aceder ao barramento propriamente dito, existem três funções de nível 0. A função *SPI_bSend* serve para realizar envio de informação e a *SPI_bReceive* para receber. A função *SPI_bDummy* tem como objectivo a realização de espera de leitura de um determinado carácter.

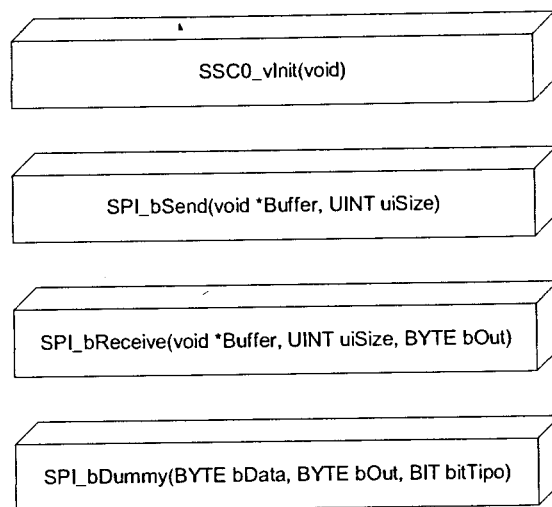


Figura 49 - Funções genéricas do módulo "spi"

5 Programa de teste e configuração

5.1 Especificação

Pretende-se o desenvolvimento de um *software* a ser ligado a todas as aplicações criadas para o *BlueTerm*. As funcionalidades principais desse *software* deverão ser a configuração e o teste dos periféricos do terminal. Estas funcionalidades deverão ser acedidas de forma transparente sem interferir com a aplicação implementada para o terminal. Uma forma de o conseguir será implementando o acesso a essas funcionalidades através do pressionar de uma determinada tecla após a iniciação do terminal.

Resumindo, este *software* deverá possuir duas vertentes:

1. Configuração do terminal - possibilidade de configuração de parâmetros, como por exemplo velocidades das interfaces de comunicação, endereços, acerto do relógio, etc.
2. Teste dos dispositivos (manual ou automático) - permitir realizar um teste aos diversos periféricos do terminal de modo a garantir fiabilidade do produto aquando da sua comercialização.

5.2 Implementação

Foi desenvolvido, de acordo com a especificação, um módulo de *software*, denominado *setup* do *BlueTerm*, que deverá ser compilado e ligado (*link*) com as aplicações que forem desenvolvidas para o terminal.
Este *software* de configuração e teste foi desenvolvido com recurso às funções de baixo nível referidas no capítulo 4.

Para se aceder ao *setup* do *BlueTerm* é necessário premir a tecla ‘.’ imediatamente após a iniciação do terminal. Na Figura 50 demonstra-se o menu inicial que permite o acesso à configuração ou ao teste, em função da opção escolhida. Se nenhuma das teclas, UP ou DOWN, for premida nos dois segundos seguintes, o terminal abandona o *setup*.

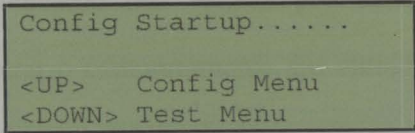


Figura 50 - Menu inicial do programa de setup

Se o utilizador premir a tecla UP, o terminal entra no modo de configuração, menu este que serve para realizar diversas configurações como por exemplo alterar os *baud rates* das portas série e do CAN, acertar o relógio, etc.
A primeira vez que se entra no menu de configuração não é pedida palavra chave, mas daí em diante será necessária a sua introdução, Figura 51, para acesso a este menu. A palavra chave por defeito é “0000”, sendo conveniente que o utilizador efectue a alteração desta o mais breve possível.

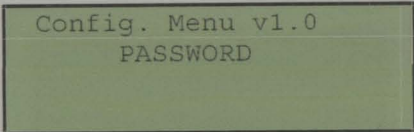


Figura 51 - Menu inicial do programa de configuração

No menu de configuração existem nove opções de escolha conforme de pode ver na Figura 52. Para visualização das várias opções disponíveis o utilizador deverá utilizar as teclas ^ e v para navegação.

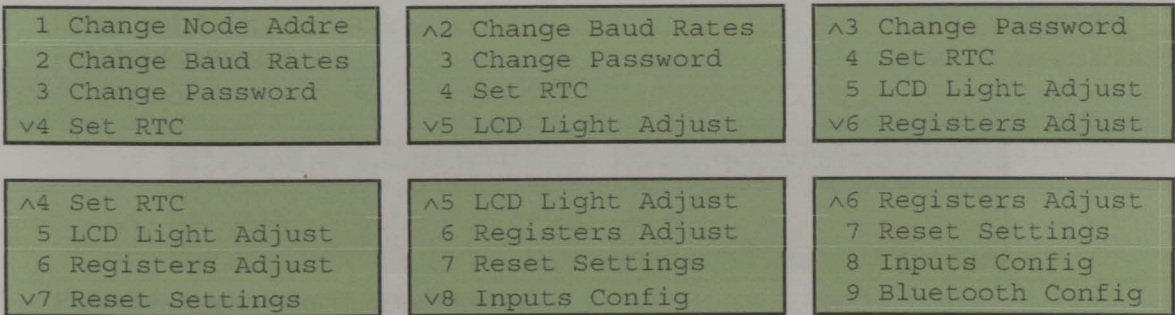


Figura 52 - Opções do menu de configuração

A opção 1 permite a alteração do endereço de CAN pelo qual o terminal é reconhecido em rede.

A opção 2 permite a alteração das velocidades de comunicação das interfaces série e dos canais de CAN.

A opção 3 permite a alteração da palavra chave de acesso do utilizador ao menu de configuração.

A opção 4 permite o acerto do relógio de tempo real.

A opção 5 permite o ajuste do nível de iluminação do visor.

A opção 6 permite configurar registos variados que se pretendam utilizar para diversas funcionalidades. Actualmente foi implementado a configuração de dois registos; um para determinar os periféricos a incluir no teste automático do terminal e outro para definir o comportamento da tecla FN, pois esta tecla poderá ficar sempre activa após a sua ligação, ou desactivar-se após o premir da primeira tecla após a sua activação.

A opção 7 permite a colocação dos valores por defeito nos vários parâmetros configuráveis.

A opção 8 permite configurar o comportamento das entradas do terminal – digital ou analógico.

A opção 9 permite configurar o módulo de *bluetooth*, a nível dos parâmetros da sua ligação série com o terminal, ou seja, velocidade de comunicação, número de bits de dados, etc..

O menu de teste divide-se em automático e manual. Para a realização do teste automático é necessário a ligação de uma caixa de interface ao *BlueTerm*, através do seu conector de cinquenta pinos, preparada para esse efeito. Os periféricos a serem testados nesta opção são escolhidos na opção 6 do menu de configuração.

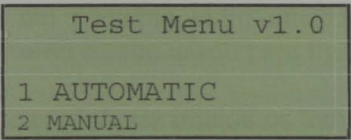


Figura 53 - Menu inicial do programa de teste

O teste manual possui dez opções, divididas em dois menus. No menu 1 está acessível o teste às memórias *flash*, *ram* e *eprom*, ao teclado, ao cartão de memória e ao relógio de tempo real. No menu 2 pode-se aceder ao teste das interfaces série e CAN e às entradas e saídas do terminal.

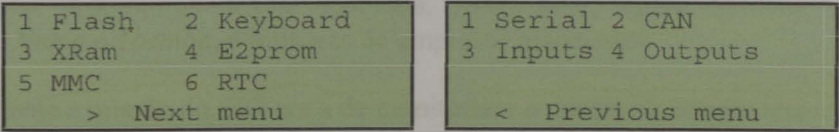


Figura 54 - Opções do menu de teste manual

6 Bluetooth no BlueTerm

6.1 Resumo

A tecnologia sem fios *Bluetooth* engloba *hardware*, *software* e requisitos de interoperabilidade. Foi adoptada não só pelas grandes indústrias das telecomunicações, computadores e entretenimento, mas também em diversas áreas como a indústria automóvel, cuidados de saúde, etc..

É um standard global que:

- ✓ elimina fios e cabos entre as estações e os dispositivos móveis.
- ✓ facilita as comunicações de dados e voz.
- ✓ possibilita o acesso directo a redes e fornece a forma mais recente de sincronismo entre todos os dispositivos pessoais.

6.2 História

Harald Bluetooth foi um *Viking* e rei da Dinamarca entre os anos de 940 e 981. Um dos seus dons era colocar as pessoas a falar entre si, e durante o seu reinado a Dinamarca e a Noruega foram cristianizadas e unidas.

Hoje em dia, a tecnologia sem fios *Bluetooth* permite a dispositivos comunicarem entre si, mas desta vez por intermédio de uma ligação rádio de curta distância e de baixo custo.

A ideia que resultou na tecnologia sem fios *Bluetooth* nasceu em 1994 quando a *Ericsson Mobile Communications* decidiu investigar a possibilidade de criar uma interface de baixo consumo e de baixo custo entre os telefones móveis e os seus acessórios. A ideia era a de um pequeno rádio integrado no telefone portátil e no computador portátil que substituiria o cabo usado para ligar os dois dispositivos.

Um ano mais tarde a equipa de engenharia iniciou os trabalhos e o verdadeiro potencial da tecnologia começou a surgir. Para além de possibilitar a substituição dos cabos, a tecnologia rádio mostrou possibilidades de se tornar uma ponte universal para as actuais redes de dados, uma interface para periféricos, e um mecanismo para criar pequenos grupos de dispositivos privados conectados além das infra-estruturas de rede fixas.

6.3 O SIG

Em Fevereiro de 1998, o *Special Interest Group* (SIG) foi formado. Hoje o SIG inclui as empresas promotoras *3Com*, *Ericsson*, *IBM*, *Intel*, *Agere Systems*, *Microsoft*, *Motorola*, *Nokia* e *Toshiba*, e milhares de empresas associadas.

Originalmente a missão do SIG era a de monitorizar o desenvolvimento técnico do rádio de curto alcance e a criação de um standard global aberto, para prevenir que a tecnologia não se tornasse propriedade de uma única empresa. Deste trabalho resultou a primeira especificação do *Bluetooth* em Julho de 1999. A continuação do desenvolvimento da especificação continua a ser uma das principais tarefas do SIG.

sendo outras igualmente importantes os requisitos de interoperabilidade, a harmonização da banda de frequência e a promoção da tecnologia.

6.4 Interoperabilidade

Desde o início, um dos principais objectivos do SIG foi a inclusão de um sistema de regulação na especificação o que iria garantir a interoperabilidade entre diferentes dispositivos de vários fabricantes - desde que partilhem o mesmo Perfil (*Profile*).

Como referido em [1], enquanto que os modelos usados descrevem aplicações e dispositivos pretendidos, os *Profiles* especificam como usar a pilha (*stack*) do protocolo *Bluetooth* para uma solução interoperável. Cada *Profile* determina como reduzir opções e activar parâmetros no standard base e como usar procedimentos a partir de vários *standard* base. Uma experiência comum de utilização também é definida. Por exemplo, um rato de computador não necessita comunicar com um *headset*, e assim são construídos para cumprir com diferentes *Profiles*.

Os *Profiles* são uma parte da especificação *Bluetooth*, e todos os dispositivos devem ser testados em função de um ou mais *Profiles* de modo a preencher os requisitos de certificação do *Bluetooth*. O número de *Profiles* irá continuar a crescer assim que novas aplicações do *Bluetooth* surjam.

O programa de qualificação do *Bluetooth* garante interoperabilidade global entre dispositivos, qualquer que seja o fornecedor, qualquer que seja o país em que são usados. Durante o procedimento de teste a que todos os dispositivos deverão ser submetidos, terá que ser verificado se eles cumprem todos os requisitos tendo em atenção: a qualidade da ligação rádio, os protocolos dos níveis inferiores, os *profiles* e informação para os utilizadores finais. Todos os dispositivos capacitados estão listados em www.bluetooth.com.

6.5 Especificação

A especificação *Bluetooth* define um pequeno (cerca de 10 metros) ou opcionalmente um médio alcance (cerca de 100 metros) de ligação rádio capaz de transmissão de voz ou dados até um máximo de 720 Kb/s por canal.

A frequência de operação está localizada na banda industrial, científica e médica (*Industrial Scientific and Medical* - ISM) não licenciada de 2.40 a 2.48 GHz, usando *spread spectrum*, *frequency hopping*, *full-duplex* sinal à taxa nominal de 1600 hops/sec. O sinal salta (*hops*) entre 79 frequências com 1 MHz de intervalo de modo a possuir um elevado grau de imunidade à interferência. A frequência rádio é especificada como 0 dBm (1 mW) na versão de 10 metros de alcance e -30 a +20 dBm (100 mW) na versão de maior alcance.

Quando foi elaborada a especificação rádio, foi dado grande relevância aos aspectos que possibilitam custos baixos, mínimo consumo e pequeno tamanho do componente necessário para a implementação em dispositivos móveis.

Voz

Até três canais simultâneos síncronos de voz são usados, ou um canal que simultaneamente suporta dados assíncronos e voz síncrona. Cada canal de voz suporta um canal de 64 Kb/s síncronos (voz) em cada sentido.

Dados

O canal assíncrono de dados pode suportar no máximo 723.2 Kb/s assimétrico (e ainda até 57.6 Kb/s em sentido contrário) ou 433.9 Kb/s simétricos.

- um *master* pode partilhar um canal assíncrono com até 7 *slaves* activos simultâneos num *piconet*.
- trocando *slaves* activos e estacionários dentro e fora de um *piconet*, podem ser virtualmente conectados 255 *slaves* (um dispositivo pode intervir com um intervalo de 2 ms). não existe um limite teórico no número de *slaves* que podem ficar estacionários.

Os *slaves* podem participar em diferentes *piconets* e o *master* de um *piconet* pode ser *slave* noutra, resultando naquilo que é conhecido como *Scatternet*. Até 10 *piconets* dentro do mesmo alcance podem formar uma *scatternet*.

6.6 Topologia de rede

Unidades *Bluetooth* que se podem alcançar mutuamente podem estabelecer ligações ponto-a-ponto e/ou ponto a multi-ponto. As unidades podem dinamicamente ser adicionadas ou desligadas da rede. Duas ou mais unidades *Bluetooth* que partilham um canal formam um *piconet*.

Podem-se construir vários *piconets* ligados entre si em *scatternets* de modo a permitir a comunicação e troca de dados em configurações flexíveis. Se vários outros *piconets* estão em alcance eles podem trabalhar independentemente e cada um tem acesso a uma largura de banda completa. Cada *piconet* é construído por um esquema de frequências de salto (*hopping*) diferente. Todos os utilizadores participando no mesmo *piconet* estão sincronizados com este esquema/padrão de salto (*hopping*). Ao contrário dos infravermelhos, as unidades *Bluetooth* não estão limitadas a comunicação em linha de visão.

Para controlar o tráfego em cada esquema de salto dos *piconets*, uma das unidades participantes torna-se o *master* do *piconet*, enquanto que todas as outras unidades tornam-se *slaves*. Com a actual especificação do *Bluetooth*, até sete *slaves* podem comunicar activamente com um *master*. Contudo, pode haver virtualmente um número ilimitado de unidades ligadas a um *master*, prontas a iniciar a comunicação instantaneamente.

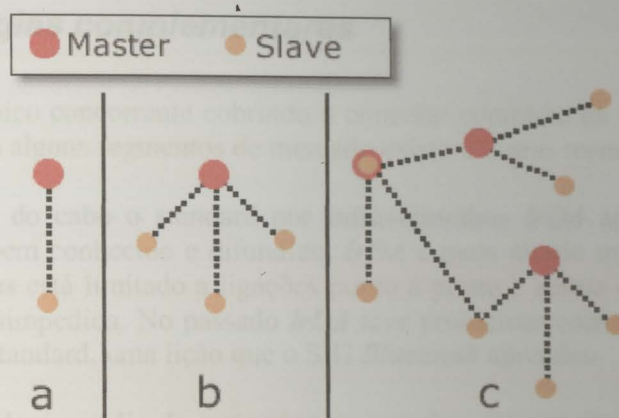


Figura 55 - Piconets com uma única ligação Master-Slave (a), uma ligação multi-slave (b), e uma ligação scatternet (c) (figura retirada de www.ericsson.com)

6.7 Segurança

Dado que os sinais rádio podem facilmente ser interceptados, os dispositivos *Bluetooth* possuem métodos de segurança intrínsecos de modo a prevenir escuta ou falsificação das mensagens (*spoofing*).

As características principais de segurança são:

1. um procedimento de desafio - resposta (*challenge - response*) - para autenticação, que previne escutas e acessos não pretendidos a dados e funções críticas.
2. trama codificada (*stream cipher*) - para encriptação, que previne escutas e mantém a privacidade da ligação.
3. geração de chave de sessão (*session key generation*) - chaves de sessão podem ser geradas a qualquer momento durante a ligação.

Três entidades são usadas nos algoritmos de segurança:

1. O endereço de dispositivo *Bluetooth* (48 bits), é uma entidade pública única para cada dispositivo. O endereço pode ser obtido por um processo de inquirição.
2. Uma chave de utilizador privada (128 bits), é a entidade secreta. A chave privada é produzida durante a inicialização e nunca é revelada.
3. Um número aleatório (128 bits), é diferente para cada nova transacção. O número aleatório é calculado por um processo pseudo-aleatório na unidade de *Bluetooth*.

Em adição a estas funções do nível de ligação, os saltos de frequência e o alcance de transmissão limitado também ajudam à prevenção de escutas.

6.8 Tecnologias complementares

Não existe um único concorrente cobrindo o conceito completo da tecnologia sem fios *Bluetooth* mas em alguns segmentos de mercado existem outras tecnologias.

Para substituição do cabo o standard por infravermelhos *IrDA* andou por perto por alguns anos e é bem conhecido e difundido. *IrDA* é mais rápido que a tecnologia sem fios *Bluetooth* mas está limitado a ligações ponto a ponto e acima de tudo requer uma linha de visão desimpedida. No passado *IrDA* teve problemas com a incompatibilidade entre aplicações standard, uma lição que o SIG *Bluetooth* aprendeu.

Duas outras tecnologias rádio de curto alcance usando a técnica de saltos de frequência residem na banda dos 2.4 GHz:

- Redes de área local (LAN's) sem fios baseadas no standard IEEE 802.11. A tecnologia é usada para substituir a rede de cabo por todo um edifício. A capacidade de transmissão é elevada assim como o numero de utilizadores em simultâneo. Por outro lado, comparado com a tecnologia sem fios *Bluetooth*, é mais cara e de mais alto consumo, e o hardware requer mais espaço. Logo não é apropriada para pequenos dispositivos móveis.
- O outro rádio de 2.4 GHz é *Home RF*, que possui muitas semelhanças com a tecnologia sem fios *Bluetooth*. o *Home RF* pode operar em redes de dados apenas ou estar sobre o controlo de um ponto de ligação coordenando o sistema e fornecendo um *gateway* para a rede de telefone (dados e voz). A frequência de salto é 8 Hz enquanto que na ligação *Bluetooth* o salto é de 1600 Hz.
- *Ultra-Wideband Radio* (UWB) é uma nova tecnologia de rádio ainda em desenvolvimento. Pulsos curtos são transmitidos in a broad frequency range. A capacidade aparenta ser superior enquanto que se espera que o consumo seja mais baixo.

Como é referido em [2], [3] e [4], existem vários estudos com o objectivo de determinar possíveis interferências existentes na coexistência das várias tecnologias sem fios. Este é um estudo ainda não totalmente conclusivo, devido a algumas das tecnologias serem demasiado recentes o que implica a inexistência de uma grande amostra de aplicações com mais que um tipo de tecnologia.

6.9 Pesquisa de controladores de Bluetooth

Esta foi uma das fases mais morosas do projecto devido à pequena oferta existente em módulos que implementem a interface *bluetooth* de uma forma integrada.

A maioria dos dispositivos encontrados para comunicação *bluetooth* possuem pouca integração o que implica a existência de vários componentes para a sua implementação e algum desenvolvimento de software.

Foram encontrados dois fabricantes de módulos que possuem num único encapsulamento o controlador, o rádio e a antena, sendo desnecessário o desenvolvimento de software visto comportarem-se como conversores do protocolo RS232 para o *bluetooth*.

Tabela 4 - Módulos de *bluetooth* integrados

Fabricante	Referência	Alcance	Alimentação	Interface série	Antena	Slaves
connectblue	OEM serial port adapter 13i	10-30 mt	3-6 V	RS232; Baud:300-921.6k	interna	3
connectblue	OEM serial port adapter 13x	10-30 mt	3-6 V	RS232; Baud:300-921.6k	externa	3
connectblue	OEM serial port adapter 33i	100 mt	3.3-6 V	RS232; Baud:300-921.6k	interna	7
connectblue	OEM serial port adapter 33i	100 mt	3.3-6 V	RS232; Baud:300-921.6k	externa	7
wireless futures	BlueWAVE DTE terminal	100 mt	3.3-6 V	RS232; Baud:300-115.2k	interna ou externa	7
wireless futures	BlueWAVE DCE terminal	100 mt	3.3-6 V	RS232; Baud:300-115.2k	interna ou externa	7

A escolha foi para o fabricante *wireless futures* devido à existência de maior apoio técnico destes e da disponibilização de um *dongle* USB para o computador pessoal de modo a se realizar a comunicação entre o computador e o terminal. A nível de custos, as soluções entre os dois fabricantes são equivalentes.

O módulo escolhido foi o BlueWAVE DCE terminal pois para a solução pretendida é o que melhor se adapta.

6.10 Teste funcional do BlueWAVE

Para a realização do teste funcional ao módulo de *bluetooth* foi colocado numa das interfaces USB de um computador portátil o *dongle* para o capacitar com a interface *bluetooth*, visto não existir de raiz esta interface no computador.



Figura 56 - Dongle USB-Bluetooth

Seguidamente foi instalada uma aplicação no computador, fornecida pelos fabricantes dos módulos de *bluetooth*, que cria uma porta série virtual no computador para envio e recepção de informação. Abriu-se a aplicação *Hyperterminal* do *Windows* e configurou-se a porta série virtual criada, com oito bits de dados, um stop bit, sem bit de paridade, sem controlo de fluxo e com uma velocidade de 115200 bps.

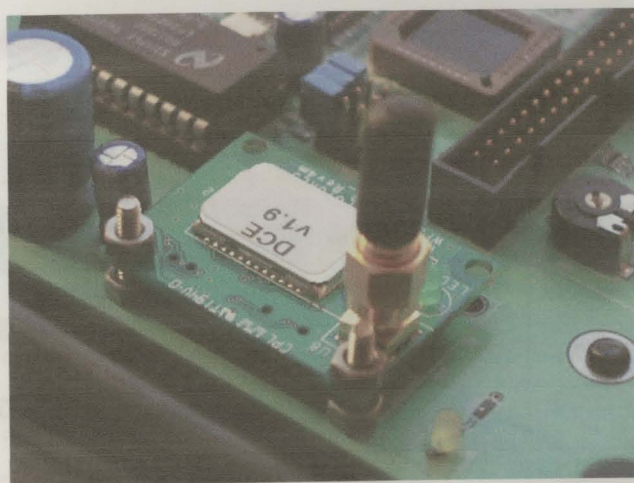


Figura 57 - Módulo RS232-Bluetooth

No *BlueTerm* foi inserido o módulo de *bluetooth* no hardware e desenvolveu-se uma função para envio permanente de dados através do módulo de *bluetooth*. Com a comunicação iniciada entre o computador portátil e o *BlueTerm*, foi feita a movimentação do computador de modo a verificar-se a distância máxima permitida para comunicação. O resultado do teste está demonstrado na Figura 58.

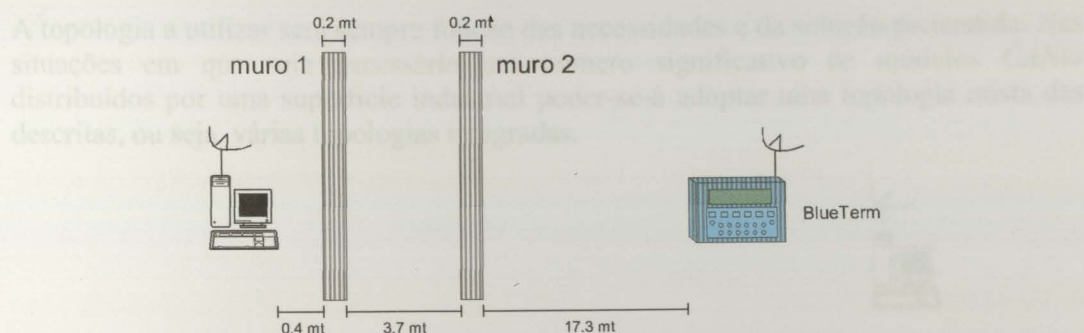
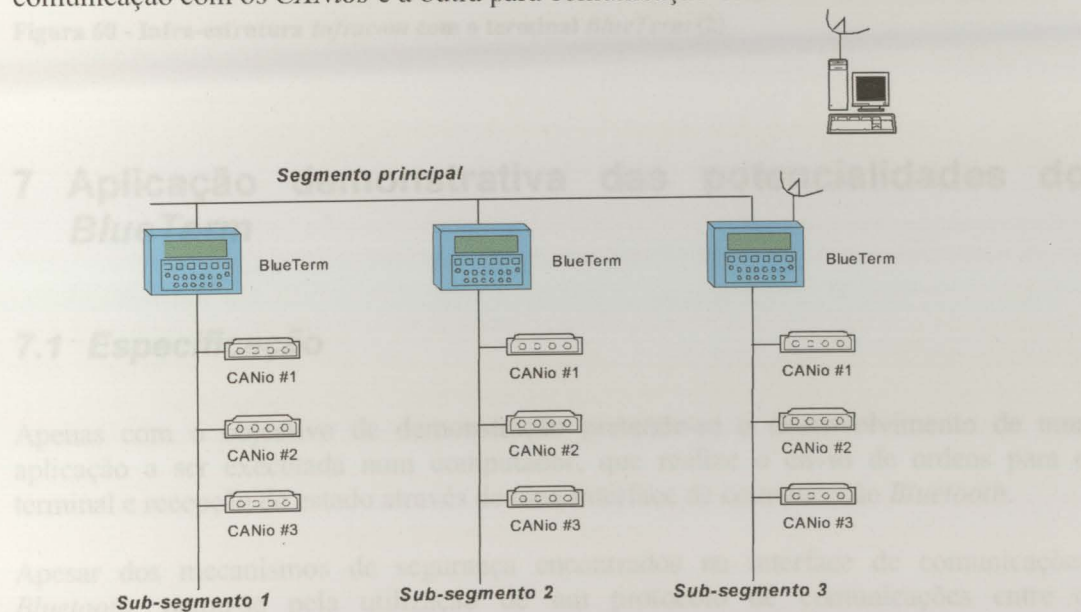


Figura 58 - Teste de distância

6.11 Valor acrescentado pelo Bluetooth à Infracom

O *BlueTerm* pode ser inserido na infra-estrutura *Infracom* em várias topologias:

Na Figura 59 temos o caso em que são usados dois terminais *BlueTerm* sem e um com a opção de interface *Bluetooth*. O terminal que possui a interface *Bluetooth* centraliza toda a informação do segmento industrial e realiza a interface com o computador do centro de operações via interface *Bluetooth*. o segmento principal e os subsegmentos são realizados sobre a interface *CAN*; uma das interfaces *CAN* do terminal é usada para comunicação com os *CANios* e a outra para comunicação com os outros terminais.

Figura 59 - Infra-estrutura Infracom com o terminal *BlueTerm* (1)

Na Figura 60 temos uma integração um pouco distinta da anterior. Neste caso todos os terminais possuem interface *Bluetooth* podendo cada um deles comunicar independentemente com o computador do centro de operações. Esta solução fica mais dispendiosa que a anterior devido à necessidade de interface *Bluetooth* em todos os terminais.

A topologia a utilizar será sempre função das necessidades e da solução pretendida. Nas situações em que seja necessário um número significativo de módulos *CANio* distribuídos por uma superfície industrial poder-se-á adoptar uma topologia mista das descritas, ou seja, várias topologias integradas.

Tabela 5 – Alguns comandos e respostas para estrutura remota do terminal

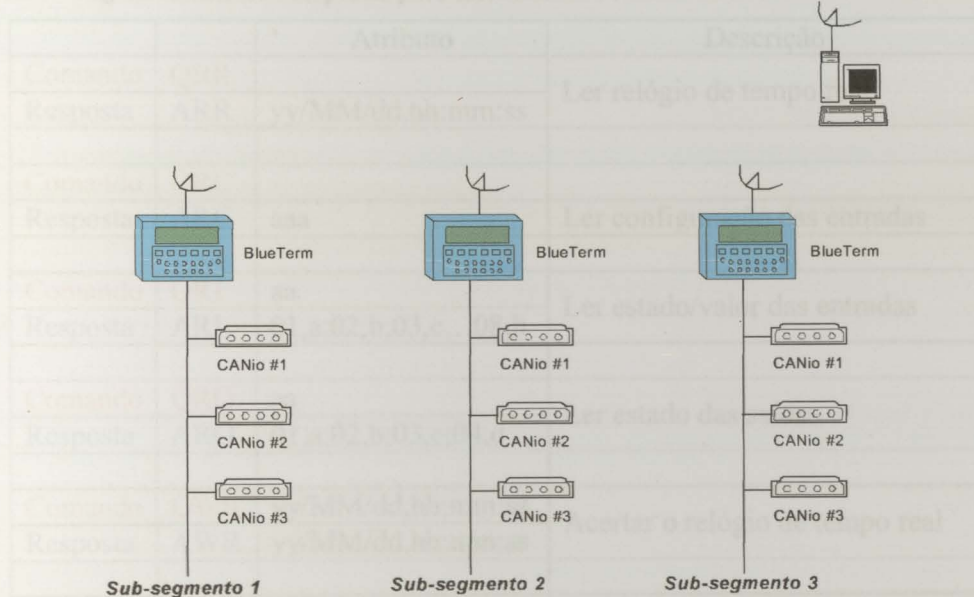


Figura 60 - Infra-estrutura Infracom com o terminal BlueTerm (2)

7 Aplicação demonstrativa das potencialidades do BlueTerm

7.1 Especificação

Apenas com o objectivo de demonstração pretende-se o desenvolvimento de uma aplicação a ser executada num computador, que realize o envio de ordens para o terminal e recepção de estado através de uma interface de comunicação *Bluetooth*.

Apesar dos mecanismos de segurança encontrados na interface de comunicações *Bluetooth*, optou-se pela utilização de um protocolo de comunicações entre o computador e o terminal proprietário que possui autenticação por *password*, ou seja, na sua memória de configuração, o terminal possui uma *password* numérica configurável de quatro dígitos que serve para aceitar ou recusar as tramas em função da sua *password*. Com a tecnologia sem fios *Bluetooth* este mecanismo de segurança não era necessário, mas se for necessário recorrer a outro tipo de interface sem este tipo de segurança o protocolo pode-se manter - flexibilidade implementada a nível da programação.

Na trama de comando os primeiros quatro zeros 0000 correspondem à *password*, o QWR corresponde ao comando, 03 ao argumento *ago*, 10 ao argumento *mês*, 04 ao

Na Tabela 5 apresenta-se um conjunto de comandos e respectivos argumentos para troca de informação entre um computador e o terminal *BlueTerm*.

Tabela 5 – Alguns comandos e respostas para escrita/leitura remota do terminal

		Atributo	Descrição
Comando	QRR		Ler relógio de tempo real
Resposta	ARR	yy/MM/dd,hh:mm:ss	
Comando	QRC		Ler configuração das entradas
Resposta	ARC	aaa	
Comando	QRI	aa	Ler estado/valor das entradas
Resposta	ARI	01,a;02,b;03,c...08,h	
Comando	QRO	aa	Ler estado das saídas
Resposta	ARO	01,a;02,b;03,c;04,d	
Comando	QWR	yy/MM/dd,hh:mm:ss	Acertar o relógio de tempo real
Resposta	AWR	yy/MM/dd,hh:mm:ss	
Comando	QWI	aa;b	Configurar comportamento das entradas
Resposta	AWI	aa;b	
Comando	QWO	aa;b	Actuar determinada saída
Resposta	AWO	aa;b	

Neste protocolo de comunicação as tramas enviadas ao módulo de *Bluetooth* serão constituídas por um delimitador inicial (carácter '\$'), seguido pela *password*, seguido por um comando e respectivo argumento (ver Tabela 5), seguido por um *checksum* de dois caracteres *ascii* e por fim um delimitador final (carácter '#'). Numa trama os comandos são separados dos respectivos argumentos (caso existam) pelo carácter ';' e a separação dos argumentos é variável (ver Tabela 5).

O *checksum* é igual ao resto da divisão por 256 do somatório do código *ascii* de todos os caracteres da trama com excepção dos delimitadores.

Exemplo de um comando enviado de um computador para um terminal *BlueTerm*:

Função: acerto do relógio de tempo real.

Trama de comando:

\$0000;QWR;03/10/04,12:00:00;D8#

Na trama de comando os primeiros quatro zeros 0000 correspondem à *password*, o QWR corresponde ao comando, 03 ao argumento ano, 10 ao argumento mês, 04 ao

argumento dia, 12 ao argumento horas, 00 ao argumento minutos e 00 ao argumento segundos. O *checksum* é representado pelos caracteres D8.

Trama de resposta:

\$0000;AWR;03/10/04,12:00:00;C8#

Na trama de resposta os primeiros quatro zeros 0000 correspondem à *password*, o AWR corresponde ao comando, 03 ao argumento ano, 10 ao argumento mês, 04 ao argumento dia, 12 ao argumento horas, 00 ao argumento minutos e 00 ao argumento segundos. O *checksum* é representado pelos caracteres C8.

7.2 Implementação

Foi desenvolvida uma aplicação com recurso à ferramenta *Microsoft Visual Basic 6.0* para demonstração de algumas das potencialidades da comunicação remota – *Bluetooth*. Na Figura 61 é apresentada a janela inicial da aplicação. Existem quatro menus com opções disponíveis.

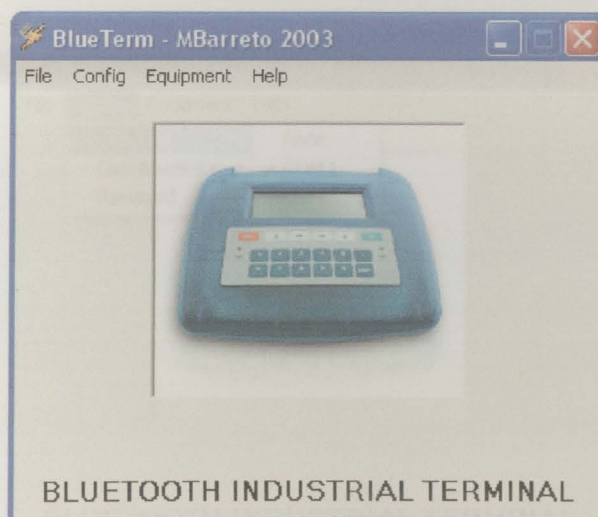


Figura 61 - Aplicação demonstrativa

No menu **File** existe a opção **Exit** que serve para abandonar a aplicação.



Figura 62 - Aplicação demonstrativa - menu *File*

No menu **Config** existem três opções. A opção **Com Port** permite a escolha da porta de comunicação série de entre quatro opções disponíveis.

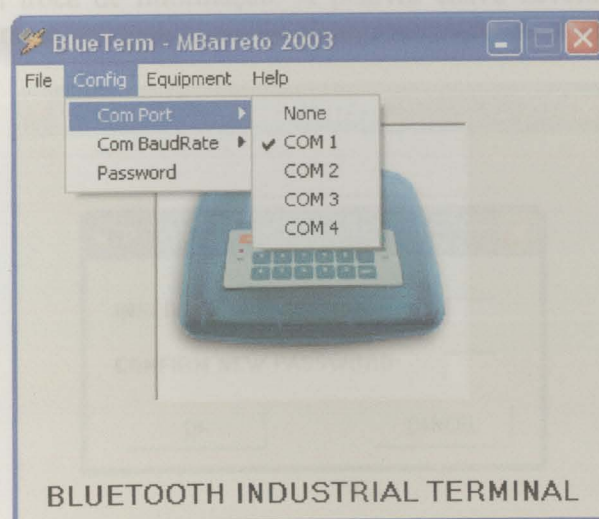


Figura 63 - Aplicação demonstrativa - menu *Config* - *Com Port*

A opção **Com BaudRate** do menu **Config** permite a escolha da velocidade de comunicação da ligação série. Existem cinco opções disponíveis.

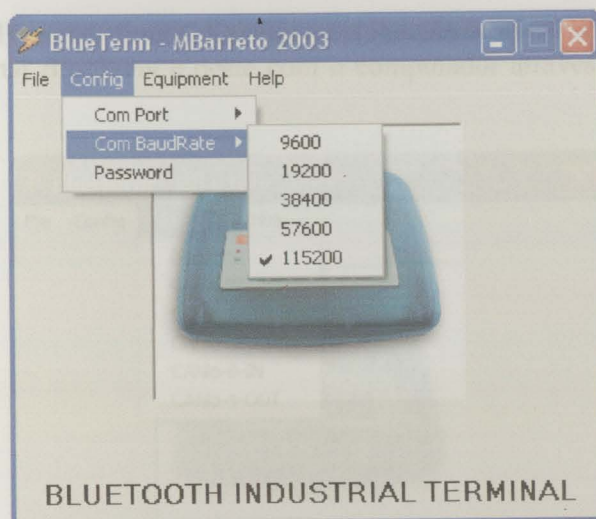


Figura 64 - Aplicação demonstrativa - menu *Config* - *Com BaudRate*

A opção **Password** do menu **Config** permite configurar a palavra chave utilizada na ligação série, para troca de informação. A palavra chave deverá ser constituída por quatro dígitos decimais.

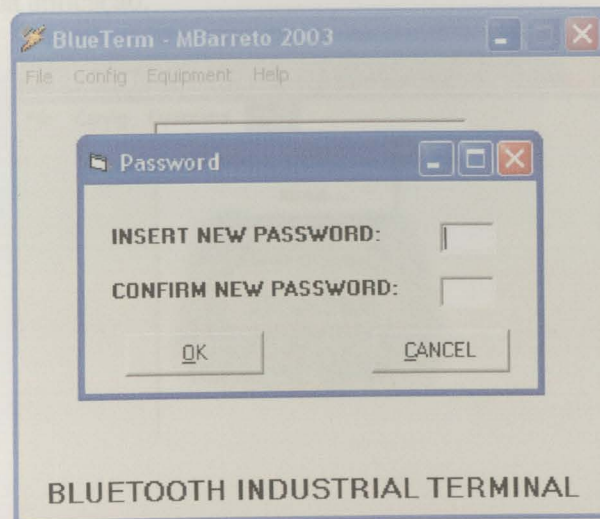


Figura 65 - Aplicação demonstrativa - menu *Config* - *Password*

O menu **Equipment** permite escolher o equipamento com o qual se pretende comunicar. O **BlueTerm** pode ser acedido através deste menu ou pressionando a figura presente na janela principal da aplicação que possui o desenho do terminal. A comunicação com o terminal **BlueTerm** pode ser feita através de uma interface por cabo RS232 ou pela interface sem fios **bluetooth**. A comunicação com os outros equipamentos é objecto de desenvolvimento posterior, mas foi pensado a sua inclusão nesta aplicação de forma a

facilitar o desenvolvimento futuro. Existirá uma ligação através da interface diferencial *CAN* com o terminal *BlueTerm* e deste com o computador através da interface RS232 ou *bluetooth*.

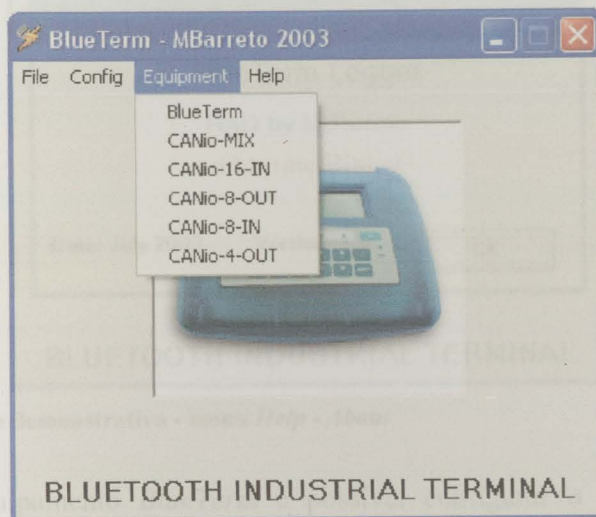


Figura 66 - Aplicação demonstrativa - menu *Equipment*

No menu *Help* existem duas opções mas apenas o acesso ao *About* está disponível. Futuramente poderá ser desenvolvida a opção *Contents* incluindo nesta um pequeno manual funcional da aplicação.

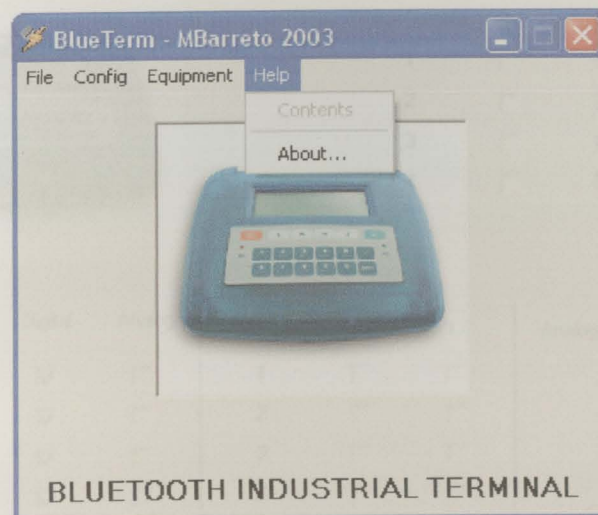


Figura 67 - Aplicação demonstrativa - menu *Help*

A opção *About* do menu *Help* possui o nome da aplicação e alguma informação sobre a sua criação.



Figura 68 - Aplicação demonstrativa - menu *Help* - *About*

Na janela do equipamento *BlueTerm* é possível configurar o comportamento das entradas (digital ou analógico) e o estado das saídas. Pode ser visualizado o estado ou o valor das entradas em função das suas configurações. Existem dois botões: *Exit* permite abandonar a janela do *BlueTerm* e regressar à janela principal e *Update* permite estabelecer uma ligação com o terminal e actualizar a sua informação.

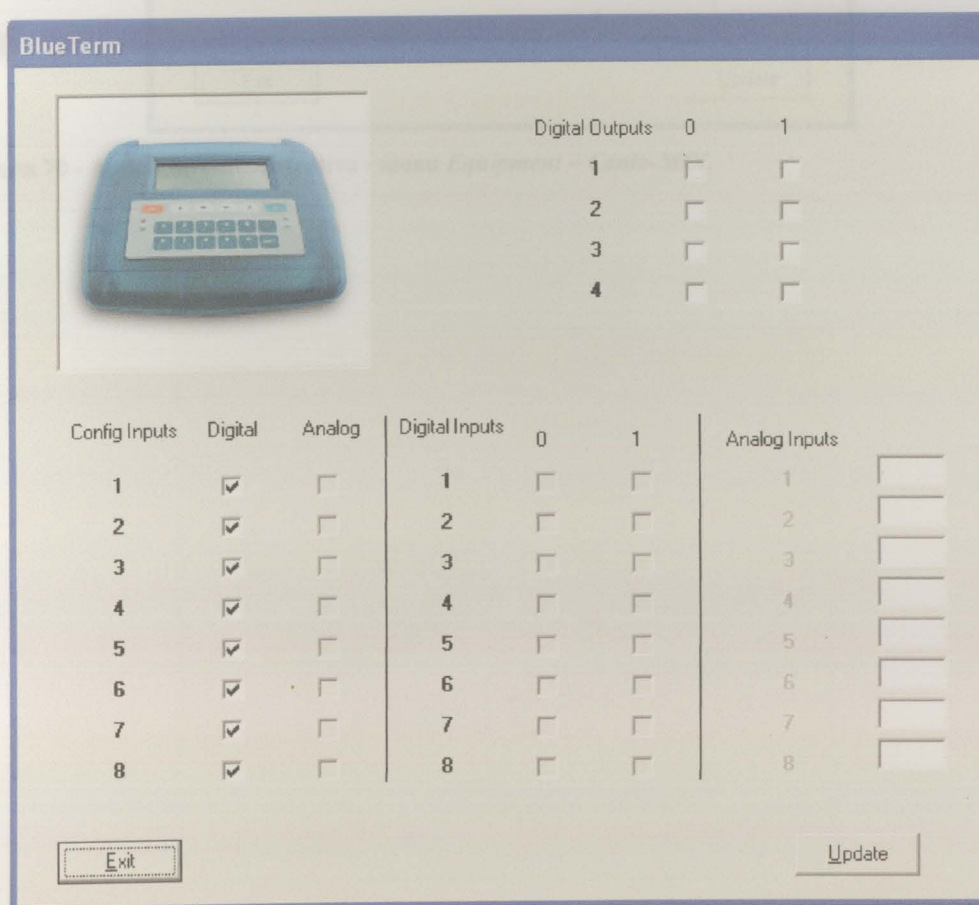


Figura 69 - Aplicação demonstrativa - menu *Equipment* - *BlueTerm*

A janela de acesso ao equipamento *CANio-MIX* permitirá actuar nas suas saídas digitais, ler o estado das entradas digitais e o valor das entradas analógicas.

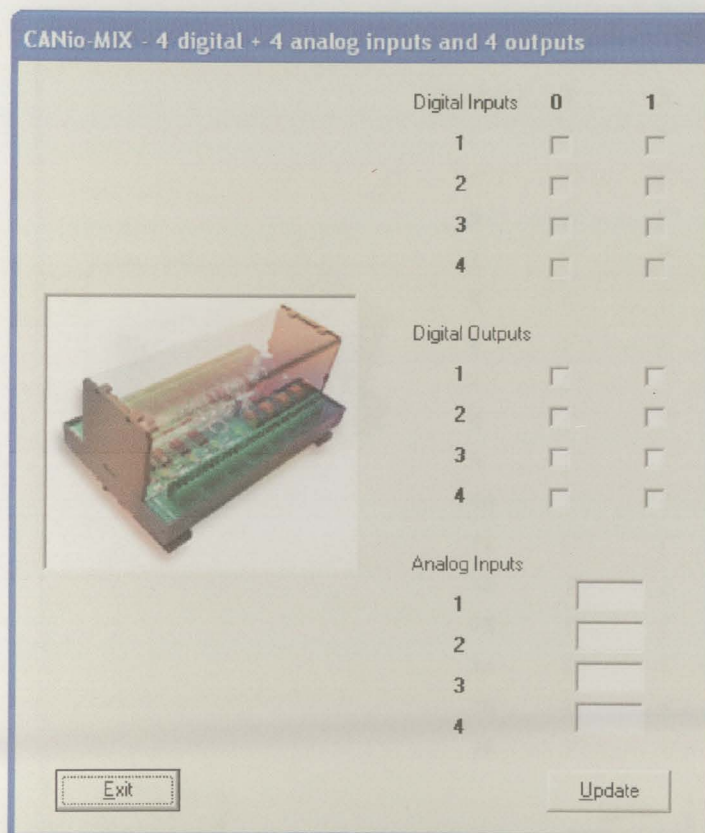
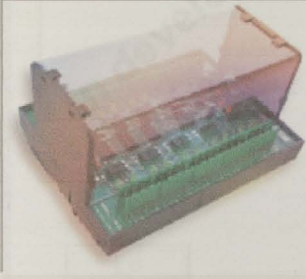


Figura 70 - Aplicação demonstrativa - menu *Equipment* – *Canio-MIX*

A janela de acesso ao equipamento *CANio-16-IN* permitirá ler o estado das suas entradas digitais.

CANio-16-IN - 16 opto-isolated digital inputs

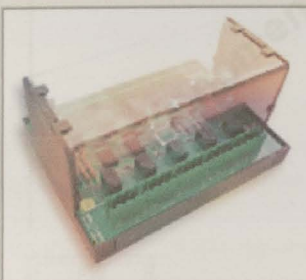


Digital Inputs	0	1
1	☐	☐
2	☐	☐
3	☐	☐
4	☐	☐
5	☐	☐
6	☐	☐
7	☐	☐
8	☐	☐
9	☐	☐
10	☐	☐
11	☐	☐
12	☐	☐
13	☐	☐
14	☐	☐
15	☐	☐
16	☐	☐

Figura 71 - Aplicação demonstrativa - menu *Equipment* - *CANio-16-IN*

A janela de acesso ao equipamento *CANio-8-OUT* permitirá alterar o estado das suas saídas digitais.

CANio-8-OUT - 8 digital outputs



Digital Outputs	0	1
1	☐	☐
2	☐	☐
3	☐	☐
4	☐	☐
5	☐	☐
6	☐	☐
7	☐	☐
8	☐	☐

Figura 72 - Aplicação demonstrativa - menu *Equipment* - *CANio-8-OUT*

A janela de acesso ao equipamento *CANio-8-IN* permitirá ler o valor das suas entradas analógicas.

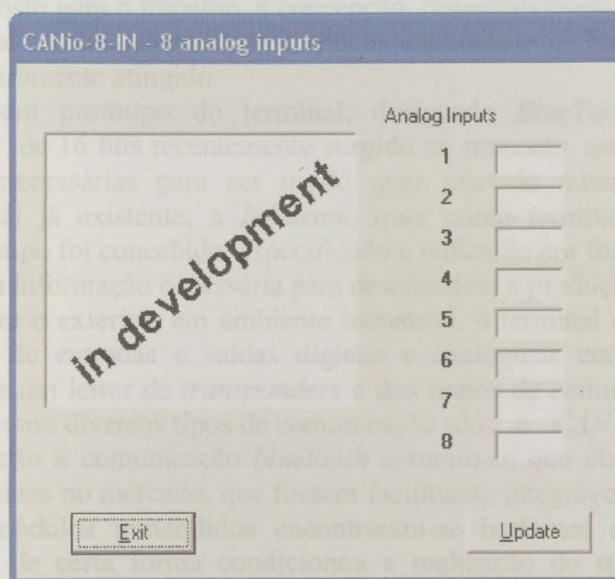


Figura 73 - Aplicação demonstrativa - menu *Equipment* - *CANio-8-IN*

A janela de acesso ao equipamento *CANio-4-OUT* permitirá escrever o valor das suas saídas analógicas.

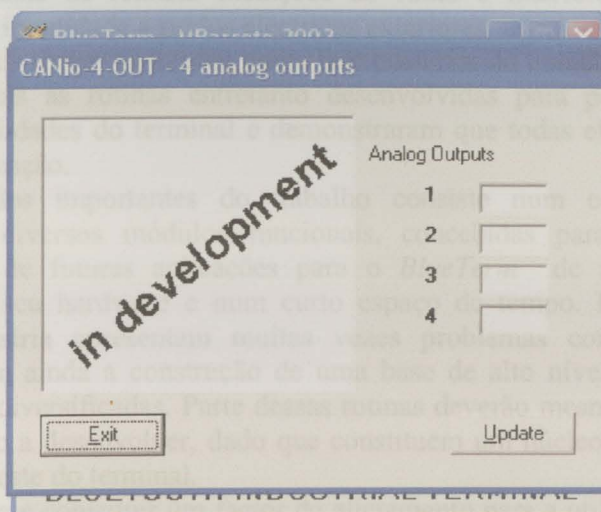


Figura 74 - Aplicação demonstrativa - menu *Equipment* - *CANio-4-OUT*

8 Conclusões e trabalho futuro

O objectivo proposto para o trabalho, a concepção, desenvolvimento e realização de um terminal industrial de uso genérico dispondo da capacidade de comunicação sem fios *bluetooth* foi inteiramente atingido.

Foi construído um protótipo do terminal, designado *BlueTerm*, com base num microcontrolador de 16 bits recentemente surgido no mercado, que dispõe de todas as funcionalidades necessárias para ser usado quer inserido numa infra-estrutura de controlo industrial já existente, a *Infracom*, quer como terminal industrial de uso genérico. O protótipo foi concebido, especificado e realizado por forma a que no final se pudesse dispor da informação necessária para desencadear a produção do terminal.

Para interagir com o exterior, em ambiente industrial, o terminal dispõe para além de visor e teclado, de entradas e saídas digitais e analógicas configuráveis, de uma interface *USB*, de um leitor de *transponders* e dos meios de comunicação mais usuais nesse ambiente, como diversos tipos de comunicação série e *CAN*.

No que diz respeito à comunicação *bluetooth* assumiu-se que ela seria realizada por módulos já existentes no mercado, que fossem facilmente integráveis no terminal. Após seleccionar os módulos pretendidos encontraram-se bastantes dificuldades na sua obtenção, o que de certa forma condicionou a realização do trabalho, limitando o número de testes e não permitindo obter uma amostra significativa do comportamento do terminal com esse tipo de ligação em ambiente industrial, como se pretendia. A instalação do equipamento em ambiente industrial e a monitorização do seu desempenho durante um espaço de tempo considerado significativo constituem assim um dos trabalhos futuros a realizar.

O terminal foi alvo de um conjunto alargado de testes eléctricos e funcionais, sendo os primeiros realizados logo após a sua montagem. Estes testes iniciais permitiram avaliar a qualidade do circuito no que diz respeito à colocação dos componentes e sua interligação e deram boas indicações para o desempenho das diversas funcionalidades, visto não revelarem sintomas de ruídos eléctricos capazes de provocar funcionamento anómalo de qualquer um dos componentes. Como trabalho futuro pode-se desde já referir a necessidade de realizar medições do ruído e interferência causados pelo terminal e da sua imunidade a ruídos eléctricos exteriores.

Os testes funcionais foram realizados numa fase posterior do trabalho, à medida que iam ficando disponíveis as rotinas entretanto desenvolvidas para permitir o acesso às diversas funcionalidades do terminal e demonstraram que todas elas estavam em boas condições de utilização.

Um dos resultados importantes do trabalho consiste num conjunto de rotinas, organizadas em diversos módulos funcionais, concebidas para tornar possível o desenvolvimento de futuras aplicações para o *BlueTerm* de forma transparente relativamente ao seu hardware e num curto espaço de tempo. Dado que diferentes sectores da indústria apresentam muitas vezes problemas comuns, este tipo de abordagem facilita ainda a construção de uma base de alto nível para utilização em aplicações muito diversificadas. Parte dessas rotinas deverão mesmo ser integradas em qualquer aplicação a desenvolver, dado que constituem um núcleo que permite fazer a configuração e o teste do terminal.

Para servir de base e constituir um factor de aliciamento para a utilização do *BlueTerm* em futuras aplicações foi desenvolvida uma aplicação demonstrativa que permitiu testar todas as funcionalidades do *BlueTerm* e nomeadamente realizar o seu controlo a partir de comandos enviados de um computador pessoal por ligação *bluetooth*.

Um terminal industrial com as características do *BlueTerm* traz mais valias para o mercado entre as quais podemos destacar a sua flexibilidade por via da adaptação das suas funcionalidades à aplicação pretendida, o seu baixo custo relativo, a facilidade da sua programação e a capacidade de integração em diferentes tipos de redes e infra-estruturas de comunicação.

Este trabalho pretendeu constituir um ponto de partida, uma base, para o desenvolvimento de soluções de problemas muito diversos no âmbito da automação industrial.

Como trabalho a realizar no futuro aponta-se a optimização dos recursos fornecidos pelo processador utilizado e o aumento das potencialidades do terminal tendo em vista as aplicações alvo. A utilização de um processador bastante recente e com imenso potencial proporciona uma boa plataforma para próximas realizações mesmo tendo em conta que o tempo de adaptação a um novo processador é normalmente elevado assim como o da obtenção e familiarização com as ferramentas necessárias para o desenvolvimento das aplicações.

A novidade da utilização da tecnologia *bluetooth* em ambiente industrial deixa antever que num futuro próximo estarão disponíveis mais e melhores componentes permitindo explorar e integrar melhor as potencialidades desta tecnologia promissora no domínio da automação industrial.

Referências

- [1] Ericsson Technology Licensing AB – Bluetooth Beginner's Guide.
- [2] HODGDON, Charles - Adaptive Frequency Hopping for Reduced Interference between Bluetooth and Wireless LAN - White paper. Ericsson Technology Licensing AB. 2003.
- [3] Silicon Wave, Inc. - Bluetooth and Wi-Fi Coexistence Overview - White Paper. 2003.
- [4] BOURK, Terry - Techniques mitigate interference between 802.11 and Bluetooth. Silicon Wave, Inc.. 2003.

Bibliografia

BEACH, Michael; BARSTOW, John; SMITH, Karl - C166 Designer's Guide - Second Edition. Coventry: University of Warwick Science Park. 1999.

Infineon Technologies AG - XC161 - 16-Bit Single-Chip Microcontroller with 166SV2 Core Volume 1 (of 2): System Units. München: Infineon Technologies AG. 2003.

Infineon Technologies AG - XC161 - 16-Bit Single-Chip Microcontroller with 166SV2 Core Volume 2 (of 2): Peripheral Units. München: Infineon Technologies AG. 2003.

Sgs-Thomson Microelectronics - M24C16, M24C08, M24C04, M24C02, M24C01, 16K/8K/4K/2K/1K SERIAL I2C BUS EEPROM Product Specification. 1998.

Mitsubishi Electric - M5M51008CP,FP,VP,RV,KV,KR -55HI, -70HI, -55XI, -70XI; 1048576-BIT(131072-WORD BY 8-BIT) CMOS STATIC RAM Product Specification.

Philips Semiconductors - SC26C92 Dual Universal Asynchronous Receiver/Transmitter (DUART) - Product Specification. 2000.

Data Image Corporation - LCD Module Specification: ITEM NO.: CM2041S1LY-J6. 1999.

STMicroelectronics - M41T56 512 bit (64 bit x 8) Serial Access TIMEKEEPER® SRAM. 2001.

Scandisk Corporation - SanDisk MultiMediaCard Product Manual. 1998.

AMD's - Am29F010B 1 Megabit (128 K x 8-bit) CMOS 5.0 Volt-only, Uniform Sector Flash Memory: Product Specification. 2000.

National Semiconductor Corporation - LM2825 Integrated Power Supply 1A DC-DC Converter Product specification. 1997.

Keil Software, Inc. - C16x/ST10 Assembler and Utilities - User's Guide 11.00. 1992-2000.

Keil Software, Inc. - C166 Compiler - Optimizing 166/167 C Compiler and Library Reference - User's Guide 09.00. 1992-2000.

Keil Software, Inc. - Getting Started and Create Applications with µVision2 and C166/ST10 Microcontroller Development Tools - User's Guide 5.99. 1992-2000.

Intel - 87C196CA/87C196CB 16 MHz Advanced 16-bit CHMOS Microcontroller with Integrated CAN 2.0. 1998.

Motorola - Motorola Semiconductor Technical Data - MC9S12D-Family. 2002.

Anexo A – Protótipos das funções de baixo nível

A.1 Módulo “can”

- ***void CAN_vInit(void)***

Descrição: função de nível 0, usada para iniciação dos canais de CAN.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void CAN_vGetMsgObj(BYTE ubObjNr, TCAN_SWObj *pstObj)***

Descrição: função de nível 0, esta função preenche o objecto de software da mensagem com o conteúdo do objecto de hardware.

Parâmetros:

ubObjNr : objecto a actualizar

**pstObj* : apontador para o objecto de software da mensagem a preencher

Valor devolvido: nenhum.

- ***BYTE CAN_ubRequestMsgObj(BYTE ubObjNr)***

Descrição: função de nível 0, usada para determinar quando um certo objecto está livre para acesso (por exemplo para reconfiguração).

Parâmetros:

ubObjNr : objecto a verificar

Valor devolvido:

0 : objecto está ocupado

1 : objecto está livre

- ***BYTE CAN_ubNewData(BYTE ubObjNr)***

Descrição: função de nível 0, usada para determinar quando um objecto de recepção recebeu uma nova mensagem.

Parâmetros:

ubObjNr : objecto a verificar

Valor devolvido:

0 : objecto não possui mensagens novas

1 : objecto recebeu uma mensagem nova

- ***void CAN_vTransmit(BYTE ubObjNr)***

Descrição: função de nível 0, esta função despoleta o envio de uma mensagem de determinado objecto.

Parâmetros:

ubObjNr : objecto a enviar

Valor devolvido: nenhum.

- ***void CAN_vSendDataFrame(BYTE bNode, UINT uiAddress, BYTE *bData, BYTE bQty)***

Descrição: função de nível 1, usada para enviar um número pré-determinado de caracteres por um dos canais CAN com um endereço de destino.

Parâmetros:

bNode : canal de CAN a utilizar. CAN_NODE_A ou CAN_NODE_B

uiAddress : endereço de destino da mensagem

**bData* : apontador para a origem da mensagem

bQty : quantidade de caracteres da mensagem

Valor devolvido: nenhum.

- **void CAN_vConfigMsgObj(BYTE ubObjNr, TCAN_SWObj *pstObj)**

Descrição: função de nível 0, usada para configuração dos objectos.

Parâmetros:

- ubObjNr* : número do objecto a configurar
- *pstObj* : apontador para a nova configuração a realizar no objecto

Valor devolvido: nenhum.

- **void CAN_vReleaseObj(BYTE ubObjNr)**

Descrição: função de nível 0, usada para libertar um determinado objecto. Esta função deve ser chamada se a função *CAN_ubNewData* detecta que novos dados estão presentes no objecto e esses dados já foram lidos através da chamada da função *CAN_vGetMsgObj*.

Parâmetros:

- ubObjNr* : número do objecto

Valor devolvido: nenhum.

A.2 Módulo “coms”

- **void COM_vTakeChar(BYTE bPort)**

Descrição: função de nível 0, usada nas interrupções de recepção dos canais de comunicação de modo a retirar os caracteres do *buffer* interno de recepção da porta e guarda-los no respectivo *buffer* de recepção (*bComBufferIn[bPort]*) para posterior utilização.

Parâmetros:

- bPort* : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

Tabela 6-A - Definições para o módulo de comunicações

Valor	Definição	Descrição
0	ASC0	Porta série 0
1	ASC1	Porta série 1
2	DUART_A	Porta série de comunicação com o módulo de Bluetooth
3	DUART_B	Porta série de comunicação com o módulo de leitura de transponders

- **void COM_vSendChar(BYTE bPort)**

Descrição: função de nível 0, usada nas interrupções de transmissão dos canais de comunicação de modo a colocar os caracteres no *buffer* interno de transmissão da porta. Os caracteres são retirados do respectivo *buffer* de transmissão (*bComBufferOut[bPort]*).

Parâmetros:

- bPort* : canal de comunicação pretendido - utilizar uma das

definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

- **BYTE COM_bGetChar(BYTE *bValorLido, BYTE bPort)**

Descrição: função de nível 1, que deve ser utilizada para retirar um carácter do *buffer* de recepção pretendido (*bComBufferIn[bPort]*). Os caracteres são automaticamente recebidos e guardados pela função *COM_vTakeChar*.

Parâmetros:

**bValorLido* : apontador para posição onde guardar o carácter retirado do *buffer*
bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

- **void COM_vPutChar(BYTE bData, BYTE bPort)**

Descrição: função de nível 1, que deve ser utilizada para colocar um carácter no *buffer* de transmissão pretendido (*bComBufferOut[bPort]*). Estes serão automaticamente enviados pela função *COM_vSendChar*.

Parâmetros:

bData : carácter a enviar
bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

- **BYTE COM_bRead(void *apvStr, BYTE bNrChars, BYTE bPort)**

Descrição: função de nível 2, que deve ser utilizada para retirar um ou mais caracteres do respectivo *buffer* de recepção (*bComBufferIn[bPort]*).

Parâmetros:

**apvStr* : apontador para a primeira posição de memória onde se pretende colocar os caracteres retirados do *buffer* de recepção
bNrChars : quantidade de caracteres a retirar do *buffer* de recepção
bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

- **void COM_vWrite(BYTE *bText, BYTE bPort)**

Descrição: função de nível 2, que deve ser usada para colocar um ou mais caracteres no respectivo *buffer* de transmissão (*bComBufferOut[bPort]*). O número de caracteres a copiar é determinado pela colocação do carácter '\0' no *buffer* de origem.

Parâmetros:

**bText* : apontador para a primeira posição de memória onde se encontram os caracteres que se pretendem colocar no *buffer* de transmissão
bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

- **BYTE COM_bBuffSize(BYTE bBufferInOut, BYTE bPort)**

Descrição: função de nível 1, que serve para determinar o número de caracteres presentes nos *buffers* de transmissão e recepção.

Parâmetros:

bBufferInOut : BUFFER_IN - *buffer* de recepção (*bComBufferIn[bPort]*)
: BUFFER_OUT - *buffer* de transmissão
(*bComBufferOut[bPort]*)

bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: número de caracteres presente no *buffer*.

• **BYTE COM_bWait4String(BYTE *sChars, UINT uiTimeout, BYTE bPort)**

Descrição: função de nível 2, que tem como objectivo esperar a recepção de um determinado conjunto de caracteres (*string*) durante um tempo pré-definido.

Parâmetros:

**sChars* : apontador para a primeira posição onde se encontra o conjunto de caracteres que se espera receber

uiTimeout : tempo durante o qual aguarda pela recepção. Uma unidade corresponde a 1.5 ms

bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido:

0 : sucesso na recepção do conjunto de caracteres
1 : insucesso na recepção do conjunto de caracteres

• **void COM_vFlush(BYTE bPort)**

Descrição: função de nível 0, que serve para ajustar os apontadores do *buffer* de recepção pretendido de modo a apagar deste todos os caracteres.

Parâmetros:

bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

• **void COM_vSendMessage(BYTE Message[], BYTE bPort)**

Descrição: função de nível 3, que serve para a realização da transmissão de uma mensagem com o seguinte formato: '\$x...xy#'. Os caracteres '\$' e '#' são os delimitadores da mensagem e 'y' é o *checksum*; os delimitadores e o *checksum* são colocados automaticamente por esta função, o programador apenas tem que fornecer o corpo da mensagem 'x...x'.

Parâmetros:

Message[] : apontador para a primeira posição onde se encontra o corpo da mensagem a enviar

bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido: nenhum.

• **BYTE COM_bReceiveMessage(BYTE Message[], BYTE bPort)**

Descrição: função de nível 3, que serve para a recepção de uma mensagem no seguinte formato: '\$x...xy#'. Os caracteres '\$' e '#' são os delimitadores da mensagem e 'y' é o *checksum*; os delimitadores e o *checksum* são retirados automaticamente por esta função.

Parâmetros:

Message[] : apontador para à primeira posição onde se pretende guardar o corpo da mensagem

bPort : canal de comunicação pretendido - utilizar uma das definições indicadas na Tabela 6-A

Valor devolvido:

0 : recepção de mensagem pretendida

1 : recepção nula

2 : carácter de início de mensagem inválido

3 : carácter de fim de mensagem inválido

4 : *checksum* da mensagem incorrecto

A.3 Módulo “display”**Tabela 7-A - Definições do módulo display**

Item	Definição	Descrição
1	<i>BLINK_ON</i>	a colocar no parâmetro <i>bOptions</i> das funções <i>vprint</i> quando se pretende que a informação a escrever no <i>display</i> fique intermitente
2	<i>BLINK_OFF</i>	a colocar no parâmetro <i>bOptions</i> das funções <i>vprint</i> quando se pretende que a informação a escrever no <i>display</i> fique estática
3	<i>ALIGN_LEFT</i>	a colocar no parâmetro <i>bOptions</i> das funções <i>vprint</i> quando se pretende que a informação a escrever no <i>display</i> fique alinhada à esquerda
4	<i>ALIGN_RIGHT</i>	a colocar no parâmetro <i>bOptions</i> das funções <i>vprint</i> quando se pretende que a informação a escrever no <i>display</i> fique alinhada à direita
5	<i>ALIGN_CENTER</i>	a colocar no parâmetro <i>bOptions</i> das funções <i>vprint</i> quando se pretende que a informação a escrever no <i>display</i> fique alinhada ao centro
6	<i>AS_ASTERISKS</i>	a colocar no parâmetro <i>bOptions</i> das funções <i>vprint</i> quando se pretende que a informação a escrever no <i>display</i> seja substituída por asteriscos
7	<i>LEADING_ZEROS</i>	a colocar no parâmetro <i>bOptions</i> das funções <i>vprint</i> quando se pretende que com o valor a afixar apareçam os zeros à esquerda correspondentes
8	<i>SHOW_SECONDS</i>	a colocar no parâmetro <i>uiOptions</i> da função <i>vPrintTime</i> quando se pretende escrever além da hora e dos minutos os segundos
9	<i>SHOW_WEEK_DAY</i>	ver Tabela 11-A
10	<i>ENGLISH_DAY</i>	ver Tabela 11-A
11	<i>IFEN_SEPARATOR</i>	ver Tabela 11-A

- ***void CC2_vInit(void)***

Descrição: função de nível 0, tem como objectivo a iniciação da unidade de comparação/captura 2 (CAPCOM2) do microprocessador, que é usada para a geração do PWM para ajuste da luminosidade do *display*. Esta função deve ser invocada nas iniciações do programa.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void DISPLAY_vReset(BYTE bCursor, BYTE bBlink)***

Descrição: função de nível 0, tem como objectivo a realização dos procedimentos necessários à iniciação do *display*. Esta função deve ser invocada nas iniciações do programa.

Parâmetros:

bCursor : serve para indicar se o cursor deve ao não aparecer (CURSOR_ON ou CURSOR_OFF)
bBlink : serve para indicar se o cursor deve ao não piscar (BLINK_ON ou BLINK_OFF)

Valor devolvido: nenhum.

- ***void DISPLAY_vRefresh(BYTE Frame)***

Descrição: função de nível 0, serve para realizar a actualização da informação do *display* com o conteúdo do *buffer DisplayBuffer*. A actualização é efectuada em parcelas sendo cada parcela constituída por meia linha (dez caracteres); o *display* está dividido em oito parcelas. Esta função é chamada com a periodicidade de 26 ms aquando da interrupção de um temporizador (*timer*) pré-programado.

Parâmetros:

Frame : indica qual a parcela a actualizar

Valor devolvido: nenhum

- ***void DISPLAY_vClearLine(BYTE bLine)***

Descrição: função de nível 1, serve para limpar o conteúdo de uma linha do *display*.

Parâmetros:

bLine : indica qual a linha a apagar

Valor devolvido: nenhum.

- ***void DISPLAY_vClearScreen(void)***

Descrição: função de nível 1, serve para limpar o conteúdo de todo o *display*.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void DISPLAY_vPrintString(BYTE bLine, BYTE bColumn, char cString[], BYTE bOptions)***

Descrição: função de nível 1, serve para escrever um conjunto de caracteres (*string*) na posição indicada por *bLine* e *bColumn*. O canto superior esquerdo corresponde à posição (0,0) e o inferior direito à posição (3,19). A *string* tem de ser sempre terminada com '\0'.

Parâmetros:

bLine : indica qual a linha a escrever
bColumn : indica qual a coluna de início da escrita

- cString* : mensagem a escrever
bOptions : pode tomar um dos valores das definições dos itens de 1 a 5 da Tabela 7-A

Valor devolvido: nenhum.

- ***void DISPLAY_vPrintByte(BYTE bLine, BYTE bColumn, BYTE bValue, BYTE bDigits, BYTE bOptions)***

Descrição: função de nível 1, serve para escrever um número do tipo carácter (*char* ou *byte*) no *display* em decimal, na posição indicada.

Parâmetros:

- bLine* : indica qual a linha a escrever
bColumn : indica qual a coluna de início da escrita
bValue : carácter a escrever
bDigits : indica o número de algarismos que devem ser utilizados para escrever o número. Se o número tiver menos algarismos que os indicados, são inseridos espaços à esquerda. Se o número tiver mais algarismos que os indicados apenas são escritos os algarismos menos significativos. Valores possíveis: 1 a 3. Ver Tabela 7-A
bOptions : pode tomar um dos valores das definições dos itens de 1 a 7 da Tabela 7-A

Valor devolvido: nenhum.

Tabela 8-A - Formato da escrita de um número

Dígitos	Numero: -12345	Numero: -2
2 dígitos	4 5	-2
5 dígitos	1 2 3 4 5	--- - 2
6 dígitos	- 1 2 3 4 5	----- - 2

- ***void DISPLAY_vPrintInt(BYTE bLine, BYTE bColumn, int iValue, BYTE bDigits, BYTE bOptions)***

Descrição: função de nível 1, funciona de forma idêntica à função *vPrintByte*, e serve para escrever um número do tipo inteiro com sinal (*int*) no *display* (-32768 a 32767). O número de dígitos pode variar entre 1 e 6. O sinal menos do número também é contado como dígito (ver Tabela 8-A).

- ***void DISPLAY_vPrintLong(BYTE bLine, BYTE bColumn, long lValue, BYTE bDigits, BYTE bOptions)***

Descrição: função de nível 1, funciona de forma idêntica à função *vPrintByte*, e serve para escrever um número do tipo longo com sinal (*long*) no *display* (-2147483648 a 2147483647). O número de dígitos pode variar entre 1 e 10. O sinal menos do número também é contado como dígito (ver Tabela 8-A).

- ***void DISPLAY_vPrintUINT(BYTE bLine, BYTE bColumn, UINT uiValue, BYTE bDigits, BYTE bOptions)***

Descrição: função de nível 1, funciona de forma idêntica à função *vPrintByte*, e serve para escrever um número do tipo inteiro sem sinal (*unsigned int*) no *display* (0 a 65535). O número de dígitos pode variar entre 1 e 5.

- **void DISPLAY_vPrintULONG(BYTE bLine, BYTE bColumn, ULONG uiValue, BYTE bDigits, BYTE bOptions)**

Descrição: função de nível 1, funciona de forma idêntica à função *vPrintByte*, e serve para escrever um número do tipo longo sem sinal (*unsigned long*) no *display* (0 a 4294967295). O número de dígitos pode variar entre 1 e 5.

- **void DISPLAY_vPrintFloat(BYTE bLine, BYTE bColumn, float fValue, BYTE bIntDigits, BYTE bDecDigits, BYTE bOptions)**

Descrição: função de nível 1, permite escrever um número em formato “vírgula flutuante” (*float*) no *display*. O parâmetro *bDecDigits* indica o número de algarismos a serem utilizados para escrever o número. São contados apenas os algarismos à esquerda da vírgula e o sinal menos. À direita da vírgula existe sempre um algarismo, correspondente às décimas. O número está limitado à gama -3200,0 a 3200,0. Ver exemplo na Tabela 8-A.

Parâmetros:

<i>bLine</i>	: indica qual a linha a escrever
<i>bColumn</i>	: indica qual a coluna de início da escrita
<i>fValue</i>	: <i>float</i> a escrever
<i>bIntDigits</i>	: indica o número de algarismos da parte inteira que devem ser utilizados para escrever o número. Se o número tiver menos algarismos que os indicados, são inseridos espaços à esquerda. Se o número tiver mais algarismos que os indicados apenas são escritos os algarismos menos significativos. Valores possíveis para bDigits : 1 a 3 (ver Tabela 7-A).
<i>bDecDigits</i>	: indica o número de algarismos da parte decimal que devem ser utilizados para escrever o número.
<i>bOptions</i>	: pode tomar um dos valores das definições dos itens de 1 a 7 da Tabela 7-A.

Valor devolvido: nenhum.

Tabela 9-A - Formato da escrita de um número em vírgula flutuante

Dígitos	Numero: -1234.5	Numero: 0
2 dígitos	3 4 . 5	_ 0 . 0
5 dígitos	- 1 2 3 4 . 5	___ 0 . 0
6 dígitos	_ - 1 2 3 4 . 5	___ 0 . 0

- **void DISPLAY_vPrintTime(BYTE bLine, BYTE bColumn, TypeTime HourDate, UINT uiOptions)**

Descrição: função de nível 2, permite escrever a hora fornecida pelo relógio de tempo real (RTC) na posição pretendida no *display*.

Parâmetros:

<i>bLine</i>	: indica qual a linha a escrever
<i>bColumn</i>	: indica qual a coluna de início da escrita
<i>HourDate</i>	: apontador para uma variável definida com a estrutura TypeTime
<i>bOptions</i>	: define como a hora deverá aparecer (ver Tabela 9-A)

Valor devolvido: nenhum.

A.4 Módulo "duart"

Tabela 10-A - Formatos de apresentação da hora

Valor de uiOptions	Formato apresentado
sem SHOW_SECONDS	"hh:mm"
com SHOW_SECONDS	"hh:mm:ss"

- **void DISPLAY_vPrintDate(BYTE bLine, BYTE bColumn, TypeTime HourDate, UINT uiOptions)**

Descrição: função de nível 2, permite escrever a data fornecida pelo relógio de tempo real (RTC) formatada de várias formas. Esta formatação é obtida com o valor introduzido no valor *uiOptions*.

A soma destes valores permite obter as configurações presentes na Tabela 11-A. Quando não são somadas algumas das opções, é assumido sem dia da semana e símbolo barra como separador.

Parâmetros:

bLine : indica qual a linha a escrever
bColumn : indica qual a coluna de inicio da escrita
HourDate : apontador para uma variável definida com a estrutura *TypeTime*
bOptions : define como a data deverá aparecer (ver Tabela 11-A)

Valor devolvido: nenhum.

Tabela 11-A – Formatos de apresentação da data

Valor de Option	Formato apresentado
sem IFEN_SEPARATOR	"04/10/03"
com IFEN_SEPARATOR	"04-10-03"
com SHOW_WEEK_DAY	"SAB 04/10/03"
com IFEN_SEPARATOR+SHOW_WEEK_DAY	"SAB 04-10-03"
com SHOW_WEEK_DAY+ENGLISH_DAY	"SAT 04/10/03"
com IFEN_SEPARATOR+SHOW_WEEK_DAY+ENGLISH_DAY	"SAT 04-10-03"

- **void ScreenCopy(void)**

Descrição: função de nível 1, serve para guardar a informação presente no *display*.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- **void ScreenRestore(void)**

Descrição: função de nível 1, serve para recuperar a informação presente no *display* aquando da invocação da função *ScreenCopy*.

Parâmetros: nenhum.

Valor devolvido: nenhum.

A.4 Módulo “duart”

- ***void DUART_vStart(BYTE bParametroDuart, BYTE bDuart)***

Descrição: função de nível 1, tem como objectivo permitir a configuração (antes da sua utilização) do canal do periférico DUART indicada em *bDuart*. Algumas configurações especiais de *baud rate* são incompatíveis, quando utilizadas as duas portas: se for configurado o *baud rate* 28800 bps numa porta não é possível obter 57600 bps na outra. Esta função deve ser chamada nas iniciações do programa.

Parâmetros:

bParametroDuart : é constituído pela junção (OR lógico) das configurações da Tabela 12-A

bDuart : indica o canal *DUART_A* ou *DUART_B*

Valor devolvido: nenhum.

Tabela 12-A - Definições do módulo Duart

Parâmetro	Valores possíveis	Descrição
Baudrate	DUART_BDR_110	Define o baud rate a utilizar.
	DUART_BDR_200	Admite valores entre 110bps e 57600 em conformidade com os valores standard.
	DUART_BDR_300	
	DUART_BDR_600	
	DUART_BDR_1050	
	DUART_BDR_1200	
	DUART_BDR_1800	
	DUART_BDR_2000	
	DUART_BDR_2400	
	DUART_BDR_4800	
	DUART_BDR_7200	
	DUART_BDR_9600	
	DUART_BDR_19200	
	DUART_BDR_28800	
	DUART_BDR_38400	
	DUART_BDR_57600	
Paridade	DUART_PARITY_NONE	Tipo de bit de paridade utilizado.
	DUART_PARITY_EVEN	
	DUART_PARITY_ODD	
Número de bits de dados	DUART_7_DATA_BITS	Define o número de bits a enviar/receber por carácter.
	DUART_8_DATA_BITS	
Número de bits de STOP	DUART_1_STOP_BIT	Define o número de bits de stop enviados/recebidos.
	DUART_2_STOP_BIT	

Exemplo de utilização:

```
DUART_vStart(BAUD_9600|DUART_PARITY_NONE|DUART_8_DATA_BITS |
DUART_1_STOP_BIT, DUART_A);
```

- ***void INT_viEx0(void) interrupt CC1_CC8INT***

Descrição: função de nível 0, é invocada sempre que exista um pedido de interrupção do periférico DUART. Nesta função são realizados os procedimentos para as comunicações com este periférico (envio e recepção de caracteres).

Parâmetros: nenhum.

Valor devolvido: nenhum.

• **BYTE DUART_vDetection(void)**

Descrição: função de nível 0, serve para detectar a existência e o bom funcionamento do periférico DUART no terminal. Esta função deve ser chamada antes da configuração dos canais pois se existir algum problema não se deve tentar realizar a iniciação do periférico.

Parâmetros: nenhum.

Valor devolvido:

- 0 : DUART detectada e com bom funcionamento
 ≠0 : DUART não detectada ou com problemas

A.5 Módulo “e2prom”

Definição	Descrição
E2PROM_SIZE	define o tamanho da e2prom
para a 24C02 – 256 bytes	0x00FF
para a 24C04 – 512 bytes	0x01FF
para a 24C08 – 1024 bytes	0x03FF
para a 24C16 – 2048 bytes	0x07FF

• **BYTE E2PROM_bRead(UINT uiAddress, void* ptrValue, BYTE bSize, BYTE bPermission)**

Descrição: função de nível 1, permite ler um conjunto de caracteres da memória eeprom, a partir do endereço indicado. A gama de endereços varia de 0 a 127 até 0 a 2047, dependendo da EEPROM utilizada. Antes da utilização desta função é necessário adquirir o direito de aceder à interface iic (através da invocação da função *I2C_bEnable*). Após a sua utilização deve ser invocada a função *I2C_bDisable* de modo a libertar o acesso à interface iic para os processos seguintes.

Parâmetros:

- uiAddress* : endereço da primeira posição a ler
**ptrValue* : apontador para a primeira posição de memória que irá receber a leitura
bSize : quantidade de caracteres a ler (máximo 255)
bPermission : E2P_CONFIG_ZONE - para ler na zona de configurações (esta zona pode ser definida no ficheiro e2prom.h)
 : E2P_USER_ZONE - para ler fora da zona de configurações

Valor devolvido:

- 0 : sucesso na leitura
 I2C_TRAP_ERROR : sem possibilidade de acesso ao recurso. Existe outro processo com o recurso ou não foi efectuado pedido de acesso ao recurso através da função *I2C_bEnable*
 1, 2, 3 : erros nas comunicações IIC

- **BYTE E2PROM_bWrite(UINT uiAddress, void* ptrValue, BYTE bSize, BYTE bPermission)**

Descrição: função de nível 1, permite escrever um conjunto de caracteres da memória EEPROM, a partir do endereço indicado. A gama de endereços varia de 0 a 127 até 0 a 2047, dependendo da EEPROM utilizada. Antes da utilização desta função é necessário adquirir o direito de aceder à interface iic (através da invocação da função *I2C_bEnable*). Após a sua utilização deve ser invocada a função *I2C_bDisable* de modo a libertar o acesso à interface iic para os processos seguintes.

Parâmetros:

<i>uiAddress</i>	: endereço da primeira posição a escrever
<i>*ptrValue</i>	: apontador para a primeira posição de memória onde estão os caracteres a escrever
<i>bSize</i>	: quantidade de caracteres a escrever (máximo 255)
<i>bPermission</i>	: E2P_CONFIG_ZONE - para escrever na zona de configurações (esta zona pode ser definida no ficheiro e2prom.h) : E2P_USER_ZONE - para escrever fora da zona de configurações

Valor devolvido:

0	: sucesso na escrita
I2C_TRAP_ERROR	: sem possibilidade de acesso ao recurso. Existe outro processo com o recurso ou não foi efectuado pedido de acesso ao recurso através da função <i>I2C_bEnable</i>
1, 2, 3	: erros nas comunicações IIC

A.6 Módulo “flash”

- **void FLASH_vReadSector(ULONG ulFirstAddress, BYTE *bData, BYTE bSize)**

Descrição: função de nível 0, permite ler um sector da memória flash, a partir do endereço indicado.

Parâmetros:

<i>ulAddress</i>	: endereço do sector a ler
<i>*bData</i>	: apontador para a primeira posição de memória que irá receber a leitura
<i>bSize</i>	: quantidade de caracteres a ler (máximo 256, bSize+1)

Valor devolvido:

0	: sucesso na leitura
FLASH_TIMEOUT_ERROR	: a <i>flash</i> não respondeu no tempo permitido

- **BYTE FLASH_bWriteSector(ULONG ulFirstAddress, BYTE *bData, BYTE bSize)**

Descrição: função de nível 0, permite escrever um sector da memória *flash*, a partir do endereço indicado.

Parâmetros:

<i>ulAddress</i>	: endereço do sector a escrever
<i>*bData</i>	: apontador para a primeira posição de memória onde se encontram os dados a escrever

bSize : quantidade de caracteres a escrever (máximo 256, *bSize*+1)

Valor devolvido:

0 : sucesso na escrita
FLASH_TIMEOUT_ERROR : a *flash* não respondeu no tempo permitido

- **BYTE FLASH_bEraseAll(void)**

Descrição: função de nível 0, permite apagar toda a memória.

Parâmetros: nenhum.

Valor devolvido:

0 : sucesso
FLASH_TIMEOUT_ERROR : a *flash* não respondeu no tempo permitido

A.7 Módulo “i2c”

- **void I2C_vInit(void)**

Descrição: função de nível 0, permite a iniciação do controlador iic interno ao microprocessador. Deve ser invocada nas iniciações do programa e antes de ser tentado qualquer acesso à interface iic através das funções de escrita e leitura da eeprom e do relógio de tempo real.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- **BYTE I2C_bEnable(void)**

Descrição: função de nível 0, que gere o acesso dos processos ao recurso iic. Esta função verifica se o recurso está livre e em caso afirmativo ocupa-o. Esta função deve ser chamada antes da invocação das funções de escrita e leitura na eeprom e no relógio de tempo real (RTC).

Parâmetros: nenhum.

Valor devolvido:

FALSE : recurso ocupado
TRUE : recurso livre anteriormente e colocado agora como ocupado

- **void I2C_vDisable(void)**

Descrição: função de nível 0, que serve para libertar o acesso ao recurso iic. Após a invocação desta função todos os processo vêm o recurso livre e podem tentar ocupa-lo (através do recurso à função *I2C_bEnable*).

Parâmetros: nenhum.

Valor devolvido: nenhum.

- **BYTE I2C_bAccess(BYTE bI2CCommand, BYTE bI2CDeviceAddress, UINT uiI2CSubAddress, BYTE *bI2CBuffer, UINT uiI2CBufferSize)**

Descrição: função de nível 0, que serve para aceder à interface iic para escrita ou leitura. Antes da sua utilização deve ser invocada a função *I2C_bEnable* para

determinar a possibilidade de acesso à interface e no fim deverá ser invocada a função *I2C_Disable* para libertar o acesso ao recurso.

Parâmetros:

<i>bI2CCommand</i>	: acesso pretendido IIC_WRITE ou IIC_READ (escrita ou leitura)
<i>bI2CDeviceAddress</i>	: endereço do dispositivo IIC E2PROM_SLAVE_ADDRESS ou CLOCK_SLAVE_ADDRESS
<i>uiI2CSubAddress</i>	: endereço que se pretende aceder dentro do dispositivo
<i>*bI2CBuffer</i>	: apontador para a primeira posição onde estão os caracteres a escrever ou onde colocar os caracteres a ler
<i>uiI2CBufferSize</i>	: quantidade de caracteres a escrever ou a ler

Valor devolvido:

0	: sucesso no acesso
1, 2, 3	: erros nas comunicações IIC

A.8 Módulo “io”

- *void ADC_vInit(void)*

Descrição: função de nível 0, serve para efectuar a configuração do ADC. Deve ser chamada nas iniciações do programa e antes da utilização do ADC.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- *BYTE ADC_bEnable(void)*

Descrição: função de nível 0, gere o acesso aos valores do ADC para cada entrada - não permite mais que um processo aceder aos valores do ADC em simultâneo. Deve ser chamada antes de qualquer acesso de leitura dos valores do ADC.

Parâmetros: nenhum.

Valor devolvido:

<i>TRUE</i>	: permite a leitura dos valores do ADC
<i>FALSE</i>	: não permite a leitura dos valores do ADC - outro processo a aceder ao recurso

- *void ADC_vDisable(void)*

Descrição: função de nível 0, liberta o acesso aos valores do ADC para cada entrada - não permite mais que um processo aceder aos valores do ADC em simultâneo. Deve ser chamada após leitura dos valores do ADC de modo a que outros processos possam aceder a este.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- *BYTE ADC_bKernelRead(void)*

Descrição: função de nível 0, realiza a leitura do valor do ADC de cada canal para a posição respectiva da variável global *uiADCValues[bChannel]*.

Parâmetros: nenhum.

Valor devolvido:

0 : sucesso
 !=0 : insucesso - o ADC está a ser acedido por outro processo

- **UINT ADC_uiRead(BYTE bChannel)**

Descrição: função de nível 0, permite ler um canal do ADC.

Parâmetros:

bChannel : indica a entrada que se pretende ler, podendo ter um dos valores de 0 a 7.

Valor devolvido:

0-1023 : valor do ADC

- **BYTE Input_bBehaviourConfig(BYTE bChannel, BYTE bBehaviour)**

Descrição: função de nível 0, permite configurar comportamento das entradas: digital ou analógico.

Parâmetros:

bChannel : Indica qual o canal de entrada a configurar. Valores de 0 a 7.
bBehaviour : Comportamento da entrada. 0: digital, 1: analógica

Valor devolvido: estado de configuração de todas as entradas.

- **BYTE InputDigital_bRead(BYTE bChannel)**

Descrição: função de nível 0, permite ler uma entrada digital.

Parâmetros:

bChannel : Indica qual o canal de entrada a ler. Valores de 0 a 7.

Valor devolvido:

0 : se a entrada estiver OFF
 1 : se a entrada estiver ON
 255 : função não executada devido a parâmetros fora dos limites

- **BYTE OutputDigital_bWrite(BYTE bChannel, BYTE bValue)**

Descrição: função de nível 0, permite actuar numa saída digital.

Parâmetros:

bChannel : Indica qual o canal de saída a escrever. Valores de 0 a 3.
bValue : Valor a escrever na saída. Valores 0 ou 1.

Valor devolvido:

0 : operação realizada com sucesso
 255 : função não executada devido a parâmetros fora dos limites

A.9 Módulo “kernel”

- **void IO_vInit(void)**

Descrição: função de nível 0, tem como objectivo efectuar a configuração de todos os pinos de entradas e saídas do microprocessador. Deve ser a primeira função chamada nas iniciações.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void GPT1_vInit(void)***

Descrição: função de nível 0, tem como objectivo efectuar iniciação da unidade número 1 dos temporizadores para uso geral (GPT1). São usados dois temporizadores com os seguintes objectivos:

O temporizador número 3 é usado provocar uma interrupção com a periodicidade de 26 ms de modo a se realizarem diversas tarefas:

- actualização da informação do *display*
- leitura e filtragem do teclado
- leitura dos canais do ADC
- leitura dos relógio de tempo real (RTC)
- actualização dos registo que provocam a intermitência dos caracteres no *display* e o som do besouro

O temporizador número 2 é usado pela função *vDelay* de modo a se conseguir obter pausas com o máximo de precisão possível e independente da frequência de oscilação escolhida.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void KERNEL_vStart(void)***

Descrição: função de nível 0, tem como objectivo efectuar todas as iniciações necessárias da aplicação.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void GPT1_viTmr3(void) interrupt T3INT***

Descrição: função de nível 0, rotina de serviço à interrupção do temporizador número 3 da unidade GPT1. Esta função é invocada com a periodicidade de 26 ms e nela são realizadas as seguintes acções:

- actualização da informação do *display*
- leitura e filtragem do teclado
- leitura dos canais do ADC
- leitura dos relógio de tempo real (RTC)
- actualização dos registo que provocam a intermitência dos caracteres no *display* e o som do besouro

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void GPT1_viTmr2(void) interrupt T2INT***

Descrição: função de nível 0, rotina de serviço à interrupção do temporizador número 2 da unidade GPT1. Esta função é invocada com a periodicidade de 1.5 ms e serve de apoio à função *vDelay* para determinação das pausas no processamento.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void MAIN_vUnlockProtecReg(void)***

Descrição: função de nível 0, tem como função permitir a escrita em registos protegidos. Certos registos especiais do microprocessador estão protegidos contra



escritas acidentais. Para aceder a este registos é necessário efectuar vários procedimentos, os quais são realizados nesta função.

Parâmetros: nenhum

Valor devolvido: nenhum.

- ***void vError(BYTE bCodigoDeErro)***

Descrição: função de nível 1, serve de recolha dos erros fatais e fornece ao programador a sua causa através de afixação no *display*. Nesta função o processamento do programa em execução é parado e deverá se reiniciar o terminal (*power-off/power-on*) de modo a iniciar novamente a execução da aplicação.

Parâmetros:

bCodigoDeErro : ver Tabela 13-A

Valor devolvido: nenhum.

Tabela 13-A - Códigos de erro do kernel

Valor	Definição	Descrição
0	0-OUT OF DISPLAY	
1	1-NO I2C ENABLE	

- ***void vBeep (BYTE bDuration , BYTE bType)***

Descrição: função de nível 1, tem como objectivo iniciar um aviso sonoro.

Parâmetros:

bDuration : Duração do Aviso. Cada unidade representa 110 ms.

bType : BEEP_CONTINUED: som continuo

BEEP_BLINK: som intermitente

Valor devolvido: nenhum.

- ***void vDelay(UINT uiDelayFactor)***

Descrição: função de nível 1, tem como objectivo provocar uma pausa no processamento da aplicação.

Parâmetros:

uiDelayFactor : Duração da pausa. Cada unidade corresponde a 1.5 ms

Valor devolvido: nenhum.

A.10 Módulo “keyboard”

Tabela 14-A - Definições do módulo keyed

Item	Definição	Descrição
1	BEEP_CONTINUED	opção do parâmetro bType; aviso sonoro continuo
2	BEEP_BLINK	opção do parâmetro bType; aviso sonoro intermitente

3	LER_TIPO_NORMAL	
4	LER_TIPO_PASSWORD	
5	KEY_LEFT	
6	KEY_DELETE	
7	KEY_UP	
8	KEY_DOWN	
9	KEY_RIGHT	
10	KEY_ESC	
11	KEY_OK	
12	KEY_AZ	
13	KEY_FN	
14	KEY_POINT	
15	KEY_SHIFT	
16	KEY_x	x: 0-9
17	KEY_Fx	x: 0-20

• **ULONG KEYBOARD_ulRead(void)**

Descrição: função de nível 0, destina-se a fazer o barrimento do teclado para determinação das teclas premidas.

Parâmetros: nenhum.

Valor devolvido: A cada bit do longo sem sinal corresponde a uma tecla (ver Tabela 15-A)

Tabela 15-A - Matriz do teclado

bit	Tecla
0	KEY_ESC
1	KEY_1
2	KEY_6
3	não usado
4	KEY_LEFT
5	KEY_2
6	KEY_7
7	não usado
8	KEY_UP
9	KEY_3
10	KEY_8
11	não usado
12	KEY_DOWN
13	KEY_4
14	KEY_9
15	não usado
16	KEY_RIGHT
17	KEY_5
18	KEY_0
19	não usado
20	KEY_OK
21	KEY_POINT
22	KEY_AZ

23	não usado
24	KEY_FN
25	não usado
26	não usado
27	KEY_SHIFT
28	não usado
29	não usado
30	não usado
31	não usado

- **BYTE KEYBOARD_bHit(void)**

Descrição: função de nível 0, não espera que o utilizador actue o teclado, devolvendo o código da tecla que estiver actuada no momento da chamada à função. Realiza procedimentos de filtragem às teclas premidas.

Parâmetros: nenhum.

Valor devolvido: Devolve o código da tecla actuada. Quando não está nenhuma tecla actuada é devolvido o valor zero. O código que cada tecla devolve está definido na Tabela 14-A itens 7 a 15.

- **BYTE KEYBOARD_bGetKey(BYTE bKeyboardTimeout, BYTE bKey)**

Descrição: função de nível 1, destina-se a ler uma tecla premida, esperando que o utilizador actue no teclado.

Parâmetros: nenhum.

Valor devolvido: Devolve o código da tecla actuada.

- **void KEYBOARD_vWaitKeyRelease(void)**

Descrição: função de nível 1, espera a libertação de uma tecla premida.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- **BYTE KEYBOARD_bReadBYTE(BYTE bLine, BYTE bColumn, BYTE *bData, BYTE bSize, BYTE bOptions)**

Descrição: função de nível 2, que permite a introdução de um número do tipo carácter (byte) utilizando o teclado numérico. A tecla 'KEY_DELETE' apaga um algarismo e as teclas 'KEY_OK' e 'KEY_ESC' terminam a introdução do número.

Parâmetros:

bLine : define a linha do *display*.

bColumn : define a coluna do *display*.

**bData* : é um apontador para a variável que vai receber o número introduzido.

bSize : número de dígitos esperados

bOptions : pode tomar um dos valores das definições dos itens de 1 a 7 da Tabela 14-A

Valor devolvido:

0 : se premida KEY_OK após introdução de bSize caracteres

1 : se premida KEY_OK sem a introdução de bSize caracteres

2 : se premida KEY_ESC

- **BYTE KEYBOARD_bReadUINT**(*BYTE bLine, BYTE bColumn, UINT *uiData, BYTE bSize, BYTE bOptions*)

Descrição: função de nível 2, funciona de forma idêntica à função *bReadBYTE*, a diferença é que permite a introdução de um número do tipo inteiro sem sinal (*unsigned int*) em lugar do carácter (byte).

- **BYTE KEYBOARD_bReadULONG**(*BYTE bLine, BYTE bColumn, ULONG *ulData, BYTE bSize, BYTE bOptions*)

Descrição: função de nível 2, funciona de forma idêntica à função *bReadBYTE*, a diferença é que permite a introdução de um número do tipo longo sem sinal (*unsigned int*) em lugar do carácter (byte).

- **BYTE KEYBOARD_bReadString**(*BYTE bLine, BYTE bColumn, BYTE *ApValue, BYTE bSize, BYTE bOptions*)

Descrição: função de nível 2, permite a introdução de uma conjunto de caracteres do teclado, sob a forma de uma *string*. A tecla 'KEY_DELETE' apaga um algarismo e as teclas 'KEY_OK' e 'KEY_ESC' terminam a introdução do número. A *string* é sempre terminada com carácter '\0'.

Parâmetros:

bLine : define a linha do *display*.
bColumn : define a coluna do *display*.
**bData* : é um apontador para a variável que vai receber o número introduzido.
bSize : número de dígitos esperados
bOptions : pode tomar um dos valores das definições dos itens de 1 a 7 da Tabela 14-A

Valor devolvido:

0 : se premida KEY_OK após introdução de *bSize* caracteres
 1 : se premida KEY_OK sem a introdução de *bSize* caracteres
 2 : se premida KEY_ESC

A.11 Módulo “mmcard”

- **BYTE CARD_bInit**(*void*)

Descrição: função de nível 1, que realiza a iniciação do cartão.

Parâmetros: nenhum.

Valor devolvido:

0 : inicialização correcta
 ≠0 : erro na inicialização; o cartão pode não estar colocado no terminal

- **BYTE MMC_bCommand**(*BYTE bCommand, ULONG ulParameter, BYTE bCrc*)

Descrição: função de nível 1, que permite o envio de comandos para o cartão.

Parâmetros:

bCommand : comando a enviar
ulParameter : parâmetros do comando (4 caracteres)
bCrc : carácter de controlo

Valor devolvido:

255 : erro de transmissão
 ≠255 : transmissão correcta

- **BYTE MMC_bStatus(void *Data)**

Descrição: função de nível 1, que permite ler o estado do cartão. O estado é constituído por dois caracteres.

Parâmetros:

*Data : apontador para a primeira posição onde guardar o estado

Valor devolvido:

0 : recepção de estado correcta
 ≠0 : erro na recepção do estado

- **BYTE MMC_bReadSector(void *Data, ULONG ulSector)**

Descrição: função de nível 1, que permite a leitura de sectores do cartão.

Parâmetros:

*Data : apontador para a primeira posição onde colocar os dados lidos do sector

ulSector : número do sector a ler

Valor devolvido:

0 : leitura correcta
 ≠0 : erro na leitura

- **BYTE MMC_bWriteSector(void *Data, ULONG ulSector)**

Descrição: função de nível 1, que permite a escrita de sectores do cartão.

Parâmetros:

*Data : apontador para a primeira posição onde se encontram os dados a escrever no sector

ulSector : número do sector a escrever

Valor devolvido:

0 : escrita correcta
 ≠0 : erro na escrita

- **BYTE CARD_bReadSector(ULONG ulSector)**

Descrição: função de nível 2, tem como objectivo efectuar a leitura de um sector do cartão para o *CardBuffer*. Só efectua a leitura do *ulSector* se este for diferente do sector no *buffer*.

Parâmetros:

ulSector : número do sector a ler

Valor devolvido:

0 : leitura correcta
 ≠0 : erro na leitura

- **BYTE CARD_bWriteSector(ULONG ulSector)**

Descrição: função de nível 2, tem como objectivo efectuar a escrita de um sector do *CardBuffer* para o cartão. Só efectua a escrita do *ulSector* se este se encontrar no *buffer*.

Parâmetros:

ulSector : número do sector a escrever

Valor devolvido:

0 : escrita correcta
 ≠0 : erro na escrita

- **BYTE CARD_bCheckCID(void)**

Descrição: função de nível 1, que permite verificar se o CID actual é igual ao anterior. (descrever o cid)

Parâmetros: nenhum.

Valor devolvido:

0 : CID correcto
≠0 : erro no CID

- **BYTE CARD_bGetCSD(BYTE *bData)**

Descrição: função de nível 1, que permite ler o CSD do cartão. (descrever o csd)

O CSD é composto por 16 bytes

Parâmetros:

*bData : apontador para a primeira posição onde guardar o CSD

Valor devolvido:

0 : CSD lido
≠0 : erro na leitura do CSD

- **BYTE MMC_bWrite(ULONG ulEndereco, void *vData, UINT uiSize)**

Descrição: função de nível 2, que permite a escrita de apenas um carácter (*byte*) no cartão.

Parâmetros:

ulEndereco : primeiro endereço dos caracteres a escrever

*vData : apontador para a primeira posição onde se encontram os dados a escrever no cartão

uiSize : número de caracteres a escrever

Valor devolvido:

0 : escrita correcta
≠0 : erro na escrita

- **BYTE MMC_bRead(ULONG ulEndereco, void *vData, UINT uiSize)**

Descrição: função de nível 2, que permite a leitura de apenas um carácter (*byte*) do cartão.

Parâmetros:

ulEndereco : primeiro endereço dos caracteres a ler

*vData : apontador para a primeira posição onde colocar os dados a ler do cartão

uiSize : número de caracteres a ler

Valor devolvido:

0 : leitura correcta
≠0 : erro na leitura

A.12 Módulo “mystring”

- **void vAddULONG2HexString(BYTE bMsg[], UINT uiPosition, BYTE bPlaces, ULONG ulValue)**

Descrição: função de nível 2, permite inserir um valor do tipo longo sem sinal de *bPlaces* dígitos, na posição *uiPosition* da *string* *bMsg* em formato hexadecimal.

Parâmetros:

bMsg[] : *string* onde será inserido o valor.
uiPosition : posição dentro da *string* onde se pretende inserir o valor.
bPlaces : número de casas decimais do valor a inserir.
ulValue : valor a inserir.

Valor devolvido: Nenhum.

- **char *btoa(void *String, BYTE bNumber, BYTE bDigits, BIT Zeros)**

Descrição: função de nível 2, permite converter um valor do tipo carácter (byte) numa *string*.

Parâmetros:

**String* : apontador para a *string* onde colocar o resultado
bNumber : carácter que se pretende converter
bDigits : número de dígitos que se pretende converter (começa pelo menos significativo)
Zeros : 1- substitui os zeros à esquerda do número pelo carácter espaço; 0- mantém os zeros

Valor devolvido: apontador para a *string* que contem o resultado

- **char *uitoa(void *String, UINT uiNumber, BYTE bDigits, BIT Zeros)**

Descrição: função de nível 2, permite converter um valor do tipo inteiro sem sinal (*unsigned int*) numa *string*

Parâmetros:

**String* : apontador para a *string* onde colocar o resultado
uiNumber : inteiro sem sinal que se pretende converter
bDigits : número de dígitos que se pretende converter (começa pelo menos significativo)
Zeros : 1- substitui os zeros à esquerda do número pelo carácter espaço; 0- mantém os zeros

Valor devolvido: apontador para a *string* que contem o resultado

- **char *ultoa(void *String, ULONG ulNumber, BYTE bDigits, BIT Zeros)**

Descrição: função de nível 2, permite converter um valor do tipo longo sem sinal (*unsigned long*) numa *string*

Parâmetros:

**String* : apontador para a *string* onde colocar o resultado
ulNumber : longo sem sinal que se pretende converter
bDigits : número de dígitos que se pretende converter (começa pelo menos significativo)
Zeros : 1- substitui os zeros à esquerda do número pelo carácter espaço; 0- mantém os zeros

Valor devolvido: apontador para a *string* que contem o resultado

- ***char *ZeroClean(void *String)***

Descrição: função de nível 2, permite substituir os zeros à esquerda numa *string* pelo carácter espaço

Parâmetros:

**String* : apontador para a *string* a tratar

Valor devolvido: apontador para a *string* que contem resultado

- ***BYTE *bDec2Bin(BYTE bValue, BYTE *bResult)***

Descrição: função de nível 2, permite converter um valor do tipo carácter (*byte*) do formato decimal para o formato binário

Parâmetros:

bValue : valor a converter

**bResult* : apontador para a *string* com o resultado

Valor devolvido: apontador para a *string* que contem o resultado

- ***BYTE bStringisdigit(BYTE *ptrString)***

Descrição: função de nível 2, permite testar se a *string* é composta por dígitos decimais '0'-'9'. Termina a pesquisa quando encontra o carácter NULL.

Parâmetros:

**ptrString* : apontador para a *string* a testar

Valor devolvido:

1 : *string* composta apenas por dígitos decimais '0'-'9'

0 : *string* contem dígitos não decimais

- ***BYTE bNStringisdigit(BYTE *ptrString, BYTE bQty)***

Descrição: função de nível 2, permite testar se *n* caracteres da *string* são dígitos decimais '0'-'9'. A pesquisa começa no primeiro carácter da *string*

Parâmetros:

**ptrString* : apontador para a *string* a testar

bQty : número de caracteres a testar

Valor devolvido:

1 : *string* composta apenas por dígitos decimais '0'-'9'

0 : *string* contem dígitos não decimais

- ***BYTE bStringExtract(BYTE *strOrig, BYTE *strDest, BYTE *strFirst, BYTE *strLast)***

Descrição: função de nível 2, permite extrair da *string* apontada por *strOrig* a primeira *string* encontrada que esteja compreendida entre a *string* apontada por *strFirst* e a *string* apontada por *strLast*.

Parâmetros:

**strOrig* : apontador para a *string* onde procurar

**strDest* : apontador para a *string* onde colocar o resultado (caso exista)

**strFirst* : apontador para a *string* limite inicial

**strLast* : apontador para a *string* limite final

Valor devolvido:

0 : sucesso - *string* extraída

1 : não encontrou *strLast* na *strOrig*

2 : não encontrou *strFirst* na *strOrig*

A.13 Módulo “rtc_ext”

```
typedef struct TYPETIME {
    BYTE Second;
    BYTE Minute;
    BYTE Hour;
    BYTE Day;
    BYTE Month;
    BYTE WeekDay;
    BYTE Year;
}TypeTime;
```

- **BYTE RTC_EXT_bGetTime(void)**

Descrição: função de nível 2, permite ler todos os registos de data e hora do relógio (7 registos 0-6) para a variável global *RTC_EXT_Time*. Antes de utilizar esta função é necessário chamar a função *I2C_bEnable()*. Após a chamada desta função deve-se invocar a função *I2C_vDisable()* para libertar o acesso ao i2c.

Parâmetros: nenhum.

Valor devolvido:

= 0 : sucesso - é actualizada a variável *RTC_EXT_Time*
 ≠ 0 : insucesso - não é actualizada a variável *RTC_EXT_Time*

- **BYTE RTC_EXT_bPutTime(TypeTime NewTime)**

Descrição: função de nível 2, permite escrever todos os registos do relógio de data e hora (7 registos 0-6) a partir da variável *NewTime*.

Parâmetros:

* *bNewTime* : apontador para a estrutura a ler do tipo *TypeTime*

Valor devolvido:

= 0 : sucesso
 ≠ 0 : insucesso

- **void RTC_EXT_vAdjust(void)**

Descrição: função de nível 2, permite acertar o relógio de tempo real externo a partir da introdução dos valores (ano, mês, dia, hora, minutos, segundos) a partir do teclado.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- **BYTE bCalcWeekDay(UINT yr, BYTE mo, BYTE day)**

Descrição: função de nível 2, permite calcular o dia da semana em função da data.

Parâmetros:

Yr : indica o ano em que se pretende encontrar o dia da semana.

Mo : indica o mês em que se pretende encontrar o dia da semana.

Day : indica o dia em que se pretende encontrar o dia da semana.

Valor devolvido:

: dia da semana; 0=Dom., 1=Seg., 2=Ter, 3=Qua., 4=Qui., 5=Sex.,
 6=Sab.

- **BYTE bGetSunday(BYTE bSunday, BYTE bMonth, BYTE bYear)**

Descrição: função de nível 2, permite calcular o dia do mês em que acontece um determinado domingo. Os anos de 98 e 99 correspondem ao século XX, os restantes ao século XXI.

Parâmetros:

- bSunday*

: indica qual número do domingo que se deseja determinar o dia (1/2/3/4/5 para 1º/2º/3º/4º/5º domingo do mês; maior que 5 determina o dia do último domingo do mês).
- bMonth*

: indica o mês em que se pretende encontrar o domingo.
- bYear*

: indica o ano em que se pretende encontrar o domingo.

Valor devolvido:

- = 0*

: se houve erro
- ≠ 0*

: o dia do mês correspondente ao domingo

Tabela 16-A - Mapa de registos do relógio

Endereço	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Grandeza	Gama
0	ST								Segundos	00-59
1	X								Minutos	00-59
2	X	X							Hora	00-23
3	X	X	X	X	X				Dia da Semana	01-07
4	X	X							Dia	01-31
5	X	X							Mês	01-12
6									Ano	00-99
7	OUT	FT	S	C4	C3	C2	C1	C0	Controlo	-----
8-63									Livres	0-FF

O bit ST permite parar o relógio. Os bits C4 a C0 destinam-se a corrigir a precisão do relógio. O bit S indica o sinal da correcção (‘1’ indica acelerar o relógio e ‘0’ abrandar). Cada unidade positiva representa um aumento de 10.7s por mês e cada unidade negativa uma diminuição de 5.35s por mês. O bit FT permite colocar na saída de teste do relógio uma onda quadrada para verificar a precisão do relógio.

Quando o terminal está desligado, o relógio é alimentado pela pilha de lítio existente. Nas funções de escrever ou ler um byte do relógio são efectuadas respectivamente cinco ou três tentativas se a operação não for conseguida imediatamente.

A.14 Módulo “serial”

Tabela 17-A - Velocidades disponíveis nas comunicações RS232

Baud Rate	Definição
1200 bps	ASC_BDR_1200
2400 bps	ASC_BDR_2400
4800 bps	ASC_BDR_4800
9600 bps	ASC_BDR_9600
19200 bps	ASC_BDR_19200
57600 bps	ASC_BDR_57600
115200 bps	ASC_BDR_115200

- `void ASC0_vInit(BYTE bBaudRate)`

Descrição: função de nível 0, permite configurar a porta série 0 do terminal. As velocidades (*baud rates*) disponíveis são os presentes na Tabela 17-A. São sempre utilizados oito bits de dados, um stop bit e sem bit de paridade.

Parâmetros:

bBaudRate : selecciona o baud rate pretendido. Consultar a Tabela 17-A.

Valor devolvido: nenhum.

- ***void ASCI_vInit(BYTE bBaudRate)***

Descrição: função de nível 0, permite configurar a porta série 1 do terminal. As velocidades (*baud rates*) disponíveis são os presentes na Tabela 17-A. São sempre utilizados oito bits de dados, um stop bit e sem bit de paridade.

Parâmetros:

bBaudRate : selecciona o baud rate pretendido. Consultar a Tabela 17-A.

Valor devolvido: nenhum.

- ***void ASC0_Receive(void) interrupt ASC0_RINT***

Descrição: função de nível 0, serviço à interrupção de recepção da porta série 0. É chamada sempre que é recebido qualquer dado.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void ASC0_Transmit(void) interrupt ASC0_TINT***

Descrição: função de nível 0, serviço à interrupção de transmissão da porta série 0. É chamada quando termina o envio de um dado.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void ASC1_Receive(void) interrupt ASC1_RINT***

Descrição: função de nível 0, serviço à interrupção de recepção da porta série 1. É chamada sempre que é recebido qualquer dado.

Parâmetros: nenhum.

Valor devolvido: nenhum.

- ***void ASC1_Transmit(void) interrupt ASC1_TINT***

Descrição: função de nível 0, serviço à interrupção de transmissão da porta série 1. É chamada quando termina o envio de um dado.

Parâmetros: nenhum.

Valor devolvido: nenhum.

A.15 Módulo “spi”

- ***void SSC0_vInit(void)***

Descrição: função de nível 0, permite configurar a unidade de comunicação síncrona de alta velocidade. Esta unidade é configurada para implementar o protocolo spi para comunicação com o cartão de memória (*multimedia card*).

Parâmetros: nenhum.

Valor devolvido: nenhum.

- **BYTE SPI_bSend(void *Buffer, UINT uiSize)**

Descrição: função de nível 0, permite enviar caracteres usando o protocolo spi.

Parâmetros:

**Buffer* : apontador para a primeira posição onde se encontram os caracteres a enviar

uiSize : quantidade de caracteres a enviar

Valor devolvido:

0 : envio bem sucedido

≠0 : falha no envio

- **BYTE SPI_bReceive(void *Buffer, UINT uiSize, BYTE bOut)**

Descrição: função de nível 0, permite receber caracteres usando o protocolo spi. Como o protocolo está a ser usado em modo *full-duplex* (comunicação simultânea nos dois sentidos) é necessário enviar um carácter para receber.

Parâmetros:

**Buffer* : apontador para a primeira posição onde colocar os caracteres a receber

uiSize : quantidade de caracteres a receber

bOut : carácter a colocar na interface de modo a efectuar a leitura

Valor devolvido:

0 : recepção bem sucedida

≠0 : falha na recepção

- **BYTE SPI_bDummy(BYTE bData, BYTE bOut, BIT bitTipo)**

Descrição: função de nível 0, tem como objectivo aguardar a recepção de determinado carácter. São efectuadas um número pré-determinado de tentativas antes do abandono da função.

Parâmetros:

bData : carácter que se espera receber

bOut : carácter a enviar para desencadear a recepção

bitTipo : 0 - aguardar o primeiro carácter igual a *bData*

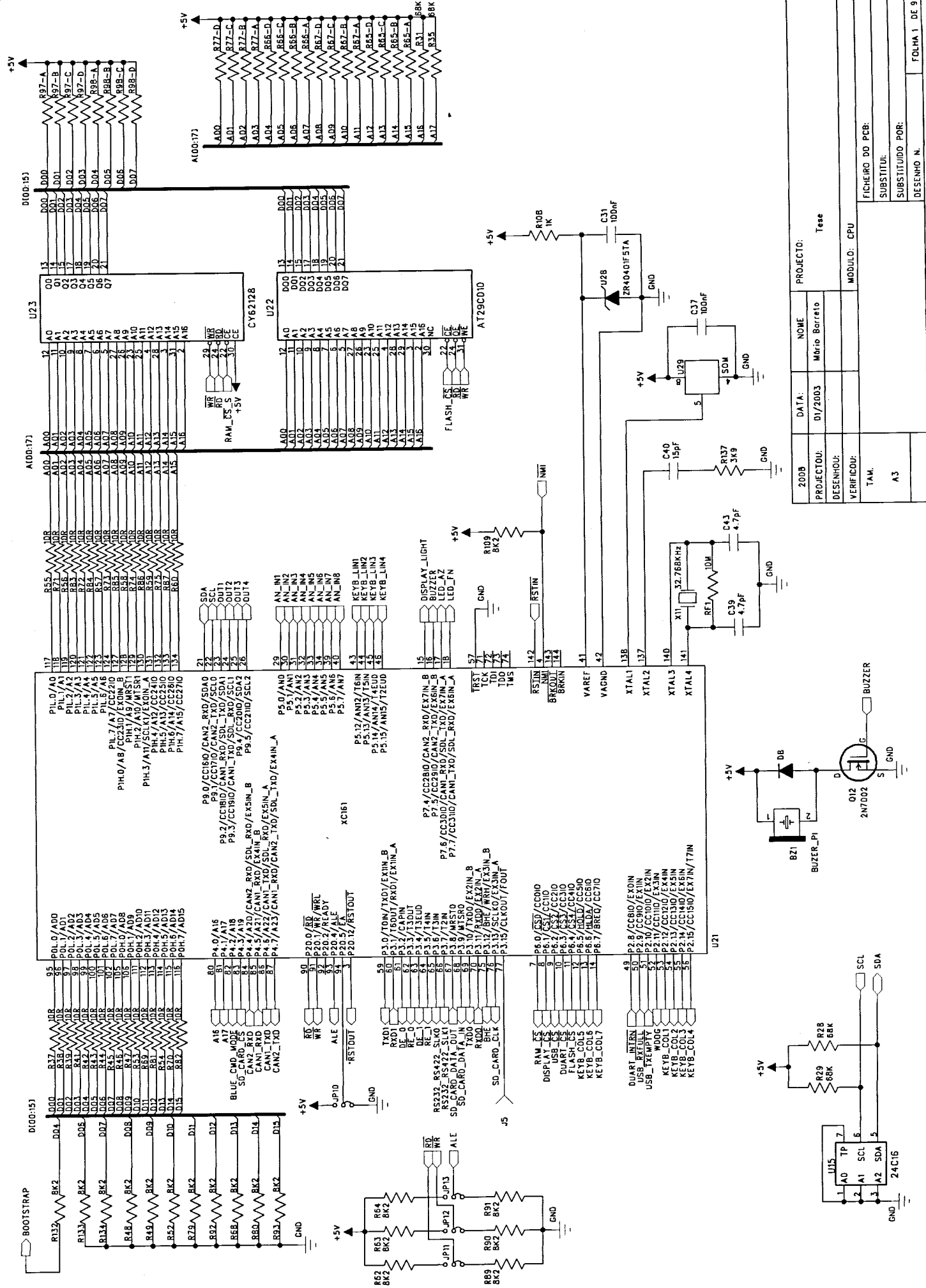
: 1 - aguardar o primeiro carácter diferente de *bData*

Valor devolvido:

0 : operação bem sucedida

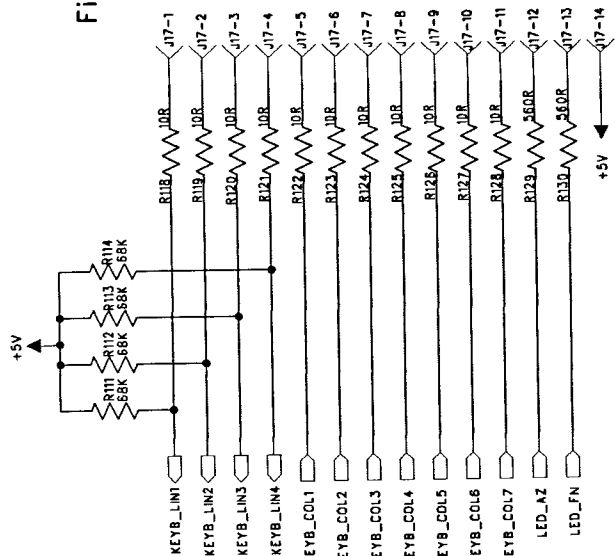
≠0 : operação mal sucedida

Anexo B – Desenho do esquemático

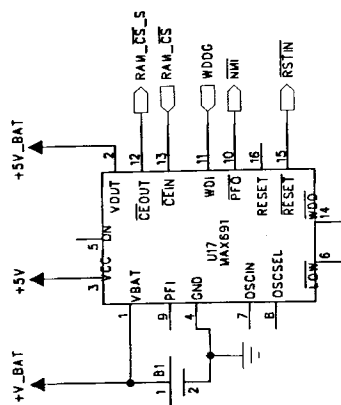
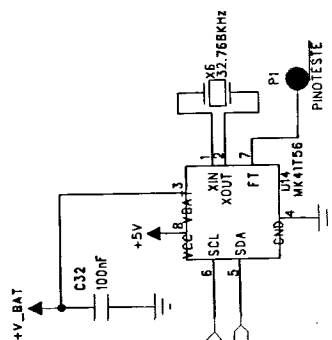
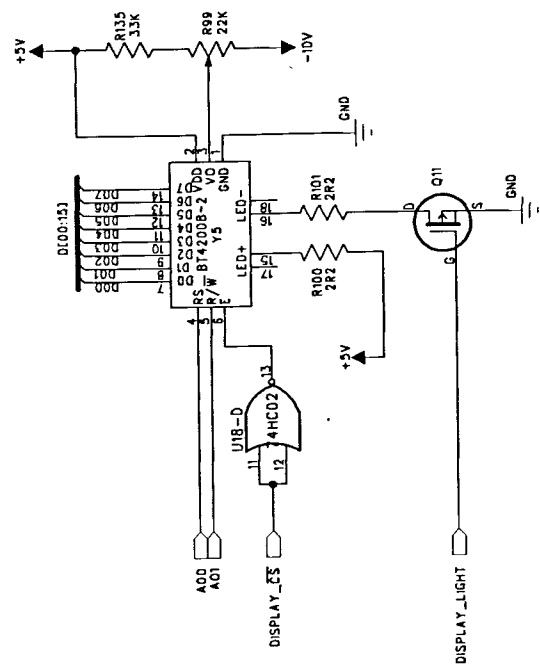


2008	DATA:	NOME	PROJECTO:	
PROJECTO:	01/2003	Mário Barreto	Tese	
DESENVOL:				
VERIFICOU:			MÓDULO: CPU	
TAM.				FICHEIRO DO PCB:
A3				SUBSTITUI:
				SUBSTITUÍDO POR:
				DESENHOU N.º
				FOLHA 1 DE 9

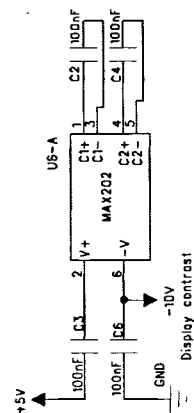
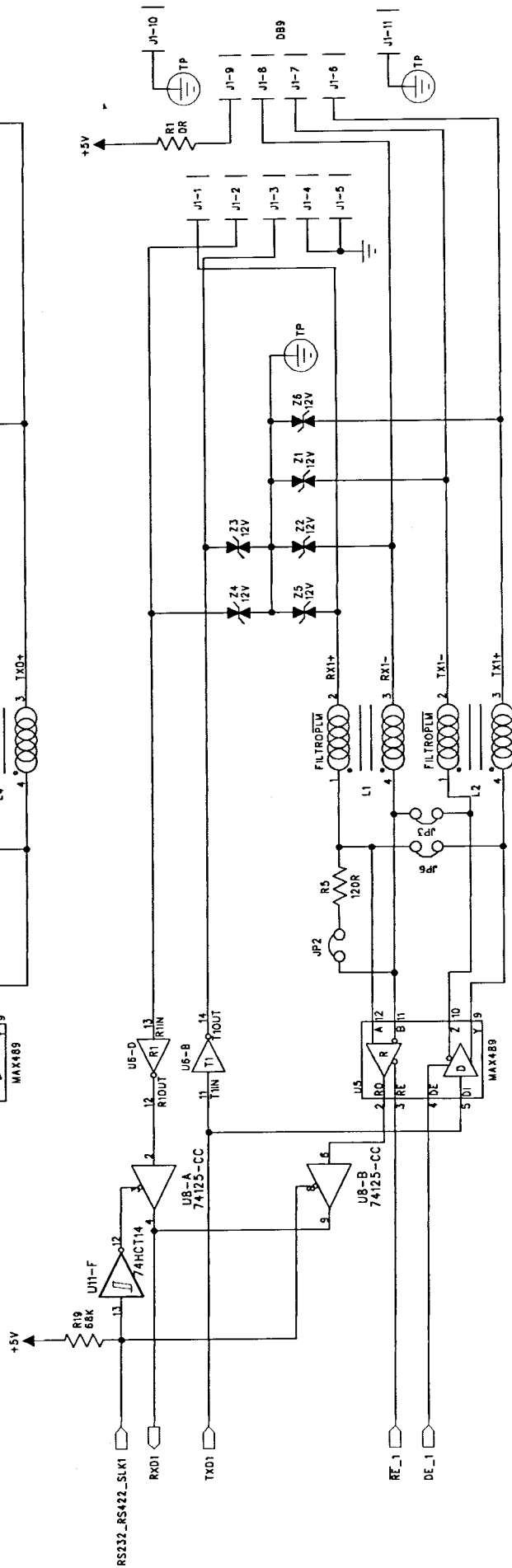
Linha 1
Linha 2
Linha 3
Linha 4
Coluna
Coluna
Coluna
Coluna
Coluna
Coluna
Coluna
Led AZ
Led FN
Led Co



Pino 1 fica à esquerda

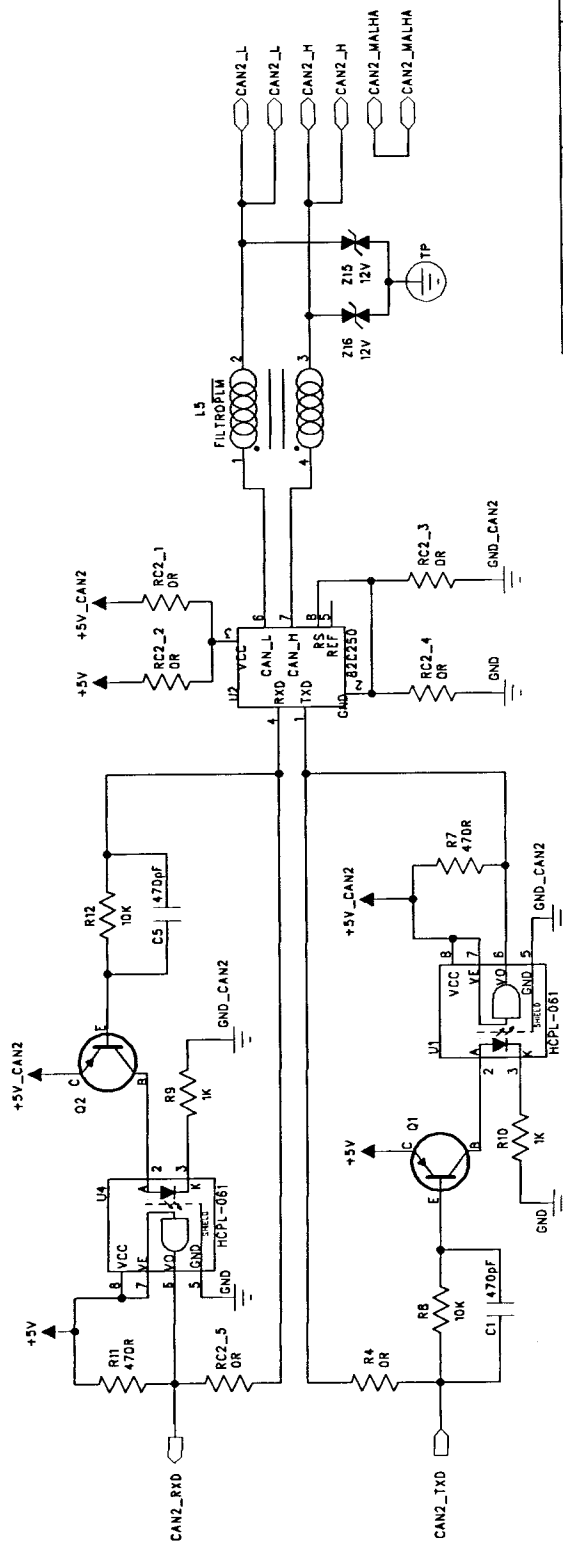


2009		DATA:	NOME	PROJECTO:	Tese
PROJECTOU:		01/2003	Mário Barreto		
DESENHO:					
VERIFICOU:				MODULO: DISPLAY+KEYBOARD	
TAM.				FICHEIRO DO PCB:	
A3				SUBSTITUTE:	
				SUBSTITUIDO POR:	
				DESENHO N.	FOLHA 2 DE 9

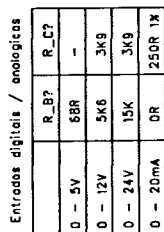
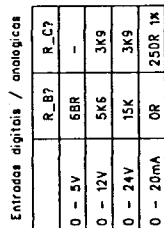


Display contrast

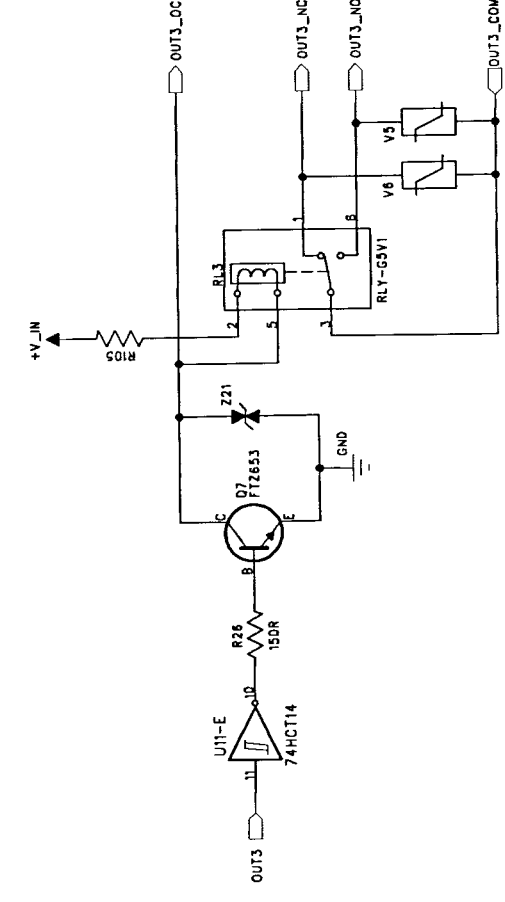
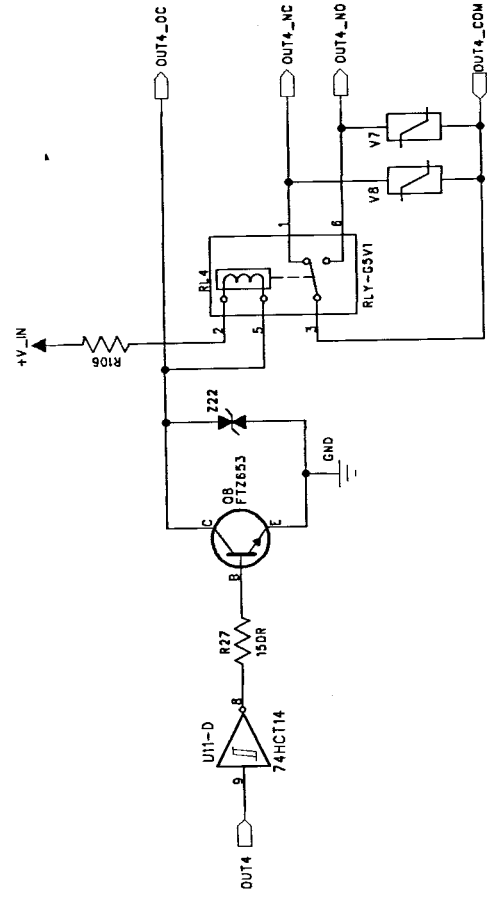
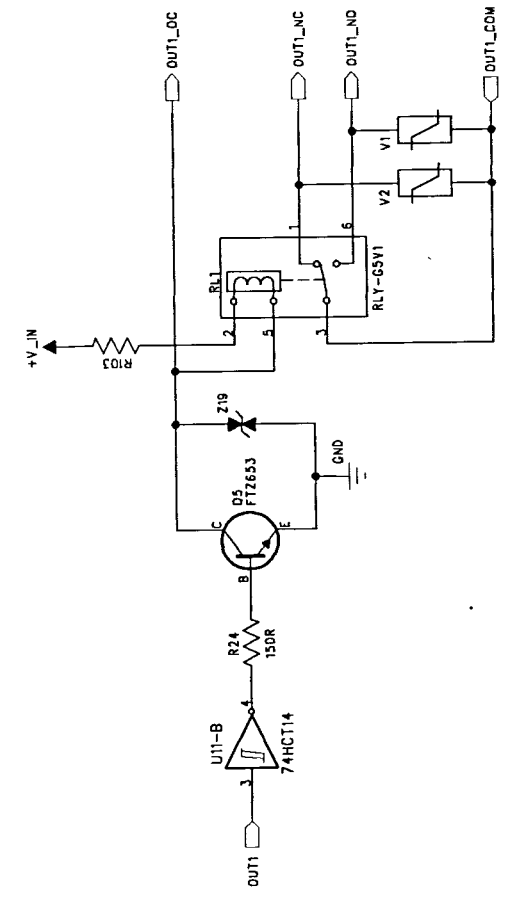
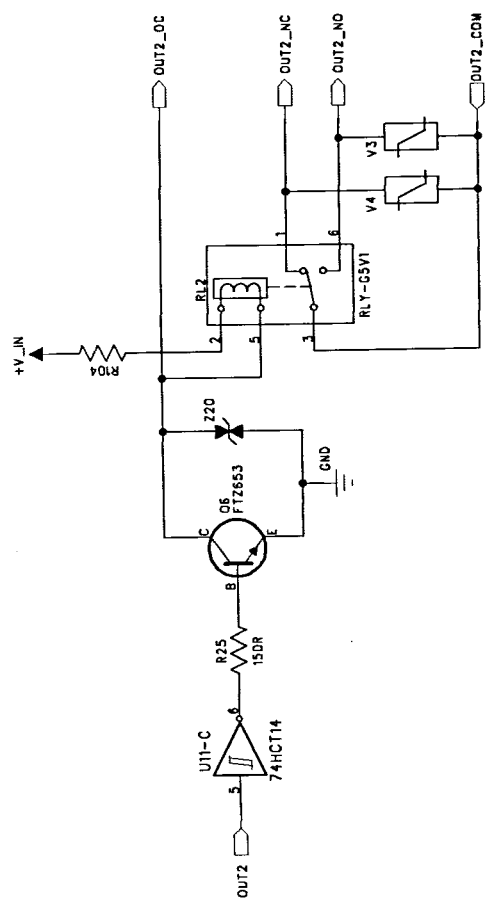
	R_C2_1	R_C2_2	R_C2_3	R_C2_4	R_C2_5
CAN2					
sem isolamento	DR	DR	open	DR	open
com isolamento	open	open	DR	open	DR



FICHEIRO DO PCB:	
SUBSTITUI:	
SUBSTITUIDO POR:	
DESENHO N.	

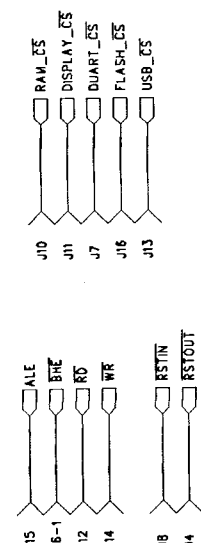
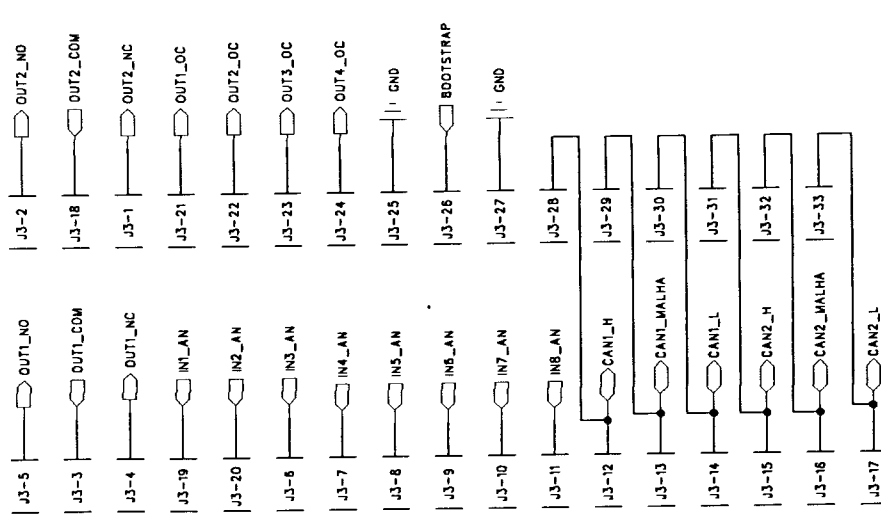
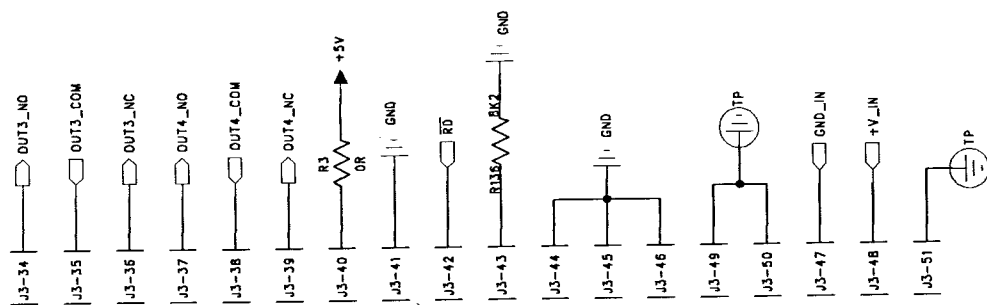
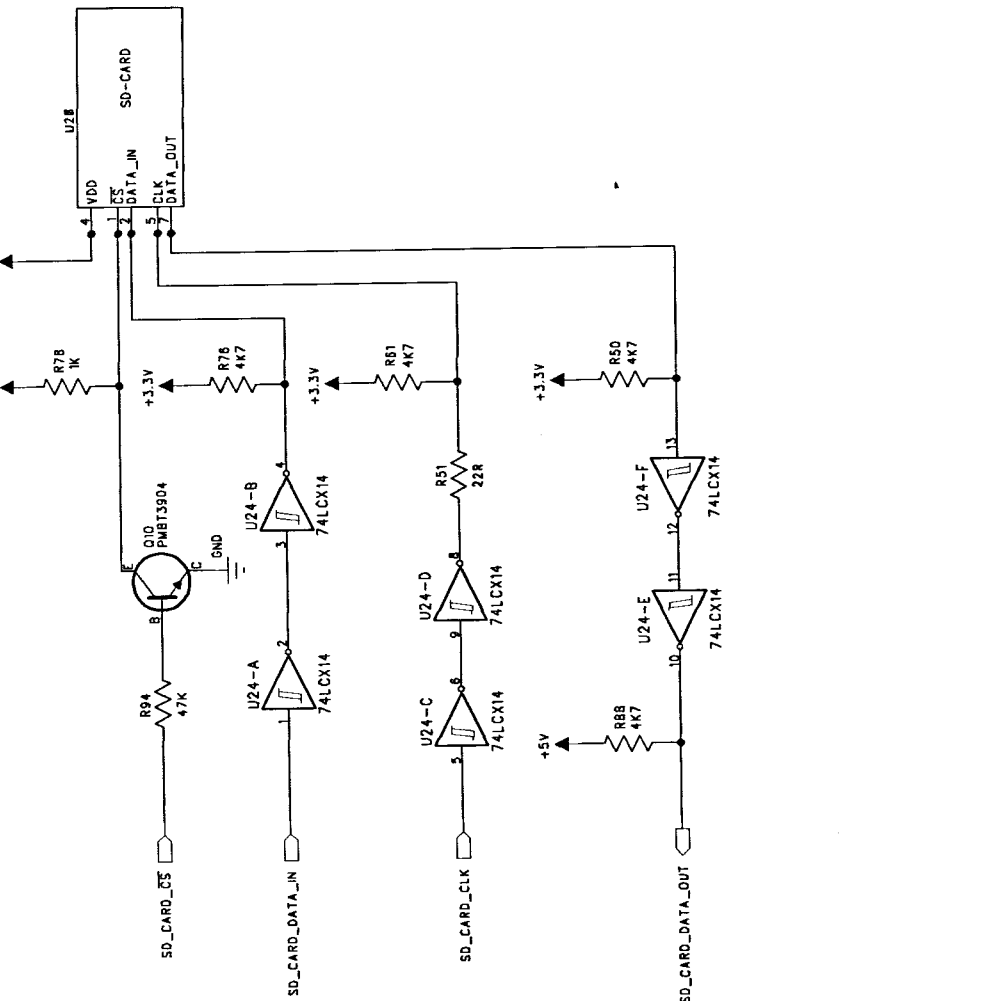


2008	DATA:	NOME	PROJECTO: Tese
PROJECTO:	DI/2003	Mário Barreto	
DESENHO:			
VERIFICOU:			
TAM.	MODULO: INPUTS		
	FICHERO DO PCB:		
	SUBSTITUI:		
A3	SUBSTITUIDO POR:		
	DESENHO N.		FOLHA 5 DE 9

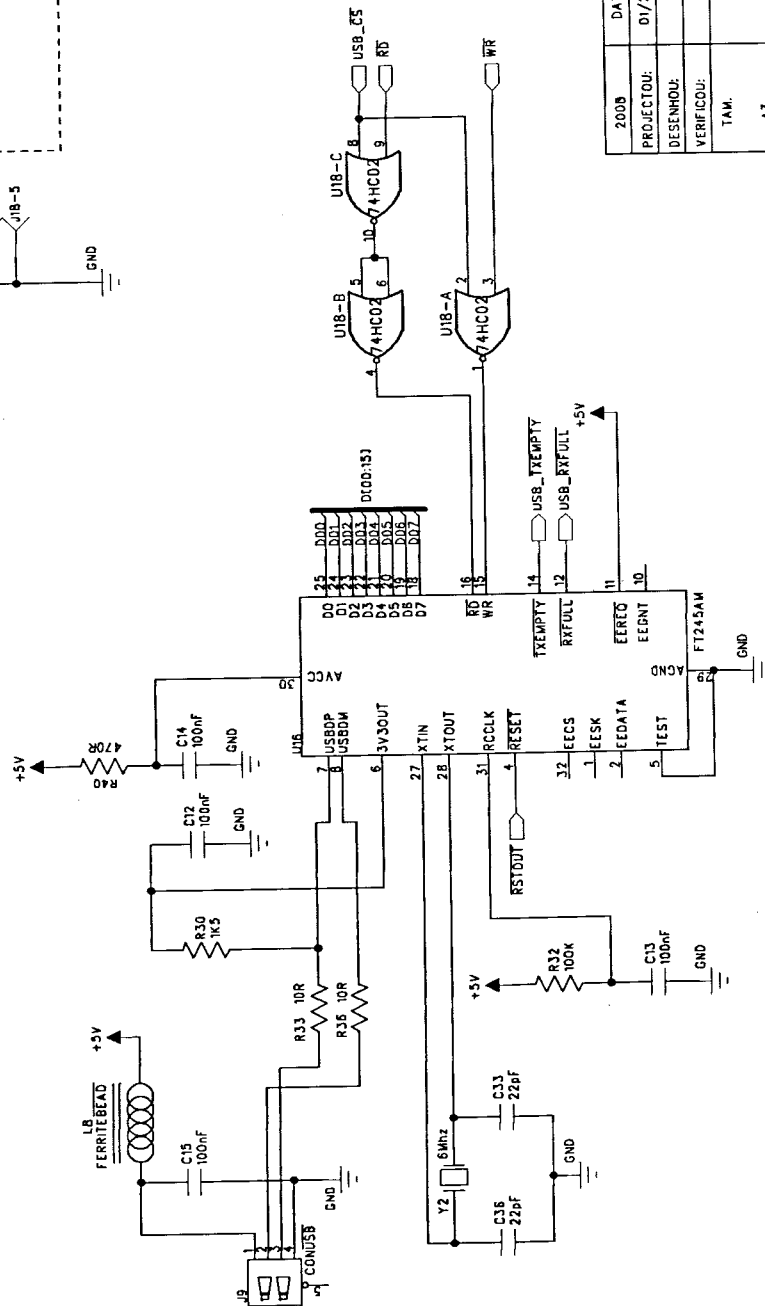


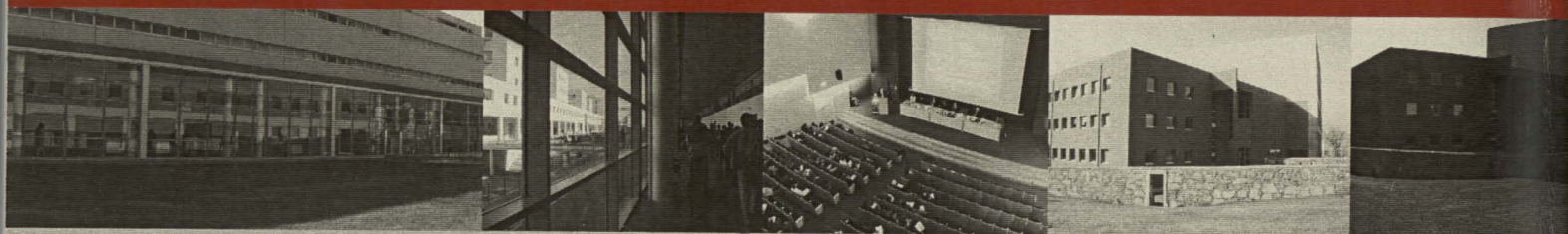
2008	DATA:	NOME	PROJECTO:
PROJECTO:	01/2003	Mário Barreto	Teste
DESENHO:			
VERIFICOU:			
TAM.			
A3			

FICHEIRO DO PCB:
SUBSTITUI:
SUBSTITUIDO POR:
DESENHO N.



PROJECTO:		Tese	
2008	DATA:	NOME	
PROJECTO:	DI/2003	Mário Barreto	
DESENHO:			
VERIFICOU:			
TAM.			
A3			
MODULO: CARD+INTERFACE			
FICHEIRO DO PCB:			
SUBSTITUI:			
SUBSTITUIDO POR:			
DESENHO N.			

[illegible]



FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

BIBLIOTECA



0000079718

62
BA