CONTROLADOR DEDICADO DE ECRÃS LCD/TFT

Emanuel Ângelo Leites Pinto

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2011

Este relatório satisfaz os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Emanuel Ângelo Leites Pinto, Nº 1020350, 1020350@isep.ipp.pt
Orientação científica: João Miguel Queirós Magno Leitão, jml@isep.ipp.pt
Co-orientação científica: João Paulo da Costa Baptista, jpb@isep.ipp.pt
Empresa:
Supervisão: Helder Parracho, hparracho@fpelectronica.com

Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
15 de Junho de 2011

Agradecimentos

A elaboração deste trabalho contou com a colaboração de pessoas e organizações que gostaria de salientar e agradecer profundamente:

Ao Doutor João Miguel Leitão, orientador deste trabalho, pela disponibilidade, pela preciosa revisão de todo este texto, comentários e sugestões;

Ao Engenheiro João Paulo Baptista, co-orientador deste trabalho, pela disponibilidade, acompanhamento pontual do projecto, discussão de ideias e comentários;

À empresa Francisco Parracho – Electrónica Industrial, Lda. por ter suportado este projecto, bem como aos seus colaboradores pelo convívio e bons momentos passados;

Ao Engenheiro Helder Parracho, orientador deste trabalho na empresa, pela sua inteira disponibilidade, pela inestimável ajuda, pelas suas ideias e discussão de tantas outras, pelo apoio e incentivo e pelo auxílio na tradução do resumo deste texto para Inglês;

Por último mas não menos importante, à minha família pelo apoio, compreensão e força e por tudo o que representa para mim.

Resumo

Para dar resposta aos grandes avanços tecnológicos e, consequentemente, à postura mais exigente dos clientes, a empresa Francisco Parracho – Electrónica Industrial, Lda., que tem actividade no ramo dos elevadores, decidiu introduzir no mercado um controlador dedicado de ecrãs *Liquid Crystal Display* / *Thin Film Transistor* (LCD / TFT). O objectivo é substituir um sistema suportado por um computador, caracterizado pelas suas elevadas dimensões e custos, mas incontornável até à data, nomeadamente para resoluções de ecrã elevadas.

E assim nasceu este trabalho. Com uma selecção criteriosa de todos os componentes e, principalmente, sem funcionalidades inúteis, obteve-se um sistema embebido com dimensões e custos bem mais reduzidos face ao seu opositor.

O ecrã apontado para este projecto é um *Thin Film Transistor – Liquid Crystal Display* (TFT-LCD) da Sharp de 10.4” de qualidade industrial, com uma resolução de 800 x 600 píxeis a 18 bits por píxel.

Para tal, foi escolhido um micro-controlador da ATMEL, um AVR de 32 bits que, entre outras características, possui um controlador LCD que suporta resoluções até 2048 x 2048 píxeis, de 1 a 24 bits por píxel.

Atendendo ao facto deste produto ser inserido na área dos elevadores, as funcionalidades, quer a nível do *hardware* quer a nível do *software*, foram projectadas para este âmbito. Contudo, o conceito aqui exposto é adjacente a quaisquer outras áreas onde este produto se possa aplicar, até porque o *software* está feito para se tornar bem flexível.

Com a ajuda de um kit de desenvolvimento, foram validados os *drivers* dos controladores e periféricos base deste projecto. De seguida, aplicou-se esse *software* numa placa de circuito impresso, elaborada no âmbito deste trabalho, para que fossem cumpridos todos os requisitos requeridos pela empresa patrocinadora:

- Apresentação de imagens no ecrã consoante o piso;
- Possibilidade de ter um texto horizontalmente deslizante;

- Indicação animada do sentido do elevador;
- Representação do piso com deslizamento vertical;
- Descrição sumária do directório de pisos também com deslizamento vertical;
- Relógio digital;
- Leitura dos conteúdos pretendidos através de um cartão SD/MMC;
- Possibilidade de actualização dos conteúdos via USB *flash drive*.

Palavras-Chave

Controlador, ecrã, LCD, píxeis, elevador, piso, micro-controlador (µC), *hardware*, *software*, camada da aplicação.

Abstract

With the intent to keep up with the major technological breakthroughs and therefore answer to the most demanding customers, the company Francisco Parracho – Electrónica Industrial, Lda., well established in the elevators field, has decided to market a dedicated Liquid Crystal Display / Thin Film Transistor (LCD / TFT) screen controller. The aim is to replace a computer based system, characterized by its inevitable large dimensions and high price, particularly for high screen resolutions.

And that's how this work began. With a careful selection of all components and, especially, without unnecessary features, we have obtained an embedded system with much lower size and cost compared to the previous solution.

The screen selected for this project is an industrial grade 10.4” Sharp Thin Film Transistor - Liquid Crystal Display (TFT-LCD), with 800 x 600 pixels of resolution at 18 bits per pixel.

To accomplish this work, an ATMEL 32-bit AVR microcontroller was chosen, with features including a LCD controller with support for resolutions up to 2048 x 2048 pixels, 1 to 24 bits per pixel.

Because this product was targeted to the elevators field, the features, both in hardware and software, have been directed to this area. However, because the software was made flexible enough, the same concept can be applied to any other areas where this product would be suitable.

Using a software development kit, the base controller and peripheral drivers were validated for this project. Then, a printed circuit board was designed and fabricated to accommodate all the requirements required by the sponsoring company:

- Displaying floor directory with images;
- Possibility of having an horizontal text banner;
- Animated elevator direction sign;
- Vertical sliding floor number animation;

- Brief description of the floor directory also with vertical sliding animation;
- Digital clock;
- SD/MMC support for media contents;
- Update procedure using an USB flash drive.

Keywords

Controller, screen, LCD, pixels, elevator, floor, microcontroller (µC), hardware, software, application layer.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

API – Application Programming Interface

ASCII – American Standard Code for Information Interchange

CCFL – Cold Cathode Fluorescent Lamp

CK – Clock

CRC – Cyclic Redundancy Check

DC – Direct Current

DMA – Direct Memory Access

DMIPS – Dhrystone Million Instructions Per Second

DST – Daylight Saving Time

EBI – External Bus Interface

EPC – Emanuel Pinto Controller

FatFs – File Allocation Table File System

GPIO – General Purpose Input/Output

HS – Horizontal Sync

HSB – High Speed Bus

IDE – Integrated Development Environment

ISP – In-System Programming

I^2C – Inter-Intergrated Circuit

LCD – Liquid Crystal Display

LED – Light-Emitting Diode

MCI – MultiMedia Card Interface

MMC – MultiMedia Card

OCD – On Chip Debug

OLED – Organic Light-Emitting Diode

PCB – Printed Circuit Board

PDC – Peripheral DMA Controller

PLL – Phase-Locked Loop

RISC – Reduced Instruction Set Computing

RTC – Real Time Clock

SD – Secure Digital

SDRAM – Synchronous Dynamic Random Access Memory

SMC – Static Memory Controller

SPI – Serial Peripheral Interface

SRAM – Static Random Access Memory

TFT – Thin-Film Transistor

TWI – Two-Wire Interface

USART – Universal Synchronous Asynchronous Receiver Transmitter

USB – Universal Serial Bus

VS – Vertical Sync

1. INTRODUÇÃO

As áreas onde hoje em dia se podem aplicar ecrãs *Liquid Crystal Display* (LCD) (e mais recentemente os *Organic Light-Emitting Diode* (OLED), nalguns casos) são inúmeras: desde os televisores aos relógios, passando pelos telemóveis, electrodomésticos, molduras digitais, etc. Nalgumas áreas até são imprescindíveis, como por exemplo nos telemóveis, molduras digitais, etc.

Mas a aplicação a salientar neste trabalho é, efectivamente, na área dos elevadores. Pelo menos no interior de qualquer elevador encontra-se um ecrã. No exterior, isto é, no patamar de cada piso, nem sempre se verifica a sua presença.

Como acontece noutras áreas, na de elevação os ecrãs não são todos do mesmo tipo, do mesmo tamanho nem têm todos as mesmas particularidades. Aqui os ecrãs costumam ter dimensões reduzidas e podem ser a LEDs, de 7 segmentos ou LCD monocromáticos. Mais recentemente, já se podem encontrar ecrãs LCD policromáticos (a cores), nomeadamente *Thin Film Transistor – Liquid Crystal Display* (TFT-LCD), cuja finalidade pode ser de multimédia e onde as dimensões já são consideráveis.

Para controlar estes ecrãs é habitual recorrer-se a integrados ou mesmo a micro-controladores, mas no último caso, como os recursos de processamento são elevados, é necessária a utilização de um computador, pelo menos até há pouco tempo.

Com o objectivo de estar na vanguarda, a empresa Francisco Parracho – Electrónica Industrial, Lda. decidiu desenvolver um sistema dedicado para controlar ecrãs LCD / TFT,

com o intuito de substituir o dispendioso computador, quer a nível de tamanho quer a nível de custos.

Sendo esta a motivação, surgiu, então, este projecto cujo presente documento pretende descrever todo o seu desenvolvimento.

Como a empresa referida, onde foi desenvolvido todo este trabalho, tem uma área de negócios no ramo de elevadores, o projecto tem incidência particular nesta mesma área. Contudo, o conceito aqui exposto é adjacente a quaisquer outras áreas onde este produto se possa aplicar. Também pelo motivo deste trabalho ter sido patrocinado pela empresa supracitada, os requisitos estabelecidos por esta foram estritamente respeitados. No entanto, este projecto pode ter outras finalidades pois o *software* está feito para se tornar bem flexível, desde que os requisitos de *hardware* coincidam.

Tratando-se de um sistema embebido, a decisão inicial principal recaiu na selecção do micro-controlador (µC), sendo escolhido um da Atmel, um AVR de 32 bits, de alto desempenho mas com baixo consumo – o AT32AP7000 [1] –, tendo como principais características para este projecto memória *Static Random Access Memory* (SRAM), controladores de memória externa volátil *Synchronous Dynamic Random Access Memory* (SDRAM) e de memórias não voláteis standard como *MultiMedia Card* (MMC), *Secure Digital* (SD)-*card*, entre outras, e, fundamentalmente, um controlador LCD, suportando ecrãs TFT-LCD, com configurações de resoluções até 2048 x 2048 píxeis desde 1 a 24 bits por píxel.

O ecrã para o qual este projecto foi concebido é um módulo da Sharp de 10.4” de aplicabilidade multimédia – o LQ104S1DG61 [2] –, contendo um painel TFT-LCD a cores, circuito de alimentação, circuitos de controlo e uma unidade de *backlight* cujo respectivo inversor tem de ser adquirido separadamente. Os gráficos e/ou os textos podem ser exibidos a uma resolução de até 800 x 600 píxeis, com 262,144 cores, através da interface de 18 bits de dados, correspondente a 6 bits por cada componente de cor do píxel.

O trabalho desenvolvido consistiu fundamentalmente em duas fases: a primeira foi constituída pela aquisição de um kit de desenvolvimento – o ATNGW100 [3], também da Atmel – onde estão presentes os elementos base do projecto. A segunda fase, após estar desenvolvido e testado o respectivo *software*, ficou entregue ao desenho do *Printed Circuit Board* (PCB) final, já com todos os periféricos propostos inicialmente, e ao

desenvolvimento e teste do restante código de programa que cumpra as funcionalidades apontadas para este projecto.

Com o ATNGW100 foi possível desenvolver a base deste trabalho a nível do *software*, sem qualquer preocupação com algum eventual problema de concepção do *hardware*, pois neste já estão integrados, para além do µC, 32 MB de memória SDRAM, 16 MB de memória *flash*, suporte para cartões de memória SD/MMC, conector JTAG para programar ou depurar a *flash*, conectores de expansão dos restantes periféricos disponíveis pelo µC, nomeadamente a interface para ecrãs, entre outros.

Posto isto, nasceu a placa electrónica Emanuel Pinto *Controller* (EPC) – designação dada a este projecto –, depois de ser elaborado o seu respectivo esquema eléctrico e desenhado o respectivo PCB. A placa EPC contém um módulo, adquirido no mercado, de seu nome ICnova AP7000 OEM da In-Circuit GmbH [4]. Com esta aquisição, dissiparam-se dois problemas: as particularidades inerentes em se projectar um PCB com várias camadas e com os componentes específicos nela contidos, e o próprio custo elevado que seria incomportável para este trabalho. Assim, de uma forma rápida e menos dispendiosa, reúne-se o µC, 64 MB de memória SDRAM, 8 MB de memória *flash*, circuito de alimentação imprescindível para o µC e conectores que funcionam tanto para encaixe como para tornar acessível todos os periféricos que o µC disponibiliza para o seu exterior. Os restantes módulos presentes são: três fontes de alimentação, um conector JTAG para programar ou depurar a *flash*, uma interface com o ecrã, outra com o inversor do *backlight*, um bloco de comunicação RS485 para receber dados do exterior, uma interface para cartões SD/MMC, um bloco responsável pelo relógio constituído por um *timekeeper*, um super condensador, com o respectivo circuito de recarregamento, e três botões para o seu acerto e, por último, outro bloco encarregue por ler e interpretar os ficheiros contidos numa eventual USB *pen drive* introduzida no sistema.

Os objectivos deste projecto são, mais uma vez, no âmbito dos elevadores, ou seja, como a aplicabilidade deste ecrã e respectivo controlador é para as cabinas dos elevadores, tanto o *layout* do que é mostrado no ecrã como o tipo de informação recebida e mostrada são destinados para este propósito. Sendo assim, no *layout* é possível visualizar o piso actual, a indicação do sentido do movimento do elevador, e até mesmo se está em movimento ou não, um directório onde é caracterizado sucintamente cada piso nas imediações do actual, um texto à escolha horizontalmente deslizante, uma sequência de imagens para cada piso,

trocadas periodicamente e também à escolha, e um relógio digital constituído por data e hora. De salientar que tanto os dígitos do piso, como as setas indicativas do sentido e o directório descritivo são animados, isto é, não se nota as trocas de imagens repentinamente pois são feitas com movimento. Para acrescentar, caso o sistema receba, via RS485, certas informações, como indicação de fogo, elevador em manutenção, excesso de carga, etc., o próprio controlador substitui a imagem principal actual por uma que corresponda ao sucedido, sendo esta também definida pelo cliente. Quanto ao relógio, para além de ser possível acertá-lo através dos botões, este tem em atenção aos anos bissextos e actualiza-se automaticamente perante o horário de Verão, mesmo se se encontrar desligado. A geração do relógio é assegurada, caso o sistema se encontre sem alimentação, pela bateria (super condensador, mais propriamente) por uma duração superior a 2 semanas. O carregamento das imagens e dos dados que definem o comportamento do sistema, como por exemplo o número de pisos e respectivos nomes e descrições, é feito pela leitura do cartão SD/MMC que tem de conter, para além das imagens desejadas, um ficheiro com todos esses dados para ser interpretado pelo µC. Para além disto, também é possível actualizar estes dados pela introdução de uma USB *pen drive*, contendo o ficheiro de dados de actualização.

1.1. CONTEXTUALIZAÇÃO

A empresa Francisco Parracho – Electrónica Industrial, Lda., que tem actividade no ramo dos elevadores, e cujos clientes são as empresas de elevação, procurava introduzir no mercado um controlador de ecrãs *Thin Film Transistor – Liquid Crystal Display* (TFT-LCD) que substituísse o computador, até então implementado.

E assim nasceu este trabalho: projectar um controlador dedicado de ecrãs LCD com um custo e tamanho bem mais reduzidos, face a um sistema com computador, e que possa ser integrado com o ecrã para ocupar o menor espaço possível.

1.2. REQUISITOS

Sendo este projecto proposto e suportado pela empresa Francisco Parracho – Electrónica Industrial, Lda., esta estabeleceu os requisitos a implementar no produto. Isto permite definir muito bem as características e os limites das funcionalidades apontadas para este projecto, sendo as seguintes:

- Ecrã TFT-LCD de 10.4” com resolução de 800 x 600 píxeis – o Sharp LQ104S1DG61 [2];

- Alimentação de 24 V DC;
- Implementar o tipo de comunicação adoptado pela empresa: RS485;
- Ler as imagens (no formato BMP) apresentadas no ecrã por um cartão de memória SD/MMC, tal como as informações da aplicação, como por exemplo, o número de pisos;
- Mostrar o piso onde o elevador se encontra;
- Apresentar uma pequena descrição respectiva a cada piso;
- Indicar o sentido da deslocação do elevador, bem como se está ou não em movimento;
- Possibilidade de apresentar vários conjuntos de imagens, onde cada conjunto pode corresponder a cada piso configurado;
- Mostrar um relógio capaz de se auto-ajustar tanto nos anos bissextos como no horário de Verão, mesmo que este se encontre desligado (até uma certa duração);
- Ser possível actualizar as informações da aplicação através de uma USB *pen drive*, eventualmente introduzida (requisito não obrigatório);
- Ficar o mais barato possível mas sem descurar a qualidade.

1.3. OBJECTIVOS

O objectivo principal deste projecto é o desenvolvimento de um controlador dedicado de ecrãs LCD. Dada a complexidade inerente a este objectivo, sentiu-se a necessidade de o subdividir em múltiplas tarefas de realização mais simples, tais como:

- A análise dos produtos já existentes no mercado;
- O levantamento dos requisitos procurados pelo cliente, conciliados com os requeridos pela própria empresa, para, assim, reunir todo o *hardware* que irá compor o sistema;
- A pesquisa e a escolha adequada do micro-controlador a implementar, bem como a plataforma de desenvolvimento;
- A análise e a decisão do dilema subjacente entre um programa *standalone* e um programa que corre baseado num sistema operativo;
- A procura de possíveis kits de desenvolvimento que facilitem a elaboração do *software* base do projecto;
- Se viável, a aquisição do kit e o desenvolvimento do código de programa dos principais componentes comuns ao projecto;
- A elaboração ou o complemento do *hardware* necessário para a interligação de todos os intervenientes do sistema, tal como o respectivo *software*;

- A realização dos testes e possíveis melhorias pontuais no projecto.

1.4. ORGANIZAÇÃO DO RELATÓRIO

No presente capítulo, 1, é efectuada uma introdução ao trabalho realizado. No capítulo seguinte, o 2º, é abordado o estado do conhecimento na área onde este projecto se insere. O 3º capítulo apresenta uma visão geral do projecto elaborado. No capítulo 4 é descrita toda a componente física que compõe este projecto – o *hardware*. Enquanto no capítulo 5 é descortinado todo o *software* desenvolvido no âmbito deste trabalho. No último capítulo, o 6º, são reunidas as principais conclusões e perspectivados futuros desenvolvimentos.

2. ESTADO DO CONHECIMENTO NA ÁREA

Este capítulo relata o conhecimento actual na área dos controladores de ecrãs, com particular incidência nos *Liquid Cristal Display* (LCD).

Como o âmbito deste projecto insere-se no ramo dos elevadores, as referências são direccionadas para este propósito, contudo, é um tipo de projecto muito abrangente. Por este mesmo motivo, é importante esta secção começar por uma breve explicação do funcionamento de um elevador.

2.1. FUNCIONAMENTO DE UM ELEVADOR [5]

Basicamente, um elevador pode ser dividido em quatro áreas, identificadas na Figura 1: na casa das máquinas (1), na cabina (2), na caixa (3) e no poço (4).

Na casa das máquinas estão localizados o quadro de comando, que é o "cérebro" do elevador, a máquina de tracção, que no fundo é um motor, e o limitador de velocidade, que é responsável pelo controlo de velocidade e accionamento do travão de segurança.

A cabina tem como função transportar passageiros e possíveis cargas.

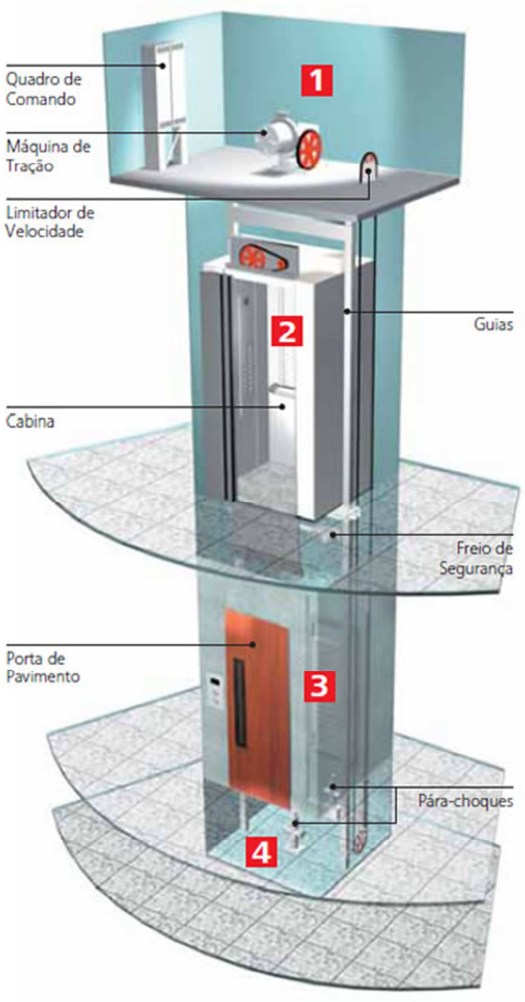

Figura 1 Representação gráfica de um elevador [5]

Na caixa ficam as guias, que servem para manter a direcção e o equilíbrio da cabina, e também as portas de patamar.

No poço do elevador encontra-se o pára-choques, um item de segurança desenvolvido para desacelerar a cabina no movimento de descida em caso de emergência.

O princípio de funcionamento básico é o mesmo desde que o elevador foi inventado há 150 anos atrás: a cabina fica ligada a um contra-peso, através de cabos e roldanas, e é movimentada por um motor que torna possível o sobe-e-desce vertical. Obviamente, os modelos mais recentes já contam com vários aditivos, tudo para garantir a segurança e a rapidez no transporte das pessoas: travões de emergência, um comando que calcula o caminho mais lógico a ser percorrido pela cabina e sensores que impedem que a porta se feche quando há pessoas no caminho.

Para se ter a noção da importância da existência dos elevadores, até já ficou provado que os elevadores impulsionaram as metrópoles.

Nos elevadores mais modernos do mundo, os motores são tão potentes que conseguem fazer com que as cabinas se desloquem a quase 50 Km/h.

Como foi referenciado em cima, uma cabina só consegue subir e descer porque fica ligada a um contra-peso, através de um sistema de roldanas e engrenagens. Para que essa operação seja possível, o contra-peso deve ter pelo menos 40% do peso da cabina com a máxima carga. Por exemplo, se uma cabina cheia pesar 1000 Kg, o contra-peso terá que ter no mínimo 400 Kg. O resto da força que movimenta a cabina é o motor que suporta.

Os elevadores são suspensos por cabos de aço ligados à cabina e ao contra-peso, num conjunto que inclui também uma roldana para evitar o desgaste do material. No mínimo, os elevadores contam com três cabos de aço de 10 mm, mas podem ter até 6 ou 8 cabos de 12 mm ou 16 mm. Se um deles se romper, os outros são suficientes para garantir o transporte da cabina por um curto período de emergência. Contudo, se todos os cabos de aço se romperem, os travões de emergência são accionados automaticamente pelo limitador de velocidade.

As portas abrem ou fecham graças a um sistema de braços mecânicos movimentados por um pequeno motor. Os elevadores mais modernos têm sensores infravermelhos que só permitem que a porta automática se feche quando não há qualquer actividade entre ela, isto é, quando não há pessoas no meio. Alguns deles também impedem o fecho quando o limite de peso é excedido.

No poço do elevador existe um sistema de amortecimento com molas para evitar que a cabina choque com o chão. Este mecanismo diminui os danos de uma paragem brusca, mas não tem a capacidade de amortecer uma queda de mais de dois andares.

Por último, uma pequena abordagem ao modelo aplicado para atender várias chamadas de passageiros. Antigamente, a ordem pela qual eram atendidas as chamadas correspondia exactamente à rota do elevador. Nos elevadores de hoje, como são computorizados, o processador recorre a um algoritmo que decide o caminho mais rápido para a cabina. Para isso o sistema leva em conta onde estão as pessoas que chamaram o elevador e o andar em que a cabina se encontra.

2.1.1. INDICADORES

Nos elevadores também estão presentes indicadores visuais das informações fundamentais requeridas pelos passageiros de um elevador: o piso e a direcção. Estes indicadores são ecrãs (ou *displays*) que podem ser de vários tipos, como serão apresentados de seguida, estando localizados tanto no patamar como dentro da cabina. Consoante o tipo e o tamanho de ecrã utilizado, as informações disponíveis para os passageiros do elevador podem ser mais enriquecidas, como por exemplo, podem ser acrescentadas imagens, horário, informação descritiva do piso, etc.

Para ser melhor compreendido como é que estes indicadores visuais se integram no sistema de um elevador, é apresentado um digrama de blocos na Figura 2. Há dois tipos base de comunicação entre o comando e o exterior, inclusive o ecrã: paralelo e série. Dentro da interface paralela, as entradas podem ser de vários tipos, desde discretas decimais, em binário, em código Gray, etc. No segundo caso, a interface física pode ser RS485/RS422, CANbus ou mesmo imiscuir-se com a própria alimentação, isto é, os dados são transmitidos nas mesmas linhas da alimentação. A nível de protocolo, existe desde o *opensource* até aos protocolos criados propositadamente para uma empresa de elevação específica. Aqui a variedade é muita e a preocupação em desenvolver métodos que não sejam descortinados por terceiros é ainda maior.

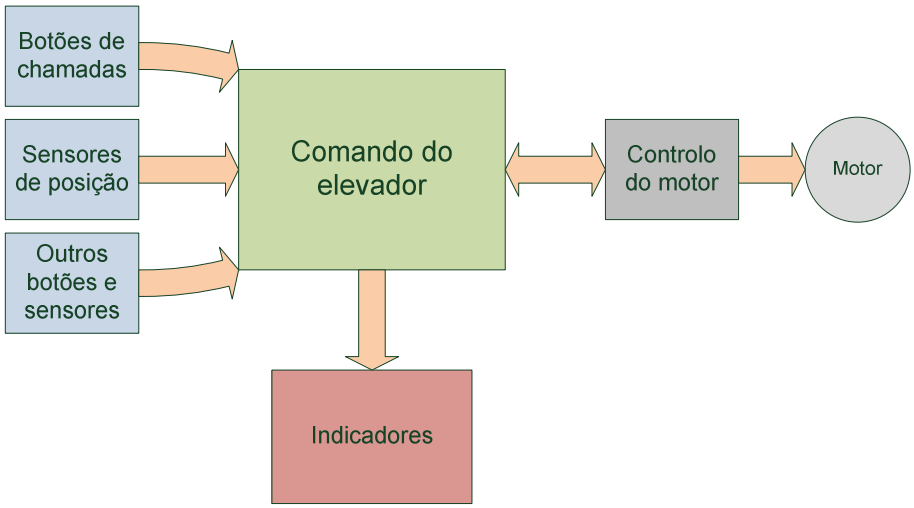

Figura 2 Diagrama de blocos integrante dos indicadores

2.2. ECRÃS

Nesta subsecção são dadas explicações sucintas sobre os vários tipos de ecrãs utilizados na área de elevação, desde os ecrãs de 7 segmentos até aos ecrãs LCD. É feita também uma pequena abordagem aos ecrãs *Organic Light-Emitting Diode* (OLED), porém, apenas com um carácter informativo para o futuro pois até hoje não há conhecimento de alguma aplicação com este tipo de ecrã nesta área.

Embora os ecrãs de 7 segmentos e de matriz de pontos ainda estejam amplamente implementados nesta área, são os ecrãs LCD, nomeadamente os *Thin Film Transistor – Liquid Crystal Display* (TFT-LCD), que têm maior destaque neste trabalho, pois é este o ecrã que começa a dar os primeiros passos nesta área. Inclusive, é apresentado um exemplo de ecrã TFT-LCD indicado para aplicações em elevadores.

2.2.1. ECRÃ DE 7 SEGMENTOS [6]

O ecrã (ou *display*, como também é comummente conhecido) de sete segmentos (Figura 3) é um invólucro com oito leds (um deles é o ponto decimal) em que cada um tem um formato de segmento, estando posicionados de modo a possibilitar a formação de números decimais e algumas letras utilizadas no código hexadecimal.

A Figura 4 representa uma unidade do ecrã genérica, com a nomenclatura de identificação dos segmentos usual em manuais práticos.

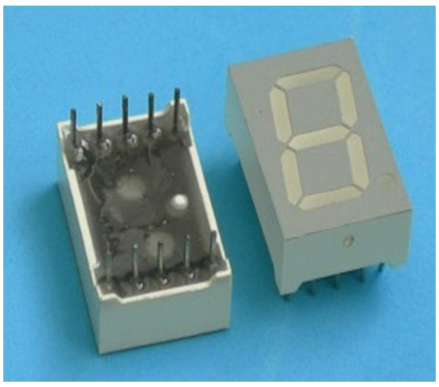

Figura 3 Ecrã de 7 segmentos [6]

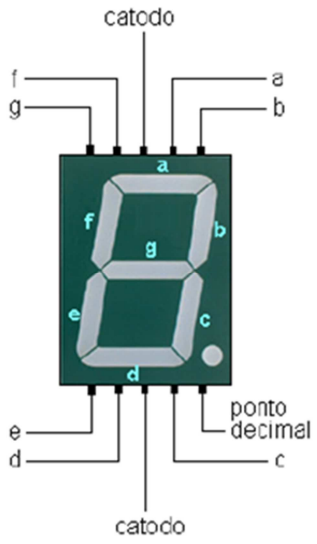

Figura 4 Nomenclatura dos segmentos de um ecrã genérico de 7 segmentos [6]

Entre as tecnologias de fabrico das unidades de ecrã, o mais comum é o ecrã a leds, que possui cada segmento composto por um led, conforme a Figura 5.

O ecrã pode ser do tipo ânodo comum, ou seja, os terminais ânodo de todos os segmentos estão interligados internamente e, para o ecrã funcionar, este terminal comum deverá ser ligado ao V_{CC}, enquanto o segmento que se quer ligar precisa de estar ligado à massa. Já no ecrã do tipo cátodo comum é o contrário, ou seja, o terminal comum deverá ser ligado à massa e para ligar o segmento é necessário aplicar V_{CC} ao terminal. Actualmente, o ecrã mais comercializado é o do tipo ânodo comum.

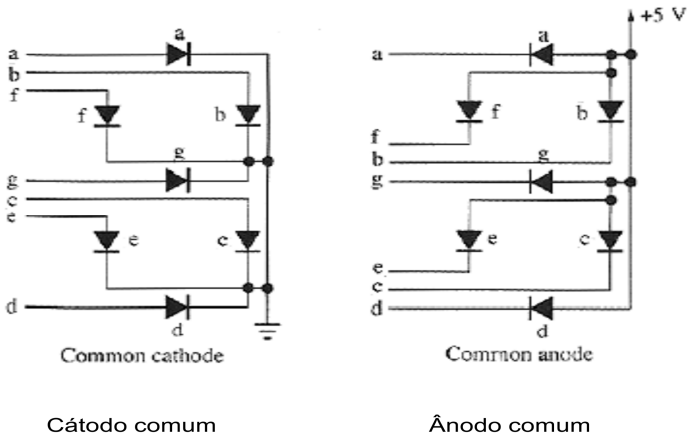

Figura 5 Tipos de ecrãs de 7 segmentos [6]

Tabela 1 Segmentos de saída de um ecrã de 7 segmentos em cátodo comum [6]

segmentos de saída							DISPLAY
a	b	c	d	e	f	g	
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
0	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	0	0	1	1	9

Como já foi referido anteriormente, o ecrã de sete segmentos é formado por sete leds, dispostos em forma de oito. Quando é necessário acender o número “0”, ligam-se os leds correspondentes ao digito “0”, neste caso, os segmentos a, b, c, d, e, f. Para os outros casos pode-se consultar a Tabela 1.

Como os segmentos são leds, então é preciso limitar a corrente, e para isso deve-se usar uma resistência em cada segmento. A corrente utilizada depende do brilho que se queira do ecrã. Normalmente, utilizam-se resistências entre 220 e 560 Ω para uma fonte de 5 V, o que equivale a uma corrente entre 9 mA a 20 mA. Não se deve usar valores de resistência muito baixos pois está-se a reduzir a vida útil do ecrã, inclusive pode-se queimar o segmento. Ao usar um ecrã de 7 segmentos é melhor testar primeiro cada segmento, isto para se ter a certeza que nenhum segmento do ecrã está queimado.

2.2.2. ECRÃ DE MATRIZ DE PONTOS [6]

Um ecrã de matriz de pontos é muito semelhante a um ecrã de 7 segmentos, ou seja, também contém leds, mas neste caso os leds estão no formato de uma matriz. Como se pode ver pela Figura 6, existem vários formatos de matriz, contudo, o mais usual é o formato 7x5.

Ao contrário dos ecrãs de 7 segmentos, nos de matriz de pontos não se consegue ligar todos os leds ao mesmo tempo, só uma coluna de cada vez. A solução é recorrer à multiplexagem de colunas para o carácter ser mostrado.

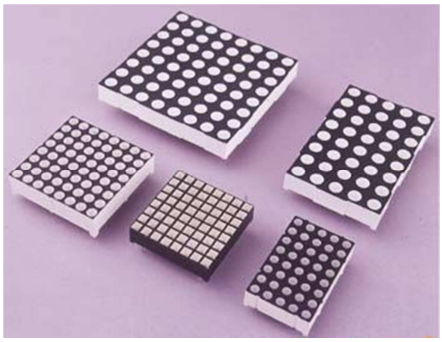

Figura 6 Ecrãs de matriz de pontos

2.2.3. LCD [7][8]

A tecnologia LCD baseia-se num ecrã composto por duas placas paralelas sulcadas transparentes, orientadas a 90°, entre as quais é colocada uma fina camada de líquido que contém moléculas (cristais líquidos) que têm a propriedade de se orientar quando são sujeitas à corrente eléctrica.

Combinada com uma fonte de luz, a primeira placa estriada age como um filtro polarizante, que só deixa passar as componentes da luz cuja oscilação é paralela às ranhuras (Figura 7).

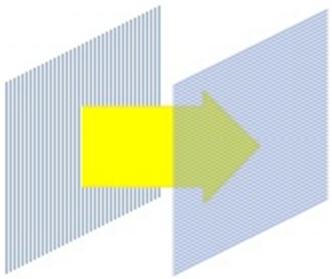

Figura 7 Filtro polarizante da primeira placa de um ecrã LCD

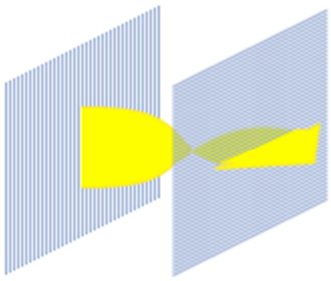

Figura 8 Filtro polarizante perpendicular da segunda placa de um ecrã LCD

Na ausência de tensão eléctrica, a luz é bloqueada pela segunda placa, agindo como um filtro polarizante perpendicular (Figura 8).

Sob o efeito de uma tensão, os cristais vão progressivamente alinhar-se no sentido do campo eléctrico e assim poder atravessar a segunda placa.

Controlando localmente a orientação destes cristais é possível constituir píxeis.

Um píxel é a menor unidade de uma imagem, podendo ser formado por 3 componentes de cor: vermelho, verde e azul (RGB).

Nos ecrãs LCD monocromáticos cada ponto do ecrã corresponde a um píxel da imagem (Figura 9), enquanto nos ecrãs policromáticos (a cores) cada píxel da imagem é formado por um grupo de 3 pontos: os ditos RGB (Figura 10).

O número de píxeis no ecrã determina a resolução do ecrã LCD.

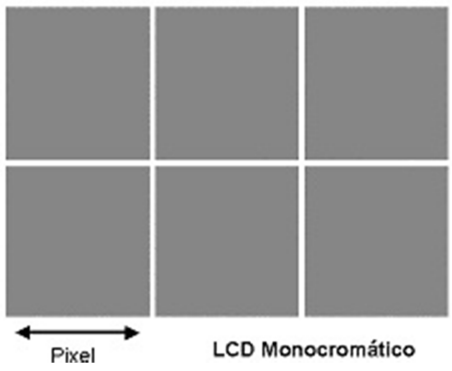

Figura 9 Píxel de um ecrã LCD monocromático

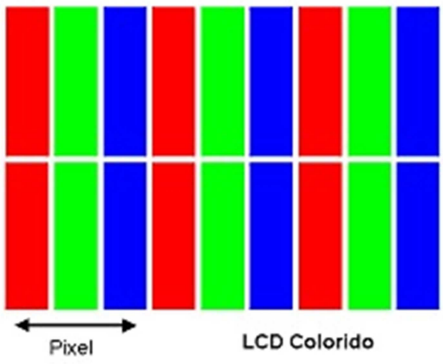

Figura 10 Píxel de um ecrã LCD policromático

Habitualmente, distinguem-se dois tipos de ecrãs planos de acordo com o sistema de comando que permite polarizar os cristais: os de matriz passiva e os de matriz activa.

Nos primeiros, os píxeis são controlados por linha e por coluna. Assim, os píxeis são dirigidos por linhas e por colunas graças a condutores transparentes situados no ladrilho. O píxel acende-se quando é endereçado e apaga-se entre dois varrimentos.

Os tipos de ecrãs de matriz passiva são os seguintes:

- *Guest Host* (GH): utiliza uma espécie de pigmento contido no cristal líquido que absorve luz. O processo ocorre de acordo com o nível do campo eléctrico aplicado. Com isso, é possível trabalhar com várias cores;
- *Twisted Nematic* (TN): as moléculas de cristal líquido com ângulos de 90º em ecrãs que usam TN podem ter a exibição da imagem prejudicada em animações muito rápidas;
- *Super Twisted Nematic* (STN): é uma evolução do padrão TN, capaz de trabalhar com imagens que mudam de estado rapidamente. As moléculas têm movimentação melhorada, fazendo com que o utilizador consiga ver a imagem do ecrã satisfatoriamente em ângulos muitas vezes superiores a 160º.

Nos ecrãs de matriz activa cada píxel é controlado individualmente.

A tecnologia mais utilizada para este tipo de afixação é a tecnologia TFT (ou TFT-LCD), permitindo controlar cada píxel com a ajuda de três transístores (correspondendo às 3 cores RGB). Assim, o transístor acoplado a cada píxel permite memorizar o seu estado e, se for caso disso, mantê-lo ligado entre dois varrimentos sucessivos. Os ecrãs de matriz activa beneficiam assim de uma melhor luminosidade e uma afixação mais fina.

Quer os ecrãs sejam de matriz activa ou passiva têm necessidade de ter uma fonte luminosa para se poder ver perfeitamente a imagem. Esta fonte de iluminação de um ecrã LCD é fornecida por uma lâmpada *Cold Cathode Fluorescent Lamp* (CCFL) ou por um conjunto de leds.

A lâmpada fluorescente funciona em frequência e em tensão, não existindo resistência de aquecimento no cátodo da lâmpada e, para isso, necessita de um inversor CCFL para funcionar.

2.2.4. OLED [9]

Um OLED é um semicondutor cerca de 200 vezes menor que um cabelo humano e pode ter duas ou três camadas de material orgânico. A base da estrutura OLED é o substrato, que pode ser um plástico, vidro ou uma lâmina.

Um eléctrodo ânodo remove os electrões quando uma corrente circula pelo dispositivo. Em seguida, entram em acção as camadas orgânicas. A primeira é a condutora, que transporta os electrões do ânodo (um condutor utilizado é a polianilina). A segunda é a camada emissora, feita com moléculas diferentes da camada condutora e que transporta os electrões para outro eléctrodo: o cátodo. É aqui que a luz é feita. Um polímero utilizado nesta camada é o polifluoreno.

Uma das grandes vantagens do OLED é ter luz própria, mas como? Uma bateria ou um alimentador fornece uma tensão de alimentação. A corrente eléctrica circula do cátodo para o ânodo através das camadas orgânicas. O cátodo dá electrões para a camada emissora, enquanto o ânodo remove os electrões da camada condutora, criando várias lacunas. Neste processo entre as camadas, os electrões preenchem essas lacunas, reduzindo a sua energia para o mesmo nível de um átomo sem um electrão. Essa energia que sobra é transformada num fotão de luz. A cor da luz depende do tipo de molécula da camada emissora. Essa camada é desenvolvida com vários tipos de filmes orgânicos para gerar cores diferentes. E assim se justifica a capacidade de fornecer luz sem ajudas externas, como *backlights* ou *sidelights* no caso dos LCD.

A intensidade da luz depende da quantidade de corrente eléctrica aplicada: quanto maior a corrente mais brilhante fica a luz.

Há diferentes tipos de OLED, em que cada um é utilizado de maneira diferente:

- *Passive-Matrix* OLED (PMOLED): tem uma estrutura composta por linhas de cátodo, camadas orgânicas e linhas de ânodo, sendo que as linhas de ânodo e cátodo são perpendiculares. Essa intersecção geral os píxeis onde a luz é emitida. Este tipo de OLED é facilmente fabricado mas consome mais energia. São indicados para ecrãs pequenos como os de telemóveis;
- *Active–Matrix* OLED (AMOLED): neste tipo as camadas de cátodo, moléculas orgânicas e ânodo são como chapas, sendo que a camada de ânodo sobrepõe um transístor que forma uma matriz e que, por fim, determina quais os píxeis são ligados

para formar a imagem. Este tipo de OLED consome menos energia e é indicado para ecrãs grandes;

- OLED transparente: é uma variação do PMOLED e do AMOLED feita somente por componentes transparentes. Quando um OLED transparente é ligado, ele permite que a luz passe em ambas as direcções. Este tipo é usado em ecrãs com informações, como por exemplo em ecrãs *heads-up*;
- *Top-emitting* OLED: este tipo tem um substrato que pode ser tanto opaco como reflexivo, bem indicados para design;
- OLED dobrável: estes OLED são feitos de substratos de materiais muito flexíveis, tornando-os muito leves e duráveis. Podem ser utilizados em telemóveis, por exemplo, diminuindo o risco de danos em caso de quedas. Estuda-se a possibilidade de usar esta tecnologia em tecidos;
- *White* OLED: este OLED emite luz mais brilhante, uniforme e económica. É esta a tecnologia que poderá substituir as luzes fluorescentes que hoje são utilizadas em casas e em prédios, reduzindo o consumo de energia de iluminação.

2.2.5. SHARP LQ104S1DG61 [2]

Um exemplo de um ecrã TFT-LCD que pode perfeitamente ser implementado em elevadores é o Sharp LQ104S1DG61, projectado para aplicações multimédia e equipamentos que requeiram fiabilidade.

Este módulo é composto por um painel a cores TFT-LCD de 10.4” (26 cm) de tamanho, circuitos integrados controladores, circuito de controlo, circuito de alimentação e uma unidade de *backlight*.

Quer os gráficos quer os textos podem ser exibidos em 800 (na horizontal) x 600 (na vertical) píxeis, em que cada píxel tem três componentes de cor: vermelho (R), verde (G) e azul (B), formando, assim, o chamado RGB; 262144 é o número de cores que este ecrã suporta.

Para exibir essas imagens ou textos, o módulo está preparado para receber um sinal de dados de 18 bits, correspondendo a 6 bits por cada componente RGB (por cada cor) e, para tal, este contém quatro sinais de sincronismo.

O ecrã suporta uma tensão contínua de 3.3 V ou 5.0 V (não estando preparado para tensões superiores a 5.5 V) para o painel TFT-LCD e outra alimentação para o *backlight*, fornecida por um inversor que será descrito mais à frente. Em relação à corrente contínua que o painel necessita, para uma tensão de 3.3 V este pede um máximo de 450 mA, sendo o mais comum de 300 mA, enquanto para uma tensão de 5.0 V a corrente máxima é de 300 mA, mas a típica é de 200 mA.

O tipo de painel TFT-LCD utilizado neste módulo é um de baixo reflexo e de elevada saturação de cores, portanto, também se adequa bem às aplicações multimédia. O ecrã tem um amplo ângulo de visão e uma elevada luminosidade: 420 cd/m^2.

Este ecrã permite que se possa alterar a posição da imagem de uma forma rápida a partir do *hardware*, ou até mesmo por *software*, se os pinos respectivos forem ligados a um controlador. Antes de serem descritas as combinações possíveis para este propósito, é importante conhecer as ligações internas do módulo, que se pode ver na Figura 11. Para além das ligações naturais dos sinais de sincronismo (*Vertical Sync* (VS), *Horizontal Sync* (HS) e *Clock* (CK)) e das linhas de dados do RGB, há dois pinos, R/L e U/D, cujas iniciais representam direita/esquerda (*right*/*left*) e cima/baixo (*up*/*down*), respectivamente, que são os tais pinos que podem ser combinados.

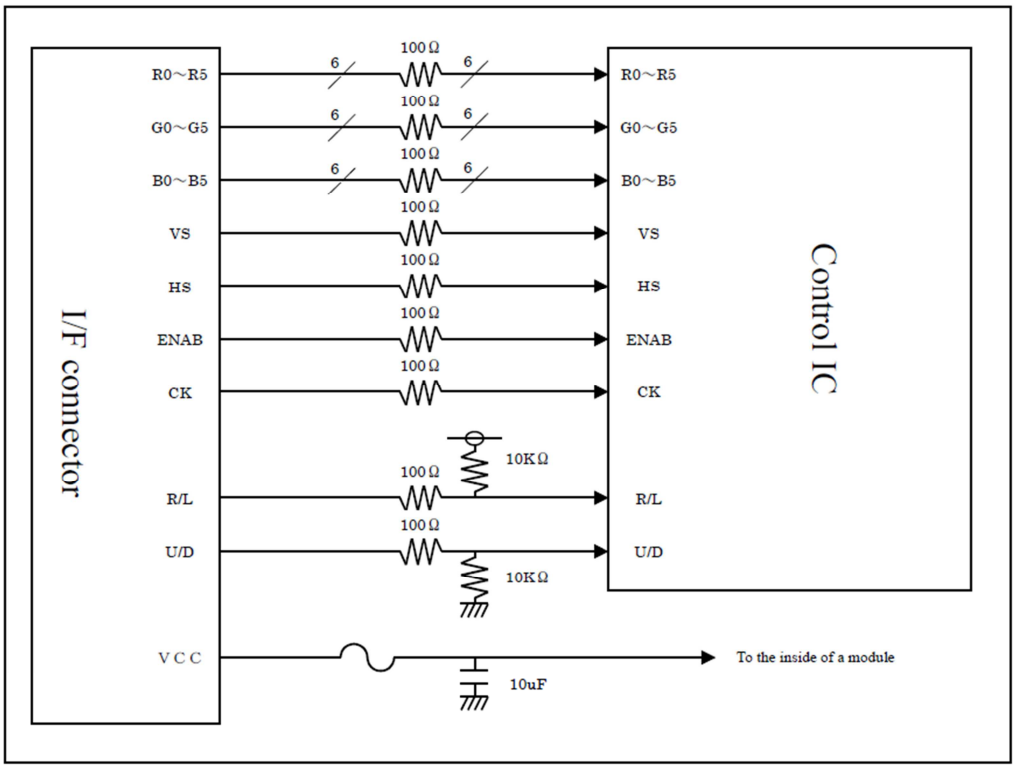

Figura 11 Ligação interna do conector do ecrã LQ104S1DG61

Pela Figura 11 pode-se notar que na ligação interna do módulo entre o conector, acessível para o exterior, e o circuito integrado controlador está ligado um *pull-up* ao pino R/L e um *pull-down* ao pino U/D. Esta situação origina a quatro combinações possíveis:

- Se R/L estiver num estado lógico alto e U/D estiver num estado lógico baixo, a imagem obtida fica na posição original, como mostra a Figura 12;
- Se R/L e U/D estiverem num estado lógico baixo a imagem obtida fica espelhada face à posição original, como mostra a Figura 13;
- Se R/L e U/D estiverem num estado lógico alto a imagem obtida fica invertida face à posição original, como mostra a Figura 14;
- Se R/L estiver num estado lógico baixo e U/D estiver num estado lógico alto, a imagem obtida fica espelhada e invertida ao mesmo tempo em comparação com a posição original, como mostra a Figura 15;

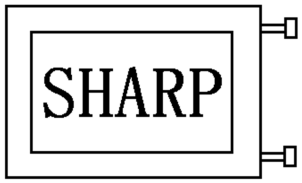

Figura 12 Posição original da imagem no ecrã LQ104S1DG61

Figura 13 Imagem espelhada no ecrã LQ104S1DG61

Figura 14 Imagem invertida no ecrã LQ104S1DG61

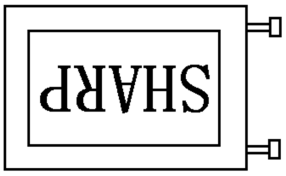

Figura 15 Imagem espelhada e invertida no ecrã LQ104S1DG61

2.3. INVERSOR PARA ALIMENTAR O *BACKLIGHT* DE UM ECRÃ LCD

O sistema de retro-iluminação (*backlight*) de qualquer ecrã LCD pode ser de dois tipos: por lâmpada(s) *Cold Cathode Fluorescent Tube(s)* (CCFT) ou por leds. Caso seja por lâmpada(s), esta(s) precisa(m) de ser alimentada(s) por uma tensão alternada elevada. Para que isso seja possível é habitual recorrer-se a um inversor.

Em traços gerais, um inversor é um componente electrónico, neste caso uma placa electrónica, que, recorrendo a interruptores electrónicos (como por exemplo transístores), converte uma tensão contínua em tensão alternada, com o acréscimo dessa tensão ser aumentada. O tipo de controlo utilizado nesse circuito caracteriza os parâmetros disponibilizados pelo fabricante do inversor.

2.3.1. TDK-LAMBDA CXA-0454 [10]

Retomando ao exemplo do ecrã TFT-LCD da Sharp modelo LQ104S1DG61 [2], o *backlight* deste tem duas lâmpadas CCFT, que precisam de ser alimentadas por uma tensão alternada máxima de 1300 Vrms.

Para alimentar estas duas lâmpadas pode-se adquirir um inversor, recomendado pelo próprio fabricante deste ecrã TFT-LCD: o CXA-0454 da marca TDK-Lambda.

Uma das principais características deste inversor é a aplicabilidade em ecrãs de 10 a 12”. Tem também duas saídas, uma para cada lâmpada, tem a possibilidade de se controlar a luminosidade do ecrã por PWM e uma funcionalidade para se desligar o inversor.

A tensão de entrada estipulada é de 12 V DC (com uma variação aceitável de ± 1.2 V), pedindo uma corrente típica de 0.7 A, sendo a máxima de 1.25 A. Com estes valores de entrada o inversor garante uma saída, para ambas as lâmpadas, de 600 Vrms.

Tabela 2 Funções dos terminais do conector de entrada do inversor [10]

Número do terminal	Símbolo	Valor	Observação
CN1-1	Vin	12 ± 1.2 V	Alimentação de entrada
CN1-2			
CN1-3	GND	0 V	Massa
CN1-4			
CN1-5	Vrmt	0 V/2.5 V – Vin	0 – 0.4 V: OFF 2.5 V – Vin: ON
CN1-6	Vst (saída)	0 V/ 5 V	Alarme
CN1-7	Vbr/Rbr	0 – 2.5 V/0 – 50 kΩ	*Dimmer*

Como o inversor tem duas saídas tem, logicamente, dois conectores na saída enquanto para a entrada da alimentação tem somente um.

Contudo, o conector da entrada tem sete terminais: os dois primeiros pertencem à tensão de entrada, os dois seguintes à massa e os três seguintes são sinais de controlo; o quinto terminal é o "interruptor" do inversor, isto é, se for aplicada uma tensão de 0 a 0.4 V o inversor permanece desligado enquanto se a tensão aplicada variar entre 2.5 V e 12 V o inversor fica a funcionar em pleno; a sexta ligação indica, caso haja algum corte na saída, um estado de alerta colocando essa saída em 5 V; o sétimo e último terminal é o controlo da luminosidade desejada no ecrã. Estes dados podem ser lidos, resumidamente, na Tabela 2.

2.4. INTERFACES PARA ECRÃS

Qualquer ecrã pode ser ligado a um micro-controlador (µC) através de um circuito integrado responsável pela interface, ou mesmo directamente. A escolha depende do que se pretende e das capacidades do µC escolhido. Se este não tiver capacidade para tal, um controlador dedicado torna-se útil e até imprescindível, pois para além de fazer a interface com o ecrã, retira, e muito, o processamento que o µC necessitaria.

Falando mais concretamente sobre os ecrãs LCD, estes não fogem à regra e também tanto podem ser ligados através de um circuito integrado que faz a interface/controlador como directamente. Neste último caso, a ligação com o µC é feita através dos endereços e/ou barramento de dados mais alguns pinos de controlo. Nalguns µC actuais até já se pode

encontrar controladores LCD integrados, porém, a maioria destes limita a resolução máxima do ecrã para valores que não satisfazem os requisitos deste projecto.

Alguns ecrãs LCD também suportam interface série.

De seguida, são referenciados alguns exemplos de interfaces com ecrãs LCD [11].

- Interface paralela com um módulo LCD de 2 linhas com 16 caracteres cada [12];
- Interface série RS-232 com um módulo LCD de 2 linhas com 16 caracteres cada [13];
- Interface paralela entre um FPGA e um módulo LCD de 1 linha com 16 caracteres [14];
- Interface paralela com um µC da família 8051 mais uma EPROM externa [15];
- Interface série através do circuito integrado EDE702 [16];
- Interface série com um µC PIC [17].

Continuando com o exemplo do ecrã TFT-LCD LQ104S1DG61 [2], a interface deste é paralela, tendo 18 bits de dados (6 para cada componente de cor do píxel), 4 bits de sincronismo e outros pinos de controlo (posição da imagem) e alimentação. Para tal, este ecrã está munido de um conector de 41 pinos, com um tamanho muito reduzido (SMD), que possibilita o seu controlo através de um controlador dedicado ou mesmo directamente com um µC capaz: o conector da Hirose DF9-41S-1V [18] (Figura 16).

As informações sobre os sinais de sincronismo e respectivos tempos desta interface específica estão disponíveis nas páginas 13 e 11, respectivamente, do *datasheet* deste ecrã [2].

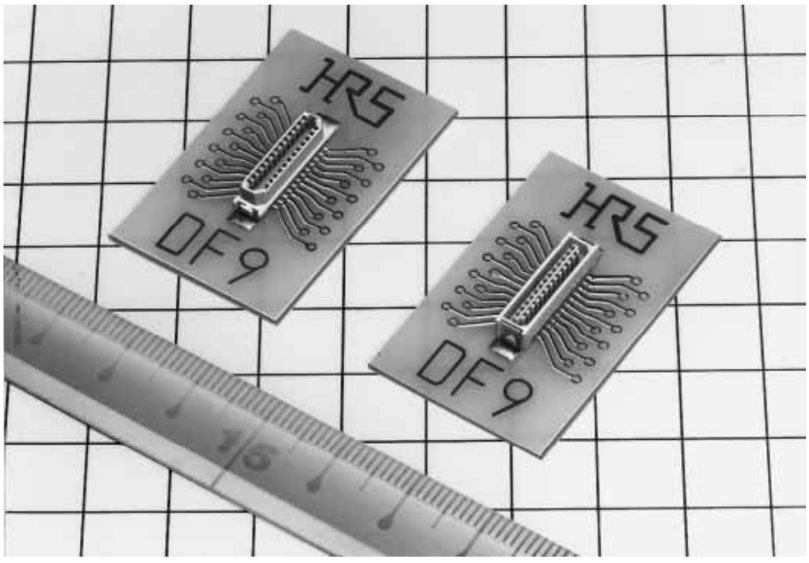

Figura 16 Conector do ecrã TFT-LCD LQ104S1DG61 [18]

2.5. CONTROLADORES DEDICADOS DE ECRÃS

Qualquer ecrã, por si só, não tem um funcionamento autónomo, isto é, para mostrar imagens e/ou textos necessita sempre de um controlador externo. Esse controlador pode ser dos mais variados tipos, dependendo do ecrã e da exigência do controlo.

2.5.1. DE 7 SEGMENTOS E DE MATRIZ DE PONTOS [6][19]

Um controlador, se assim se pode chamar, muito usado com os ecrãs de 7 segmentos é o descodificador BCD-7segmentos (Figura 17). Este descodificador tem a função de interpretar um código (*Binary-Coded Decimal* (BCD)) e gerar os sinais para ligar o dígito correspondente a este código no ecrã de 7 segmentos.

Por exemplo, para uma entrada no descodificador em código BCD de "0000", a respectiva saída será "0111111", ou seja, os segmentos ligados serão o "g", o "f", o "e", o "d", o "c", o "b" e o "a", como se pode ver na Figura 18. De notar que esta saída do descodificador corresponde a ligar os segmentos do dígito "0" de um ecrã de 7 segmentos do tipo cátodo comum.

Os descodificadores disponíveis no mercado são o 7447, para ânodo comum, e o 7448, para cátodo comum.

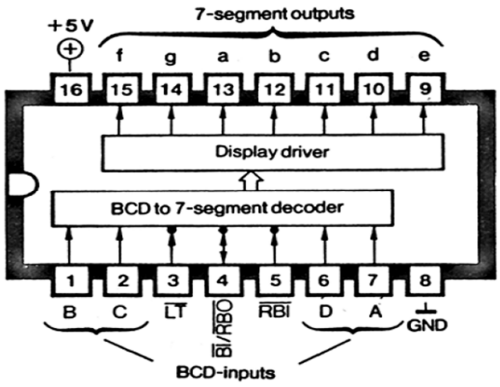

Figura 17 Pinos de um descodificador BCD – 7 segmentos [6]

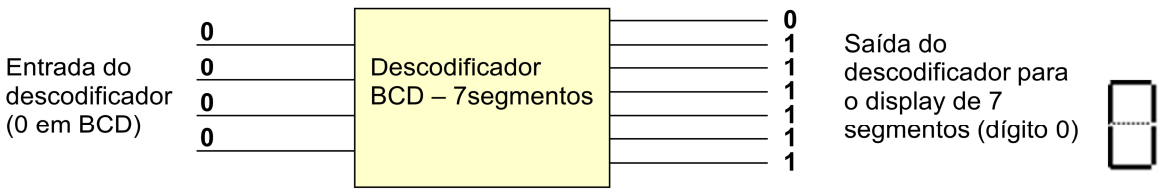

Figura 18 Descodificador BCD – 7 segmentos (exemplo) [6]

Tabela 3 Entradas BCD correspondentes a cada carácter [6]

entradas BCD				segmentos de saída							DISPLAY
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9

Para os restantes casos de entradas BCD, ou por outro lado, para outros caracteres, é apresentada a Tabela 3.

Muitas aplicações que utilizam ecrãs de 7 segmentos necessitam de várias unidades. Isto implica um consumo grande de energia. Uma solução adoptada e muito simples é multiplexar os ecrãs. Multiplexar significa activar um ecrã de cada vez, alternando o funcionamento dos ecrãs. Portanto, cada ecrã ficará ligado por um espaço de tempo e depois apagar-se-á. Mas isto é feito a uma frequência que a visão humana não consegue perceber, ou seja, se os ecrãs estiverem a ser multiplexados com uma frequência de 50 Hz ou maior, a visão humana terá a impressão que todos os ecrãs estão ligados, porém, na realidade quando um liga os outros estão desligados.

Em relação aos ecrãs de matriz de pontos, e tal como acontece com os de 7 segmentos, são habitualmente usados ecrãs com leds. Mas ao contrário dos ecrãs de 7 segmentos, nos de matriz de pontos os leds não costumam ser ligados todos ao mesmo tempo, devido ao número de leds ser considerável e, portanto, não ser eficiente a utilização dos recursos. Para tal, o conceito de multiplexagem volta a entrar nesta situação.

Nos ecrãs multiplexados os leds são ligados por um curto período de tempo e repetidamente para cada led. Cada coluna da matriz é activada sequencialmente durante um período de tempo a que se dá o nome de período de varrimento. Em cada período de varrimento são activadas também as linhas correspondentes aos leds que se querem ligar.

A frequência a que cada coluna é activada é chamada frequência de refrescamento. Se esta frequência for suficientemente elevada, o olho humano não se apercebe da transição, observando apenas um conjunto de leds acesos como se fossem ligados simultaneamente. Este procedimento é familiar com o utilizado no caso do ecrã anteriormente referido.

O controlo multiplexado, apesar de inevitavelmente requerer processamento mais complexo, é preferido ao controlo directo nos ecrãs de matriz de pontos (ou leds) com muitos elementos. O que se deve principalmente ao custo, pois é necessário um reduzido número de elementos de controlo, mas também às reduzidas interconexões e simplicidade de desenho da placa de circuito impresso.

2.5.2. LCD

Como há variadíssimos tipos e formatos de ecrãs LCD, também há no mercado vários controladores dedicados para cada caso ou para cada associação de ecrãs com características próximas. Ora, tendo presente esta constatação, é inviável a citação de todos esses controladores.

Contudo, há um controlador para ecrãs TFT-LCD importante de ser salientado no contexto deste trabalho: o PrismaECO II, da Data Display Teknoloji [20]. Este destaque tem de ser dado pois este componente faz parte de um sistema cujo intuito é ser substituído pelo projecto deste trabalho.

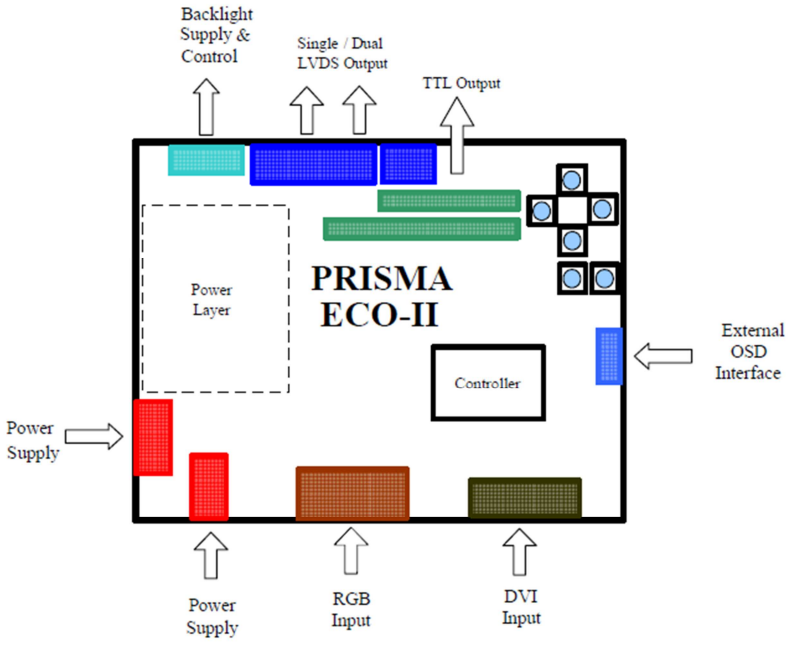

Figura 19 Interfaces do controlador PrismaECO II [21]

O PrismaECO II é uma placa de processamento gráfico, oferecendo imagens de alta qualidade para ecrãs TFT-LCD. A placa suporta resoluções até SXGA (1280 x 1024 píxeis) e pode ser utilizada nos mais variados sistemas. Foi desenvolvida pela Distec GmbH e é capaz de se adaptar a quase todos os ecrãs TFT-LCD.

Para acompanhar a enumeração das características deste conversor de vídeo (ou controlador de vídeo), estão ilustradas na Figura 19 as interfaces desta placa [21]:

- Entradas RGB analógica (VGA) e *Digital Visual Interface* (DVI);
- Saídas individual/dupla *Low-Voltage Differential Signaling* (LVDS) e TTL;
- Controlo da intensidade do *backlight* por PWM;
- Suporta VESA DDC2B e um subconjunto do padrão VESA DPMS;
- Placa adequada para ser montada atrás de um ecrã LCD;
- Interface de 4 botões e menus no ecrã que permitem ajustar o sistema.

2.6. MICRO-CONTROLADORES (µC)

O outro método para que um ecrã seja controlado é o directo, ou seja, num sistema, em vez de haver um controlador que faça a interface entre o ecrã e um computador ou um µC, pode haver um controlo directo do ecrã com um µC. Para tal, este tem de ter capacidade de processamento e o respectivo controlador integrado.

Para ecrãs que exigem pouco processamento e requerem poucos pinos para a sua interface há um número infindável de µCs no mercado. Por outro lado, à medida que este grau de exigência aumenta o número de produtos à altura diminui. Esta situação é bem notória para ecrãs LCD com resoluções relativamente elevadas, nomeadamente para resoluções SVGA (800x600 píxeis) ou superiores.

2.6.1. AT32AP7000 [1]

Um dos poucos exemplos de µCs com competências multimédia e capazes de suportar as tais resoluções relativamente elevadas é o AT32AP7000.

Este componente é um µC de 32 bits da ATMEL, da família AVR32, com um sistema completo integrado que inclui um micro-processador AVR32 *Reduced Instruction Set Computer* (RISC) e que funciona a 210 *Dhrystone Millions of Instructions Per Second* (DMIPS) à velocidade de 150 MHz. O núcleo deste micro-processador, designado por

AVR32, é um RISC de 32 bits de alta performance, concebido para aplicações dedicadas de baixo custo, com ênfase particular em baixo consumo de energia, alta densidade de código e elevado desempenho de aplicativos.

Este componente implementa um *Memory Management Unit* (MMU) e um controlador flexível de interrupções que suporta os mais recentes sistemas operativos, bem como os sistemas de tempo real. O processador também inclui um rico conjunto de instruções *Digital Signal Processor* (DSP) e *Single Instruction Multiple Data* (SIMD), especialmente concebido para aplicações multimédia e de telecomunicações.

O µC contém memórias *Static Random Access Memory* (SRAM) integradas para um acesso rápido e seguro. Para aplicações que exijam memória adicional, está acessível memória externa SRAM de 16 bits. Além disso, um controlador de memória *Synchronous Dynamic Random Access Memory* (SDRAM) oferece acesso a memórias voláteis externas bem como controladores para todas as memórias não voláteis externas que sejam standards na indústria, tal como *Compact Flash*, *MultiMedia Card* (MMC), *Secure Digital* (SD) *card*, *SmartCard*, *NAND Flash* e Atmel *DataFlash*^{TM}.

O controlador de acesso directo à memória (DMA) para todos os periféricos série permite transferência de dados entre as memórias sem a intervenção do processador. Isto reduz a sobrecarga do processador quando está em transferências contínuas e de grandes fluxos entre os módulos no µC.

Os *Timers*/*Counters* (Temporizadores/Contadores) têm três canais idênticos de 16 bits. Cada canal pode ser independentemente programado para executar uma ampla gama de funções, incluindo a medição de frequências, contador de eventos, medição de intervalos de tempo, geração de impulsos, atrasos de tempo e a criação de *Pulse Width Modulation* (PWM).

Sendo esta característica a que mais realce tem, este µC também dispõe de um controlador de *Liquid Crystal Display* (LCD) integrado, que suporta varrimento simples e duplo monocromático, módulos a cores passivos LCD *Super-Twisted Nematic* (STN) e módulos de varrimento simples activos LCD *Thin-Film Transistor* (TFT). Em ecrãs monocromáticos STN, são suportados até 16 tons de cinzento usando um algoritmo baseado no tempo – *dithering* – e o método *Frame Rate Control* (FRC). Este método também é usado em ecrãs a cores STN para gerar até 4096 cores.

O controlador de LCD é programável para suportar resoluções até 2048 x 2048 de 1 a 24 bits por píxel (bpp).

O µC também tem integrado um píxel co-processador cuja responsabilidade é converter espaço de cores para imagens e vídeo, além de uma grande variedade de suporte de filtros de *hardware*.

OS módulos *Media Independent Interface* (MII) e *Reduced Media Independent Interface* (RMII) 10/100 *Ethernet Media Access Control* (MAC) integrados no µC disponibilizam soluções para dispositivos conectados à rede.

Os controladores síncronos série facilitam o acesso a protocolos de comunicação série, às normas de áudio como o *Integrated Interchip Sound* (I^2S), ao protocolo *Serial Peripheral Interface Bus* (SPI) e protocolos baseados em *frames*.

A implementação de um acelerador Java por *hardware* no AVR32 permite uma elevada velocidade de execução do código Java ao nível do *byte*.

A *Image Sensor Interface* (ISI) suporta câmaras com até 12 bits de barramentos de dados.

A conectividade PS/2 é fornecida para dispositivos de entrada padrão, tal como ratos e teclados.

O AT32AP7000 integra o sistema Nexus 2.0 *On-Chip Debug* (OCD) classe 3, com um *trace* não intrusivo em tempo real, com acesso à memória de leitura/escrita em velocidade máxima além do controlo básico do tempo de execução.

O compilador de C está proximamente ligado à arquitectura e é capaz de utilizar recursos de optimização de código, tanto para tamanho como para velocidade.

2.7. KITS DE DESENVOLVIMENTO

Com a motivação de ajudar no desenvolvimento de um projecto, quer académico quer comercial, há marcas que desenvolvem kits de desenvolvimento para os seus micro-controladores. Integrando simplesmente o µC e os seus componentes externos mais básicos, ou com o acréscimo de periféricos que tornam o kit mais atractivo para um número mais abrangente de projectos, cada kit oferece a possibilidade do projectista desenvolver apenas o código de programa sem ter a preocupação de elaborar o respectivo

circuito impresso. Assim, é afastada qualquer probabilidade de ocorrer algum problema inerente à construção de uma placa electrónica.

2.7.1. ATNGW100 [3]

No seguimento do exemplo do µC AT32AP7000, nesta sub-subsecção é descrito o respectivo kit de desenvolvimento, comercializado pela própria ATMEL: o *Network Gateway* (ATNGW100) [3].

Na Figura 20 estão identificados os principais componentes do ATNGW100 que contribuem directamente para que se consiga tirar partido de todos os recursos do µC.

No centro está, inevitavelmente, o µC, acompanhado por vários tipos de memória. Está disponível uma memória volátil SDRAM de 32 MB com barramento de 16 bits e, também, dois tipos de memória não volátil: 8 MB de memória *flash* paralela e outra memória *flash* também de 8 MB mas série. Como possibilidade de expansão, tem também uma ranhura para se poder introduzir cartões de memória SD ou MMC e, assim, obter-se mais um tipo de memória, sendo esta de extrema flexibilidade pois pode ser retirada, acedida, por exemplo, por um computador pessoal, e voltar a ser colocada no kit para que o µC possa ler e/ou escrever dados sobre ela.

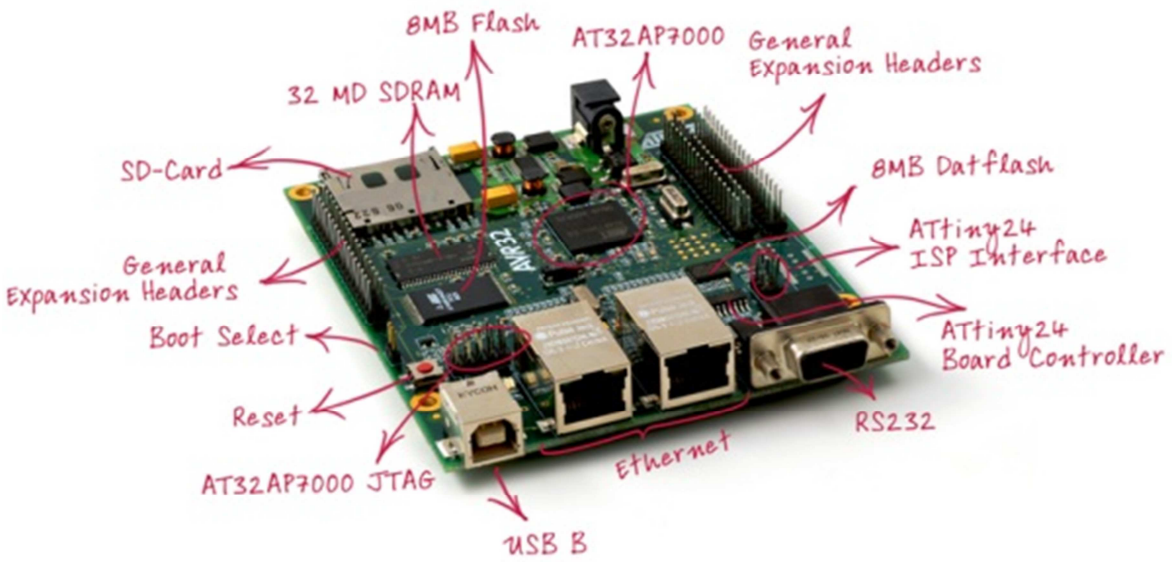

Figura 20 Visão geral dos componentes do ATNGW100 [3]

O controlo desta placa é realizado por um pequeno µC de seu nome ATtiny24. É ainda possível ter acesso a três *Light Emitting Diodes* (LEDs). Um deles, só com acesso visual, corresponde ao estado da alimentação do kit. Os outros dois podem ser controlados pelo programador. Para uma fácil e rápida reinicialização do sistema está disponível um botão de *reset*.

Outras características deste kit são os conectores disponíveis de vários tipos.

O mais importante é o conector JTAG que serve para programar o µC e, até, fazer o *debug* do programa que estiver a correr. Por outro lado, também está disponível o acesso à interface NEXUS [22], que é outro meio para programar o µC.

Ter acesso ao que se passa dentro do sistema a partir do exterior pode ser muito importante para o programador que necessite de depurar o seu código. Para este efeito, este kit tem incluída uma porta COM que possibilita uma comunicação RS232, por exemplo, com um computador pessoal.

Outro conector que se encontra nesta placa é um conector USB do tipo B, o que faz com que este sistema se possa comportar como um dispositivo tal como, por exemplo, uma impressora.

Por último, mas não menos importante, tem-se acesso a dois conectores *ethernet* que possibilitam a ligação deste sistema a redes de dados.

Nem todos os recursos do µC estão acessíveis directamente neste kit. Porém, pode-se colocar posteriormente conectores na placa que façam a expansão dos outros recursos que não estão a ser aproveitados de origem, tais como a interface para o ecrã, as comunicações RS232 ou RS485, SPI, etc.

O ATNGW100, esquematizado na Figura 21, é um sistema computacional embebido totalmente funcional.

Este sistema tem uma fonte de alimentação flexível que aceita tensões de alimentação em corrente contínua desde 9 a 15 V. Os conversores DC/DC incorporados convertem essa tensão recebida do exterior para a tensão do núcleo do CPU bem como para a tensão do resto do sistema.

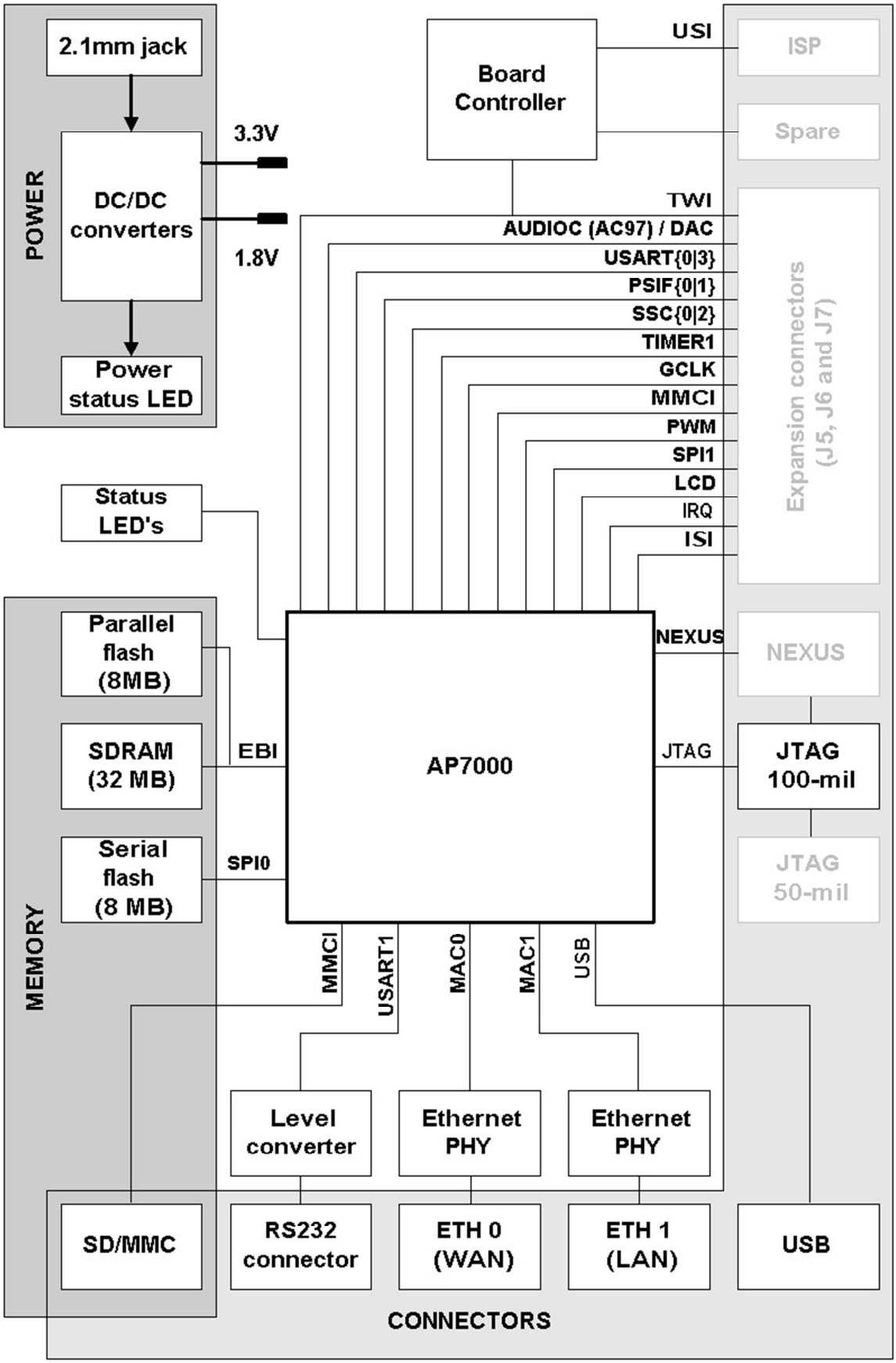

Figura 21 Diagrama de blocos do ATNGW100

O ATNGW100 tem, no total, 16 MB de memória *flash* e 32 MB de memória SDRAM mas esta memória interna pode ser expandida através de um cartão SD ou MMC inserido numa ranhura para tal propósito. O sistema é capaz de correr Linux a partir da memória *flash* incorporada. Todavia, expandindo o tamanho da memória permite aos programadores testar programas em Linux, ou mesmo sem sistema operativo algum (em *standalone*), com ocupação de memória maior que aquela que está disponível de fábrica.

Estão disponíveis três tipos de interfaces de comunicação no ATNGW100. A interface RS232 é útil para se fazer a depuração (*debug*) de baixo nível ou comunicar com o *boot loader*. As interfaces de rede são essenciais para o propósito deste kit e a ligação USB permite explorar o controlador USB 2.0 que o AP7000 contém.

O conector JTAG é usado para programar a *flash* incorporada e para fazer a depuração com um *debugger* JTAG, como o "AVR JTAGICE mkll". É também possível utilizar um *debugger* topo de gama, como o "AVR ONE!", que possui interface NEXUS.

Os conectores de expansão fornecem aos projectistas o acesso às interfaces não usadas no kit do µC, ou seja, permite-lhes criar placas expansíveis que acrescentem as funcionalidades pretendidas.

Passando agora a documentar mais extensamente cada bloco individualmente, o sistema de alimentação do ATNGW100 tem de ser alimentado externamente por uma fonte de alimentação, cuja tensão esteja compreendida entre 9 e 15 V DC e que seja capaz de fornecer 0.5 A para garantir que a potência suficiente chegue à entrada dos conversores de tensão. O LED verde do sistema acende-se quando o kit é alimentado.

O ATNGW100 tem dois circuitos de *reset*. Um é usado para o CPU e para o controlador da placa de desenvolvimento e o outro circuito de *reset* é usado para o circuito físico de rede e para a memória *flash* paralela.

Ambos os circuitos de *reset* são ligados à massa pelo mesmo botão de *reset*, se este for pressionado. Os diferentes atrasos nos tempos de subida de cada um dos dois circuitos garantem que a memória *flash* sai do estado de *reset* antes do CPU.

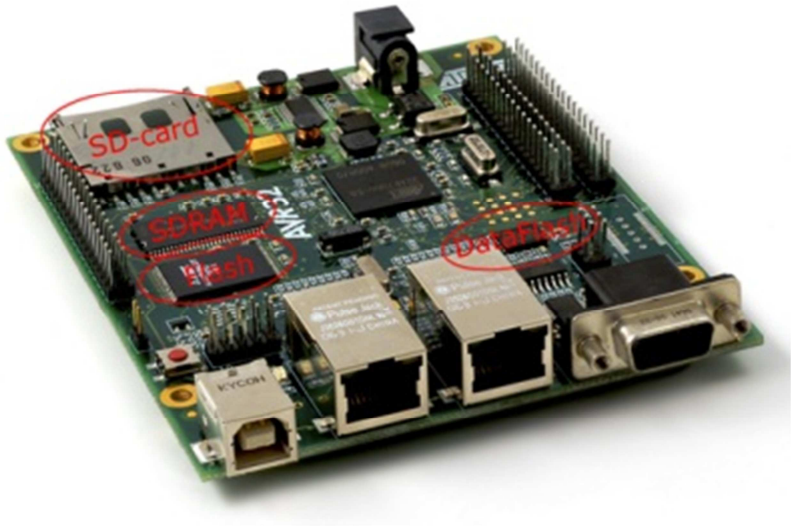

Figura 22 Localização das memórias no ATNGW100

A Figura 22 mostra a localização dos vários tipos de memórias presentes no ATNGW100, enquanto a Figura 23 apresenta a organização típica do sistema de memória num sistema Linux sobre o ATNGW100.

Uma das memórias é a SDRAM de 256 Mb, que corresponde a 4 MB x 16 bits x 4 bancos, ou seja, 32 MB cujo barramento de dados entre esta e o µC é de 16 bits. Esta memória tem alguns parâmetros importantes a reter, principalmente para as configurações do lado do *software*, tais como:

- Número de colunas: 512 (9 bits)
- Número de filas: 8192 (13 bits)
- Número de bancos: 4 (2 bits)
- Latência *Column Address Strobe* (CAS): 2
- Taxa de actualização: 15.6 µs
- Tempo de recuperação: 2 clocks

O ATNGW100 tem uma memória *flash* paralela de 64 Mb, ou 8 MB, com um barramento de dados entre esta e o µC também de 16 bits. Esta memória serve para armazenar o código de programa com que o µC vai correr.

No ATNGW100 está presente outra memória *flash*, também de 8 MB, mas esta é serie, ao contrário da anterior citada.

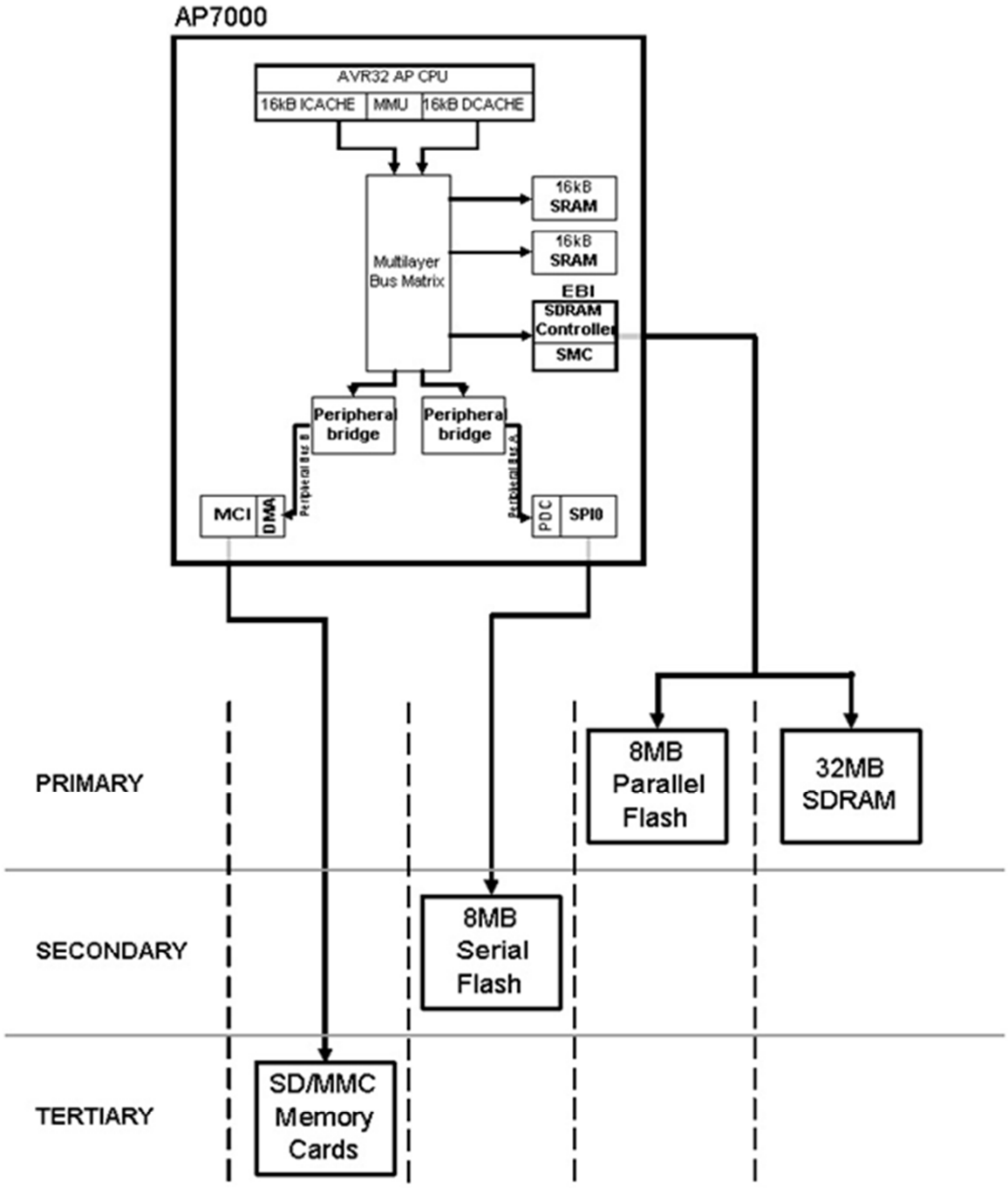

Figura 23 Organização do sistema de memória no ATNGW100

Esta memória *flash* série de dados está ligada ao µC através da interface SPI e está pré-carregada com o sistema de ficheiros do sistema operativo Linux, isto é, programa e dados para as funcionalidades do ATNGW100. De salientar ainda que o acesso a esta tem uma velocidade inferior comparado com a *flash* paralela.

Por último, mas não menos importante, é a ranhura para cartões de memória SD ou MMC. Isto cria a possibilidade de se expandir a memória já presente no kit de uma forma muito

prática e facilmente removível ou substituível. Para acrescentar, a interface entre a ranhura e o µC adoptada no ATNGW100 é a *MultiMedia Card Interface* (MCI), proprietária da ATMEL, sendo mais rápida que a outra interface maioritariamente adoptada: a SPI.

Como o ATNGW100, como o próprio nome indica, é um *network gateway*, não poderiam faltar as ligações *Ethernet* físicas. Estão presentes dois conectores, o mesmo número de interfaces que o µC disponibiliza para este propósito, um designado por MACB0 e outro por MACB1, podendo ser utilizados como *Wide Area Network* (WAN) e *Local Area Network* (LAN), respectivamente. A interface do AP7000 com estas *Ethernet* físicas chama-se *Media Independent Interface* (MII).

O µC AP7000 presente no ATNGW100 tem três fontes de relógio: dois cristais, XC0 e XC1, a funcionarem a 20 e 12 MHz, respectivamente, e um outro, mais lento, de 32 KHz (XC2) que tem como função gerar os impulsos para relógio de tempo real.

Os diferentes módulos no AP7000 podem ser configurados recorrendo directamente ao XC0, XC1 ou mesmo às multiplicações do relógio através dos módulos *Phase-Locked Loop* (PLL) disponíveis no µC.

Os módulos PLL, PLL0 e PLL1, estão ligados directamente ao XC0 e XC1, respectivamente, cujas saídas são filtradas para que se atinjam as frequências máxima e mínima do ATNGW100.

O ATNGW100 tem um circuito integrado, mais propriamente um µC mas mais pequeno e bem mais simples que o AP7000, que tem como função controlar a placa de desenvolvimento – o ATtiny24. Este guarda internamente as identificações do fabrico, do produto e da revisão, estando as identificações do fabricante e do produto inseridas no *firmware* do controlador do kit. Este controlador também pode ler as tensões principal e do sistema e reportá-las para o CPU.

É utilizado o *Power Management Bus* (PMBus™) padrão para fazer a comunicação entre o CPU e o ATtiny24. E o tipo de comunicação física entre o AP7000 e este controlador é o *Two Wire Interface* (TWI).

O *firmware* do ATtiny24 pode ser substituído utilizando as ferramentas habituais de programação dos µC AVR8 (de 8 bits), e programado por *In System Programmer* (ISP).

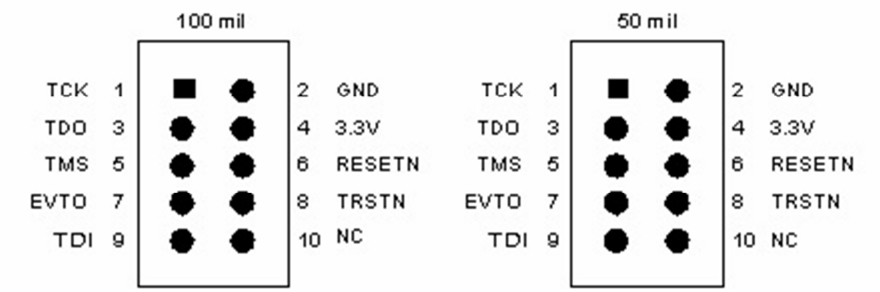

Figura 24 Conectores JTAG do ATNGW100

Figura 25 JTAGICE-mkll

O ATNGW100 tem dois tipos de interfaces para programar e depurar o AP7000: o JTAG e o NEXUS. Mas apenas o conector do primeiro vem, desde a fábrica, montado no sistema. E de notar que é o conector de 100 mil, pois ainda há outro de 50 mil, como se pode ver na Figura 24.

A interface JTAG permite programar a *flash* externa e fazer a depuração do *software* que esteja a correr no AP7000. Para tal, é imprescindível ter um programador compatível com o JTAG, nomeadamente o "JTAGICE-mkll", visível na Figura 25.

Para terminar a descrição do ATNGW100, há que referenciar os importantes conectores de expansão. Eles são três e disponibilizam para o seu exterior todos os periféricos que não foram aproveitados pela placa de desenvolvimento, tais como:

- *Universal Synchronous Asynchronous Receiver Transmitter* 0 (USART0) e USART3;
- *PS/2 Interface* 0 (PSIF0) e PSIF1;
- *Synchronous Serial Controller* 0 (SSC0) e SSC2;

- *Two-Wire Interface* (TWI) (ou *Inter-Intergrated Circuit* (I^2C));
- *Pulse Width Modulation* (PWM);
- *TIMER1*;
- *Serial Peripheral Interface* 1 (SPI1);
- *Image Sensor Interface* (ISI);
- Interface do conversor digital-analógico (DAC);
- Interface do controlador AC97 (AC97C);
- *MultiMedia Card Interface* (MCI);
- Interrupções externas;
- Linhas de saída do controlador de LCD;
- Linhas *General Purpose Input/Output* (GPIO).

De salientar duas notas. A primeira é o facto dos pinos do µC serem multiplexados, isto é, embora cada pino possa ter duas funcionalidades, apenas uma pode estar configurada. A segunda direcciona-se na necessidade de, caso se tenha de recorrer a um destes periféricos supracitados, serem criadas todas as condições para que se faça a interface a nível do *hardware* entre o ou os conectores do ATNGW100 e o periférico à parte.

2.8. APLICAÇÕES DE ECRÃS EM ELEVADORES

Nesta sub-secção são citados e referenciados alguns exemplos de ecrãs aplicáveis em elevadores, que tanto podem ser implementados na cabina do elevador como no patamar. De sublinhar que embora sejam exemplos, não diferem muito dos restantes produtos existentes no mercado de elevadores, principalmente a nível do aspecto para o utilizador. Com excepção dos ecrãs com resoluções elevadas, devido à diversidade de *layouts* e conteúdos possíveis.

No que diz respeito aos ecrãs de segmentos, habitualmente são constituídos por 2 dígitos de 8 ou 16 segmentos cada e 2 representações de setas, que se pode ver no documento [23] e na página da internet [24].

Ainda dentro dos ecrãs a leds, há os de matriz de pontos, cujos exemplos podem ser vistos nas páginas de internet [25] e [26].

Mais recentemente, os ecrãs implementados nos elevadores são os LCD monocromáticos, que estão exemplificados com 2 elementos na página [27].

Entrando agora na chamada próxima geração de ecrãs na área dos elevadores, cita-se os ecrãs LCD a cores, mais propriamente os TFT-LCD, que podem ter tamanhos bem maiores face aos casos anteriores. Dois exemplos deste tipo de ecrãs estão identificados e especificados nas páginas de internet [28] e [29].

Por último, pode-se visualizar na Figura 26 uma imagem exemplificativa do *layout* de um ecrã TFT-LCD, que é controlado por um sistema cujo projecto deste trabalho pretende substituir. Este sistema, também projectado pela empresa patrocinadora deste trabalho, é constituído por um computador e pelo controlador PrismaECO II, descrito na sub-subsecção 2.5.2.

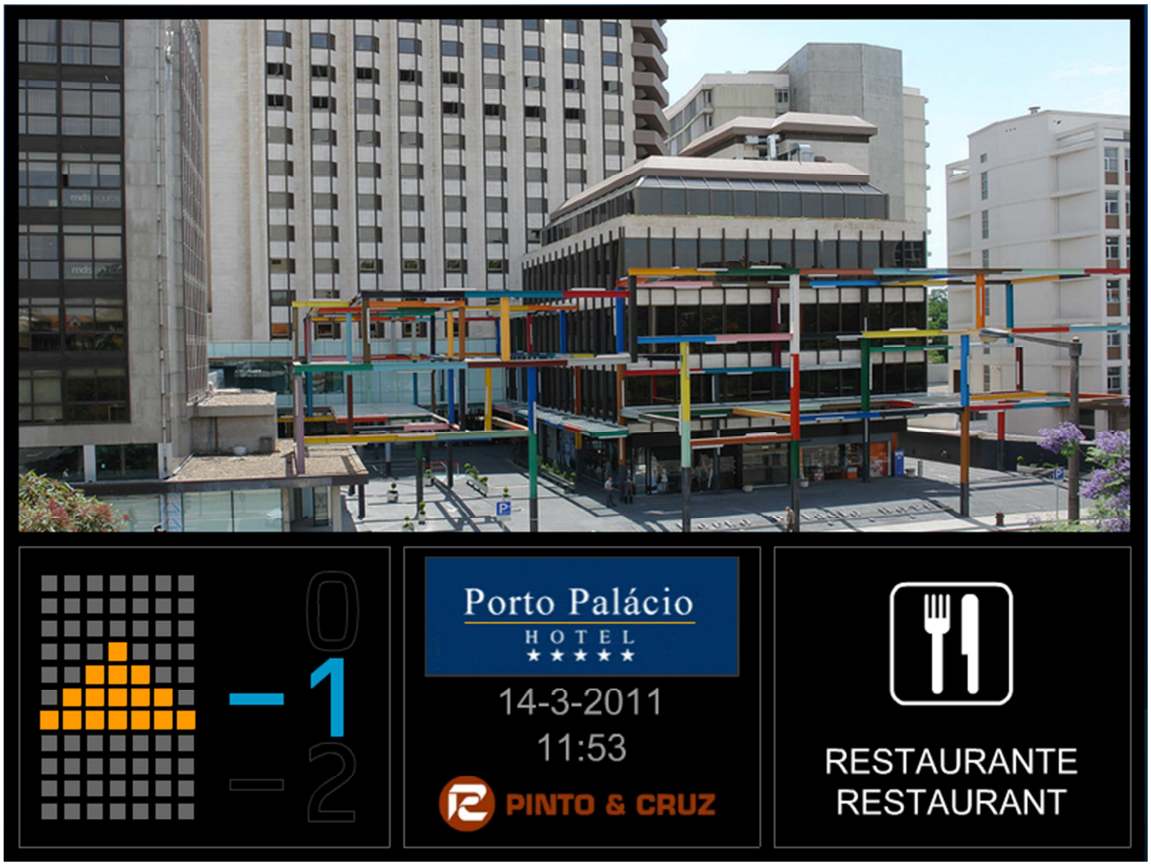

Figura 26 Imagem exemplificativa do ecrã controlado por um sistema com computador

3. APRESENTAÇÃO DO PROJECTO

Este capítulo apresenta uma visão geral do projecto realizado no âmbito deste trabalho: um controlador dedicado de ecrãs LCD/TFT.

A apresentação foi estruturada em dois módulos: no *hardware* e no *software*. No primeiro é mostrado um diagrama de blocos do sistema, complementado com uma primeira e breve explicação de cada um deles. No segundo é feita uma abordagem às áreas informativas do produto no ecrã, seguida de uma esquematização do fluxograma do código do programa.

O objectivo primordial foi projectar um sistema dedicado que cumpra os objectivos inicialmente propostos, e não por excesso de potencialidades desnecessárias. Assim, com um sistema embebido, obtém-se uma componente física do projecto – *hardware* – muito mais compacta e menos complexa face à outra proposta concorrente já introduzida no mercado: um controlador baseado num computador comum. Em suma, traduz-se num produto muito menos dispendioso que o actual concorrente.

Como já foi referido anteriormente, os objectivos deste projecto estão estritamente ligados aos objectivos definidos para este produto por parte da empresa Francisco Parracho – Electrónica Industrial, Lda., que suportou todo este trabalho. Portanto, todas as opções,

incluindo funcionalidades e disposições, quer a nível de componentes quer a nível de *layout*, foram tomadas baseadas nas necessidades da área de elevação, pois é esta a área de negócios desta empresa.

3.1. HARDWARE

Este projecto exigiu o desenvolvimento de uma placa de circuito impresso principal, designada por EPC, onde foram implementados os periféricos e onde encaixa o módulo ICnova AP7000 OEM [4]. O diagrama de blocos do sistema pode ser visualizado na Figura 27.

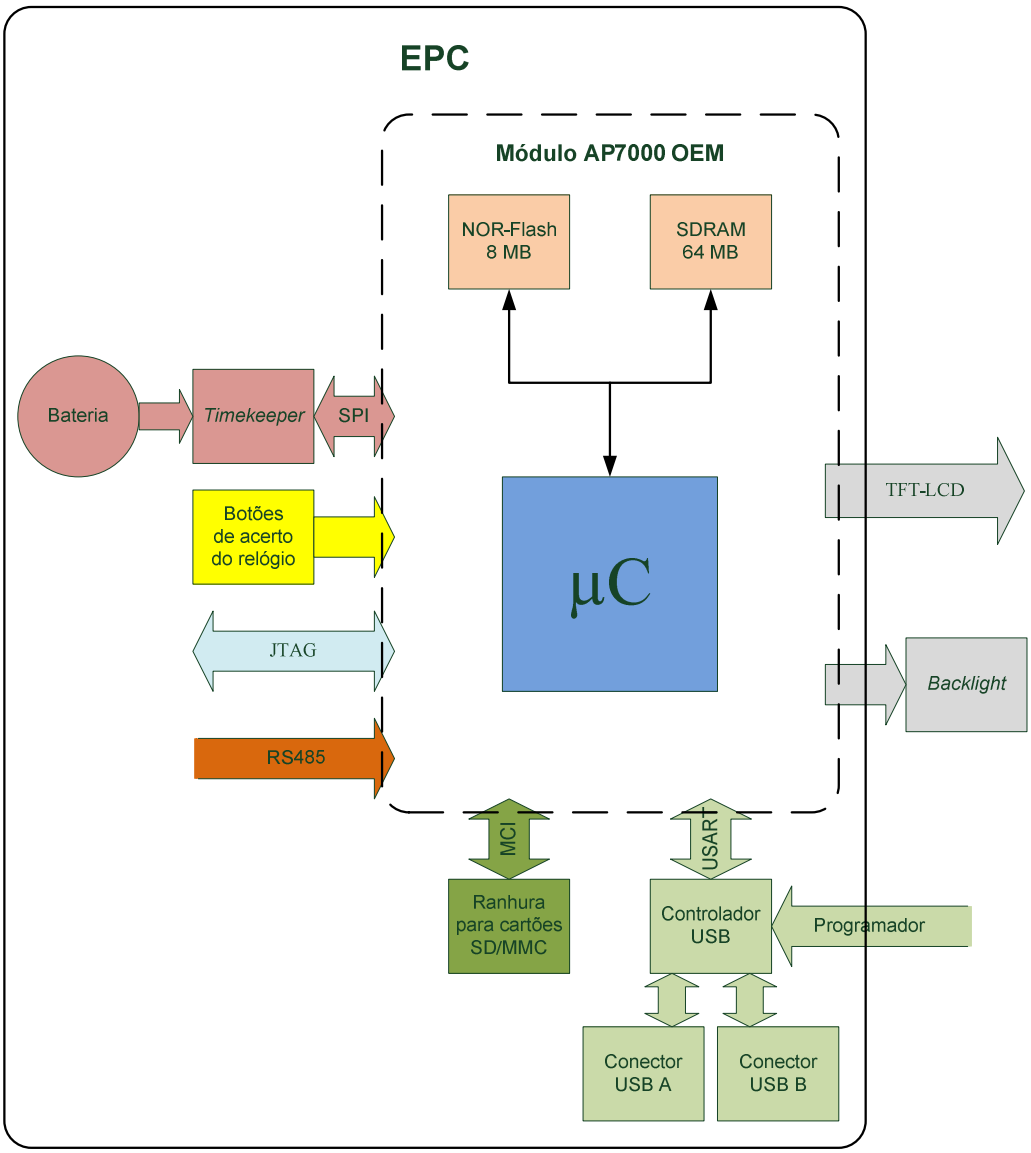

Figura 27 Diagrama de blocos sintético do *hardware*

De sublinhar que o micro-controlador (µC) [1] é o centro de todas as comunicações e, por isso, é o responsável tanto por receber e transmitir de ou para qualquer periférico, como de processar os respectivos dados. Este é programado através da interface JTAG.

O módulo ICnova AP7000 OEM [4] é constituído por um µC de 32 bits da ATMEL, de alto desempenho mas com baixo consumo, o AT32AP7000 [1]. Como principais características, tem um desempenho de 210 *Dhrystone Million Instructions Per Second* (DMIPS) a 150 MHz, tem interfaces para memória de acesso aleatório dinâmica (SDRAM) e para cartão de memória SD/MMC, entre muitas outras potencialidades. A característica mais importante para este trabalho é o controlador interno de *Liquid Crystal Display* (LCD), nomeadamente de *Thin Film Transistor* (TFT). Integrados neste módulo estão também uma memória SDRAM de 64 MB com barramento de dados de 32 bits e outra *NOR-Flash* de 8 MB. Na primeira são armazenados, temporariamente, todos os dados e as imagens processados pelo µC e, a última, serve para guardar o código de programa que é descarregado, através de um programador, pelo computador que contém o programa.

Na placa dedicada EPC estão disponíveis as funcionalidades deste *hardware*. A fundamental é a alimentação de 24 V em corrente contínua (DC) que alimenta três fontes de alimentação distintas no sistema. No mesmo conector da alimentação tem de ser ligado a comunicação RS485 para que o sistema receba os dados externos e os processe. Uma das três tensões presentes no circuito impresso serve para alimentar o *backlight* (através de um inversor, com a função de aumentar a sua tensão de entrada para ligar as lâmpadas situadas no módulo TFT-LCD) do ecrã cuja ligação é feita através de outro conector existente na placa. A imagem mostrada no ecrã também pode ser invertida, espelhada ou ambas em combinação, e para se poder definir uma dessas opções está à disposição um micro-interruptor de duas unidades que, consoante a combinação, define um de quatro possíveis modos de visualização do ecrã.

Uma outra funcionalidade potenciada por este sistema dedicado é o suporte de cartões de memória SD/MMC. No arranque, o µC tem de ler um ficheiro que indica as informações a carregar no sistema e, para tal, tem de estar disponível no referido cartão.

Para suportar todo o processo envolvente do relógio, estão presentes três blocos: o *timekeeper*, que é o integrado que gere o relógio, um super condensador com o respectivo circuito de recarregamento, para manter a alimentação do *timekeeper* e, assim, garantir o

funcionamento do relógio por um tempo superior a duas semanas quando o sistema se encontra sem alimentação, e três botões: “OK”, “para cima” e “para baixo” cujas funções serão explicitadas na subsecção seguinte.

Durante o funcionamento normal do controlador é possível actualizar as informações essenciais para o sistema. Esta actualização pode ser feita através da leitura de um *flash disk* (ou *pen drive*) que contenha esse tal ficheiro com os novos dados. Para tal, encontra-se na placa um circuito integrado – designado por Vinculum [30] – que tem a função de ler qualquer *pen drive* inserida num conector (tipo A), também disponível, e enviar os dados pretendidos para o µC (por troca de comandos). Como este integrado também tem a capacidade de comunicar com um *host*, um computador por exemplo, está também implementado um conector USB do tipo B no circuito, porém, como não faz parte do âmbito deste trabalho esta funcionalidade não está implementada no *software*. Como o Vinculum vem de fábrica em branco, é necessário que a placa tenha uma ligação (usada só na primeira utilização) para o exterior de modo que se possa programá-lo por computador. Esta programação é realizada através do programador UB232R [31], também da empresa FTDI.

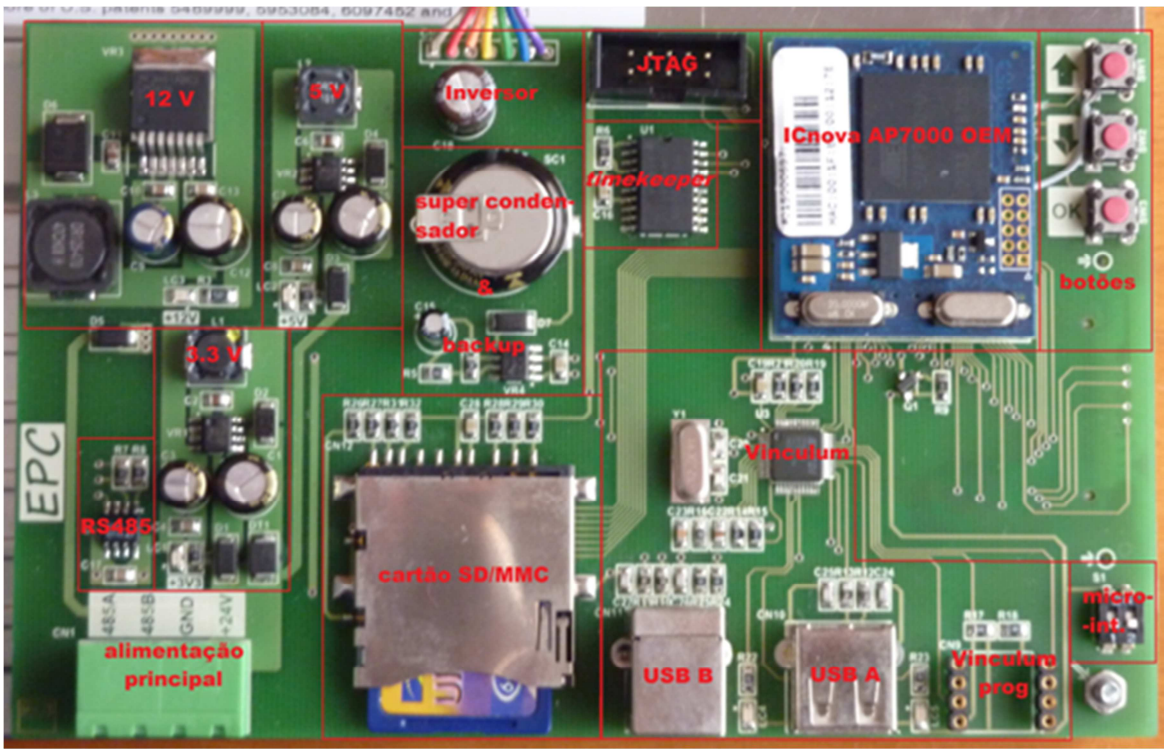

Figura 28 Identificação dos blocos na placa de circuito impresso EPC

A Figura 28 apresenta a placa de circuito impresso EPC onde estão identificados todos os blocos descritos nesta subsecção.

Já na Figura 29 pode-se constatar o tamanho reduzido do controlador por comparação com o próprio ecrã. A placa de circuito impresso EPC tem de ficar encostada ao módulo TFT-LCD para haver ligação directa, via conectores, entre estes dois. Assim, evita-se qualquer interferência face à possibilidade desta ligação ser feita através de uma fita. Fita essa que é utilizada para fazer a interface entre a placa e o inversor.

Na Figura 30 e na Figura 31 é mostrado o circuito impresso da placa EPC, vista de frente e vista de trás, respectivamente, mesmo antes de ser montado e soldado qualquer componente, inclusive o módulo ICnova AP7000 OEM. O respectivo *Printed Circuit Board* (PCB) é apresentado no Anexo A.

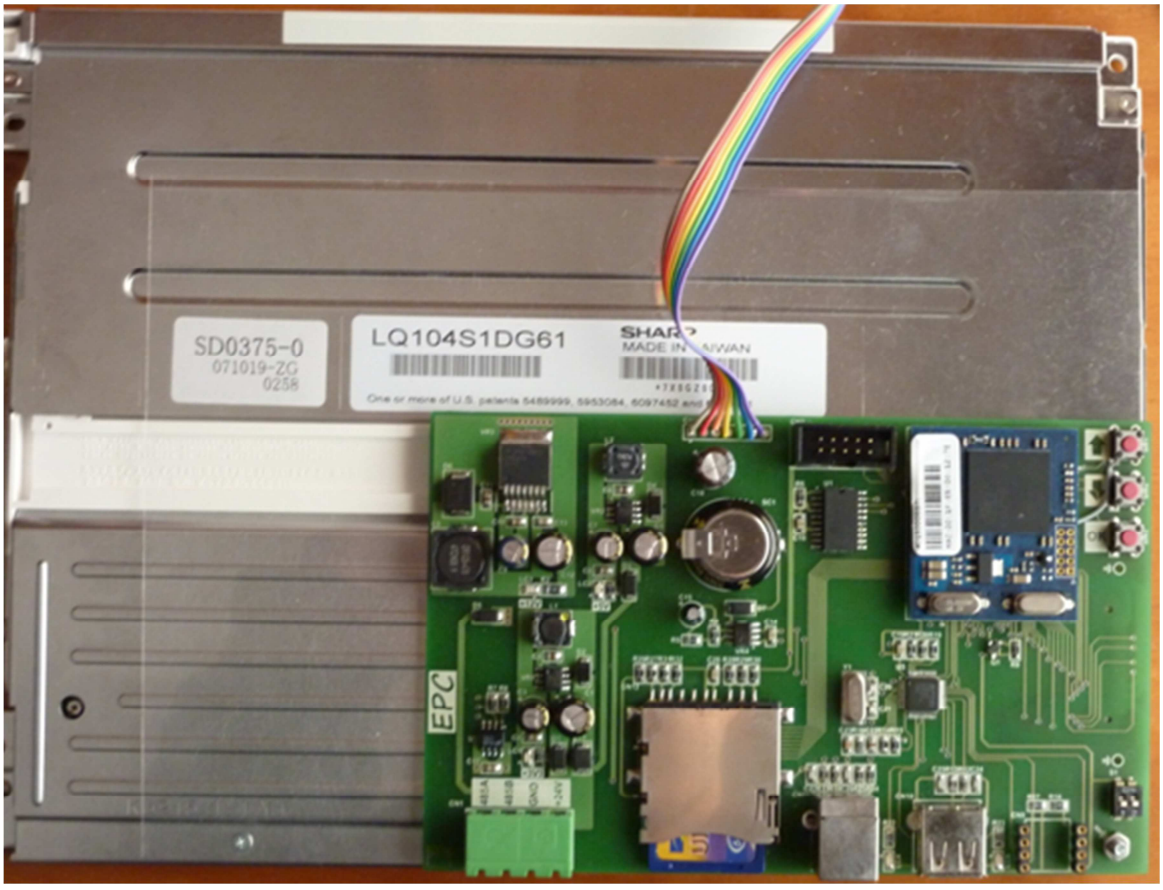

Figura 29 Enquadramento da placa de circuito impresso EPC com o ecrã

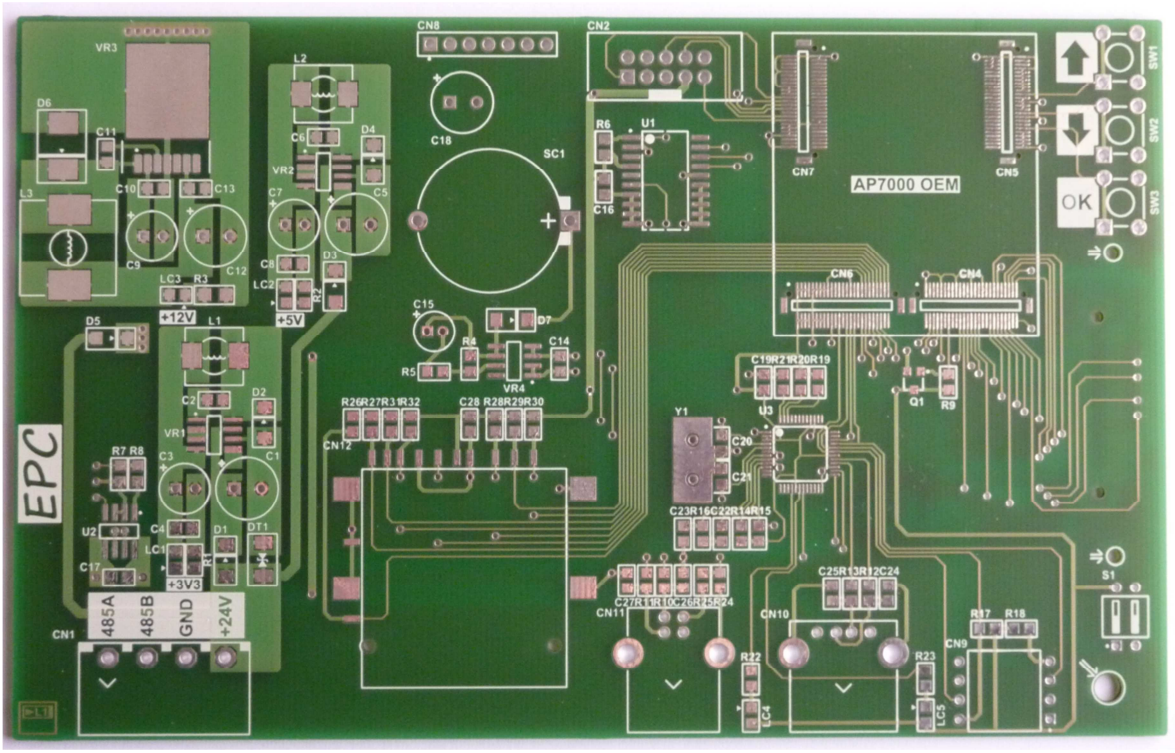

Figura 30 Circuito impresso da placa EPC (frente)

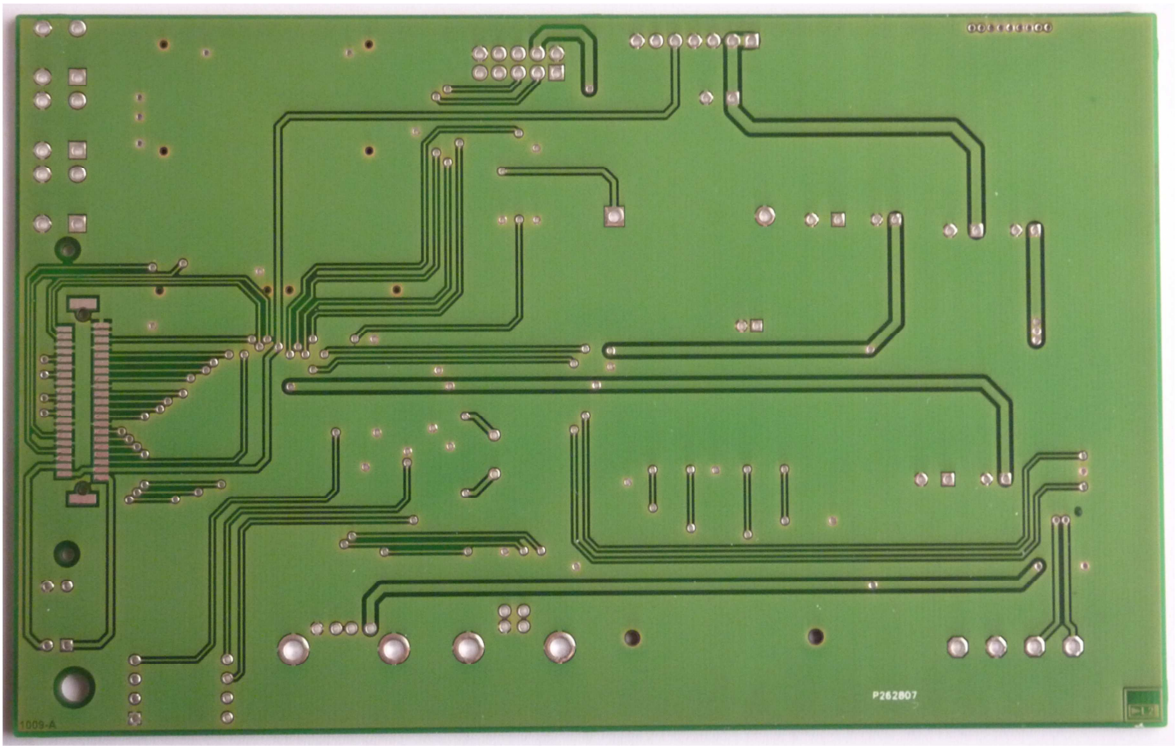

Figura 31 Circuito impresso da placa EPC (verso)

3.2. SOFTWARE

Esta subsecção está dividida em duas partes: nas funcionalidades do projecto e no fluxograma do código de programa. Na primeira, a abordagem principal são as áreas informativas disponibilizadas no ecrã. Na segunda parte, a apresentação do fluxograma sintético do código de programa do projecto é complementado com a respectiva explicação.

3.2.1. FUNCIONALIDADES

O *layout* referente à disposição das imagens/indicações no ecrã é fixo (Figura 32) para o cliente, porém, é de fácil e rápida alteração no momento de programar o µC, caso seja esse o pedido.

Na área 1 da Figura 32 enquadra-se uma imagem, única ou periodicamente alterada, que se pode destinar a publicidade do local onde o elevador e o controlador do ecrã se encontram. Há a possibilidade de atribuir uma ou várias imagens para cada piso, comuns ou distintas entre eles, em que, caso seja mais do que uma imagem carregada, são alteradas periódica e sequencialmente, bem como imagens que alertem para, por exemplo, excesso de carga, elevador em manutenção, incêndio, etc.

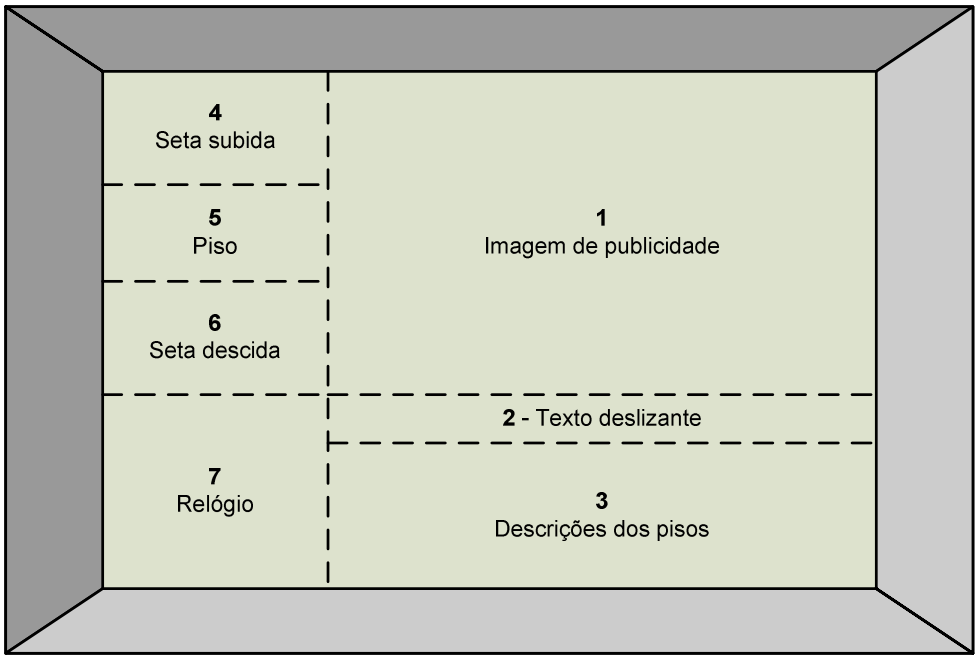

Figura 32 *Layout* predefinido das áreas informativas do produto

Associado a cada piso deve corresponder uma descrição, meramente informativa, de modo que o utilizador do elevador tenha a percepção do andar em que está [a passar], como por exemplo restaurante ou quartos de X a Y. Por defeito, esta indicação fica situada na área 3 da Figura 32 e tem uma funcionalidade relevante: o número de descrições visíveis no ecrã respectivas a cada piso pode variar (na altura da programação). Pode simplesmente aparecer uma de cada vez, três, cinco ou quantas descrições o cliente quiser (desde que seja número ímpar pela razão mais à frente indicada), desde que não exceda os limites do *layout*, que depende também do tamanho da fonte que for utilizado. Para acrescentar, pois é importante o utilizador do elevador saber em que andar está no momento, este piso estará sempre destacado dos outros com uma cor à escolha. Com o elevador em movimento, haverá, naturalmente, constantes alterações de pisos e, por conseguinte, o referido destaque também alterar-se-á mas não de uma forma repentina e sim animadamente, ou seja, a passagem entre pisos é feita suavemente. Devido a esta animação, não pode haver um número par de descrições dos pisos porque senão, em caso do piso a destacar não ser um das extremidades, este não poderia ficar no meio dos outros andares visíveis e, assim, a passagem entre pisos ocorrer de forma constante.

Na área 2 da Figura 32 pode ser definido um texto horizontalmente deslizante com qualquer tipo de informação que o cliente deseje expor aos utilizadores do elevador. Um exemplo de texto pode ser o horário de almoço e/ou de jantar, caso o ecrã (e o elevador) se encontre num hotel.

Outra informação que se enquadra no *layout*, na área 5 da Figura 32, é o insubstituível número, ou letra(s), do piso presente. Como acontece com a descrição dos pisos, esta indicação também é animada, mas neste caso só aparece uma linha de cada vez, constituída por números e/ou letras. Tanto esta como a da descrição dos pisos são animações verticais sincronizadas entre si.

Nas áreas 4 e 6 da Figura 32 ficam representadas duas setas distintas, para cima e para baixo, respectivamente. Há três estados possíveis para ambas as setas: “apagada”, que no fundo é uma imagem da seta sombreada e que indica que o elevador está parado ao piso; “activa”, que é simplesmente uma imagem da seta estática, informando que o elevador está parado mas que vai subir ou descer; e o terceiro estado é o movimento, que é representado por uma sucessão de trocas de imagens de setas de tamanhos diferentes, para que dê a sensação que a seta se afasta e aproxima, sucessivamente.

Para finalizar a apresentação do *layout*, na área 7 da Figura 32 é mostrado um relógio digital, onde se pode visualizar a data, o dia da semana, horas e minutos.

Para acertar este relógio são disponibilizados três botões na parte de trás do ecrã (no controlador deste, mais propriamente), como se pode ver na Figura 28, Figura 29 ou na Figura 30: um botão com a designação "OK", outro com uma seta para cima e o terceiro com uma seta para baixo. O primeiro tem três funções: se for carregado durante, pelo menos, dois segundos inicia o acerto e começam a piscar os primeiros dígitos a alterar (foi escolhido o ano). Após isto, se for novamente pressionado durante outros tantos segundos, deixa de piscar qualquer parte do relógio e é feito o acerto do relógio internamente. Se pressionado mas este tempo não for excedido, os dígitos que estão actualmente a piscar param de piscar e passam a ser os próximos a ter essa particularidade, e assim sucessivamente – ano, mês, dia, dia da semana, horas e depois minutos. Os outros dois botões se forem pressionados por um tempo inferior a dois segundos, o número que estiver a piscar na altura é aumentado ou diminuído num valor; se o tempo for ultrapassado, então o número será aumentado ou diminuído de uma forma bem mais rápida e sequencial até que seja largado o botão. Como é habitual, se o número atingir o valor máximo passa para o valor mínimo e, por outro lado, se o número atingir o valor mínimo passa para o valor máximo.

De salientar que o relógio deste sistema actualiza-se automaticamente de acordo com a hora do horário de Verão, em Portugal. Isto acontece em dois momentos, no último Domingo de Março e no último Domingo de Outubro e, mesmo que o sistema atravesse estas datas desligado da alimentação, a hora será actualizada logo no seu arranque, desde que a bateria (que garante internamente o funcionamento do relógio nesta situação) esteja carregada. É garantido que tanto esta actualização como o ajuste dos anos bissextos é feita até ao ano 2099.

A Figura 33 apresenta um exemplo de *layout* onde se pode verificar as áreas informativas descritas acima. É de sublinhar que este é só um exemplo, é a disposição predefinida, isto é, quer as posições das áreas informativas quer as próprias cores são facilmente alteráveis, desde que seja em conformidade.

Por sua vez, a Figura 34 apresenta um exemplo de uma indicação de alarme recebida do exterior (via RS485), neste caso é o elevador em manutenção.

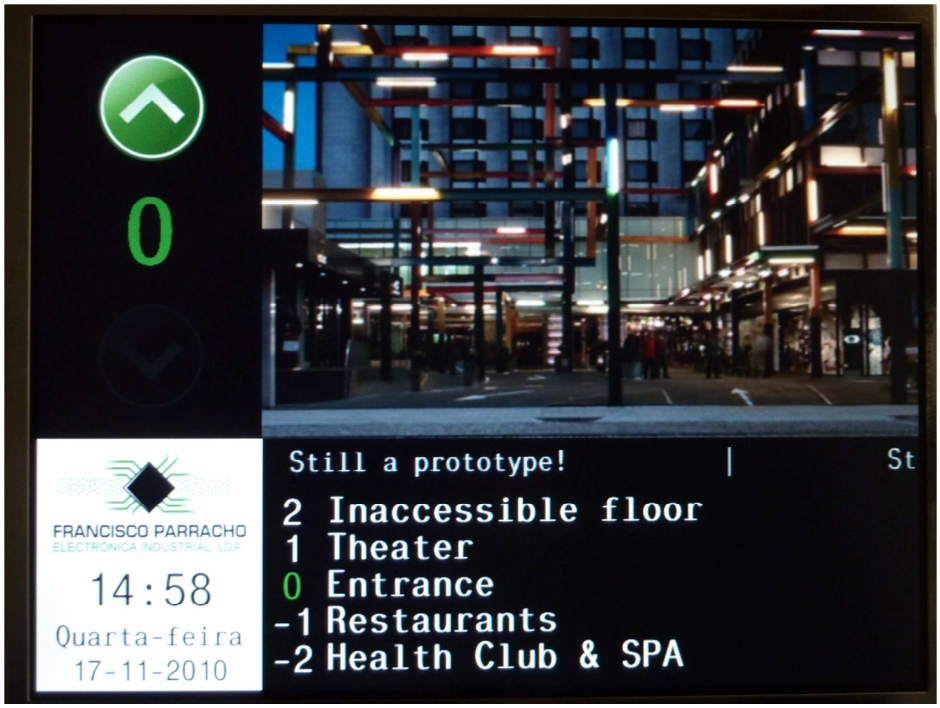

Figura 33 Exemplo de *layout* das áreas informativas do produto

Figura 34 Exemplo de uma indicação de alarme

A última funcionalidade de destaque a referir é a capacidade de se poder actualizar/alterar as definições do sistema, tais como: os pisos, e as respectivas descrições, as imagens a carregarem e a própria sequência a mostrar, e, naturalmente, a informação que será

visualizada no texto horizontalmente deslizante. Por outras palavras, é possível, através da leitura de uma memória *Universal Serial Bus* (USB) *flash drive*, substituir o ficheiro de dados que é lido sempre no arranque do sistema e que contém os dados anteriormente citados. Este ficheiro tem de estar sempre inserido no cartão de memória *Secure Digital* (SD) ou *Multi Media Card* (MMC) pois é com os dados introduzidos, num formato próprio, nesse ficheiro que o sistema "sabe" o que deverá processar.

3.2.2. FLUXOGRAMA

Na Figura 35 está esquematizado, muito resumidamente, o fluxograma do código do programa que permite que o µC processe todos os dados que lhe compete.

Logo no arranque são inicializados os recursos imprescindíveis ao bom funcionamento do µC, inclusive a definição das memórias, o controlador do cartão de memória SD/MMC e o controlador de LCD, entre outros.

Realizada esta etapa com sucesso, é imediatamente carregada e mostrada a imagem do logótipo da empresa mãe deste projecto para que, entretanto, seja lido o ficheiro que contém a informação a indicar quantas, quais e qual a sequência das imagens que devem ser carregadas no sistema (para a memória), tanto para os pisos como também para as mensagens de alerta, o número do piso e a respectiva descrição e, também, a frase inserida no texto horizontalmente deslizante. Efectuada esta leitura é, então, processado tudo o que é necessário para cumprir os dados contidos no ficheiro.

De seguida são inicializados os restantes recursos que serão utilizados ou que podem ser utilizados esporadicamente tais como o relógio, o controlador USB e a comunicação RS485.

A partir daqui o programa entra num ciclo infinito. Deixa de executar funções de arranque para passar a executar funções cíclicas.

A primeira etapa repetitiva é o processamento dos dados recebidos via RS485, que contém informações sobre o piso, se o elevador está em movimento ou não, em que sentido se está a deslocar e a mensagem de alerta que deve ser, eventualmente, mostrada no ecrã. Em anexo vem o código *Cyclic Redundancy Check* (CRC) respectivo, possibilitando a detecção de erros de transmissão ou de armazenamento.

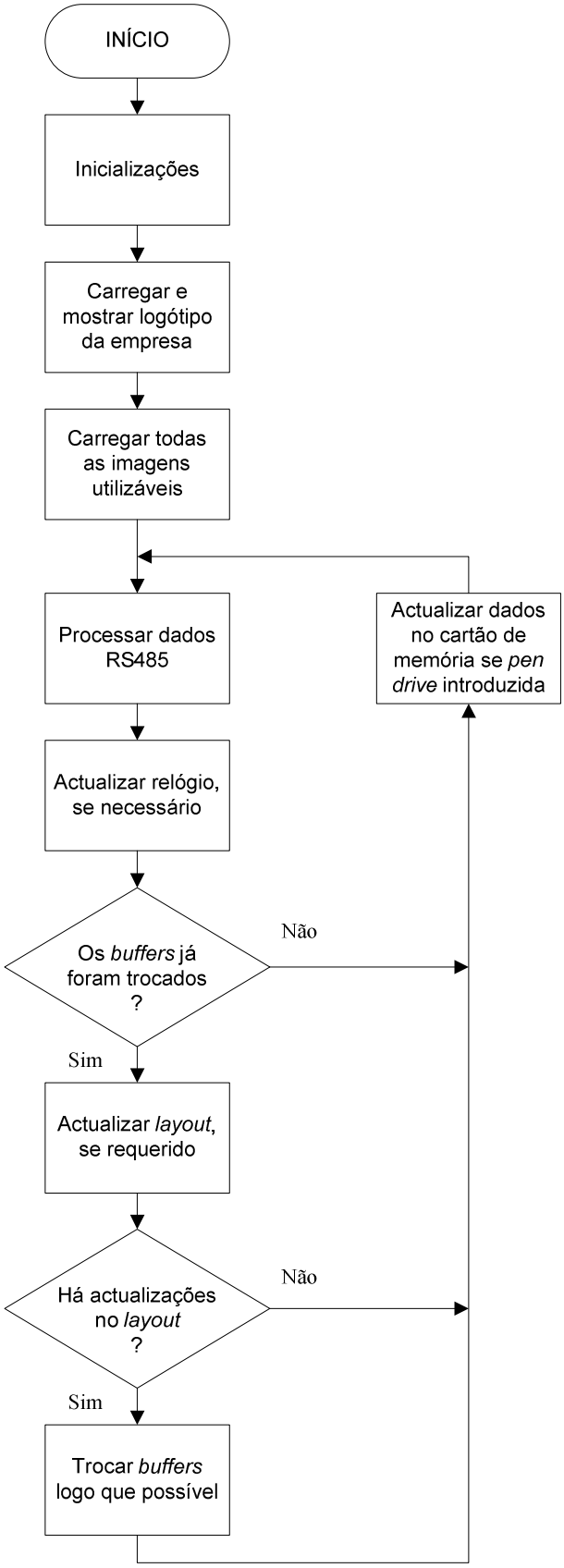

Figura 35 Fluxograma sintético do *software*

O próximo passo é verificar se está na altura de se actualizar o relógio e, se sim, ler o *timekeeper* e actualizar o relógio no ecrã. Há duas condições para que isto ocorra: ou já ter passado um minuto desde a última actualização, para que o relógio mostrado no ecrã esteja sempre sincronizado ao minuto com o relógio interno, ou então o pressionar dos botões ter despoletado o modo de actualização do relógio.

É utilizada a técnica do duplo *buffer*, ou seja, enquanto um *buffer* está a ser mostrado há um outro que já está a ser carregado e, quando este estiver pronto a ser mostrado, os *buffers* são trocados. Cada *buffer* corresponde a uma área de memória, com o tamanho de um ecrã, que contém todo o *layout* a ser mostrado.

Devido a este facto é imperativo que só seja reconstruído o novo *layout* após os *buffers* serem trocados, isto para não acontecer o caso de se estar a escrever em memória por cima de dados que já estariam prontos para serem mostrados. Depois de estar assegurada esta condição é que são realizadas as actualizações necessárias ao próximo *layout* a ser apresentado. Logo de seguida é activado o pedido de troca dos *buffers*. Esta troca não é efectuada de imediato mas sim só no final da *frame*, ou seja, no final da escrita da imagem do ecrã por parte do controlador de LCD.

Por fim, caso o cliente tenha inserido uma *pen drive* na ranhura para ela destinada, é despoletado todo o processo de actualização dos dados a carregar no arranque. Se o ficheiro pré-definido estiver presente e de acordo com o previsto, e sem erros, é feita a actualização no cartão de memória SD/MMC e o sistema é reinicializado para que o novo *layout* entre em vigor. Caso contrário, em vez de ser mostrada uma mensagem de sucesso, é apresentado um alerta de falha e o sistema retoma o seu funcionamento anterior à interrupção.

4. HARDWARE

Este capítulo é responsável por descrever toda a componente física deste trabalho – o *hardware*.

Inicialmente, foi adquirida uma placa de desenvolvimento – designada por ATNGW100 [3] – desenhada e vendida pela própria ATMEL, empresa que criou o micro-controlador (µC) escolhido, com um conjunto de periféricos que suportam grande parte dos projectos que se queira desenvolver baseado neste µC, inclusive este em concreto. Mas para que se tivesse conseguido desenvolver e testar todo o *software* que cumprisse com os objectivos propostos foi necessário criar certas interfaces, nomeadamente com o ecrã *Thin Film Transistor – Liquid Crystal Display* (TFT-LCD). No final, após estar garantido o funcionamento dos recursos básicos deste trabalho, foi, então, projectado o protótipo que inclui apenas as funcionalidades necessárias para este projecto, inclusive algumas que não estavam disponíveis na placa de desenvolvimento e, por isso, não podiam ser testadas nessa altura.

4.1. ECRÃ TFT-LCD

Neste trabalho é imprescindível haver um ecrã pois é o dispositivo final do projecto, aquele que é efectivamente visível para o cliente. O ecrã escolhido pela empresa foi um LCD de matriz activa – TFT-LCD – da marca Sharp, modelo LQ104S1DG61 [2]. É um ecrã de

10.4” (26 cm) de tamanho, com uma resolução de 800 x 600 píxeis a 18 bits por píxel (6 bits por cada componente de cor).

As suas características podem ser analisadas na sub-subsecção 2.2.5 (capítulo estado do conhecimento na área).

4.1.1. INVERSOR PARA ALIMENTAR O *BACKLIGHT* DO ECRÃ

O sistema de retro-iluminação (*backlight*) deste ecrã TFT-LCD tem duas lâmpadas *Cold Cathode Fluorescent Tubes* (CCFTs), que precisam de ser alimentadas por uma tensão alternada máxima de 1300 Vrms.

Para alimentar estas duas lâmpadas foi adquirido, à parte do ecrã pois não vem incluído neste, um inversor, o CXA-0454, da TDK-Lambda [10], recomendado pelo próprio fabricante deste ecrã TFT-LCD e apresentado na sub-subsecção 2.3.1 (capítulo estado do conhecimento na área).

4.2. MICRO-CONTROLADOR (µC)

O componente “cérebro” optado para este projecto foi um µC de 32 bits da ATMEL, da família AVR32, cuja designação é AT32AP7000 [1]. As suas características estão descritas na sub-subsecção 2.6.1 (capítulo estado do conhecimento na área).

4.2.1. JUSTIFICAÇÃO DA ESCOLHA

Há três principais razões para justificar a escolha do µC AT32AP7000.

A primeira é o facto de ser um componente da ATMEL, cujos produtos já são adquiridos e implementados noutros projectos há anos pela empresa que suporta este trabalho e, por isso, mas também pelo reconhecimento internacional, é garantida a qualidade tanto no produto em si como também em todo o seu suporte técnico e documental.

O segundo ponto aponta para o controlador de ecrãs, nomeadamente de TFT-LCD, que este µC possui internamente. O ecrã TFT-LCD deste projecto tem uma resolução de 800 x 600 a 18 bits por píxel, que já é uma resolução consideravelmente elevada para este tipo de aplicação, por isso foi escolhido o µC referido pois suporta resoluções até 2048 x 2048 de 1 a 24 bits por pixel. Aliás, à data do início deste trabalho, este era o controlador com suporte para a maior resolução inserido em µCs no mercado, e isto é muito importante, não

só porque permite boas capacidades de desenvolvimentos futuros, como também para que o componente seja capaz de fazer animações de imagens e não só apresentar uma só imagem ou várias com intervalos intercalares muito grandes.

A terceira razão, mas não a menos importante, é o facto de a ATMEL disponibilizar nos seus revendedores uma placa de desenvolvimento – a ATNGW100 – que tem, não só o µC em questão, como também outros periféricos que permitem desenvolver todo o *software* base deste projecto. Isto é fundamental pois um circuito impresso que contemple o AT32AP7000 é complexo, não só pelo formato deste e pelo número de pinos que é elevado, mas também pelas ligações às memórias que têm de ser minuciosamente bem colocadas e especificadas devido às altas velocidades. Assim, é possível desenvolver-se o respectivo *software* sem haver desconfianças de eventuais problemas de *hardware* pois este já está testado e com o bom funcionamento garantido.

4.3. KIT DE DESENVOLVIMENTO

Como já foi referido anteriormente, antes de se proceder à concepção do protótipo, foi adquirido um kit de desenvolvimento de baixo custo, da ATMEL, concebido para aplicações que recorram ao µC AT32AP7000 cuja designação é *Network Gateway* (ATNGW100) [3] e que se pode ver na Figura 36.

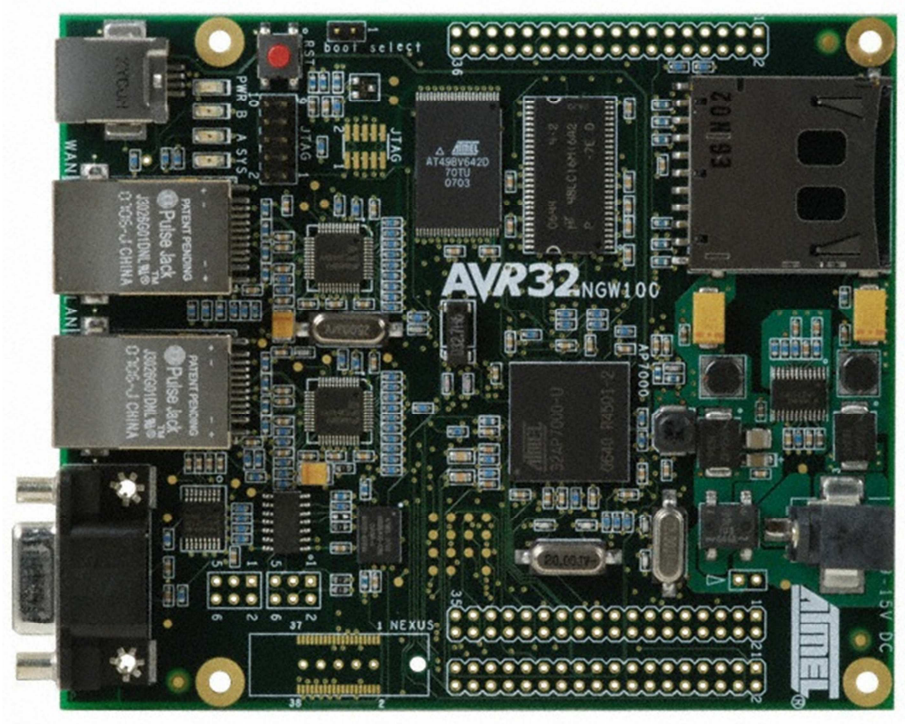

Figura 36 ATNGW100

Este kit foi desenhado especialmente para ser explorado por aplicações de rede a correr sobre Linux. Para isso estão acessíveis directamente todas as funcionalidades, nomeadamente ligações *ethernet*, imprescindíveis para tal propósito. Contudo, esta placa de desenvolvimento tem ao seu dispor conectores expansíveis que permitem que as restantes capacidades do µC possam também ficar acessíveis para o exterior. Ficando, assim, equipado com vários recursos para variadíssimos projectos. A sua utilização permitiu que a maior parte das funcionalidades apontadas para este projecto pudessem ser testadas só com a preocupação de desenvolver o programa respectivo.

A apresentação mais descritiva das suas características encontra-se na sub-subsecção 2.7.1 (capítulo estado do conhecimento na área).

4.4. PROTÓTIPO DE DESENVOLVIMENTO

O diagrama de blocos do protótipo de desenvolvimento está apresentado na Figura 37, sendo, logicamente, o ATNGW100 a base do protótipo.

O ATNGW100 tem conectores de expansão dos periféricos do µC que não são utilizados pelo kit, nomeadamente a interface do controlador LCD. Ora, para haver conexão entre este e o ecrã, foi necessário desenvolver um circuito impresso – designado por Interface com o ecrã (Figura 37) – para fazer esta ponte, especialmente para este protótipo de desenvolvimento.

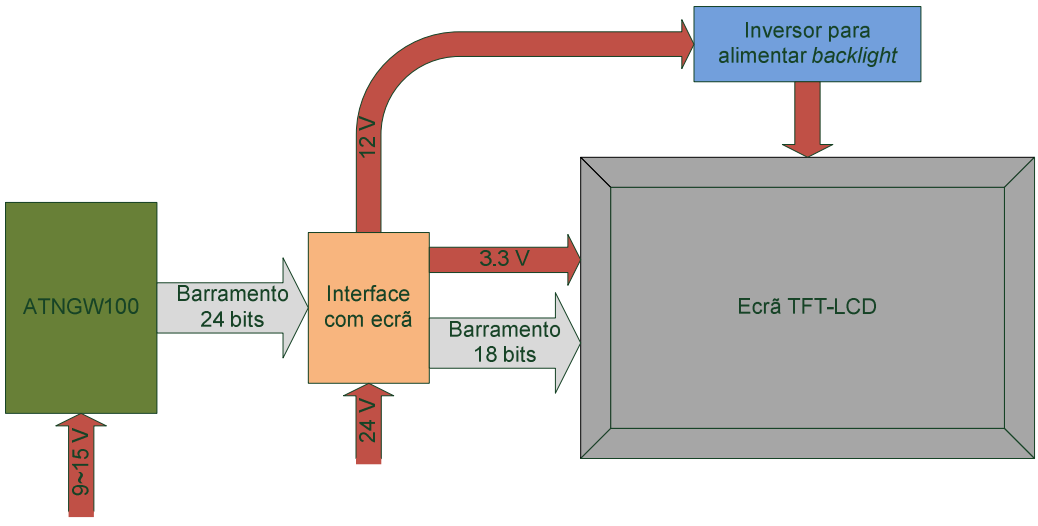

Figura 37 Diagrama de blocos do protótipo de desenvolvimento

Em suma, esta placa é constituída por duas fontes de alimentação distintas, dois conectores, um que faz a ligação com o ATNGW100 e o outro que se liga ao ecrã TFT-LCD, e dois bornes, um para receber a tensão de alimentação externa e o outro é de saída, o que alimenta o inversor do *backlight*.

Um dos três conectores de expansão do ATNGW100 é totalmente dedicado à interface do controlador LCD ficando, assim, a placa responsável por todos os terminais desse conector. A estes terminais chegam 24 linhas de dados, 4 sinais e linhas de alimentação – massa e 3.3 V –. Porém, estes 3.3 V não são aproveitados pela placa pelo simples facto da corrente fornecida pelo ATNGW100 não ser suficiente para alimentar o ecrã. Esta é a justificação para uma das fontes de alimentação da placa, pois só o módulo TFT-LCD pede uma corrente máxima de 450 mA, valor este que é muito próximo da corrente total (500 mA) que o kit pode fornecer.

Do outro lado da placa existe outro conector, fazendo este a ligação com o ecrã através de outro *flat cable* (tal como entre a placa e o ATNGW100). Nesta ligação só estão presentes 18 linhas de dados, isto porque o ecrã é de 18 bits. Ora, se o controlador de LCD do µC está configurado para 24 bits e se o módulo TFT-LCD é de 18 bits, como fazer a correspondência das linhas de dados? A solução é simples: só são ligadas, respectivamente, as 6 linhas mais significativas de cada componente RGB ou, por outras palavras, os 2 bits menos significativos de cada componente RGB da interface do controlador de LCD não vão ter interferência no ecrã. De resto, os sinais de sincronismo e de *clock* correspondem-se directamente e a alimentação é fornecida ao ecrã por uma fonte de alimentação implementada na própria placa de interface.

Esta placa está preparada para receber uma alimentação externa de 24 V DC, que é a tensão habitualmente usada na área de elevação para este propósito, chegando, então, aos terminais de entrada das duas fontes de alimentação que esta contém: uma cuja saída é de 3.3 V com capacidade para fornecer uma corrente de 1 A, e a outra com uma saída de 12 V e uma disponibilidade de 3 A.

Para a fonte de alimentação de 3.3 V foi escolhido o regulador de tensão da série LM2675 [32], mais precisamente o LM2675M-3.3. A escolha incidiu neste regulador por ser um conversor *step-down* (*buck*) de alta eficiência (maior que 90%), de tamanho reduzido e com garantias de poder fornecer uma corrente de 1 A com excelente regulação.

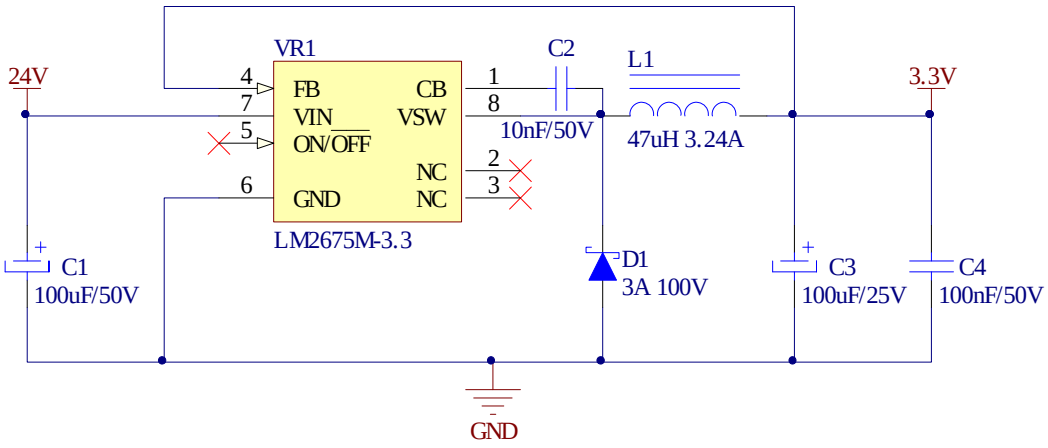

Figura 38 Fonte de alimentação de 3.3 V do protótipo de desenvolvimento

O seu circuito respectivo (Figura 38), recomendado pelo próprio fabricante [32], é simples pois requer apenas 5 componentes externos ao regulador, acrescentado de um condensador (C4) de filtragem à saída. Este regulador específico tem uma tensão de saída fixa (3.3 V), contudo, consoante o valor da tensão de entrada, a corrente de saída pode variar. O valor tabelado da bobina correspondente a 24 V e 1 A é de 47 µH [32]. Em relação aos outros componentes, inclusive o díodo de Schottky, foram definidos consoante as recomendações do fabricante, dos valores finais pretendidos e do stock disponível na empresa.

O inversor que alimenta o *backlight* do ecrã, como já foi referido anteriormente, requer uma tensão de entrada de 12 V e uma corrente, no máximo, de 1.25 A. Esta alimentação é fornecida pela outra fonte de alimentação desta placa de interface.

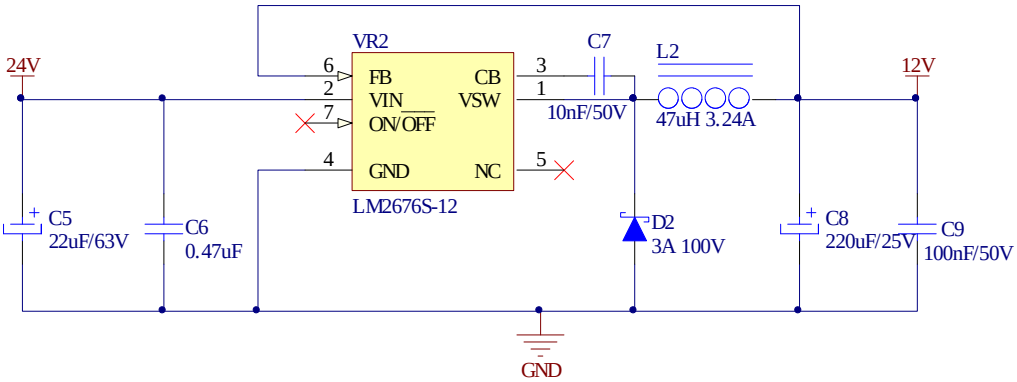

Figura 39 Fonte de alimentação de 12 V do protótipo de desenvolvimento

Esta segunda fonte (Figura 39) tem um regulador de tensão da série LM2676 [33], especificamente o LM2676S-12, que é da mesma gama do primeiro. Isto quer dizer que as características são semelhantes variando, naturalmente, a capacidade de fornecer uma maior corrente à saída, que passa a ser de 3 A, mas também da própria cápsula que é diferente, é do tipo TO-263 ao contrário do SO-8 pertencente ao LM2675M-3.3. Enquanto a escolha para um regulador de tensão de 12 V é óbvia, pois é a tensão requerida pelo inversor que alimenta o *backlight*, a escolha de um regulador que consiga debitar 3 A passa pelo simples facto de 3 A ser o valor imediatamente a seguir ao valor de 1 A entre os reguladores disponíveis e, como é preciso assegurar que a fonte de alimentação esteja preparada para a máxima corrente que lhe é pedida, esta tem de conseguir disponibilizar, pelo menos, 1.25 A que corresponde à tal corrente máxima requerida pelo inversor. Como se pode ver na Figura 39, em comparação com a Figura 38, o circuito exterior ao regulador é muito semelhante entre ambos. A bobina mais indicada a implementar neste circuito tem, por coincidência, o mesmo valor do caso anterior.

4.5. PROTÓTIPO

Após ser desenvolvido e testado o *software*, descrito no capítulo seguinte, que permitiu o funcionamento dos serviços mínimos deste projecto, nomeadamente dos controladores das memórias e do ecrã, da interface RS485, para receber os dados do exterior, entre outros periféricos, e sem nunca esquecer da própria integração conjunta para que fosse desenvolvida a aplicação final ou parte dela, foi estudado, desenhado e elaborado (este último por uma empresa exterior [34]) um circuito impresso que contém todos os periféricos imprescindíveis à prossecução dos objectivos propostos. Porém, sem funcionalidades desnecessárias a este projecto como é o caso da placa de desenvolvimento ATNGW100. O nome atribuído a este protótipo é Emanuel Pinto *Controller* (EPC).

4.5.1. MÓDULO ICNOVA AP7000 OEM [4]

Para este projecto é imprescindível o protótipo ter, para além do µC AT32AP7000, uma memória SDRAM e outra memória NOR-*Flash*, estando a primeira responsável por armazenar, durante o funcionamento da aplicação, todos os dados que estão a ser utilizados, nomeadamente as imagens a serem mostradas no ecrã, e a segunda memória encarregue de guardar o programa a ser executado durante o funcionamento. A presença destes componentes é muito importante, para não se dizer imprescindível, em muitos

projectos desenvolvidos com este µC, tanto a nível comercial como em simples trabalhos académicos ou particulares. Com esta motivação, existe à venda no mercado módulos, já com os componentes montados e prontos a serem utilizados com todas as garantias, que agrupam tanto o µC como as duas memórias acima citadas, não deixando de lado outros componentes inerentes ao bom funcionamento dos mesmos, tais como cristais, regulador de tensão de 1.8 V, etc. Assim, este módulo – o ICnova AP7000 OEM (Figura 40 e Figura 41) – foi adquirido pela In-Circuit GmbH, empresa que a vende, pois como fabrica em massa consegue disponibilizar a um preço bem mais simpático do que se esta fosse produzida no âmbito deste trabalho. Para além de ser fabricada em massa, a complexidade desta placa electrónica é elevada, não só pelo número de camadas pelo qual ela é composta como também pela minuciosidade das ligações entre o µC e as memórias e pelo distanciamento das mesmas.

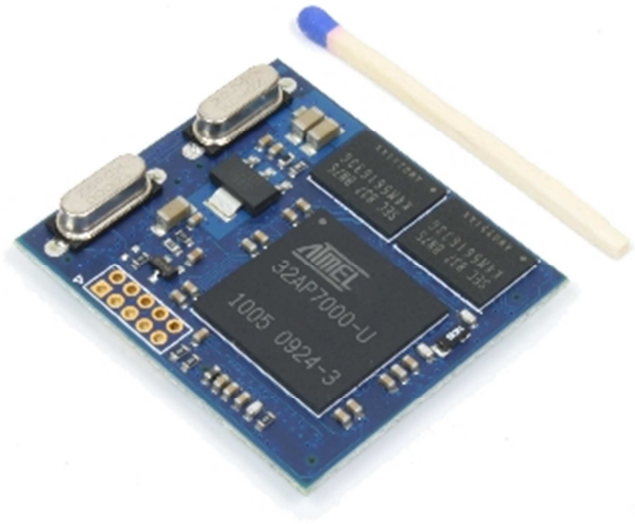

Figura 40 Módulo ICnova AP7000 OEM (parte de cima) [4]

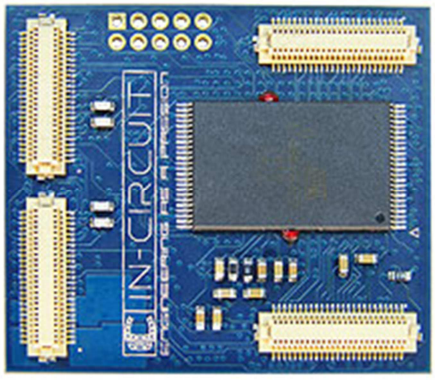

Figura 41 Módulo ICnova AP7000 OEM (parte de baixo) [4]

O módulo ICnova AP7000 OEM tem um tamanho ultra reduzido (4 cm por 3.5 cm) por ser muito compacto, como se pode verificar na Figura 40 por comparação com um fósforo, porém, é de elevado desempenho. Tem vários componentes já integrados, várias características, que passa-se a citar. Para além do inevitável µC AT32AP7000, O módulo tem 64 MB de memória SDRAM com um barramento de 32 bits, ao contrário do ATNGW100 que só possui um barramento de 16 bits possibilitando, teoricamente, um aumento para o dobro da velocidade de transferência de dados. De notar que na realidade não é só uma memória de 64 MB mas sim duas memórias de 32 MB cada. Outra memória presente é a NOR-*Flash*, de 8 MB, com um barramento de 16 bits. Como é necessário no circuito uma alimentação de 1.8 V, nomeadamente para o µC, o próprio módulo já tem um regulador de tensão para esta mesma tensão, precisando somente de uma alimentação externa de 3.3 V. O ICnova AP7000 OEM tem 4 conectores que têm duas responsabilidades: servir de encaixe no protótipo EPC, com os correspondentes conectores (conectores fêmea), e fazer as ligações entre o módulo e o protótipo. Integrada nestas ligações está a alimentação que o módulo requer do exterior, mas também várias interfaces de comunicação que o µC disponibiliza, tais como:

- LCD
- *Ethernet* de 10/100 Mb
- Dispositivo USB
- Audio *codec* (AC97)
- 4 UARTs
- SPI, MMC, SDCARD
- I²C (TWI)
- ISI
- SSC
- PS/2
- JTAG

Os 4 conectores correspondentes são do tipo Hirose fêmea, modelo DF12(3.0)-50DP-0.5V(86) [35]. São conectores de 50 pinos cada com distância (centro a centro) entre eles de 0.5 mm, cujos conectores têm de estar posicionados, entre eles, a uma distância fixa e definida, lógica e rigorosamente, pelas mesmas distâncias a que estão os conectores machos do módulo ICnova AP7000 OEM.

4.5.2. FONTES DE ALIMENTAÇÃO

Outro bloco imprescindível neste protótipo é o da alimentação. São necessárias três fontes de alimentação distintas mas com reguladores de tensão da mesma gama e respectivos circuitos eléctricos parecidos. A tensão que chega aos terminais de entrada de cada fonte é de 24 V, que tem de ser fornecida externamente através de um conector de encaixe para esse propósito e onde está ligado paralelamente aos seus terminais de entrada um díodo Supressor de Transientes de Tensão (TVS), ou transil, da marca ST, para proteger o sistema contra picos de tensão.

Uma das três fontes de alimentação existentes no protótipo EPC é a que fornece 3.3 V na sua saída e permite disponibilizar 1 A em corrente contínua (Figura 42).

O regulador de tensão utilizado é o mesmo que foi implementado no protótipo de desenvolvimento – o LM2675M-3.3 [32] – cuja tensão de saída é 3.3 V e tem capacidade para fornecer, no máximo, 1 A, mas pode variar consoante os componentes introduzidos no circuito externo que o acompanha, indicado pelo próprio fabricante.

De notar que, tanto neste circuito como no das outras duas fontes, é colocado um díodo directamente polarizado logo à entrada para protecção contra tensões negativas e também para que não haja circulação de correntes entre as fontes.

Já em relação aos componentes, por comparação com os do protótipo de desenvolvimento, não há grandes alterações, apenas de salientar que desta vez tanto o díodo de schottky como a bobina foram escolhidos mais de acordo com as necessidades do circuito, isto é, estão mais ajustados às características pretendidas e não com uma folga, ao nível dos seus limites de corrente admissível, tão grande.

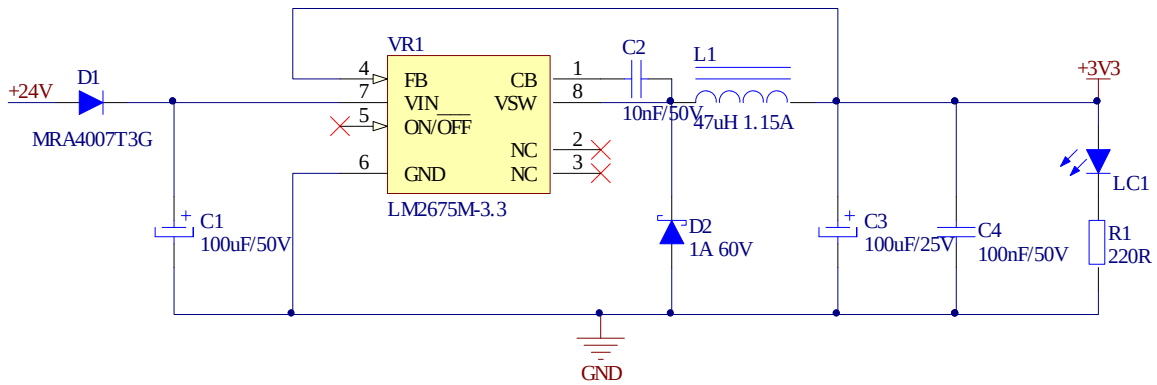

Figura 42 Fonte de alimentação de 3.3 V do protótipo EPC

Tem um acréscimo de um indicador de tensão presente na saída, em paralelo: um led, com a respectiva resistência limitadora de corrente para o led. A quantidade de brilho que se achou adequada corresponde a pouco menos de 10 mA de corrente que circula no led.

No protótipo EPC, ao contrário do protótipo de desenvolvimento, esta fonte de alimentação de 3.3 V não serve apenas para o ecrã TFT-LCD, mas também para outros blocos deste circuito eléctrico, tal como para a comunicação JTAG, para o *timekeeper*, para o leitor de cartões SD/MMC, para o Vinculum, que será explicado mais à frente, e, claro, para o próprio módulo ICnova AP7000 OEM.

Outra fonte de alimentação presente no protótipo é a de 5 V à saída, de 500 mA em corrente contínua também (Figura 43).

Mais uma vez, o regulador de tensão é da mesma gama dos outros dois, com a natural variação da tensão e corrente de saída face aos outros. É um regulador da série LM2674 [36], mais precisamente o LM2674-5.0 cuja saída de tensão, como o próprio nome indica, é de 5 V e pode proporcionar uma corrente até 500 mA.

O circuito alheio ao regulador é igual ao anterior, pois é o mesmo indicado pelo fabricante, e os componentes são praticamente os mesmos. Para além da resistência limitadora de corrente do led, que agora tem um valor maior pois o valor da tensão também é maior face ao anterior, a bobina também não é a mesma. Para uma tensão de 24 V e uma corrente máxima de 0.5 A, o valor tabelado da bobina é de 100 µH [36].

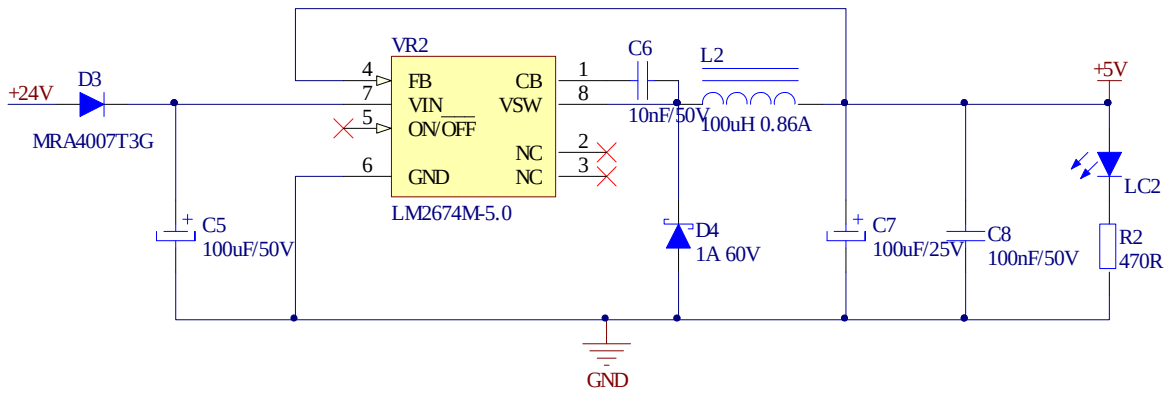

Figura 43 Fonte de alimentação de 5 V do protótipo EPC

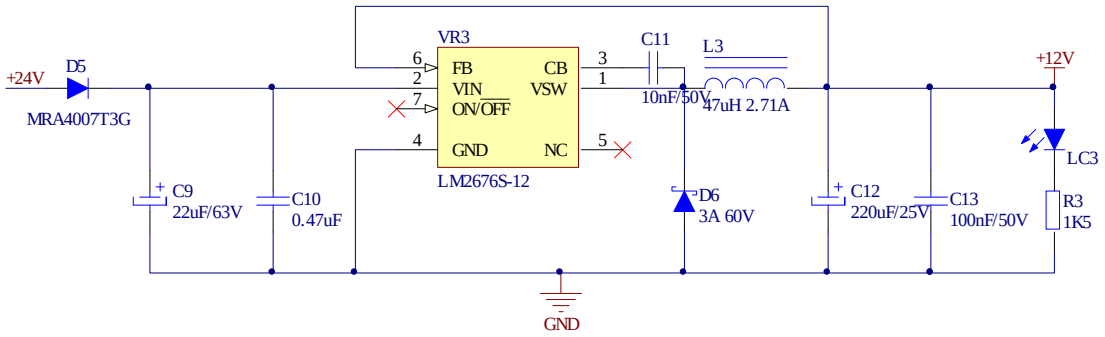

Figura 44 Fonte de alimentação de 12 V do protótipo EPC

A tensão fornecida por esta fonte tem como objectivo alimentar não só o bloco da comunicação RS485 como também alimentar uma *pen drive*, caso esta seja inserida no sistema, e dar carga à bateria do *timekeeper*, como vai ser descrito mais à frente.

Quanto à terceira e última fonte de alimentação, o reportório da fonte de 3.3 V reflecte-se aqui também, isto é, o circuito e os componentes são quase os mesmos em relação aos do protótipo de desenvolvimento, como se pode ver na Figura 44, com a diferença de ter sido adicionado o led de presença, e a respectiva resistência limitadora de corrente, o díodo à entrada para protecção contra tensões inversas e a alteração quer da bobina quer do díodo de schottky para uns com menor folga. No caso do primeiro, a corrente admissível é menor pois a carga (o inversor que alimenta o *backlight* do ecrã) desta fonte de alimentação não vai solicitar mais que 1.45 A. No caso do díodo, a tensão admissível é menor face ao implementado no protótipo de desenvolvimento pois não vai ter aos seus terminais tanta tensão. Sendo assim, esta fonte de alimentação coloca na sua saída, ou seja, tem capacidade para fornecer ao inversor uma tensão de 12 V e pouco mais de 2.5 A.

4.5.3. RELÓGIO

Como foi descrito nas funcionalidades deste projecto, este tem um relógio digital cujo horário e data têm de ser mostrados constantemente no ecrã. Na elaboração do protótipo EPC foram adicionados, entre outros, três blocos inerentes a esta funcionalidade: um bloco que gere o dito relógio, outro que garante a alimentação necessária para o carregamento da bateria do *timekeeper* e o terceiro é um conjunto de três botões que possibilitam o acerto do relógio.

O *timekeeper* escolhido (ou *Real-Time Clock* (RTC)) foi um da marca Dallas: o DS3234 [37]. A escolha recaiu neste componente específico por ter um custo reduzido, por ser um

RTC com uma interface SPI extremamente precisa, com um oscilador de cristal compensador de temperatura (TCXO) integrado e também com o seu próprio cristal incorporado.

O DS3234 integra uma tensão de referência precisa, com compensação de temperatura, e um circuito comparador para monitorizar o V_{CC}. Quando o V_{CC} baixa para além da tensão mínima (V_{PF}), o dispositivo activa a saída RST e também desabilita o acesso de leitura e escrita a este quando o V_{CC} cai para além do V_{PF} e do próprio V_{BAT}. O pino RST é monitorizado como um botão de entrada para gerar um reset ao µP. O dispositivo muda para a entrada da fonte de *backup* e mantém a sua função (gerar o relógio) intacta quando a sua alimentação principal é interrompida. A integração do oscilador de cristal aumenta a precisão a longo prazo do dispositivo, bem como reduz o número de componentes externos necessários para o seu bom funcionamento.

Este componente da Dallas também integra 256 bytes de memória SRAM resguardados pela bateria. Em caso de perda da alimentação principal, o conteúdo da memória é mantido pela fonte de alimentação ligado ao pino V_{BAT}. São vários os dados que o RTC mantém, entre eles estão os segundos, os minutos, as horas, os dias, a data, o mês e o ano. A data no final do mês é ajustada automaticamente para os meses com menos de 31 dias, incluindo as correcções para o ano bissexto. O relógio funciona quer no formato de 24 horas, quer no formato de 12 horas, tendo este a indicação de AM/PM. São também disponibilizados dois alarmes programáveis e uma saída de onda quadrada. Os endereços e os respectivos dados são transferidos pelo barramento SPI bidireccional.

Na Figura 45 está representado o circuito adjacente ao *timekeeper* DS3234, baseado na recomendação do próprio fabricante [37].

Para além da resistência de *pull-up* na linha do *Chip Select* (CS) e das linhas referentes ao barramento SPI, de notar que estão ligados no pino V_{BAT} uma bateria, mais propriamente um Super Condensador (SC), e um díodo. Este último é justificado pelo circuito de recarregamento do SC, explicado à frente.

Resumidamente, o super condensador, ou condensador de dupla camada eléctrica, é um condensador electroquímico que tem uma densidade de energia excepcionalmente alta quando comparado com os condensadores comuns, normalmente na ordem de milhares de vezes maior do que um condensador electrolítico de alta capacidade.

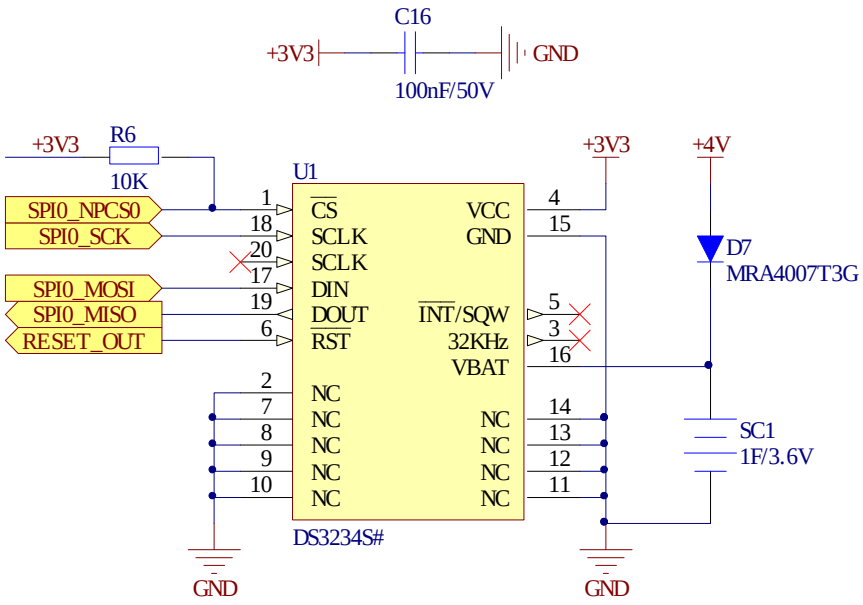

Figura 45 Esquema do *timekeeper* DS3234

O SC que foi considerado mais adequado para este projecto foi um da Panasonic da série RG, mais precisamente o EEC-RG0V105H [38], que é um SC com capacidade de 1 F a 3.6 V, para ser montado horizontalmente.

O outro bloco adjacente ao relógio é o circuito de recarregamento do SC, esquematizado na Figura 46. A máxima tensão admissível pelo SC é de 3.6 V e, como não há nenhuma alimentação no sistema com esta tensão, foi necessário introduzir um integrado, um regulador de tensão, que faça esta função.

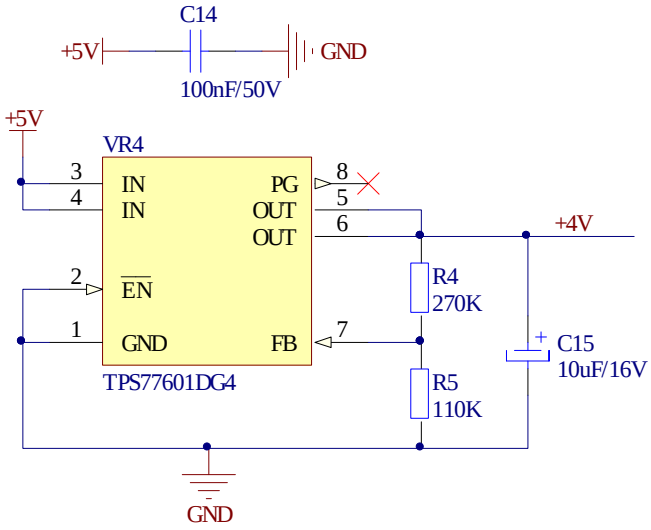

Figura 46 Esquema para recarregar o super condensador do *timekeeper* DS3234

Este integrado é um regulador de tensão com baixa queda de tensão entre a entrada e a saída e com uma rápida resposta de transitório capaz de debitar 500 mA. É da marca Texas Instruments, da série TPS776xx [39], em que o valor do "xx" é 01 pois é o correspondente a uma tensão de saída ajustável. Ora, como é imprescindível estar presente um díodo, o D_7 da Figura 45, para que não haja corrente de retorno para o TPS77601, a tensão aos terminais de saída deste tem de passar a ser de 4 V e não de 3.6 V, devido à queda de tensão que o díodo proporciona estando directamente polarizado. Seguindo a recomendação do circuito típico oferecido pelo fabricante do componente [39], e podendo ser visualizado na Figura 46, há que recorrer ao divisor de tensão para se determinar os valores das resistências R_4 e R_5. Pelo *datasheet* do componente [39], a fórmula é:

$$V_{OUT} = V_{ref} \times \left(1 + \frac{R_4}{R_5}\right)$$

Em que V_{ref} = 1.1834 V, sendo a tensão de referência interna.

Baixos valores das resistências não são aconselhados pelo *datasheet* pois obriga a um consumo de potência maior. Assim, a recomendação vai para um valor de 110 kΩ para a resistência R_5, para definir o divisor de corrente em cerca de 10 µA. Ficando assim a fórmula:

$$R_4 = \left(\frac{V_{OUT}}{V_{ref}} - 1\right) \times R_5$$

Sendo o V_{OUT} pretendido de 4 V e tendo em atenção em escolher o valor da resistência comercial mais próxima do resultado, o valor de R_4 encontrado é 270 kΩ.

A presença do condensador C_{15} deve-se ao facto dos reguladores de tensão de baixa queda de tensão requererem um condensador ligado entre a saída e a massa para estabilizar o ciclo de controlo interno. O valor de 10 µF é o valor mínimo recomendado para esta situação.

O terceiro e último bloco referente ao relógio é o conjunto dos botões destinados ao cliente para acertar o relógio mostrado no ecrã. São 3 botões que estão presentes e bem identificados da seguinte forma no circuito impresso: o "OK", que serve tanto para iniciar o acerto, como para avançar no digito a acertar e também para terminar o processo; os outros dois botões são setas, uma para cima e outra para baixo, e, como os próprios nomes indicam, incrementa e decrementa o contador do digito que estiver a piscar no momento, respectivamente.

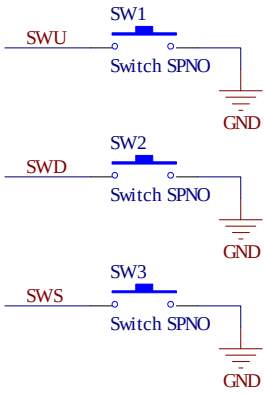

Figura 47 Esquema dos 3 botões de acerto do relógio

As ligações são muito simples, como se pode ver na Figura 47: todos os três botões têm um pino ligado à massa enquanto o outro é ligado directamente ao pino do µC.

Antecipando desde já um bocado da explicação que será feita no capítulo do *software*, o µC é que será responsável por activar as resistências de *pull-up*, para que em inactividade o estado lógico seja '1' e quando for pressionado qualquer botão o pino do µC passe a ficar ligado à massa e, assim, desencadear-se o respectivo processamento.

4.5.4. INTERFACE COM O MÓDULO TFT-LCD

Em relação às ligações com o ecrã TFT-LCD permanece tudo igual face às do protótipo de desenvolvimento, com duas pequenas nuances.

Os pinos do ecrã que definem a posição da imagem no ecrã agora podem ser controlados. Podem ser colocados em estado lógico '0' ou '1' através de um micro-interruptor (*dip switch*) de 2 unidades, estando cada uma ligada independentemente a cada pino do ecrã para este propósito: "U/D" e "R/L", como se pode ver na Figura 48.

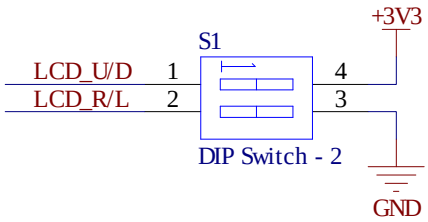

Figura 48 Esquema do micro-interruptor para controlar a posição da imagem do ecrã

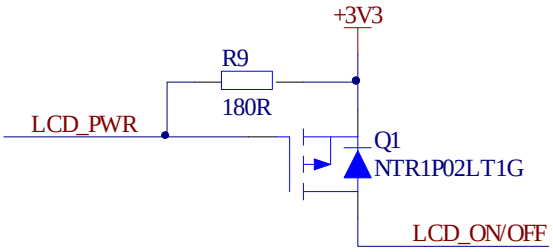

Figura 49 Esquema do transístor mosfet para controlar a alimentação do módulo TFT-LCD

A outra alteração é que agora, no protótipo EPC, a alimentação também é controlada por um transístor mosfet, ou seja, através do *software* é possível ligar ou desligar o ecrã. O transístor mosfet escolhido é um do tipo P, com um máximo de tensão admissível de -20 V e -1.3 A de corrente e, principalmente, com uma perda de tensão mínima, ao contrário dos transístores cuja queda de tensão é de 0.7 V: o NTR1P02LT1 [40]. Assim, como se pode ver na Figura 49, a tensão de 3.3 V é fornecida pela *source* do mosfet para que, consoante o estado da *gate*, que é controlada por um pino do µC, o mosfet interrompa ou deixe passar a tensão para o seu *drain* e, portanto, alimentar o módulo do ecrã (quer o próprio ecrã como o inversor que alimenta o *backlight*). Como se pode reparar, está ligada uma resistência de *pull-up* entre os 3.3 V e o pino do µC para forçar este a permanecer em estado lógico não indefinido. Tal como no protótipo de desenvolvimento, o EPC tem um conector para ser ligado ao inversor que alimenta o *backlight* do ecrã e, tal como o próprio ecrã, este também é controlado pelo transístor mosfet acima indicado e exactamente pela mesma saída. Um condensador (de 470 µF) logo nas imediações deste conector é colocado para que a tensão dos 12 V não vá para o inversor com algum ruído.

4.5.5. INTERFACE JTAG

O programador adquirido pela empresa patrocinadora deste trabalho para ser utilizado neste projecto é o JTAGICE-mkll, cuja interface é a JTAG.

A interface JTAG permite programar a *flash* externa ao µC e fazer a depuração do *software* que esteja a correr no µC AT32AP7000.

Para tal, é imprescindível que o protótipo EPC contenha um conector (*header*) de 10 vias do tipo da Figura 50. Naturalmente, os pinos do conector do EPC estão ligados aos pinos responsáveis pela interface JTAG do µC, integrado no módulo ICnova AP7000 OEM.

Figura 50 Conector (*header*) de 10 vias da interface JTAG

4.5.6. INTERFACE RS485

Para o sistema comunicar com o exterior, nomeadamente para receber as informações relativas ao piso em que o elevador se encontra no momento, em que sentido vai, entre outros dados, é utilizado o protocolo RS485, sendo o adoptado pela empresa patrocinadora nos seus projectos. E este não foge à regra.

Como o µC funciona com níveis lógicos TTL, o protótipo EPC recorre a um *driver* para fazer a ponte entre os níveis lógicos TTL do µC e o par diferencial da interface RS485. O integrado implementado já é conhecido dos projectos elaborados na empresa e, portanto, a sua aplicabilidade nesta área já foi testado e comprovada. Este é da marca Linear Technology, de seu nome LT1785 [41].

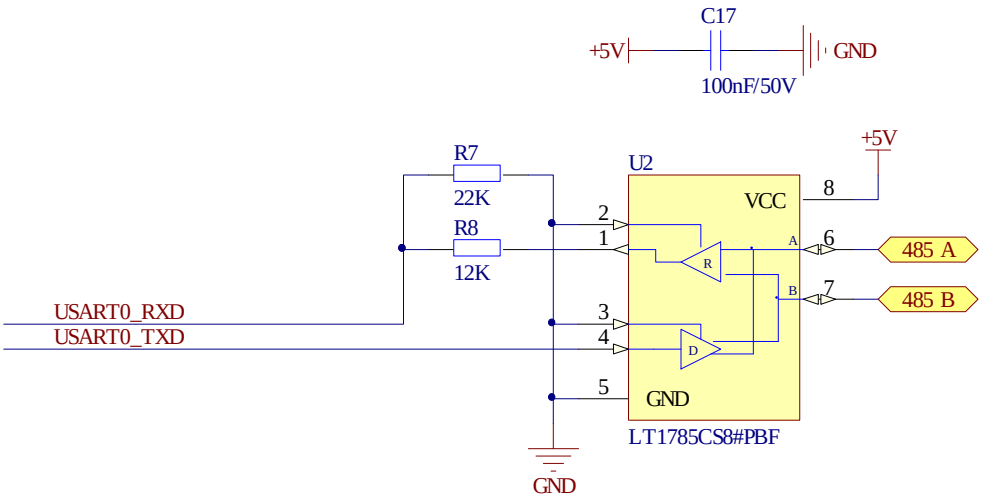

Figura 51 Esquema do *driver* LT1785 da interface RS485

Como principais características, é um *driver* de baixo custo, específico para operar com cabos de baixa impedância, mais baratos portanto, fazendo as transferências de dados em *half-duplex* ou *full-duplex*, com protecção contra sobretensões até ±60 V e com controlo *slew rates* para controlar as emissões *Electromagnetic Interference* (EMI).

Na Figura 51 está esquematizado o circuito envolvente do LT1785. “USART0_RXD” e “USART0_TXD” são os sinais que o µC recebe e transmite, respectivamente, de e para o *driver*. Como o LT1785 interpreta correctamente sinais de 3.3 V, como é o caso do sinal de transmissão do µC, a ligação entre estes é directa. Por outro lado, o estado lógico ‘1’ do pino de transmissão do *driver* é 5 V e, como não é aconselhável que o µC receber sinais com tensões superiores a 3.3 V neste pino, é introduzido um divisor de tensão para que a tensão de 5 V à saída do LT1785 seja reduzida para, aproximadamente, 3.3 V:

$$V_{RXD} = 5 \times \frac{22k}{22k + 12k} = 3.235\ V$$

Os pinos 2 e 3 do LT1785 são pinos de controlo da funcionalidade de recepção e transmissão, respectivamente. Para que as condições desejadas deste trabalho sejam consumadas, ou seja, a recepção seja feita em plenas condições e a transmissão indefinida pois não vai haver qualquer tipo de envio de dados para o exterior do sistema, e pela análise do *datasheet* do componente [41], os pinos 2 e 3 ficam permanentemente com estado lógico ‘0’, isto é, ligados à massa.

Do outro lado, é ligado o par diferencial directamente ao conector principal partilhando, assim, o mesmo conector com a alimentação.

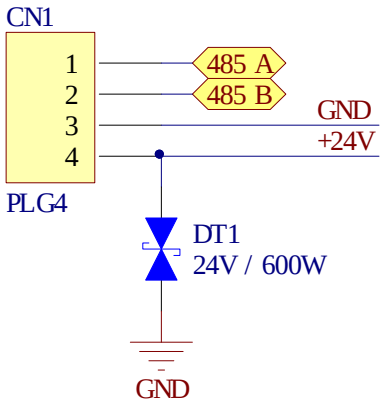

Figura 52 Esquema do conector principal do protótipo EPC

Estando agora completo o conector de encaixe principal, há uma nota a acrescentar. A ordem das ligações no conector não foi seleccionada aleatoriamente. No terminal de uma das extremidades (no terminal 4), como se pode ver na Figura 52, estão ligados os 24 V e só no terminal 3 é que se encontra a massa. Esta ordem é importante para que, caso o conector que encaixa no conector do EPC seja introduzido de forma torta, seja primeiramente a massa a ser ligada e só depois os 24 V. Esta ordem, aliás, já é implementada noutros projectos da empresa e, assim, segue-se o mesmo padrão.

4.5.7. INTERFACE PARA CARTÃO SD/MMC

Outro bloco a descrever neste protótipo EPC é o da ranhura para o cartão SD/MMC com o seu respectivo circuito.

O suporte para o cartão optado foi um da marca Kyocera cuja referência do fabricante é 14 5738 009 784 859+ [42], ilustrado na Figura 53.

Este tipo de conectores tem 12 pinos, 6 dos quais são de dados, outros 3 pertencem à alimentação e os restantes 3 servem de controlo. Começando por estes últimos, os 3 pinos podem ter duas funções: indicar se o cartão foi ou não inserido na ranhura e se este está protegido contra escrita ou não. A alimentação do conector é conhecida do EPC, sendo até a mais utilizada: 3.3 V. Em relação aos primeiros 6 pinos, 4 dos quais são sinais por onde são transferidos os dados – D3…D0 – e os outros 2 são o *clock* e o *command*, essenciais para o protocolo MCI do µC.

Através da Figura 54 pode-se constatar todas as ligações efectuadas, que interagem com os 12 pinos supracitados.

Figura 53 Suporte para cartão SD/MMC

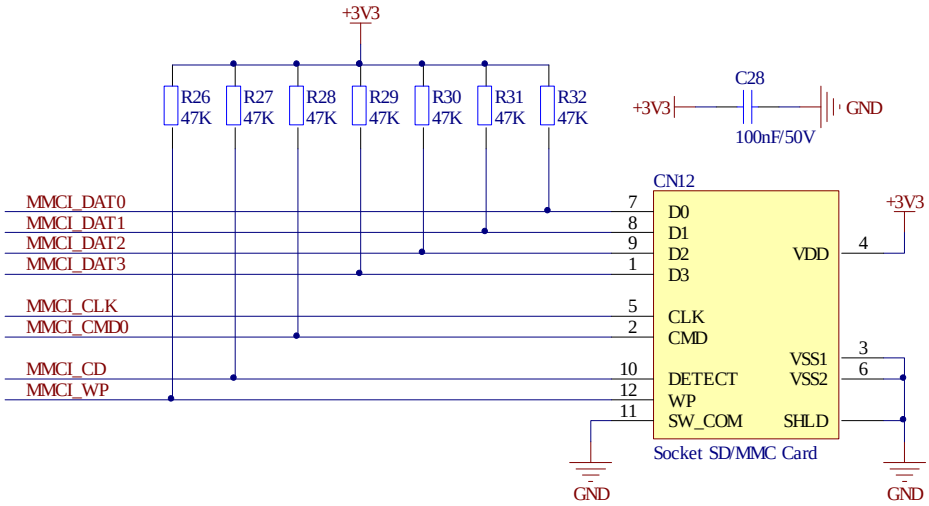

Figura 54 Esquema do suporte para cartão SD/MMC

Estas ligações correspondem à interface que foi optada – MCI – face à mais convencional – SPI – pelo simples facto da primeira ser mais rápida do que a segunda (teoricamente, a máxima velocidade de transferência é de 36 Mb contra 33 Mb) e pelos *drivers* do *software* já terem sido desenvolvidos e testados no protótipo de desenvolvimento. As resistências R26…R32 são resistências de *pull-up* para que nessas linhas fique garantido o estado lógico '1'.

4.5.8. INTERFACE USB

O último bloco referente ao *hardware* do protótipo EPC é o bloco que contempla todo o circuito envolvente do Vinculum. O que é o Vinculum? O Vinculum (ou simplesmente VNC1L) é um integrado único com dois controladores USB, da marca FTDI, com as seguintes características [30]:

- Duas portas independentes USB 2.0 *host* em baixa velocidade ou em velocidade máxima;
- As portas podem ser configuradas individualmente como *host* ou como *slave*;
- Pode ser configurado uma monitorização por comandos externa através da interface UART, FIFO ou SPI;
- O próprio VNC1L já interpreta internamente o protocolo USB;
- Resistências *pull-up* e *pull-down* já estão integradas no componente;
- O micro-processador do micro-controlador Vinculum é um proprietário da FTDI, de 8/32 bits, que recorre à tecnologia CISC;

- Contém 64 kB de memória *Flash* embebida, reconfigurável, para guardar o *firmware*, bem como 4 kB de RAM de dados;
- A actualização do *firmware* pode ser feita através da UART ou por USB, colocando o ficheiro do *firmware* numa *pen drive*;
- O VNC1L integra também um co-processador numérico para melhorar a velocidade aritmética de 32 bits;
- Dois controladores DMA, um para cada interface USB, oferecem uma aceleração no *hardware* na transferência de dados do USB para o barramento exterior;
- A FTDI disponibiliza *firmwares* para os vários tipos de configurações possíveis dos controladores USB, totalmente operacionais, sem necessidade de ser posteriormente alterado;
- É possível configurar quatro barramentos de dados e pinos de controlo oferecendo até 28 pinos de uso geral;
- O *firmware* utilizado permite que se possa ler e escrever de e para uma USB *pen drive* estando formatada num sistema de ficheiros FAT;
- O componente não vem com qualquer *firmware* de fábrica, por isso a programação do mesmo tem de ser feita via UART;
- A sua alimentação é de 3.3 V mas suportando nos pinos de entrada tensões de 5 V;
- O seu consumo é reduzido: 25 mA em funcionamento e 2 mA em *standby*.

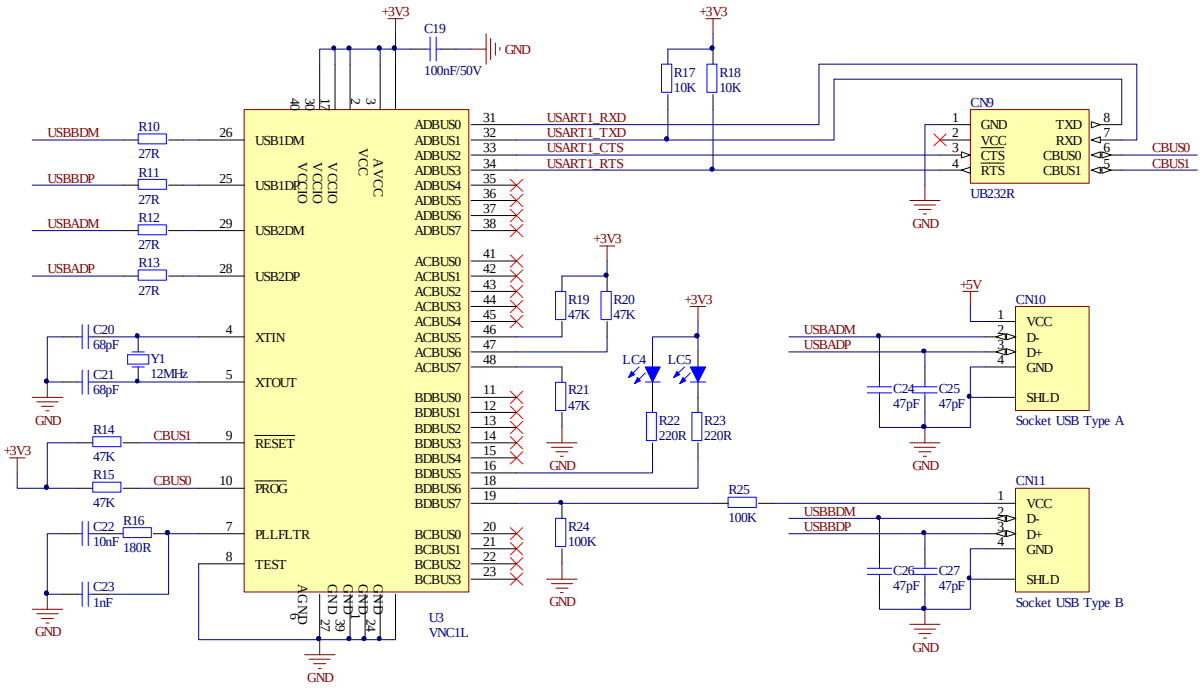

Figura 55 Esquema do bloco do Vinculum (VNC1L)

O esquema da Figura 55 é o circuito de exemplo aplicativo referido no próprio *datasheet* do VNC1L [43]. No entanto, há algumas notas a acrescentar. A interface escolhida para fazer a transmissão de dados entre o VNC1L e o µC AT32AP7000 é a UART, podendo ter velocidades de transmissão até 1 M *baud rate*. De reparar que estão presentes dois conectores, um é do tipo USB A (CN10), com a função de se poder ligar uma USB *pen drive*, e o outro é do tipo USB B (CN11), onde se pode ligar um computador comum, comportando-se o Vinculum como um dispositivo, tal como uma impressora, por exemplo. Ao se ligar um dispositivo no CN10, como uma *pen drive*, esta necessita de alimentação externa, justificando a presença de uma tensão de 5 V no pino 1 do conector. Os leds LC_4 e LC_5 indicam a presença de uma *pen* ou de uma ligação feita a um computador, respectivamente, bem como se estes estão a transferir dados ou não, ficando os leds a piscar, no primeiro caso. Por último, como o Vinculum precisa de ser programado na primeira utilização e só através da UART, o método utilizado foi recorrer a um módulo, também da FTDI, que converte os dados da ligação USB feita ao computador em dados via UART ou vice-versa, designado por UB232R [31], mostrado na Figura 56.

O UB232R é o módulo USB – série de desenvolvimento mais pequeno dos produtos da FTDI (e também o mais barato para este propósito). Consegue ser pequeno porque, para além de ser o mais simples possível, inclui o conector USB do tipo mini B. Este é baseado no integrado da FTDI FT232RQ USB para UART [44] que interpreta todas as comunicações de sinais e protocolos USB. Assim, com o UB232R obtém-se o suporte para o *hardware handshaking* RTS/CTS bem como taxas de transferência desde 300 baud até 3 Mbaud (RS422, RS485, RS232 e em níveis TTL). Para acrescentar, 2 pinos do FT232R, os pinos configuráveis CBUS, estão disponíveis no conector do UB232R, permitindo que este forneça sinais de relógio para o exterior ou que possam ser usados para indicar tráfego de sinais através de leds.

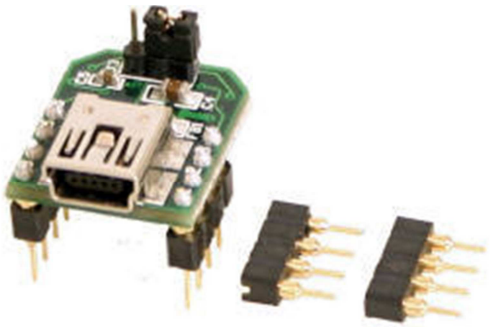

Figura 56 UB232R

A alimentação é fornecida pelo cabo USB, isto é, pelo próprio computador. Pela Figura 56 pode-se reparar que o módulo tem pinos comummente usados em projectos, sendo o seu conector fêmea de fácil e rápida implementação em PCBs. Por isto e por ser barato é que o UB232R foi escolhido para desempenhar o papel de programar pela primeira vez o VNC1L, daí a presença do conector CN9 na Figura 55.

4.6. CAMADA DO SISTEMA

Para que a camada da aplicação, descrita no capítulo seguinte (*software*), tenha sustentabilidade, tem de estar presente uma camada do sistema, isto é, um conjunto de *Applications Programming Interface* (APIs) dos controladores e dos periféricos do µC que são utilizados no projecto.

Esta subsecção é responsável por explicar as directrizes das APIs que foram implementadas neste projecto.

Para acompanhar esta explicação pode-se observar a arquitectura do µC utilizado na Figura 57.

As configurações base do protótipo EPC, análogas aos do protótipo de desenvolvimento (ATNGW100), são as seguintes.

As frequências do *clock* dos três barramentos distintos: *High Speed Bus* (HSB), que é a principal, fazendo a ponte entre o núcleo do CPU e todos os controladores e periféricos disponíveis pela arquitectura do µC, o barramento PBA, onde estão ligadas as USARTs, os SPIs, os controladores das portas de I/O, entre outros, e o PBB, onde está ligado o *Multimedia Card Interface* (MCI), a gestão da energia, os temporizadores/contadores, o *Real Timer Counter* (RTC), o controlador de interrupções, entre outros.

Embora o CPU possa funcionar a uma frequência superior a 150 MHz, é precisamente esta para a qual o sistema foi configurado, isto porque acima desta frequência o µC aquece consideravelmente e, como este produto tem como objectivo ter um funcionamento contínuo, por ser um produto comercializado, não se torna viável essa situação pois arriscar-se-ia a durabilidade do produto.

Esta frequência do CPU fica definida após uma combinação de duas configurações: a escolha de um dos dois osciladores externos, em que um é de 20 MHz e o outro é de 12

MHz, e a configuração de um dos dois PLLs, através de multiplicações e/ou divisões. Para este caso em questão, o oscilador (cristal) escolhido foi o de 20 MHz com o PLL0 a multiplicar essa frequência por 15 e a dividi-la por 2.

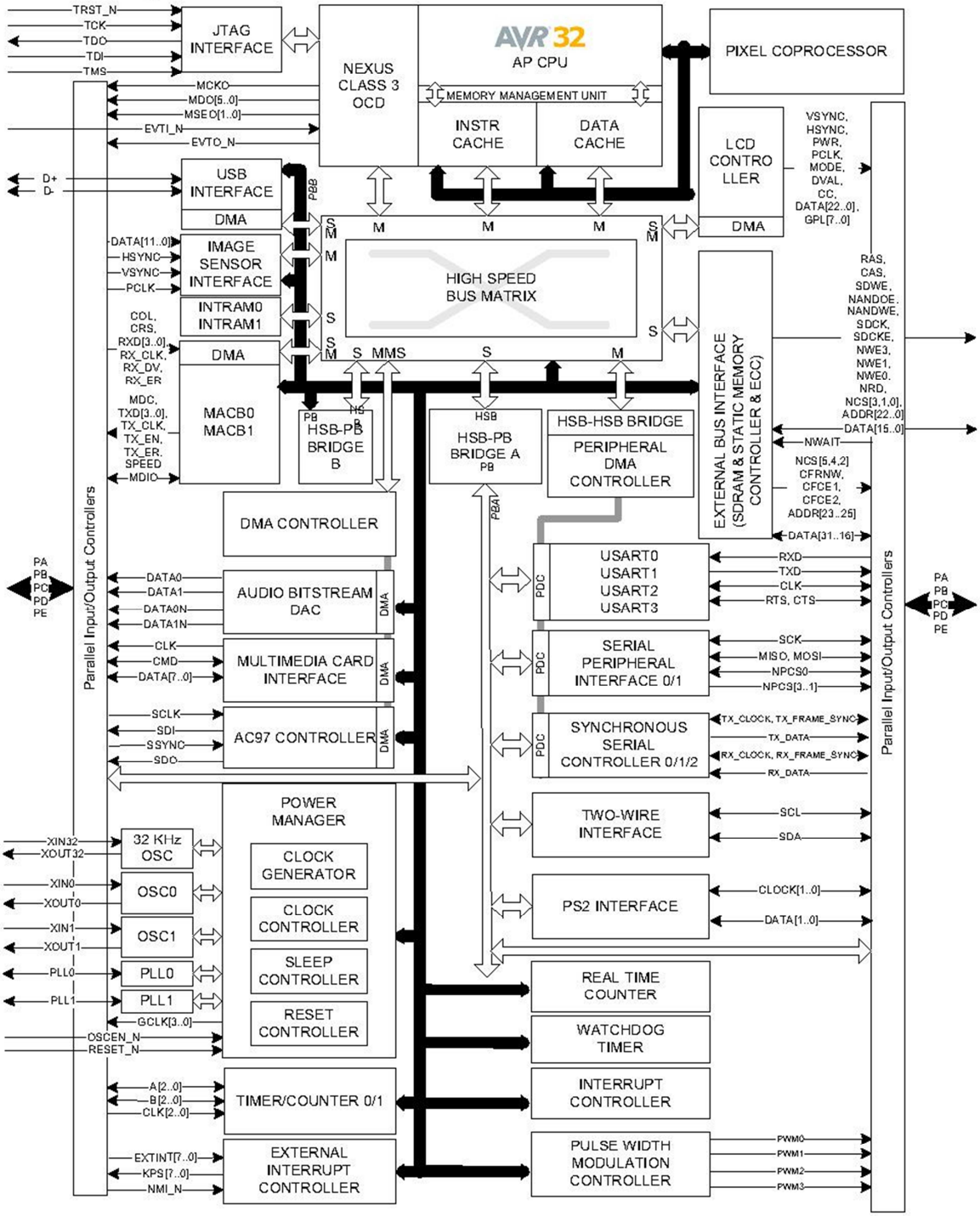

Figura 57 Diagrama de blocos do µC AT32AP7000 [1]

A relação das frequências entre os três barramentos depende da frequência definida para o CPU e é aconselhada pelo próprio *datasheet* do µC [1] da seguinte forma: as frequências do HSB e do PBB divididas por 2 e a do PBA dividida por 4. Assim, obtém-se uma frequência no HSB e no PBB de 75 MHz e uma frequência no PBA de 37.5 MHz.

Outra configuração importante é o do *Static Memory Controller* (SMC) pois é com este que a memória *flash* é controlada com velocidades elevadas de *clock*. Caso o SMC não seja configurado, a *flash* é acedida em velocidades baixas.

Para além do SMC, e para que a memória externa SDRAM se interligue com o sistema, o seu controlador também teve de ser configurado.

Na configuração do controlador da SDRAM há um destaque a fazer. Este suporta memórias SDRAM com barramento de 16 ou 32 bits. Na primeira situação, que é o caso do kit de desenvolvimento, é só definir o barramento para 16 bits e pronto. Por outro lado, já no caso do EPC (na realidade é no módulo ICnova AP7000 OEM), o barramento das duas memórias SDRAM de 32 MB cada é de 32 bits. Isto provoca que, para além da definição de 32 bits nas configurações, seja necessário activar os 16 pinos mais significativos do barramento EBI para esta função, pois estes pinos são multiplexados.

De salientar que quer o SMC quer o controlador SDRAM pertencem ao *External Bus Interface* (EBI) que, por sua vez, está ligado directamente ao HSB.

Como objectivo principal nas configurações inicias está, naturalmente, o controlador de LCD, que também se encontra ligado directamente ao HSB.

Para relembrar, o ecrã utilizado neste projecto tem uma interface com 18 bits de dados. A interligação com ecrãs TFT-LCD deste controlador do µC mais apropriada é de 24 bits. Como há bits em excesso, a solução passa por desprezar os 2 bits menos significativos de cada componente de cor do píxel. Assim, na configuração dos pinos desta interface só são activos os pinos [23…18], [15…10] e [7…2].

Após as naturais configurações do HSB para este controlador, entre elas activá-lo, são definidos os parâmetros para este ecrã em concreto. A destacar, e pela análise do *datasheet* do ecrã [2], para além da já conhecida resolução horizontal de 800 píxeis e vertical de 600 píxeis, a taxa de refrescamento definida é de 60 Hz e a frequência do *clock* do LCD recomendado (o típico) é de 40 MHz. Para o controlador LCD poder gerar este *pixel clock*,

no HSB foi definido que esta frequência seria fornecida pelo PLL1, estando este já configurado pelos mesmos moldes do PLL0 acima explicado: o oscilador optado agora foi o de 12 MHz, a multiplicação no PLL1 de valor 10 e com divisão por 3 – resultado 40 MHz.

Para que se possa testar todos os controladores acima citados, principalmente o último, e também porque é fundamental para o protótipo EPC, foi configurado o MCI para que os ficheiros, nomeadamente as imagens, situados no cartão SD/MMC sejam lidos, carregados para a memória SDRAM e possivelmente enviados para o ecrã TFT-LCD.

O MCI é um protocolo proprietário da ATMEL que, face ao seu opositor mais habitual SPI, pode atingir velocidades de transmissão mais elevadas. A taxa à qual o MCI opera pode ir até ao *master clock* dividido por 2. Simplificando, o *master clock* é a frequência do PBB.

Por esta razão e pelo facto de no kit de desenvolvimento ser esta a interface implementada, decidiu-se implementar também este protocolo no protótipo EPC e aproveitar o *software* já testado.

O acesso ao cartão SD/MMC não se resume só ao MCI mas sim a três camadas. A primeira camada destina-se à API do MCI e a segunda faz a ponte da primeira com a terceira. Esta última camada é uma estrutura *FAT File System* (FatFs) para sistemas embebidos e é totalmente independente de qualquer arquitectura de *hardware*, sendo possível, assim, ser inserida em qualquer µC. Esta é a justificação da segunda camada: interligar as funções disponibilizadas pela API do MCI com as funções do FatFs que requerem com o nível mais abaixo. Este módulo tem imensa aplicabilidade em vários projectos por reunir vários factores interessantes, tais como: tem um bom suporte pois é frequentemente actualizado, ocupa relativamente pouco código, é de fácil transporte pois a sua plataforma é independente, é compatível com o FatFs do Windows, as funções disponíveis para a camada da aplicação são amplamente conhecidas (f_mount, f_open, f_close, f_read, f_write, etc.), entre outras [45].

Com estas APIs já activas, bem definidas e testadas com sucesso, foi possível garantir o bom funcionamento de uma estrutura base e, assim, transportá-la para o protótipo EPC. Posto isto, foram acrescentadas mais algumas APIs que são essenciais para os periféricos que o EPC veio acrescentar.

O sistema precisa de receber dados exteriores, como o piso actual, o sentido do movimento do elevador, etc., para processar e mostrar no ecrã todas essas informações. O protocolo físico RS485 é largamente utilizado na indústria, nomeadamente na área de elevação, devido não só à sua robustez a ruídos e interferências como ao bom alcance que tem, já para não falar no número reduzido de condutores que necessita (2). Aliado ao facto de ser a interface adoptada pela empresa, esta também é implementada neste projecto.

Para tal, é inicializada a *Universal Synchronous Asynchronous Receiver Transmitter* 0 (USART0) a 9600 bps só em modo de recepção, isto porque não é objectivo enviar qualquer dado em sentido inverso. De salientar que este µC já integra o modo RS485, e sendo assim, este é o modo utilizado, com nenhum bit de paridade, 1 stop bit e 9 bits de dados, justificados na camada da aplicação (capítulo seguinte).

Foi justificada anteriormente a presença do integrado Vinculum (VNC1L) [43], sendo apontado como o responsável por ler e interpretar os dados de uma eventual USB *pen drive* introduzida e enviar esses dados processados para o µC via USART. Posto isto, também foi inicializada a USART1 para este propósito mas com uma variação: o VNC1L requer uma comunicação com *hardware handshaking*, isto é, para além de serem utilizados os sinais RXD e TXD, também têm de ser introduzidos os RTS e CTS. Por sinal, o µC também possui esta interface e, portanto, a USART1 foi configurada no modo *hardware handshaking*, sem bit de paridade, com 1 stop bit e 8 bits de dados, não esquecendo que agora a comunicação é bidireccional pois tem de haver, para além da recepção dos dados, troca de comandos. A taxa de transmissão máxima conseguida, sem qualquer perda de dados, foi de 460800 bps.

Só que neste modo há uma pequena complicação. Pela leitura do *datasheet* do µC [1], depara-se que este requer um canal de *Peripheral DMA Controller* (PDC) para a recepção. A utilização do PDC permite que sejam transferidos dados entre os periféricos, como a USART1, e as memórias sem a intervenção do processador. Sendo este um requisito obrigatório, foi então introduzida a API para o PDC. Cada periférico, e a USART1 não é excepção, tem um espaço em memória reservado para o canal PDC, portanto, foi integrado um canal PDC respectivo à recepção para este periférico em concreto.

A interface de comunicação do *timekeeper* (gerador do relógio) DS3234 [37] é o *Serial Peripheral Interface* (SPI). Isto significa que foi mais uma API a acrescentar ao projecto.

A taxa de transmissão é imposta pelo *timekeeper* cujo valor máximo é 4 MHz. E a comunicação foi configurada para 8 bits de dados por cada pacote SPI.

Ainda em relação ao bloco que envolve o relógio, foi necessário configurar os 3 pinos respectivos a cada botão de acerto. Todos eles foram definidos como entrada com *pull-up* e com filtro de *glitch*, isto é, qualquer impulso com menos de metade de um ciclo de relógio é rejeitado.

5. SOFTWARE

Este capítulo pretende descrever o código do programa – *software* – que permite desencadear o funcionamento de todo o *hardware* descrito no capítulo anterior.

O programa de desenvolvimento do *software*, ou o *Integrated Development Environment* (IDE), escolhido para este trabalho foi um proprietário da própria ATMEL – o AVR32 Studio [46]. É um programa de desenvolvimento de aplicações para µCs AVR de 32 bits, tal como o AT32AP7000, com várias funcionalidades. Tem um editor de C/C++, onde se pode criar e gerir projectos de ficheiros. Suporta o programador/depurador JTAGICE mkll [47], que foi adquirido pela empresa para ser utilizado neste trabalho, para executar a respectiva transferência do código para a *flash* do µC, mas também para dar a possibilidade do programador depurar o seu código do programa, onde pode visualizar os valores dos registos, a memória, os estados dos pinos de I/O, correr passo-a-passo, etc. E entre outras funcionalidades.

Foi optado a programação em modo *standalone* (aplicação sem recurso a qualquer sistema operativo). Neste modo, o AVR32 Studio tem de ser acompanhado pelo AVR32 GNU Toolchain [48]. Este conjunto de ferramentas tem utilitários que permitem compilar e depurar código *standalone*. Contém também as bibliotecas de C para o desenvolvimento das aplicações.

Como foi referido anteriormente, no início deste trabalho foi adquirido um kit de desenvolvimento (o ATNGW100). A aquisição deste kit foi feita principalmente para se poder testar as APIs dos controladores e dos periféricos base deste projecto. Estando aprovadas, estas foram transferidas para o protótipo EPC para depois serem testadas as restantes APIs. As directrizes para todas estas APIs estão referidas na subsecção 4.6 (capítulo *hardware*) – camada do sistema.

Testar é o termo mais correcto pois nesta camada não foi feito um desenvolvimento por inteiro. A ATMEL disponibiliza *online* [49], e até no próprio AVR32 Studio, através do *Framework*, várias APIs para os controladores e periféricos disponíveis pelos µCs AVR32. No entanto, nem todas as APIs foram criadas especificamente para o µC deste projecto, e também não foram desenvolvidas para este kit de desenvolvimento. Contudo, como as arquitecturas entre os µCs da gama AVR32 são semelhantes e como a base dos componentes dos kits de desenvolvimento são parecidos, é possível fazer a transferência, com as devidas alterações, dessas APIs para este caso. Para tal, recorreu-se a alguns blocos de programa disponibilizados por outro programador que já se encontram devidamente manipulados [50].

5.1. DESCRIÇÃO DO PROGRAMA DA CAMADA DA APLICAÇÃO

Enquanto no código do programa da camada do sistema a responsabilidade é partilhada maioritariamente pelos *drivers* da própria ATMEL, o da camada da aplicação foi inteiramente desenvolvido na sequência deste trabalho.

No seu aproveitamento máximo, a camada da aplicação é, efectivamente, orientada para a área dos elevadores e especialmente para os objectivos propostos para este projecto. Contudo, esta camada foi desenvolvida para ser flexível. Cada funcionalidade da aplicação, como as setas, o piso, etc., pode ser activada ou desactivada de modo que o *layout* fique com as áreas informativas pretendidas. Como exemplo, é possível que esta aplicação se torne numa simples moldura digital. Durante a explicação do programa poder-se-á verificar que, na altura do µC ser programado, estas alterações são fácil e rapidamente executáveis.

Antes de se proceder à explicação de cada bloco que compõe todo o programa da camada da aplicação, é importante ter-se uma visão global da sequência de funções implementadas que garantem o funcionamento do *software* pretendido. Na Figura 58 está esquematizada essa sequência, que corresponde à rotina “mãe” (*main*) do programa.

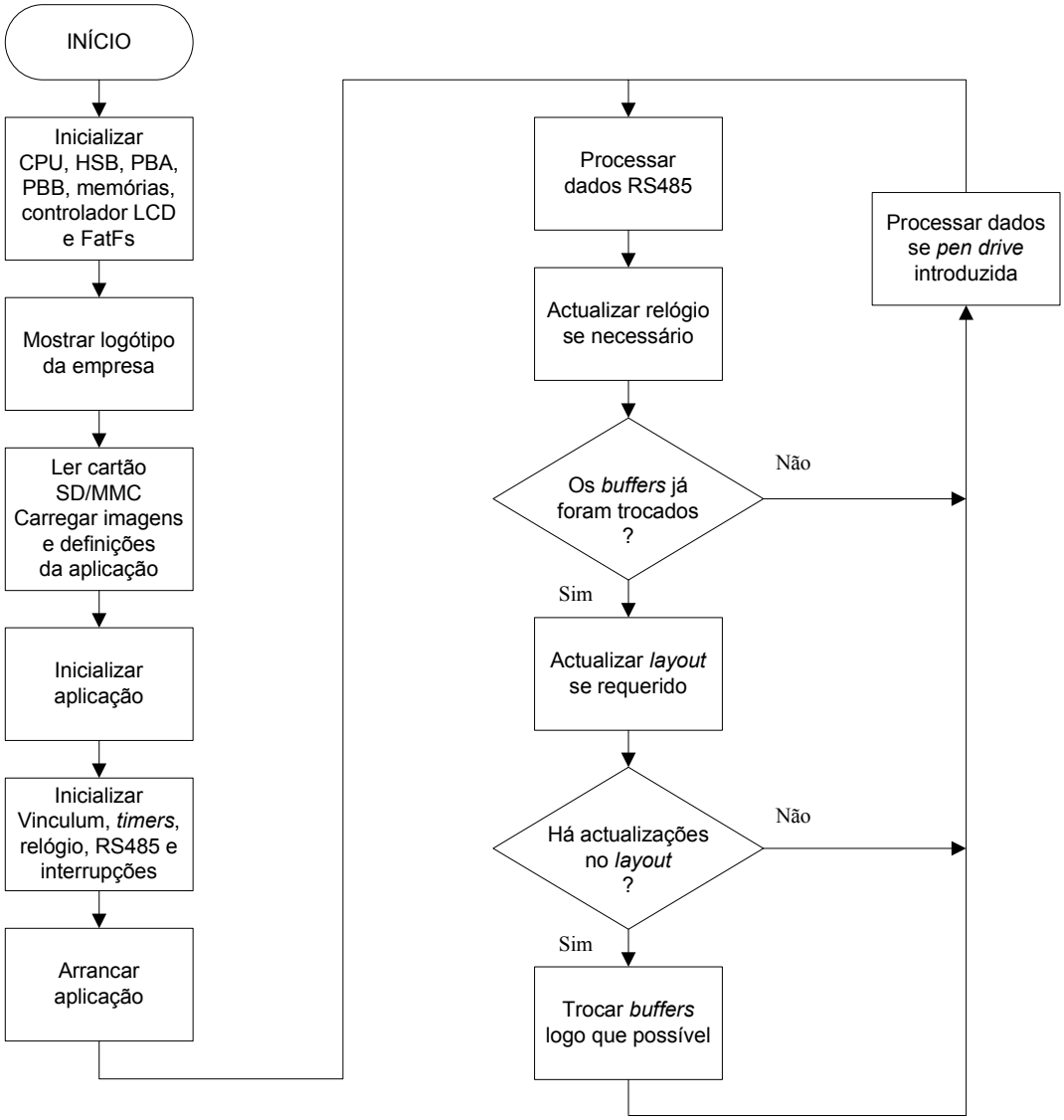

Figura 58 Fluxograma principal do *software*

As primeiras inicializações que têm de ser feitas são as configurações intrínsecas ao CPU, ou seja, a definição da frequência principal, a configuração dos barramentos HSB, PBA e PBB, bem como do SMC, por causa da memória *flash*, e a configuração do controlador SDRAM para o µC poder ler e escrever na memória SDRAM externa.

Como a inicialização da aplicação ainda demora algum tempo, e para que durante esse tempo o ecrã não permaneça sem qualquer imagem, uma imagem do logótipo da empresa que suporta este projecto é mostrada no ecrã. Para isso tiveram de ser também inicializados o controlador LCD e a interface MCI, tal como o respectivo FatFs, isto é, fazer a montagem do sistema de ficheiros.

De seguida, paralelamente com a mostragem do logótipo, é inicializada a aplicação, tendo nesta fase várias tarefas a realizar, tais como: a leitura do ficheiro que contém discriminadamente as informações que indicam os parâmetros com os quais o sistema tem de processar e, inclusive, quais as imagens dentro do cartão SD/MMC que têm de ser carregadas para a memória; a organização destas mesmas imagens consoante o piso, a indicação de alarme ou a seta de movimento a que pertencem; o carregamento das fontes para os vários textos que a aplicação mostra no ecrã; montagem das imagens em memória (SDRAM) referentes ao *marquee* horizontal (texto horizontalmente deslizante) e dos dois *marquees* verticais, sendo um referente aos pisos e o outro ao directório descritivo; entre outras tarefas explicadas posteriormente.

Posto isto, é altura de inicializar a comunicação com o Vinculum para, desde já, ficar à "escuta" de uma eventual USB *pen drive* inserida, de configurar os temporizadores essenciais para o decorrer do funcionamento, de inicializar o relógio e a comunicação RS485, bem como as várias interrupções que podem ocorrer durante o programa.

Com tudo já bem definido e configurado, é então feito o arranque da aplicação para o programa entrar num ciclo infinito.

Este ciclo, a este nível, acaba por ser simples e eficaz. A primeira função a executar é o processamento dos dados recebidos da comunicação RS485, se houver alteração dos mesmos e se forem válidos. Em seguida, vai ser verificado se está na altura do relógio ser actualizado, se já passou 1 minuto, ou se o relógio entrou em modo de acerto.

Como já foi referido no capítulo da apresentação do projecto, é utilizada a técnica do duplo *buffer*, isto é, cada *buffer*, correspondente a um ecrã inteiro (800 x 600 píxeis), tem uma área em memória (SDRAM) reservada mas só um é que é mostrado de cada vez. Assim, enquanto um *buffer* está a ser mostrado no ecrã, o outro já pode ser total ou parcialmente alterado. Para acrescentar, a troca dos buffers não é feita em qualquer momento mas sim só quando o controlador LCD chega ao final da *frame*, dissipando-se a possibilidade de se conseguir ver duas imagens em simultâneo.

Por este motivo, e agora retomando ao ciclo, é que é introduzida uma condição: saber se o LCD já está pronto ou não, ou seja, se os *buffers* já foram ou não trocados. Se esta condição for válida, e depois de serem efectuadas as actualizações necessárias, é colocada nova condição: testar se há, efectivamente, alterações no *layout* e, se assim for, "pedir"

para que os *buffers* sejam trocados. Qualquer que seja o resultado destes dois testes é verificado se o Vinculum fez com que o sistema tenha de entrar em modo de actualização (via *pen drive*).

5.1.1. ESTRUTURA DO *LAYOUT*

Nesta sub-subsecção vai ser explicitada toda uma estrutura que envolve o *layout*, isto é, todas as camadas subjacentes e que suportam todo o processo responsável pela disposição e dinâmica imposta às funcionalidades visíveis no ecrã.

Na base desta estrutura, a primeira camada descrita, está uma estrutura em C mais elementar, que permite o alocamento em memória de um vector de bytes com tamanho bem definido:

```
typedef struct ByteArray_s
{
   uint8_t *dataPtr;       // Pointer to the actual array of data
   int32_t size;           // Array size (number of elements)
   int32_t allocatedSize;  // Allocated size (number of allocated
                              elements)
}ByteArray_t;
```

Esta estrutura é transversal mas o principal objectivo dela é guardar, inicialmente, os ficheiros lidos do cartão SD/MMC, inclusive as imagens, e, posteriormente, armazenar as imagens já processadas para estarem prontas para serem copiadas directamente para o *buffer* de memória do controlador LCD.

A acompanhar esta estrutura é disponibilizado um conjunto de funções que ajudam a sua manipulação.

A primeira, cujo protótipo é “bool ByteArray_Initialize(ByteArray_t *array, int32_t size);”, diz respeito à inicialização da variável do tipo “ByteArray_t”, passada à função por referência. Ao seu apontador, “dataPtr”, é indicada a posição de memória alocada com tamanho igual ao valor que é passado por parâmetro. Para além disso, as variáveis “size” e “allocatedSize” são devidamente definidas com 0 e com o referido tamanho, respectivamente.

Outra função importante é a que apaga a variável do tipo “ByteArray_t”, que no fundo corresponde a libertar a memória apontada por “dataPtr”, para além de atribuir 0 ao “size”

e ao “allocatedSize”. O protótipo desta função é “void ByteArray_Delete(ByteArray_t *array);”.

Depois há mais quatro funções úteis para o programa. Uma delas é a função responsável por copiar um conjunto definido de bytes de uma variável do tipo “ByteArray_t” para outra do mesmo tipo, directamente de uma posição de memória para a outra. Outra função é a que acrescenta a uma variável do tipo “ByteArray_t” um byte à última posição ocupada do vector, isto se o tamanho ocupado já não tiver excedido o tamanho alocado para o vector. Para terminar este bloco de funções, há a acrescentar mais duas funções, que, por sinal, são antagónicas. Enquanto uma devolve o byte de uma posição específica do vector passada por parâmetro, a outra escreve em memória um byte numa variável do tipo “ByteArray_t” numa posição também específica, tendo de estar dentro do tamanho de memória alocado. Quer a posição quer o byte são passados por parâmetro à função.

Na camada imediatamente acima da responsável pelo vector de bytes estão presentes 3 estruturas, que tratam da manipulação dos ficheiros de imagem no formato BMP: duas delas correspondem ao *file header* e ao *info header*, sendo informações contidas nos ficheiros BMP originais; a terceira estrutura em C já diz respeito ao programa em si. As estruturas em C são as seguintes:

```
/* File header */
typedef struct
{
   uint8_t bfType[2];    // Magic number "BM"
   uint32_t bfSize;      // File size
   uint16_t bfReserved1;
   uint16_t bfReserved2;
   uint32_t bfOffBits;   // Offset to image data
}__attribute__((__packed__)) FileHeader_t;

/* Info header */
typedef struct
{
   uint32_t biSize;      // Size of bitmap info header
   int32_t biWidth;      // Image with
   int32_t biHeight;     // Image height
   uint16_t biPlanes;    // Must be equal to 1
   uint16_t biBitCount;  // Bits per pixel
   uint32_t biCompression;   // Compression type
   uint32_t biSizeImage;     // Size of pixel data
```

```
    int32_t biXPelsPerMeter;  // Pixels per meter on x-axis
    int32_t biYPelsPerMeter;  // Pixels per meter on y-axis
    uint32_t biClrUsed;       // Number of used colors
    uint32_t biClrImportant;  // Number of important colors
} __attribute__((__packed__)) InfoHeader_t;

typedef struct Bmp_s
{
    ByteArray_t dataRGB;
    uint16_t width;
    uint16_t height;
}Bmp_t;
```

Em relação à última estrutura, esta é constituída por uma variável, designada por "dataRGB", do tipo "ByteArray_t", que contém, como o próprio nome sugere, todos os bytes relativos ao RGB da imagem BMP. Para além do conteúdo da imagem, esta estrutura possui também a largura e a altura da respectiva imagem, parâmetros que são passados pela estrutura *info header*.

Mais uma vez, estas três estruturas são acompanhadas por diversas funções que têm como objectivo abrir os ficheiros de imagem no formato BMP, reter as informações úteis que contribuem para a selecção do que é realmente importante para este projecto e armazenar estes dados em memória através da estrutura acima citada.

Antes de se passar à representação de cada função individual, na Figura 59 pode-se visualizar um fluxograma que simplifica a compreensão da cadeia de funções que constituem esta camada. A função que é chamada pela camada imediatamente acima é a função "abrir BMP", que por sua vez chama a função "carregar BMP" e esta chama a "ler BMP".

Dentro desta camada, representada pelo fluxograma da Figura 59, é questionado o número de bits por pixel (bpp) que constituem a imagem e, consoante o resultado, é chamada a função correspondente. Se por acaso o bpp não estiver dentro dos previstos (1, 4, 8 ou 24), ou até mesmo se ocorrer algum erro dentro deste processo todo, as alocações feitas são anuladas e a função "abrir BMP" devolve uma indicação de erro.

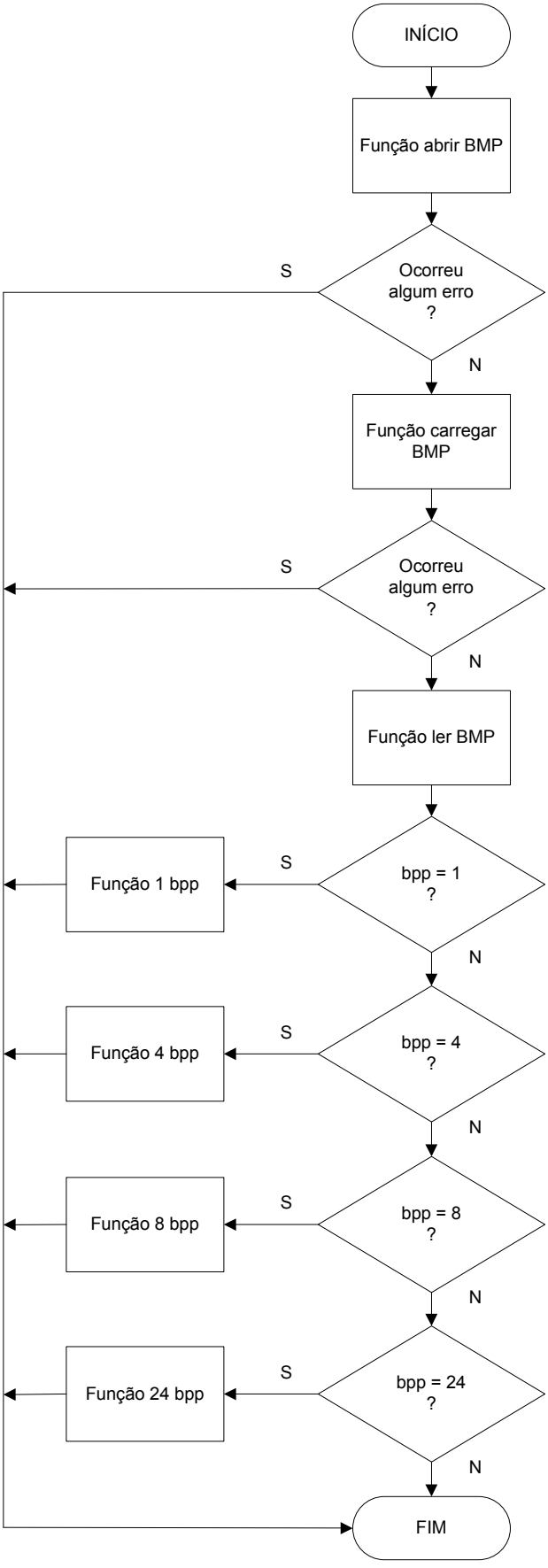

Figura 59 Fluxograma geral da camada responsável pela leitura de um ficheiro BMP

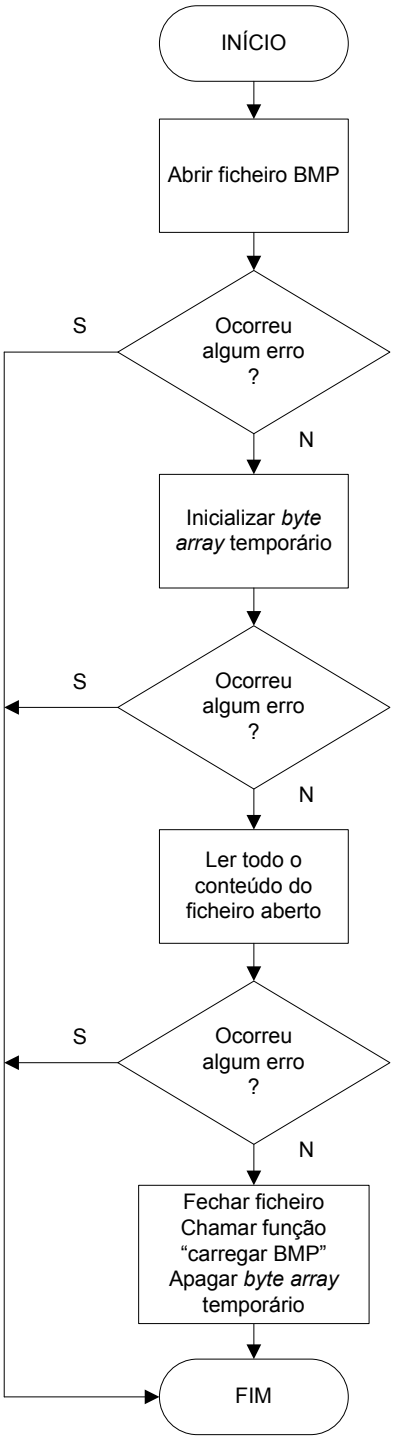

Figura 60 Fluxograma da função "abrir BMP" da camada relativa ao BMP

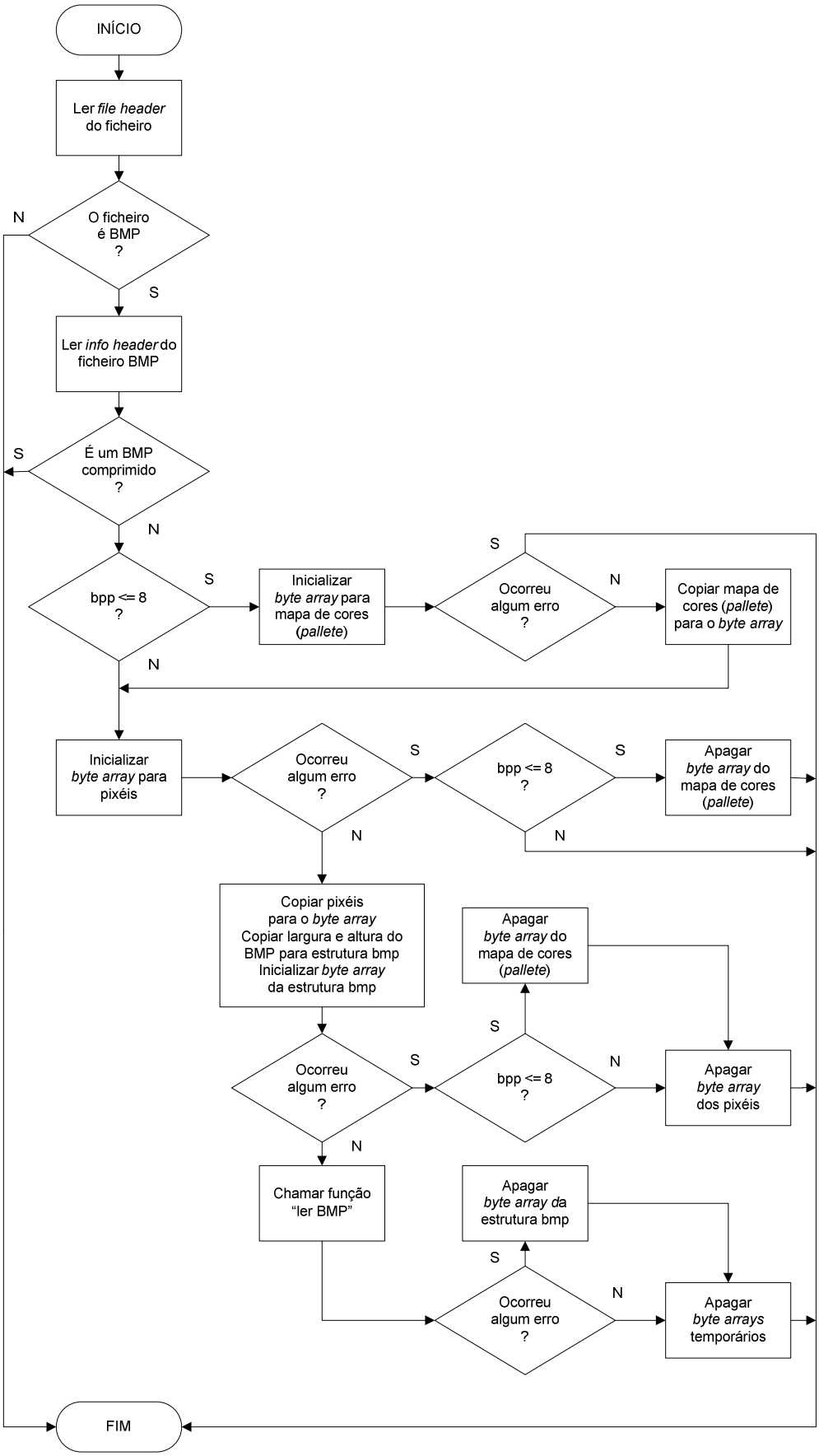

Figura 61 Fluxograma da função "carregar BMP" da camada relativa ao BMP

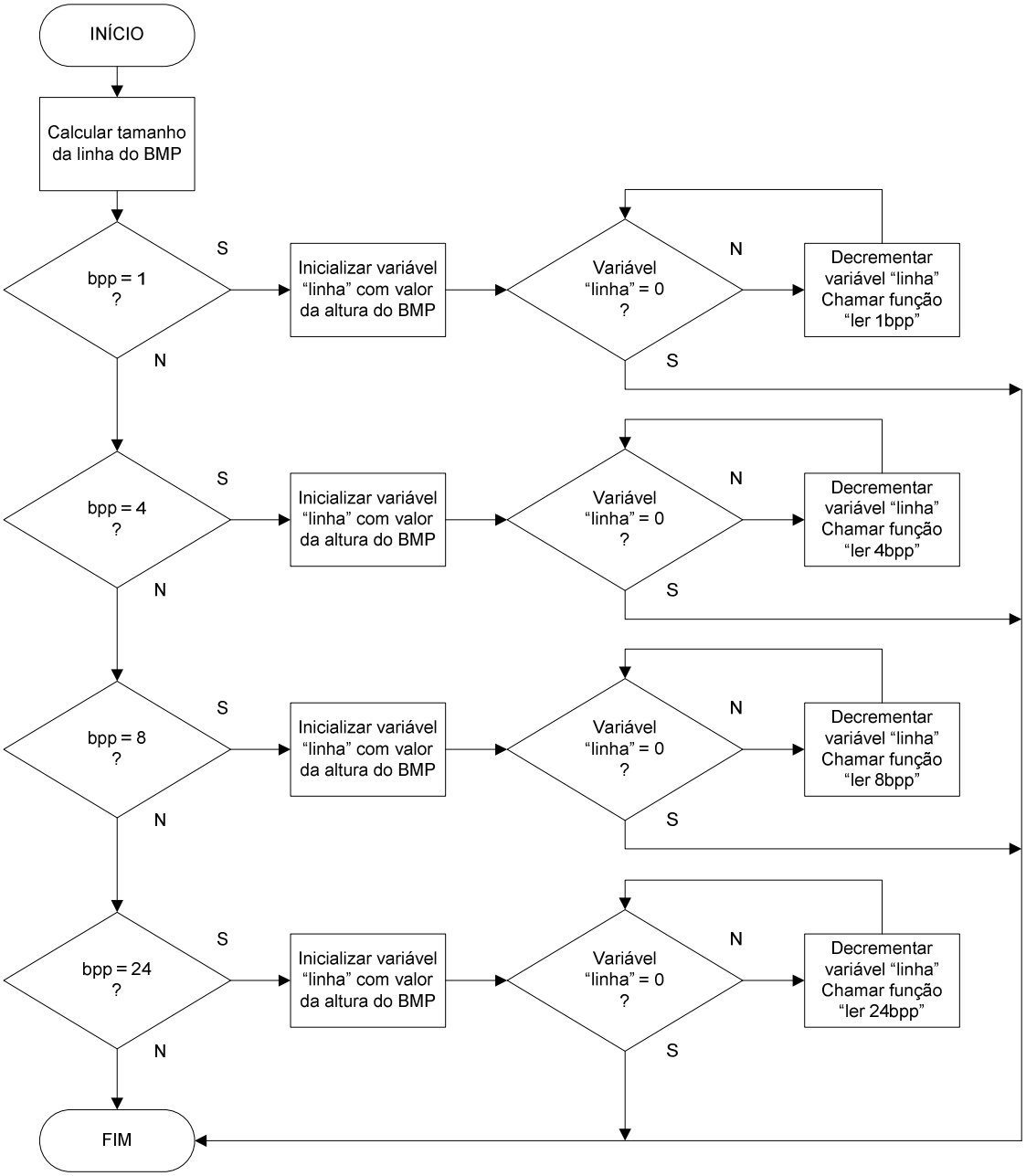

Figura 62 Fluxograma da função "ler BMP" da camada relativa ao BMP

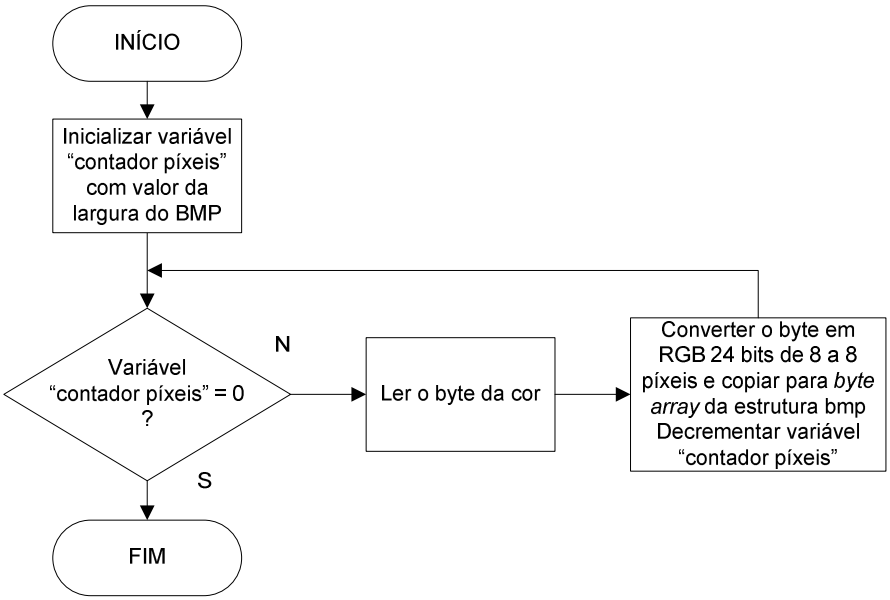

Figura 63 Fluxograma da função "ler 1bpp" da camada relativa ao BMP

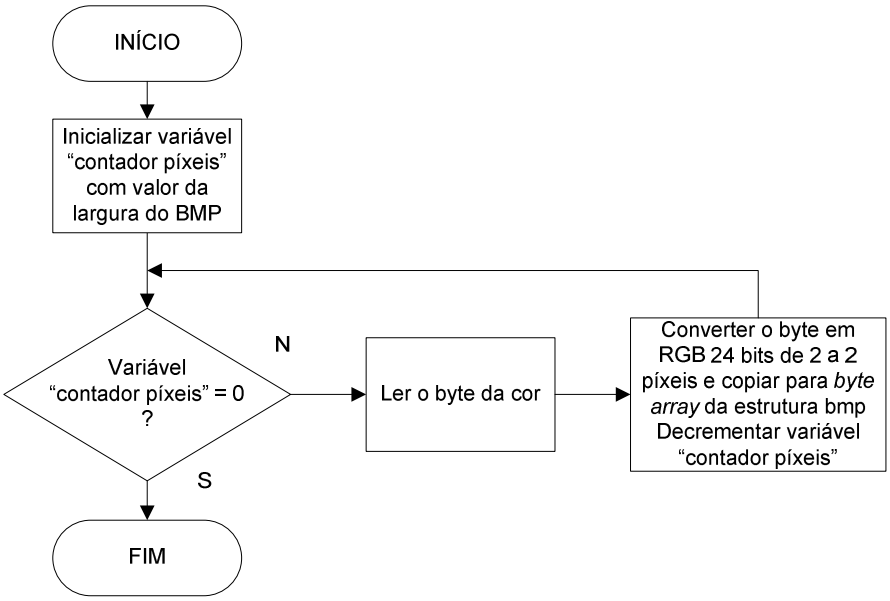

Figura 64 Fluxograma da função "ler 4bpp" da camada relativa ao BMP

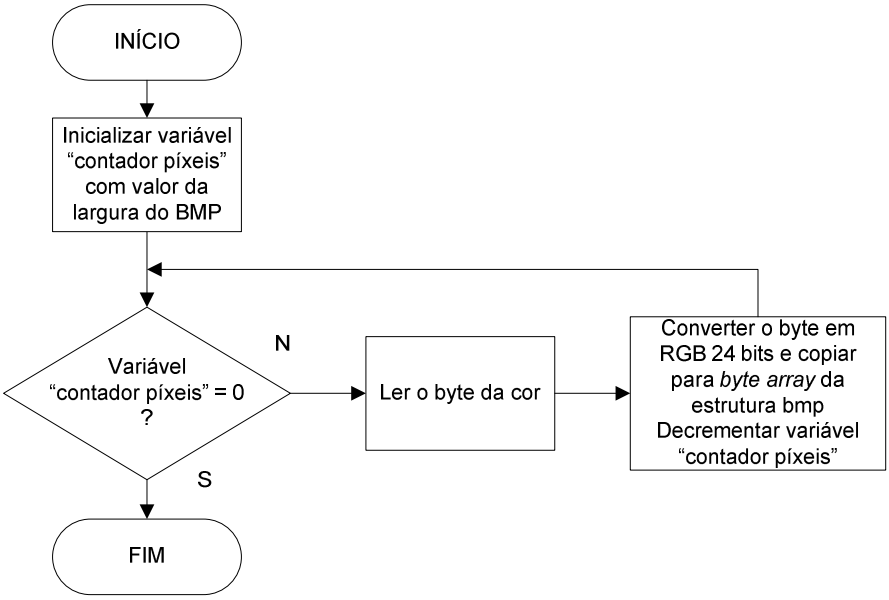

Figura 65 Fluxograma da função "ler 8bpp" da camada relativa ao BMP

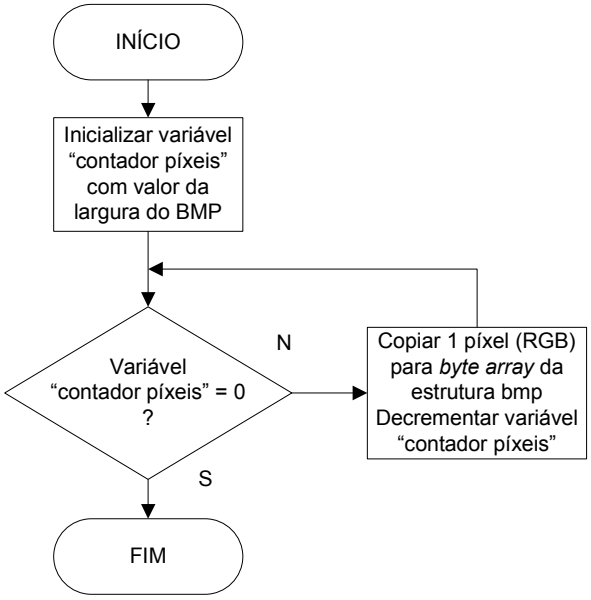

Figura 66 Fluxograma da função "ler 24bpp" da camada relativa ao BMP

Na Figura 60 está representado o fluxograma que esquematiza bem o funcionamento da função que é chamada pela camada acima. Inicialmente é utilizada a função de abertura de ficheiro do FatFs para abrir o ficheiro BMP pedido. De seguida é inicializada uma variável do tipo "ByteArray_t" temporária para guardar o conteúdo posteriormente lido do ficheiro BMP aberto. Após isto o ficheiro aberto já pode ser fechado e ser chamada a função seguinte: "carregar BMP" e dealocada a variável temporária. Se ocorrer algum erro numa

destas etapas, todo o procedimento que envolva a memória feito até então é liberto e devolvido um erro à função que a chamou.

Em relação às funções “carregar BMP” e “ler BMP”, a melhor explicação é mesmo a visualização dos fluxogramas apresentados na Figura 61 e na Figura 62, respectivamente, e não esquecendo as funções possivelmente chamadas por esta última, representadas desde a Figura 63 à Figura 66. De salientar só que, se eventualmente ocorrer qualquer erro, todas as variáveis alocadas são dealocadas e a função retoma à função que a chamou.

Continuando a subir, a camada seguinte é responsável por gerir todo um conjunto de estruturas do tipo “Bmp_t”, desde uma simples unidade até ao número que a memória permitir. Para isso foi criada outra estrutura:

```
typedef struct SpriteArray_s
{
   Bmp_t **data;
   int32_t size;
}SpriteArray_t;
```

Nesta estrutura em C está presente um vector de apontadores do tipo “Bmp_t”, ou seja, é um vector que armazena as imagens já manipuladas e, para se poder controlar este *array*, esta estrutura também tem incluído uma variável “tamanho”.

Para gerir esta estrutura, esta camada disponibiliza duas funções: a primeira inicializa a estrutura e a segunda acrescenta à última posição do vector “data” mais uma unidade.

O objectivo da primeira é muito simples, atribui “NULL” à variável “data” e 0 ao “size” a uma estrutura do tipo “SpriteArray_t”, passada por referência à função.

Na segunda, para além da estrutura passada por referência, é também recebido o nome do ficheiro da imagem BMP. Inicialmente, é alocada uma variável do tipo “Bmp_t”, isto para ser chamada a função “abrir BMP”. De seguida, há duas situações possíveis, ou o vector está vazio ou não! Mas é importante fazer este teste pois o processo respectivo difere. No caso de estar vazio, é simplesmente alocada uma posição no vector (a primeira), enquanto caso o vector já tenha algum conteúdo, o procedimento é o seguinte: é alocado um vector temporário de apontadores do tipo “Bmp_t” com tamanho “size + 1”, depois todos os apontadores contidos em “data” na estrutura passada por referência são copiados para o vector temporário para, assim, dealocar “data” e esta passar a apontar para a nova posição

de memória apontada pelo vector temporário. Justifica-se este procedimento pelo facto de ser necessário recorrer-se à função “memalign”, que aloca memória alinhada a 32 bits, neste caso. Ora, como a função “realloc” não faz este alinhamento, não pode ser usada e, portanto, tem de ser feita uma realocação “manual”. No fim de tudo é atribuído à última posição do vector “data”, depois do “size” ser incrementado, a posição de memória onde ficou alocada a imagem BMP.

No próximo nível insere-se duas estruturas em C: uma que engloba o conteúdo dos bmps e outra referente ao conteúdo dos andares (pisos do elevador).

Mas antes de se proceder à explicação destas estruturas e das funções que as acompanham, é essencial fazer-se aqui um ponto da situação. Na altura da leitura dos dados presentes no cartão de memória SD/MMC, que será explicado mais à frente, é feito o carregamento de todas as imagens BMP utilizáveis para a memória SDRAM, de uma forma sequencial e sem imagens repetidas. Estas imagens, já prontas para serem mostradas no ecrã, são armazenadas numa única variável do tipo “SpriteArray_t”, sem qualquer preocupação na ordem. A ordem e a organização destas imagens ficam ao encargo da variável ou das variáveis do tipo da primeira estrutura citada nesta camada (“BmpsContent_t”).

```
typedef struct
{
   uint16_t *bmpIndex;
   uint16_t size;
   uint16_t id;
   uint16_t posx;
   uint16_t posy;
   uint8_t updated;  // Not used in FloorsContent_t
}BmpsContent_t;

typedef struct
{
   BmpsContent_t **floorIndex;
   uint16_t floorSize;
   uint16_t floorId;
   uint8_t updated;
}FloorsContent_t;
```

A primeira estrutura, “BmpsContent_t”, contém um vector onde são armazenados os índices das imagens que estão posicionadas no vector “data” da estrutura “SpriteArray_t”. Como suporte, estão também presentes as variáveis “size”, que como o próprio nome

indica é o tamanho do *array* “bmpIndex”, “id”, que indica, numa fase posterior, o índice da imagem a ser lida/escrita e as posições em x e em y, “posx” e “posy”, respectivamente, que é a posição onde vão ser colocadas as imagens no ecrã. Esta posição é comum para todas as imagens porque cada grupo de imagens com a mesma localização no ecrã, como exemplos as mensagens de alerta e as imagens das setas, tem associado uma única variável do tipo “BmpsContent_t”. Encontra-se também uma variável “updated” que é útil, numa fase posterior, para a escrita da imagem no *buffer* do controlador LCD.

Como já foi referido, cada piso do elevador pode ter associado várias imagens de publicidade (ou de outra coisa qualquer). Assim, no conjunto de todos os pisos configurados na aplicação vai existir vários sub-conjuntos de imagens. O número destes sub-conjuntos é igual ao número de pisos mais um conjunto de imagens *default* (correspondente a imagens passadas quando o elevador está em movimento), estando este último pré-definido como o primeiro sub-conjunto do vector inserido na estrutura “FloorsContent_t”.

Como é de fácil compreensão, a estrutura “FloorsContent_t” foi implementada somente para a gestão das imagens de publicidade. Esta contém um *array* de apontadores para os tais sub-conjuntos acima denominados, sendo do tipo “BmpsContent_t”. Para além deste vector tem também, inevitavelmente, um tamanho associado ao número total de pisos, incluindo o de *default*, chamado de “floorSize”, um identificador do piso – “floorId” – e novamente a variável “updated”.

Também nesta camada estão presentes cinco funções, as duas primeiras relativas à estrutura “BmpsContent_t” e as três últimas referentes à “FloorsContent_t”. A primeira é a inicialização da estrutura respectiva, que é passada por referência à função, e tem como objectivo atribuir “NULL” ao vector “bmpIndex”, e 0 quer ao “size” quer ao “id”. Quanto à segunda, a função é acrescentar um índice, passado por parâmetro à função, à última posição livre do vector presente na variável do tipo “BmpsContent_t”, passada por referência. Para tal, o procedimento aqui presente é parecido ao já descrito na função de acrescentar um bmp ao “spriteArray”: é feito um teste se o tamanho do vector é zero ou não; se sim, é alocada uma posição de memória do vector “bmpIndex”; se o vector tiver tamanho maior que zero, é alocado numa variável temporária um número de índices “size + 1”, copiados os índices do “bmpIndex” para essa variável temporária, é dealocada a memória apontada por esta e, finalmente, atribuída a esta a posição de memória contida na

variável temporária. No final deste procedimento todo é atribuído o índice requerido à última posição do vector “bmpIndex”, já com o “size” incrementado. Depois a primeira das três funções relativas à estrutura “FloorsContent_t” é a inicialização de uma variável deste tipo, enviada por referência à função. Aqui faz-se o mesmo tipo de inicialização citada na função de inicialização anterior: “floorIndex” com atribuição “NULL”, e “floorSize” e “floorId” com 0. O objectivo da função seguinte é fazer a expansão, ou acrescentar, do vector de apontadores “floorIndex” da variável do tipo “FloorsContent_t”, passada por referência à função. Mais uma vez, é testado se o “floorSize” é nulo ou não, se sim é alocada uma posição de memória (a primeira) de um apontador do tipo “BmpsContent_t” no vector “floorIndex”, se não é alocado um número de apontadores do mesmo tipo com “floorSize + 1” numa variável temporária, de seguida são copiados os apontadores contidos no “floorIndex” para esta variável temporária para, assim, dealocar “floorIndex” e atribuir a esta a posição de memória da variável temporária. Depois do teste e dos respectivos procedimentos é, então, alocada [mais] uma posição de memória referente a [mais] uma estrutura do tipo “BmpsContent_t”. Isto é feito na última posição do vector “floorIndex”, correspondente à posição previamente alocada. Posto isto, já é possível chamar a função “inicializar BmpsArray” e incrementar, finalmente, o “floorSize”. Por último, a camada disponibiliza uma função que acrescenta um índice numa posição do “floorIndex” qualquer (na estrutura do tipo “BmpsContent_t”), ambos passados por parâmetro a esta função, numa variável do tipo “FloorsContent_t” passada por referência à função. No fundo, o que esta função “acrescentar FloorsArray” faz é chamar a função “acrescentar BmpsArray”. De salientar só que, caso ocorra algum erro em qualquer das funções descritas, nomeadamente nas alocações, essa função é retornada com uma indicação de erro.

Em cima das camadas anteriormente descortinadas está um nível com três conjuntos de funções que auxiliam o processamento da aplicação propriamente dita. Entre eles estão as funções que se responsabilizam por ler e processar os dados contidos no ficheiro de configuração presente no cartão de memória SD/MMC, as funções que copiam uma imagem para o *buffer* do controlador LCD e também uma série de funções que processam os dados recebidos por RS485.

O primeiro conjunto, designado por “carregar bmps”, é responsável por ler um ficheiro com nome “index.dat”, que tem de estar no cartão SD/MMC, e cuja motivação da

aplicação é, através dos dados inseridos nesse ficheiro, configurar o sistema de forma a correr com as definições, nomeadamente imagens e textos, pretendidas pelo cliente.

Para se compreender melhor o formato dos dados contido neste ficheiro de configuração, e principalmente para sustentar o fluxograma da Figura 68, são apresentados de seguida excertos de um exemplo desse ficheiro.

```
IMAGES
ENTRIES 37
PICTURE 1 gui/fs.bmp
PICTURE 2 gui/ol.bmp
PICTURE 3 gui/mt.bmp
PICTURE 4 advert/f_def1.bmp
PICTURE 5 advert/f_def2.bmp
…
PICTURE 34 advert/f_del2.bmp
PICTURE 35 gui/updating.bmp
PICTURE 36 gui/updated.bmp
PICTURE 37 gui/nUpdated.bmp

MESSAGES
ENTRIES 6
PICTURE 1
PICTURE 2
PICTURE 3
PICTURE 35
PICTURE 36
PICTURE 37

FLOOR --
DESCRIPTION
ENTRIES 20
PICTURE 4
PICTURE 10
…
FLOOR -2
DESCRIPTION Health Club & SPA
ENTRIES 7
PICTURE 14
PICTURE 15
…
FLOOR -1
DESCRIPTION Restaurants
```

```
ENTRIES 7
PICTURE 20
PICTURE 21
…
FLOOR 0
DESCRIPTION Entrance
ENTRIES 20
PICTURE 4
PICTURE 10
…
FLOOR 1
DESCRIPTION Theater
ENTRIES 2
PICTURE 27
PICTURE 28
…
FLOOR 28
DESCRIPTION Last floor
ENTRIES 2
PICTURE 33
PICTURE 34

EOF
```

Logo no início do ficheiro são listadas todas as imagens que serão utilizadas pela aplicação, em que cada linha tem, para além da designação “PICTURE”, o directório e nome da imagem BMP e, no meio destes, um índice. Este índice identifica cada imagem independente atribuída nas linhas seguintes. Assim, para além de ser substituído o directório (e nome) por um simples número, garante-se que não vão haver imagens repetidas armazenadas na memória SDRAM. Na parte das “MESSAGES” são definidas as imagens de alerta, tais como de incêndio, excesso de carga, etc., bem como as imagens relativas ao estado da actualização feita pela leitura da *pen* USB. Os “blocos” seguintes referem-se, visivelmente, às imagens mostradas para cada piso e respectiva descrição.

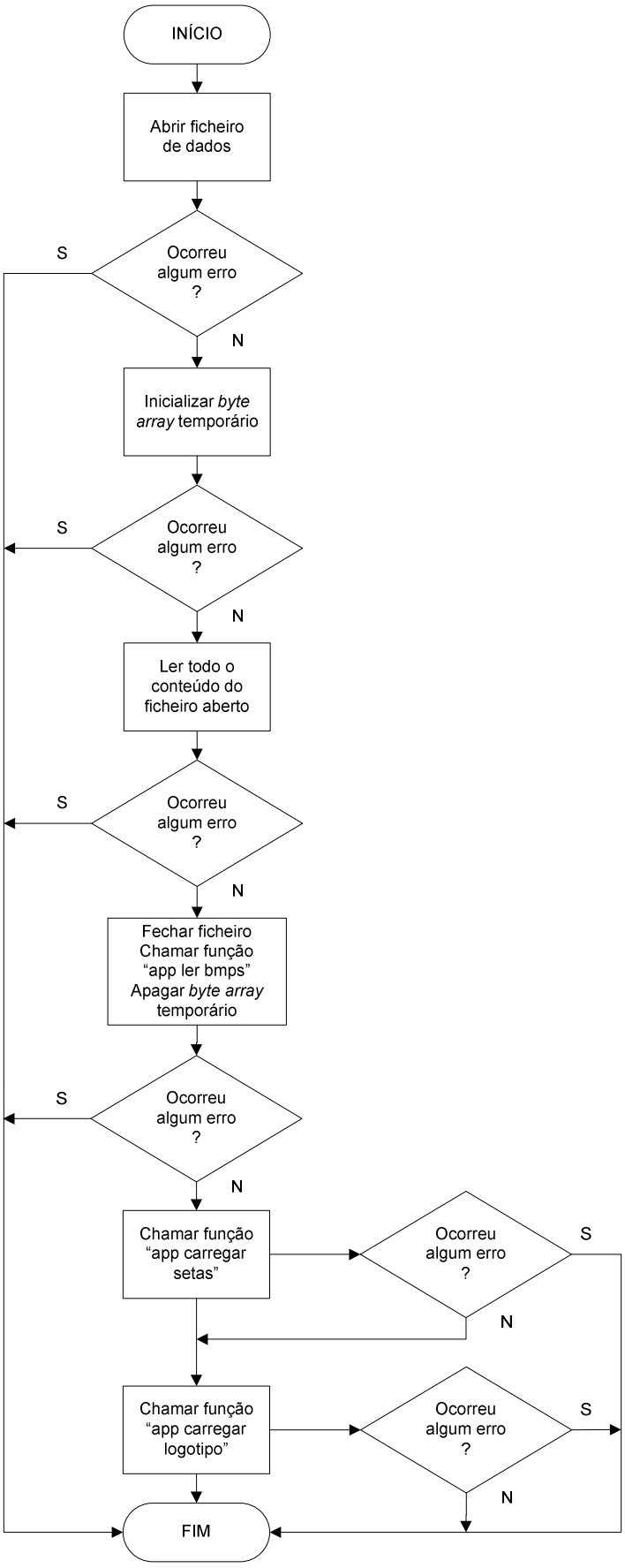

Figura 67 Fluxograma da função "carregar bmps" relativa à configuração da aplicação

Na Figura 67 está esquematizado o fluxograma da função principal que é chamada pela função de inicialização da aplicação, estando responsável não só por abrir e ler todo o conteúdo do ficheiro de configuração como também por chamar a função que processa todo esse conteúdo (dados) (Figura 68), nomeadamente a organização das imagens e dos textos, a função que processa as setas (Figura 69) e a que carrega o logótipo (Figura 70).

Figura 68 Fluxograma da função "ler bmps" relativa à configuração da aplicação

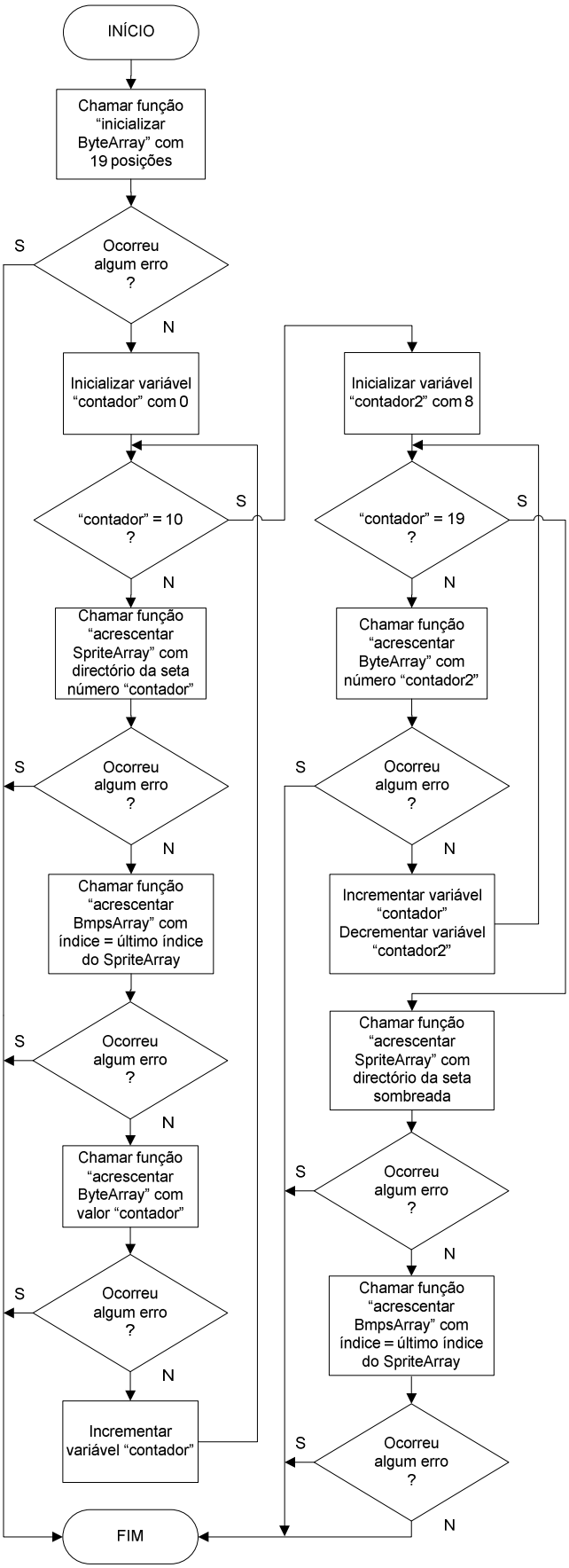

Figura 69 Fluxograma da função "carregar setas" relativa à configuração da aplicação

A função “carregar setas” inclui duas vezes o código de programa esquematizado no fluxograma da Figura 69: a primeira vez destina-se para a seta ascendente e logo de seguida outra para a seta descendente.

O princípio de funcionamento encontrado para idealizar a animação das setas, relembre-se, o objectivo é fazer parecer que o tamanho da imagem da seta vá gradualmente diminuindo até um tamanho muito reduzido, e logo de seguida vá aumentando até que volte ao tamanho original, e assim sucessivamente, é o seguinte.

Inicialmente são lidas, carregadas para a memória e organizadas num “BmpsArray” 10 imagens de setas, desde o tamanho maior (original) até ao mais pequeno, para depois também ser lida e carregada a imagem da seta sombreada, simbolizando que a deslocação do elevador não está no sentido respectivo. Paralelamente ao procedimento inicial, são guardados os índices das imagens que fazem a animação num vector de bytes (num “ByteArray”), e de seguida, nas próximas posições desse mesmo vector de bytes, são guardados os índices pela ordem inversa que foram guardados anteriormente.

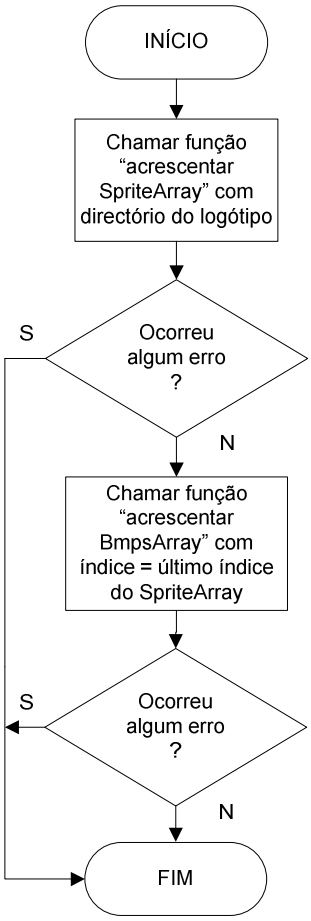

Figura 70 Fluxograma da função “carregar logótipo” relativa à configuração da aplicação

Assim, com este vector de bytes devidamente preenchido, e assumindo que o tamanho original da imagem corresponde ao índice 0 e a imagem com o tamanho mais pequeno tem índice 9, obtém-se a sequência de índices/imagens: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0. Com esta explicação o fluxograma da Figura 69 torna-se, agora, bem mais compreensível.

A função representada pelo fluxograma da Figura 70 tem a motivação de ler e carregar para a memória uma imagem no formato BMP do logótipo da empresa patrocinadora deste trabalho, destinada a ser mostrada logo no arranque do sistema.

O segundo dos três conjuntos de funções acima mencionados pertence às rotinas que se responsabilizam por preencher parcial ou totalmente o *buffer* de memória, que será enviado para o ecrã, com uma única cor ou copiando uma imagem já alocada.

A Figura 71 mostra o fluxograma da função que preenche todo um ecrã (um *buffer* de memória) com uma cor, passada por parâmetro à função. Esta função é útil, pelo menos para esta aplicação, logo no início do desenrolar da aplicação, nomeadamente após ser mostrado o logótipo da empresa, pois há a necessidade de garantir que os *buffers* estejam preenchidos com a cor de fundo pretendida.

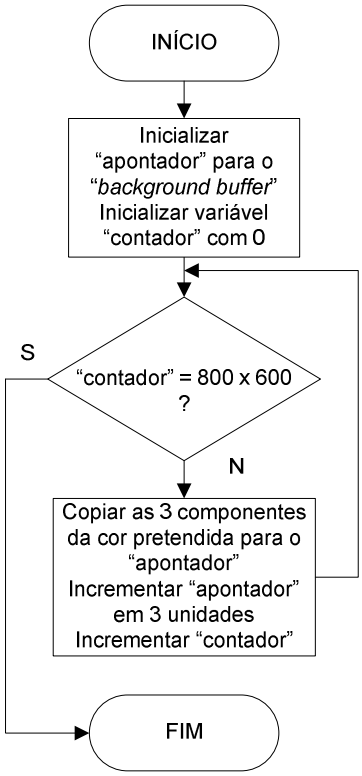

Figura 71 Fluxograma da função "preencher *background buffer*"

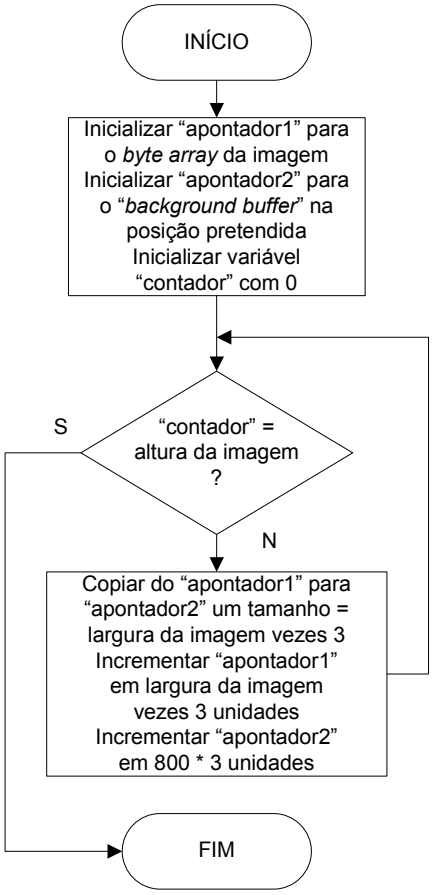

Figura 72 Fluxograma da função “escrever *background buffer*”

Já a função representada, por fluxograma, na Figura 72 escreve uma imagem contida numa variável do tipo “Bmp_t”, passada por referência à função, no *background buffer* numa posição em x e em y à escolha, ambas passadas por parâmetro à função.

De salientar uma nota implícita às duas funções anteriores: o controlador de LCD tem de ler os dados numa região *cacheable* mas a actualização do *buffer* convém ser feita numa região *non-cacheable* da memória [51]. Portanto, quando a variável do tipo apontador é criada e apontada, passa-se a redundância, para a posição de memória do *background buffer*, é feito uma operação lógica OR com o endereço 0xA0000000.

Há outras duas funções para além destas e que estão intrinsecamente relacionadas com a última. Uma é a função “escrever bmps *background buffer*”, que recebe por referência uma variável do tipo “SpriteArray_t” e outra do tipo “BmpsContent_t”, cuja responsabilidade é chamar unicamente a função “escrever *background buffer*”. A segunda função chama-se “escrever *floors background buffer*”, que também recebe por referência uma variável do tipo “SpriteArray_t” e uma segunda do tipo “FloorsContent_t”, tendo a mesma

responsabilidade da função anterior. Em suma, a primeira função destina-se a imprimir imagens como por exemplo as de alerta, e a segunda encarrega-se das imagens referentes à publicidade dos pisos.

No terceiro e último conjunto dos que foram supracitados estão as funções que processam os dados recebidos via RS485. Este conjunto é um anexo das funções que processam e geram a aplicação propriamente dita, pois as rotinas responsáveis por receber e tratar os dados recebidos via RS485 serão descritas numa sub-subsecção mais à frente.

Este conjunto subdivide-se em dois grupos de funções: nas relativas às setas e nas que correspondem à indicação de aviso. São 5 no primeiro caso enquanto no segundo são 4.

A primeira função – "*arrows off*" – do primeiro grupo processa ambas as setas "apagadas", ou seja, dá ordem à rotina encarregue das actualizações do *layout* para colocar uma imagem da seta sombreada, quer para a seta para cima quer para a seta para baixo.

A segunda, de seu nome "*arrow up*", dá a indicação à rotina que actualiza o *layout* para colocar uma seta para cima estática e uma seta para baixo sombreada.

O procedimento na terceira função – "arrow *down*" – é igual ao anterior, mas os papéis para cada seta invertem-se.

As duas últimas funções deste grupo destinam-se a indicar à mesma rotina que cada seta deverá ser mostrada no ecrã em "movimento". Isto é, fazer parecer ao observador que a seta está animada. A quarta função responsabiliza-se pela seta para cima, com o nome "*arrow up moving*", e a quinta e última função trata da seta para baixo, de seu nome "*arrow down moving*".

Passando agora para o grupo de funções das indicações de alarme, a primeira delas indica à rotina que actualiza o *layout* que tem de colocar a imagem no ecrã referente a uma indicação de incêndio.

As duas seguintes são respeitantes a excesso de carga e de elevador em manutenção.

Na última função, considerada "*default*", é simplesmente dada a ordem para "esquecer" as indicações de alarme e continuar com a mostragem das imagens referentes ao piso.

Tem de ser acrescentada uma nota. Estas funções descritas, principalmente as das indicações de alarme/aviso, podem ser, obviamente, atribuídas a outras indicações que o cliente pretenda.

A camada mais acima, a que é responsável, efectivamente, pela gestão do *layout*, tem associada duas estruturas em C:

```
typedef struct
{
   ByteArray_t arrowArray;
   uint8_t arrowId;
   bool arrowRun;
   bool arrowStop;
   bool arrowOff;
   bool arrowReady;
}ArrowContent_t;

typedef struct
{
   SpriteArray_t bmps;
   FloorsContent_t advert;
   BmpsContent_t message;
   BmpsContent_t arrowUp;
   ArrowContent_t arrowUpArray;
   BmpsContent_t arrowDown;
   ArrowContent_t arrowDownArray;
   BmpsContent_t logo;
   StringContent_t date;
   StringContent_t day;
   StringContent_t time;
   MarqueeContent_t marquee;
   VertMarqueeNumContent_t floorNumber;
   VertMarqueeDirContent_t floorDirectory;
}Layout_t;
```

Contudo, a estrutura “Layout_t” é que é a principal pois a primeira, a “ArrowContent_t”, é uma estrutura auxiliar da segunda.

A primeira variável contida nesta estrutura é a “bmps” do tipo “SpriteArray_t”, cujo conteúdo são todas as imagens possivelmente utilizadas pela aplicação.

A segunda, “advert”, contém a posição no vector presente na “bmps” de todas as imagens correspondentes de todos os pisos. A variável “message” é semelhante à anterior, porém, só contém as posições das imagens referentes às mensagens de alerta.

Segue-se um bloco de 4 variáveis relativas à gestão das setas, as duas primeiras variáveis referem-se à seta para cima e as outras duas para a seta para baixo. Mais uma vez aqui, para cada seta há uma variável do tipo “BmpsContent_t”. Mas há um complemento. Para cada seta há mais uma variável do tipo “ArrowContent_t”. Esta estrutura é a responsável por gerir o estado da seta através de algumas *flags*, como se pode ver na primeira estrutura transcrita.

Para o logótipo da empresa que pode ser mostrado junto ao relógio também está presente na estrutura uma variável do tipo “BmpsContent_t”.

De seguida, Estão presentes 3 variáveis relacionadas unicamente com o relógio: “date”, “day”, “time”. O tipo destas variáveis também é uma estrutura que vai ser explicada numa sub-secção mais à frente.

Por último, encontram-se 3 variáveis destinadas aos *marquees*. A primeira delas é o texto horizontalmente deslizante, a segunda é a indicação do piso, verticalmente deslizante, e a terceira, também na mesma direcção, é a descrição dos pisos, ou por outra palavra, o directório.

Há uma outra definição que é feita:

```
typedef void(*serieProcess_t)(Layout_t *layout);
```

Isto define o tipo “serieProcess_t” que possibilita posteriormente criar vectores de funções com o parâmetro “Layout_t *layout”.

Na prática, foram criados dois vectores de 8 posições cada, um para as mensagens de alerta e outro para as setas. Assim, cada posição de cada vector tem associação directa a cada função criada para processar os dados recebidos via RS485, já apresentadas no texto acima.

São 6 as funções responsáveis pela gestão do *layout* no ecrã: “inicializar app” (Figura 73), “mostrar logótipo” (Figura 74), “arrancar app”, “processar dados série” (Figura 75), “actualizar app” (Figura 76) e “trocar *buffers*”.

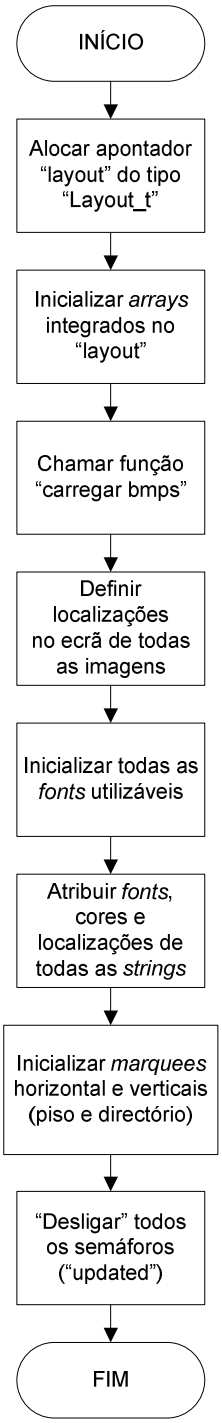

Figura 73 Fluxograma da função "inicializar app"

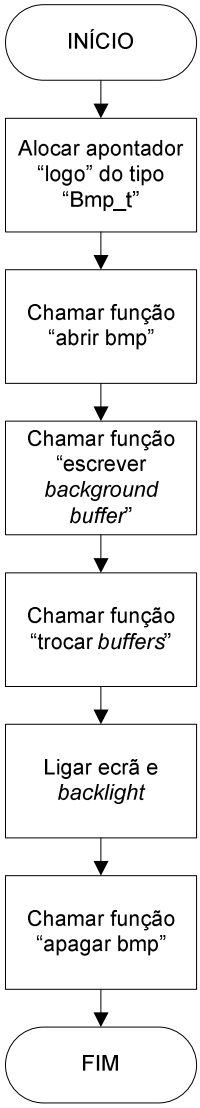

Figura 74 Fluxograma da função "mostrar logótipo"

Em relação à função "arrancar app", o objectivo dela é simples.

Como foi referido anteriormente, esta aplicação foi desenvolvida com flexibilidade ao ponto de se poder colocar no ecrã as áreas informativas que se desejar. Bem como as respectivas dimensões e localizações no ecrã, em conformidade com os limites impostos.

A missão da função "arrancar app" é precisamente esta. Aqui "liga-se" ou "desliga-se" as áreas informativas ao dispor do passageiro do elevador (ou outro utilizador qualquer), activando os "updated"s – designados neste trabalho como semáforos – das funcionalidades que se queiram ver no ecrã ou atribuindo-lhes "0", respectivamente. Também são feitas outras atribuições, tais como definições de *flags*, índices dos vectores de imagens que serão colocadas inicialmente no ecrã e piso inicial.

Executado este procedimento é feita a troca dos *buffers*, pois é de relembrar que, nesta altura, a imagem presente no ecrã é a do logótipo da empresa. E, logicamente, tem de ser feito o preenchimento do *background buffer* com uma cor à escolha, sendo chamada a função "preencher *background buffer*".

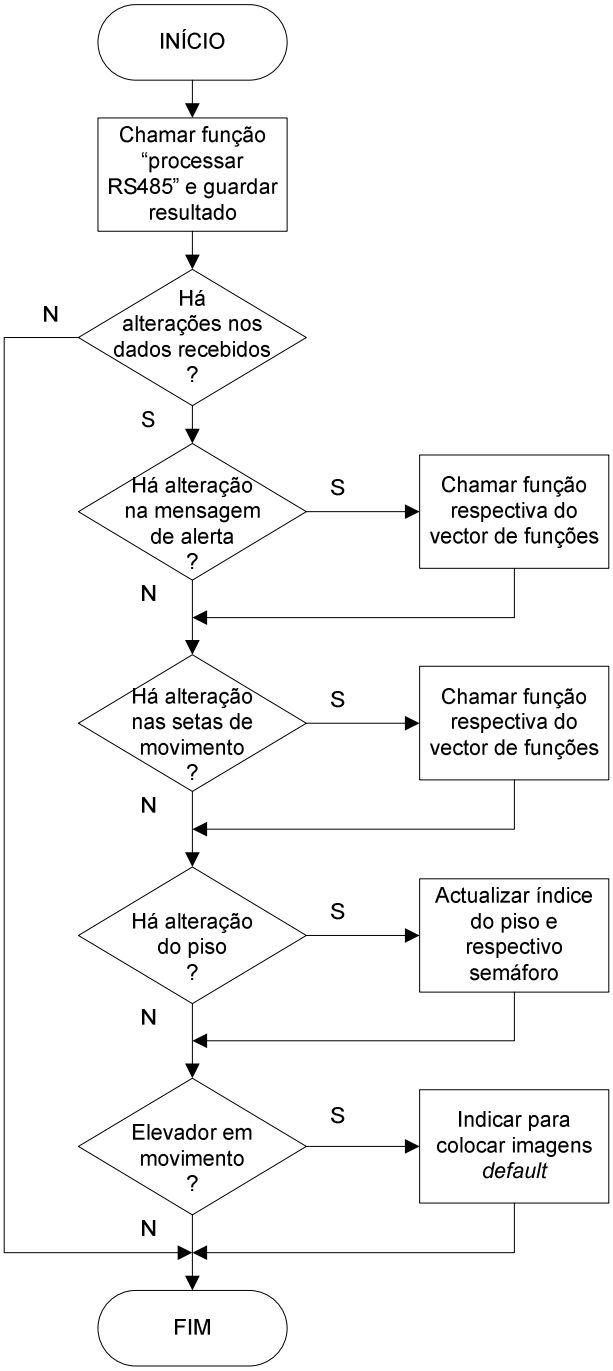

Figura 75 Fluxograma da função "processar dados série"

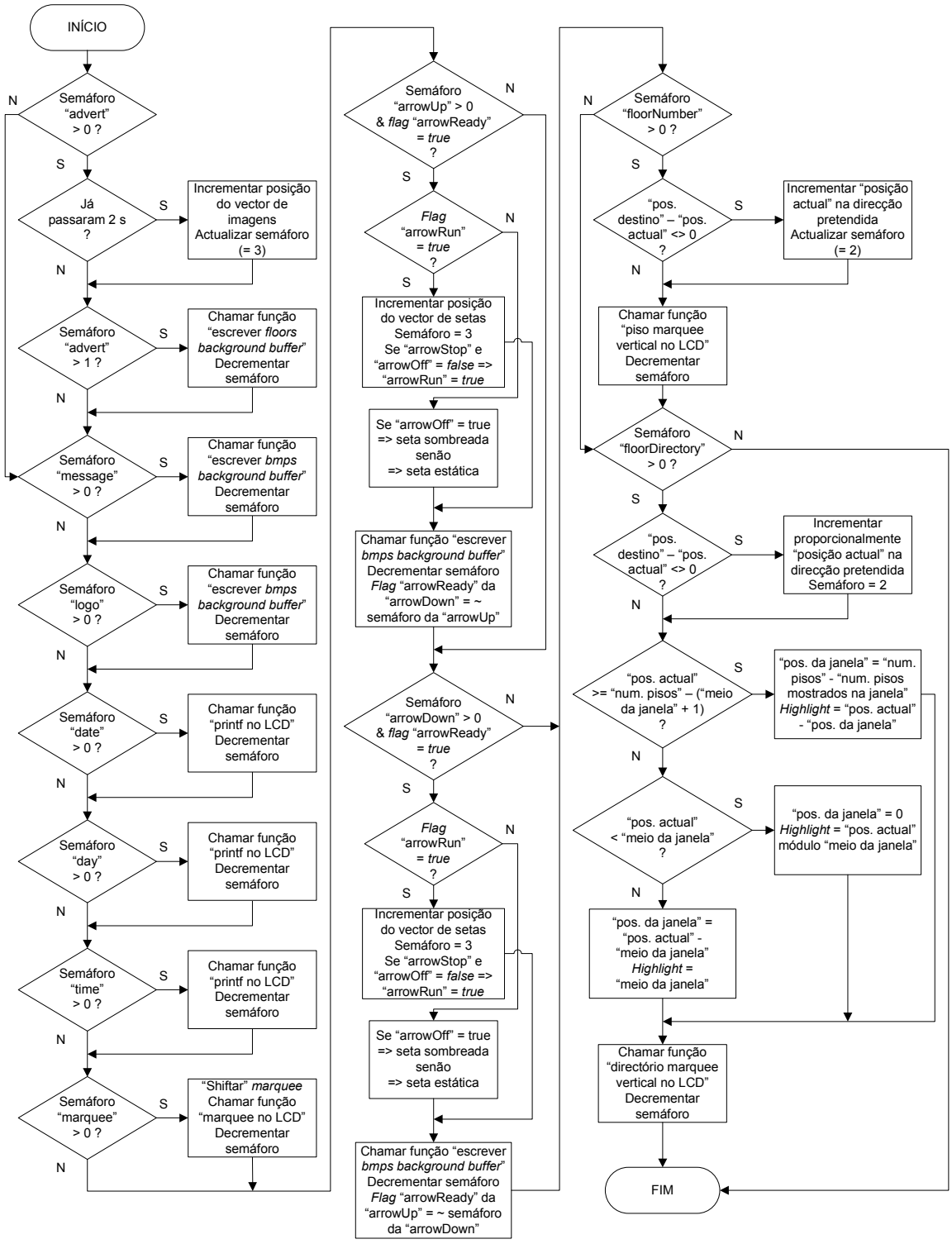

Figura 76 Fluxograma da função "actualizar app"

A última função deste conjunto referido é a "trocar *buffers*". Esta função por e simplesmente indica, através de uma *flag*, que o *background buffer* ainda não está pronto para ser actualizado e activa a interrupção do *enf of frame* para que, quando isto ocorrer, o programa entre na rotina de interrupção respectiva e, efectivamente, troque os *buffers*.

5.1.2. STRINGS

Nesta sub-subsecção são descritas todas as estruturas e funções que se encarregam ou contribuem por disponibilizar *strings* no ecrã, ou seja, todos os textos que aparecem no ecrã, como o relógio, por exemplo.

Uma estrutura muito usada nesta camada é a seguinte:

```
typedef struct
{
   uint8_t r;
   uint8_t g;
   uint8_t b;
}ColorRGB_t;
```

Nesta estrutura são armazenados os valores das componentes de cor RGB. É utilizado essencialmente para o ecrã ser preenchido com uma cor ou partes de um texto.

Depois existem mais duas estruturas, em que a primeira é um anexo da segunda:

```
typedef struct
{
   uint16_t posx, posy;
   uint16_t width, height;
   int16_t xOffset, yOffset;
   uint16_t xAdvance;
}CharDescriptor_t;

typedef struct
{
   Bmp_t *fontBmp;
   CharDescriptor_t Chars[256];
}Font_t;
```

A presença da estrutura “Font_t” justifica-se da seguinte forma. Todos os caracteres utilizáveis pelo programa têm de estar especificados e caracterizados num ficheiro de texto no cartão de memória SD/MMC. A acompanhar este ficheiro de texto, também tem de estar um ficheiro no formato BMP com todos esses caracteres desenhados. Cada carácter presente neste ficheiro BMP tem, logicamente, uma localização na imagem e dimensões bem definidas. São estas informações que têm de estar contidas no ficheiro de texto, para depois o programa poder escrever em memória a *string* pretendida.

Estes dois ficheiros, de texto e BMP, podem ser facilmente gerados com a ajuda de um *software* disponível na internet, de seu nome *Bitmap Font Generator* [52].

Com esta aplicação é, então, possível escolher quais os caracteres, o tamanho, o espaçamento, etc. para todos os conjuntos de *fonts* que se queira introduzir no código.

No ficheiro de texto ficam listados todos esses caracteres escolhidos e as respectivas informações. Estas informações, depois de ser executada a rotina que lê o ficheiro e processa os dados, ficam armazenadas na estrutura "CharDescriptor_t", enquanto os caracteres contidos, bem como a imagem com todos os caracteres desenhados, integram a estrutura "Font_t". A associação de cada carácter presente no conjunto da *font* é directa com o respectivo código ASCII. Para tal é que a estrutura "Font_t" tem um vector de 256 posições, número esse que permite contemplar todos os caracteres existentes na tabela ASCII.

Para que a estrutura "Font_t" seja total e devidamente carregada na memória existem 3 funções dependentes entre si: a "inicializar font" (Figura 78), a "abrir font" (Figura 79) e a "carregar font" (Figura 80).

Mas para se perceber melhor o contexto do conteúdo da função "carregar font", segue-se um excerto do que contém o referido ficheiro de texto, relativo à imagem da Figura 77.

```
chars count=147
char id=32   x=297   y=2     width=1     height=1     xoffset=0
yoffset=24    xadvance=14    page=0  chnl=15
char id=33   x=200   y=46    width=4     height=19    xoffset=5
yoffset=6     xadvance=14    page=0  chnl=15
char id=34   x=217   y=99    width=8     height=8     xoffset=3
yoffset=4     xadvance=14    page=0  chnl=15
char id=36   x=271   y=0     width=12    height=23    xoffset=1
yoffset=4     xadvance=14    page=0  chnl=15
…
```

Figura 77 Exemplo de uma imagem com os caracteres de uma *font*

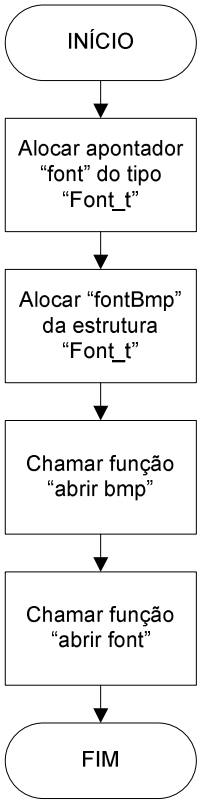

Figura 78 Fluxograma da função "inicializar font"

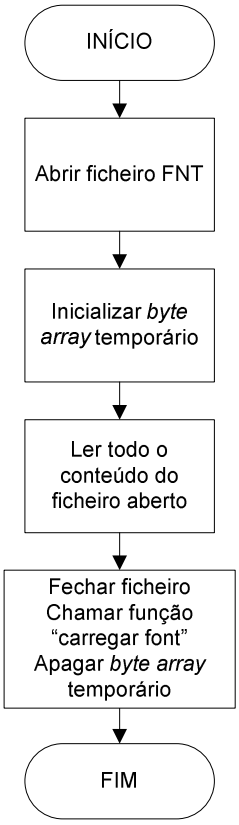

Figura 79 Fluxograma da função "abrir font"

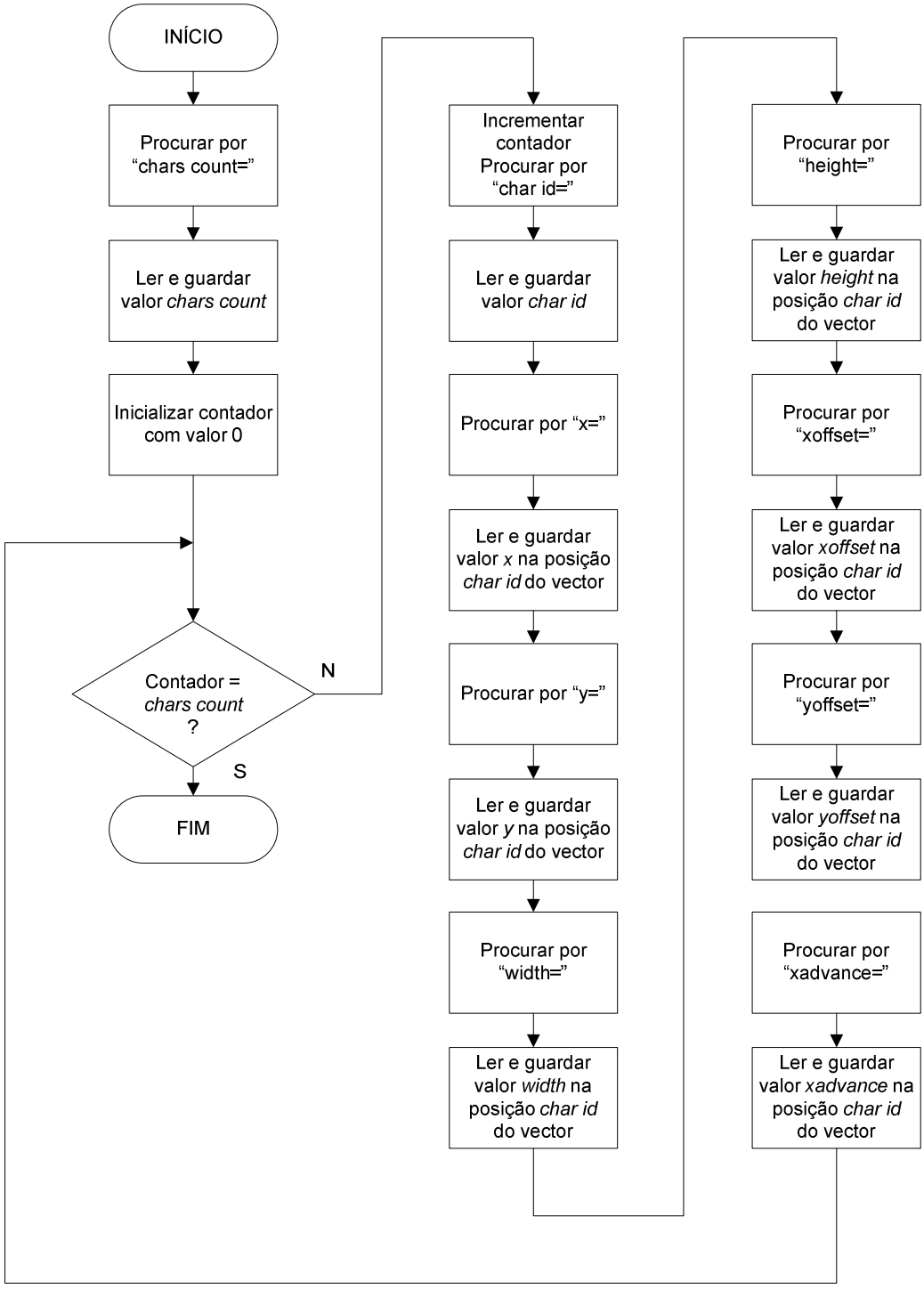

Figura 80 Fluxograma da função "carregar font"

Estando descritas as estruturas e funções responsáveis por organizar e carregar todas as *fonts* na memória, é altura de se passar para o relato das estruturas e funções que preparam antecipadamente e escrevem no ecrã as *strings* pretendidas.

Estão presentes 4 estruturas, semelhantes mas distintas, cada uma para cada área informativa diferente:

```
typedef struct
{
   uint8_t *bmpMarquee;
   uint16_t width, height;
   uint16_t posx1, posx2, posy, posxi;
   uint8_t updated;
}MarqueeContent_t;

typedef struct
{
   uint8_t *bmpVertMarquee;
   uint8_t nFloors;
   uint16_t width, height;
   uint16_t lineHeight;
   uint16_t posx, posy;
   uint16_t posyi, posyiDst;
   int16_t direction;
   uint8_t updated;
}VertMarqueeNumContent_t;

typedef struct
{
   uint8_t *bmpVertMarqueeN;    // Floor number's bmp in directory
   uint8_t *bmpVertMarqueeD;    // Floor description's bmp in directory
   uint8_t *bmpVertMarqueeNHL;  // Bmp of numbers highlighted
   uint8_t nFloors;
   uint16_t widthN, widthD;
   uint16_t height, lineHeight;
   uint16_t posx, posy;
   uint16_t posyi, posyiDst;
   uint16_t posyiWindow;
   int16_t direction;
   float posyif;
   float rateShift;
   uint8_t updated;
}VertMarqueeDirContent_t;


typedef struct
{
   Font_t *font;
```

```
    ColorRGB_t fontColor;
    uint16_t posx1, posx2;
    uint16_t posy1, posy2;
    char str[14];
    uint8_t updated;
}StringContent_t;
```

Antes de mais há que acrescentar uma nota importante. Como as *strings* não são dinâmicas ao longo do tempo, isto é, as *strings* são pré-definidas pelo cliente e carregadas no início do programa, o método utilizado foi criar e armazenar imagens no arranque do sistema já com essas *strings* formadas. Assim, o tempo que demora a escrever cada carácter em memória e, principalmente, a fazer o efeito de transparência só é gasto uma vez e logo no início. Cada imagem construída aqui é constituída pela *string* inteira, no caso do *marquee* horizontal, ou por todas as *strings* que compõem a respectiva área informativa, sequencialmente e linha a linha, como são os casos dos *marquees* verticais. No decorrer do programa simplesmente são feitas cópias dessas imagens, já elaboradas e alocadas em memória, desde uma posição até outra, consoante os dados recebidos do exterior do sistema e/ou do decorrer do próprio programa.

Portanto, após este esclarecimento, o conteúdo de cada estrutura acima transcrita torna-se mais claro na sua compreensão.

A estrutura “MarqueeContent_t” é constituída, então, por um vector de bytes, “*bmpMarquee”, que como o próprio nome sugere, armazenará a imagem com uma *string* que passará no ecrã ao estilo de uma nota de rodapé deslizante. Para o seu controlo, contém a respectiva largura e altura da imagem. As variáveis “posx1” e “posx2” são as posições no ecrã, na horizontal, inicial e final da imagem/*string*, respectivamente. Logicamente, também há a “posy” que é a posição na vertical. Por último, está presente a variável “posxi”, que é a posição da imagem na qual é iniciada a cópia para o ecrã em cada instante. Isto é, sendo o sentido do deslocamento da imagem para a esquerda, o valor de “posxi” será incrementado (avança X píxeis da imagem) periodicamente de forma a criar a ilusão que a imagem/*string* está a deslizar.

Em relação à estrutura “VertMarqueeNumContent_t”, a explicação do seu conteúdo não difere muito face à anterior. Neste caso já vai haver, muito provavelmente, várias *strings*, colocadas uma em cada linha. Por isto estão presentes as variáveis “lineHeight” e “nFloors”, pois é essencial para a escrita no ecrã saber-se a altura de todas as linhas e o

número de pisos, respectivamente. Nesta estrutura, ao contrário da anterior, a posição "i" é logicamente, na vertical. Mas há uma posição acrescentada: a "posyiDst". A diferença desta para a "posyi" é que enquanto a última é a posição actual de escrita da imagem no ecrã, a primeira é a posição final pretendida (destino), pelo menos até o sistema receber novos dados do exterior.

Na estrutura "VertMarqueeDirContent_t", embora o conceito seja o mesmo da anterior, há uma adição importante de ser realçada. Para relembrar, no directório de pisos, o(s) dígito(s) do piso em que o elevador se encontra no momento é realçado com uma cor à escolha. Um dos objectivos desta camada da aplicação é estabelecer um compromisso entre eficácia e eficiência. Com isto, e direccionando para esta estrutura, foi decidido criar 3 imagens logo no arranque do sistema: uma imagem com todos os dígitos dos pisos, outra com todos esses mesmos dígitos mas agora realçados com uma cor e uma terceira com todas as descrições dos pisos. Mais uma vez, optando por este procedimento, torna-se muito rápida a escrita no ecrã pois são simplesmente feitas cópias de imagens para o *buffer* do controlador LCD. Na estrutura "VertMarqueeDirContent_t" pode-se constatar a presença da variável "posyiWindow", que corresponde à posição da imagem total a partir da qual será escrita/mostrada no ecrã – denominada aqui por janela.

A última estrutura deste conjunto é a "StringContent_t". Esta é distinta das anteriores pelo facto de não ser criada uma imagem no início do programa. Esta é a excepção pois todas as *strings* escritas no ecrã através desta estrutura são elaboradas no momento da sua utilização. A justificação é simples. Esta estrutura foi criada a pensar na escrita dos valores do relógio digital. Ora, como há imensas combinações neste caso, o método anterior não é viável. Posto isto, esta estrutura contém a *font* associada, bem como a respectiva cor pretendida, dois pares de posições, para que a *string* fique compreendida entre esses limites, quer horizontal quer verticalmente, e o incontornável vector de caracteres.

Como há imagens que têm de ser criadas e armazenadas logo no início do programa, pode-se dividir as funções associadas a estas estruturas em dois grupos: nas de inicialização e nas que são chamadas no decorrer do programa.

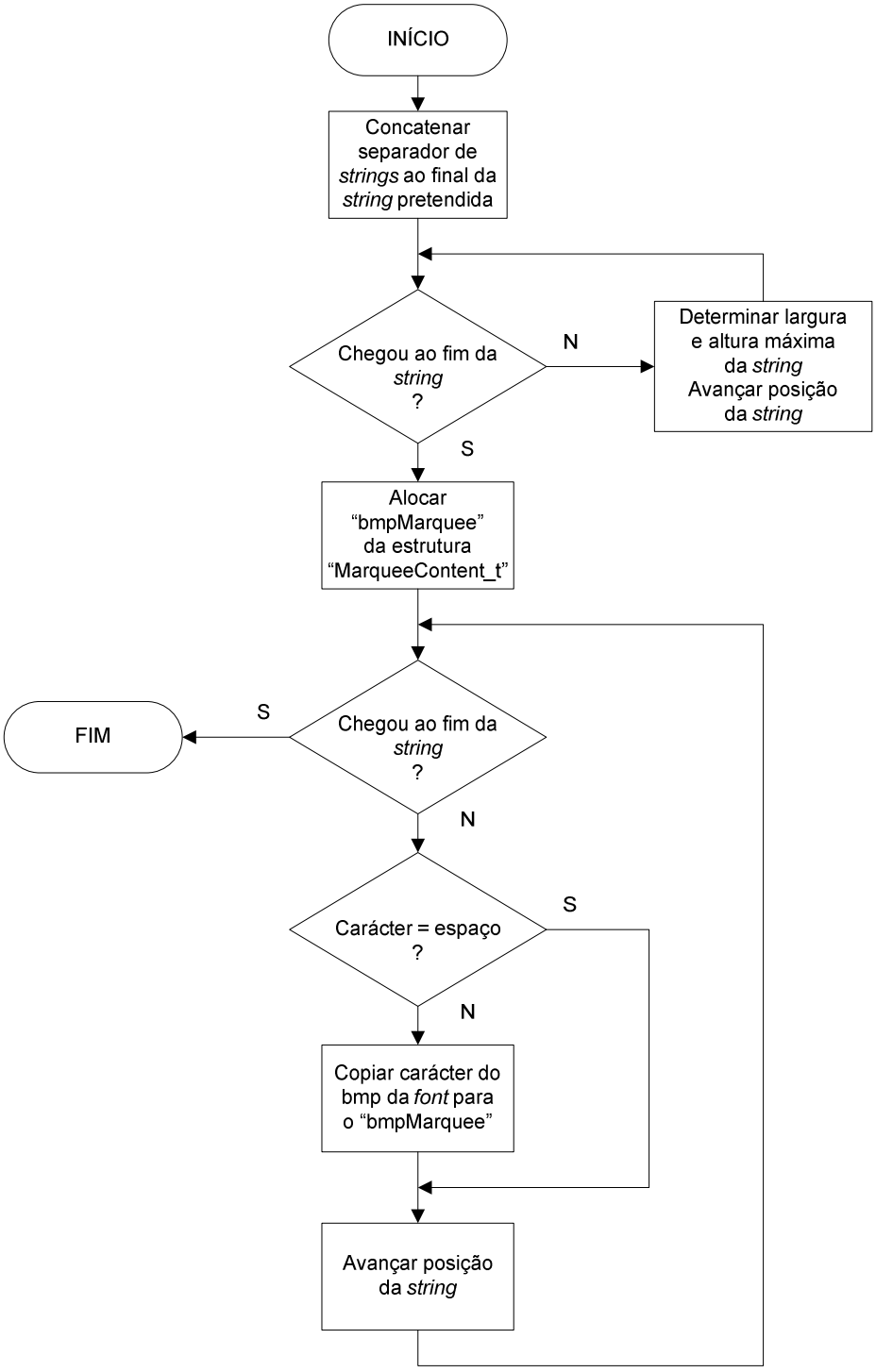

Figura 81 Fluxograma da função "inicializar marquee"

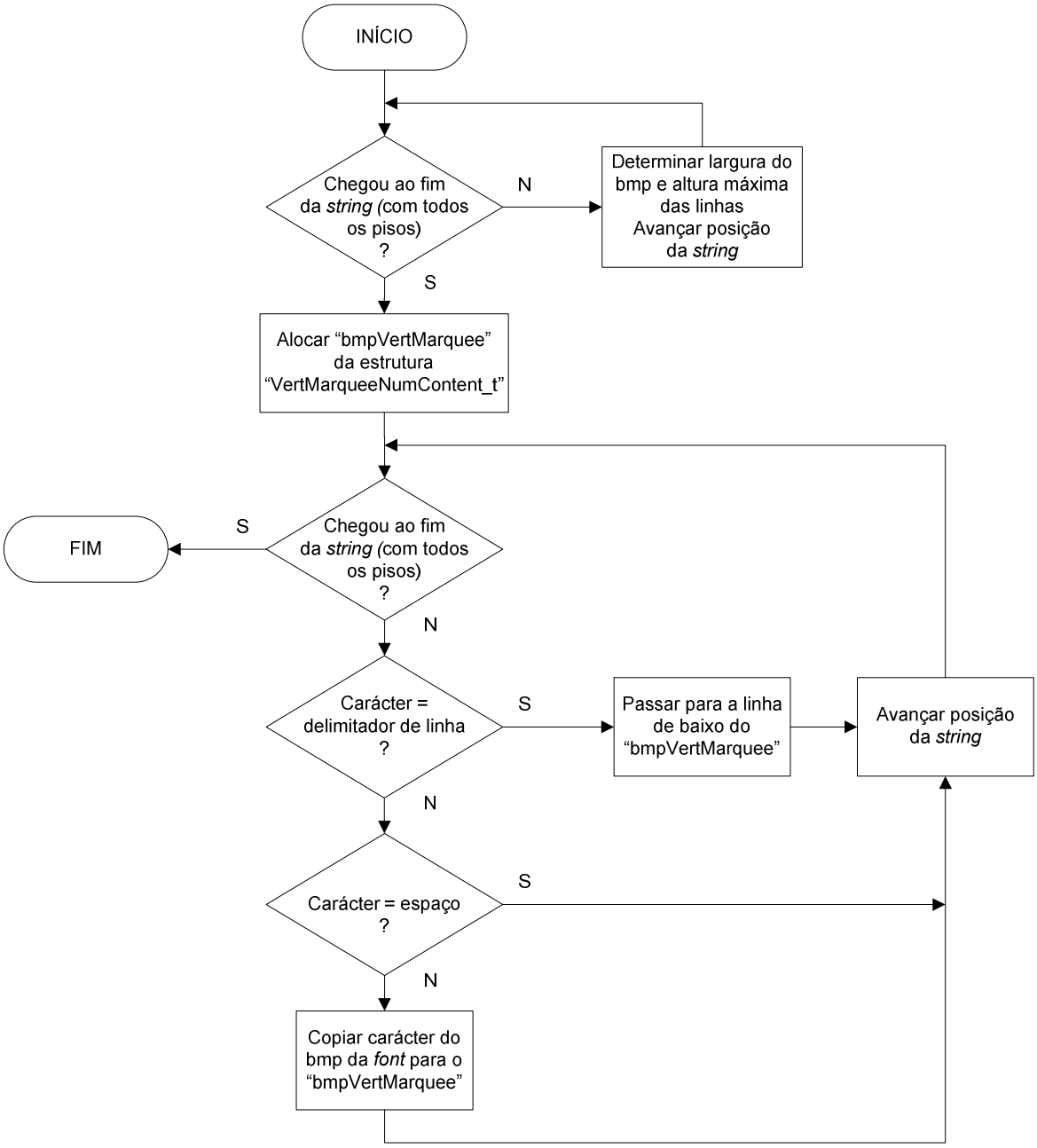

Figura 82 Fluxograma da função "inicializar marquee vertical piso"

No grupo das funções de inicialização está a responsável por inicializar o *marquee* [horizontal] (Figura 81) e a função que inicializa o *marquee* vertical correspondente à mostragem dos pisos no ecrã (Figura 82). Dentro deste grupo ainda estão presentes mais duas funções, em que a primeira é chamada dentro da segunda: a função que inicializa o *marquee* vertical relativo à parte do directório que irá ser mostrada realçada com outra cor (Figura 83), e a função que inicializa o *marquee* vertical completo do directório (Figura 84).

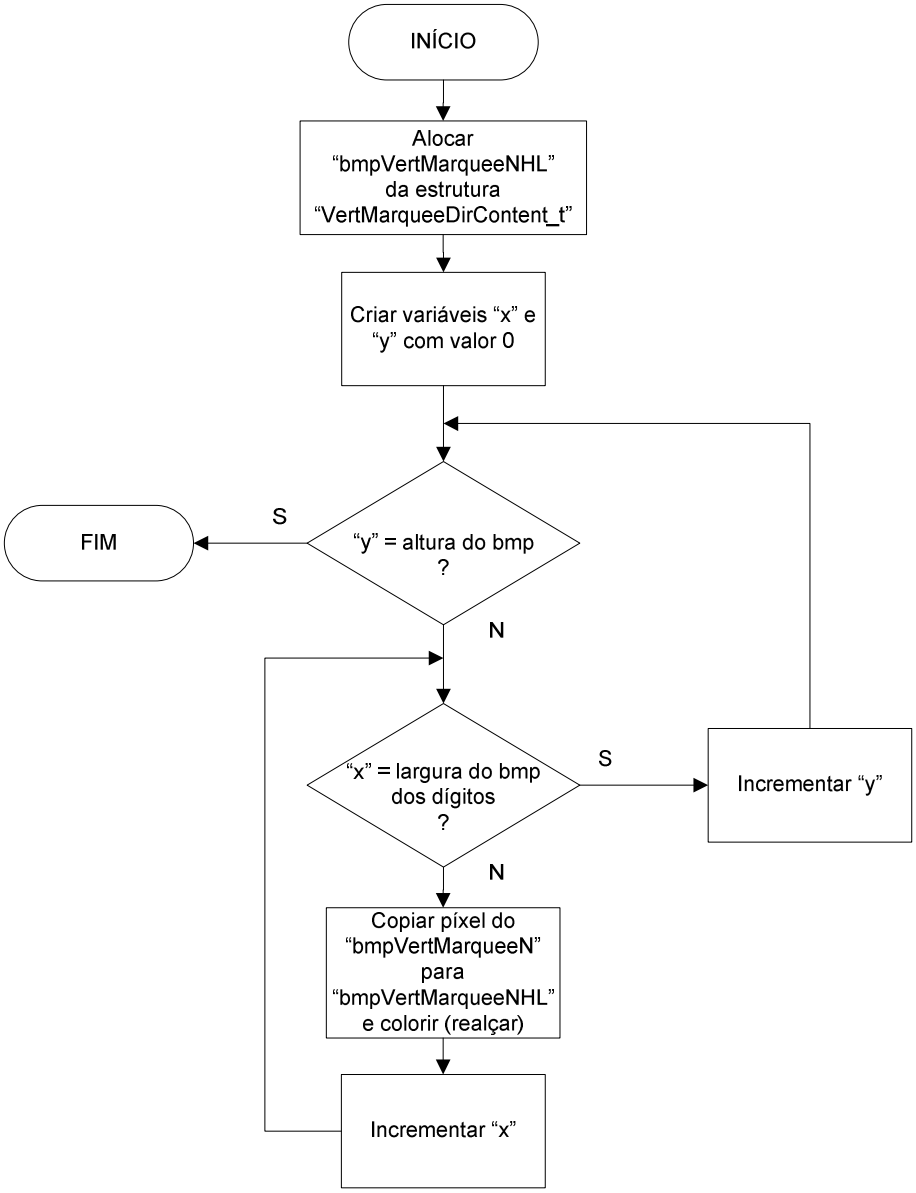

Figura 83 Fluxograma da função "inicializar marquee vertical directório highlight"

Figura 84 Fluxograma da função "inicializar marquee vertical directório"

O grupo das funções que são executadas já no decorrer do programa é constituído por outros 4 elementos. A primeira função diz respeito à escrita no ecrã do *marquee* horizontal

(Figura 85). O objectivo da segunda – escrita do *marquee* vertical relativo ao piso – é muito simples: copiar o "bmpVertMarquee" da estrutura "VertMarqueeNumContent_t" para o *background buffer* desde a posição "posyi" até uma altura da linha que representa o piso. O conteúdo da terceira função não é muito diferente da anterior. Há a acrescentar duas nuances: nesta função é necessário copiar dois bmps, um relativo aos dígitos dos pisos e outro correspondente à respectiva descrição dos mesmos. De notar que estas imagens têm de estar, logicamente, lado a lado para se fazerem corresponder. Depois há a questão do *highlight* (realce do piso onde o elevador está a passar no momento). O método é copiar posteriormente o "bmpVertMarqueeNHL" a partir da posição indicada pela função que chama esta. Por último, está representado o fluxograma da função "printf no ecrã" na Figura 86.

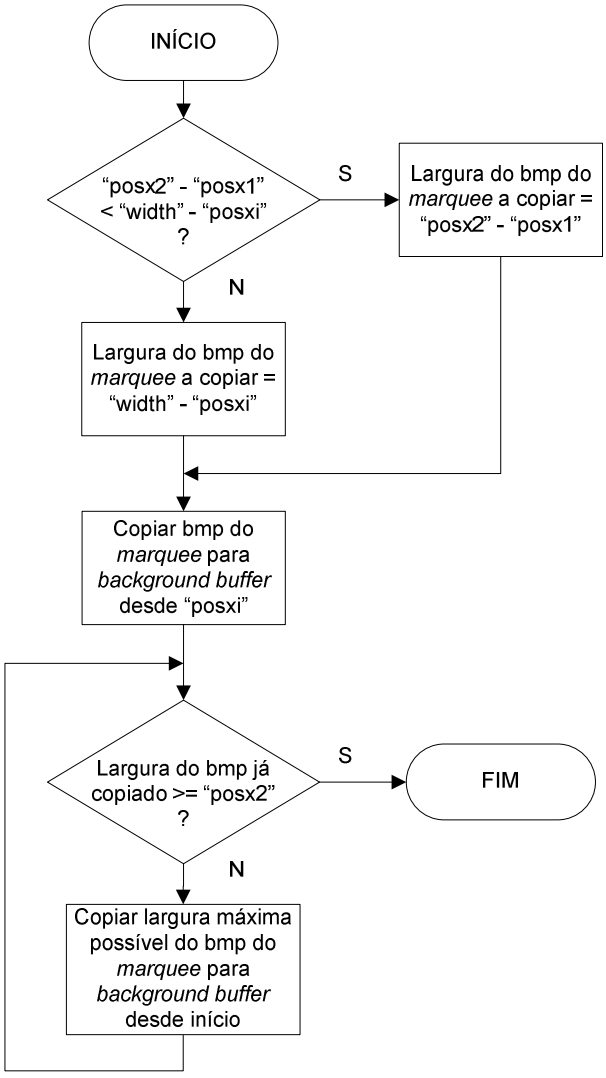

Figura 85 Fluxograma da função "marquee no ecrã"

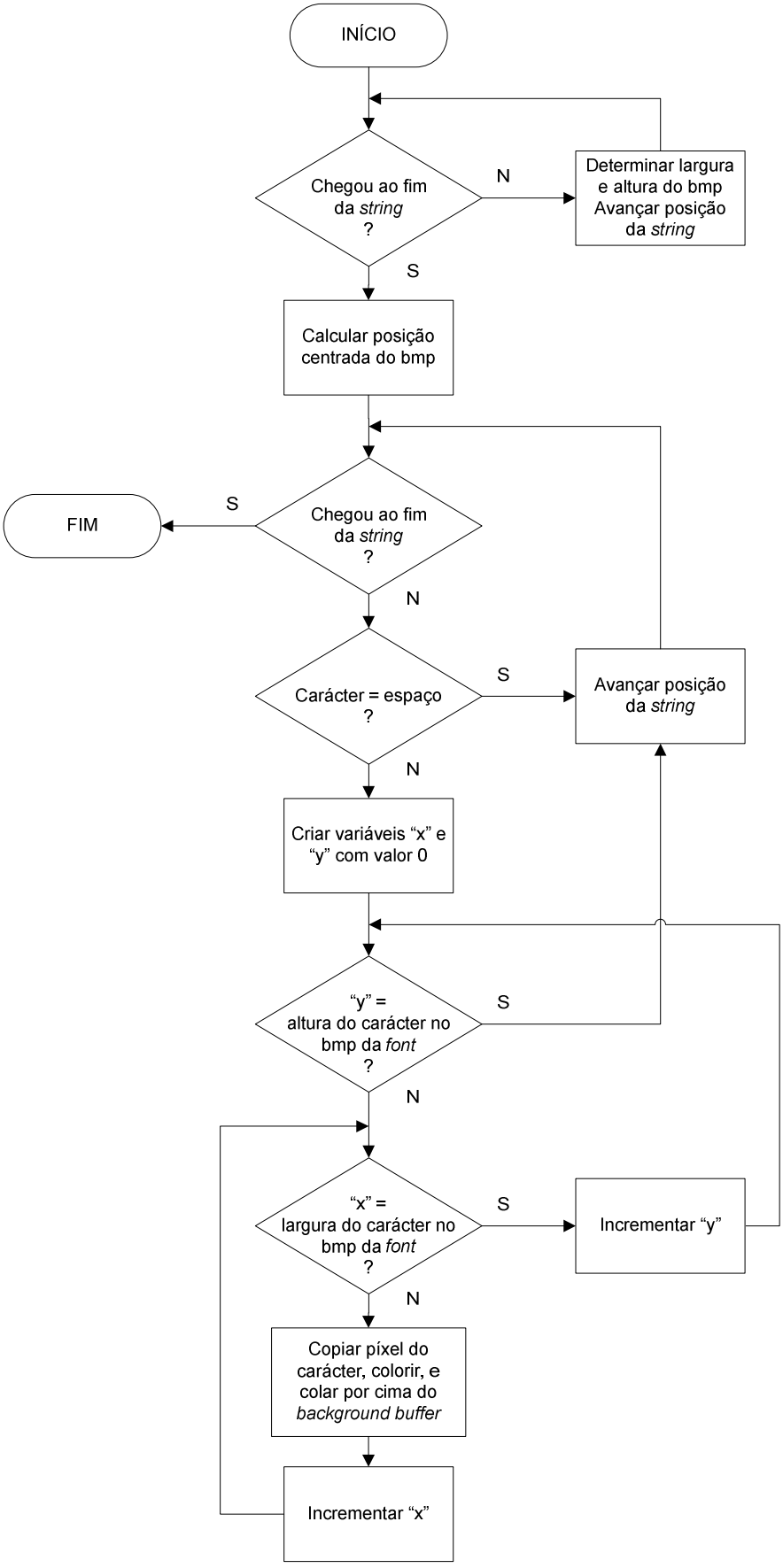

Figura 86 Fluxograma da função “printf no ecrã”

5.1.3. DADOS RECEBIDOS VIA RS485

No que diz respeito à transmissão dos dados via RS485, estão presentes 2 funções fulcrais. A primeira encarrega-se por inicializar a comunicação, ou seja, por configurar uma USART disponível em modo RS485, sem bit de paridade, com 1 stop bit e com 9 bits de dados. A comunicação é efectuada em 9 bits para que o 9º bit sirva de *trigger* de sincronismo, isto é, o sistema vai lendo as tramas da mensagem recebidas até que o 9º bit de uma trama seja '1' e, assim, ser considerada essa mensagem.

A outra função é responsável pelo processamento dos dados recebidos, sendo estes armazenados na rotina de interrupção correspondente.

Para evitar erros de transmissão, dentro de um compromisso entre eficácia e eficiência, é utilizado o método *Cyclic Redundancy Check* (CRC), com polinómio de 8 bits. Portanto, a primeira preocupação quando são processados os dados recebidos via RS485 é verificar o código CRC da mensagem.

A mensagem, para além do respectivo código CRC, é constituída por 2 tramas de 1 byte cada. A primeira é atribuída inteiramente ao binário do piso, enquanto a segunda contém as informações relativas à situação de cada seta, ao movimento e à mensagem de alerta. Cada uma destas informações é representada por 1 bit.

Posto isto, a função que processa os dados recebidos decompõe estas duas tramas em três, isto para facilitar depois a utilização destes dados na camada da aplicação, propriamente dita. Informações do estado de cada seta e do movimento num byte e a mensagem de alerta a ser mostrada noutro byte. Esta função tem outro objectivo: informar a função que a chama se estas tramas descritas são iguais às anteriores ou não.

5.1.4. RELÓGIO

Para que o programa faça a gestão do relógio digital mostrado no ecrã existem 5 funções principais a serem abordadas.

A primeira diz respeito à inicialização do relógio.

Como já foi referido anteriormente, o funcionamento inerente do relógio está ao cargo de um *timekeeper* cuja comunicação com o µC é em SPI.

Com isto, o primeiro objectivo da função inicializar é configurar devidamente essa comunicação SPI. De seguida são complementadas as inicializações com um teste fulcral. Através de um registo de controlo integrado no *timekeeper* é testado se a respectiva bateria foi descarregada. Consoante o resultado há 2 caminhos: se a bateria não se descarregou enquanto o sistema esteve desligado, há que verificar o *Daylight Saving Time* (DST) *status* e fazer a devida actualização no relógio, se necessário. O DST *status* – assim designado neste trabalho – é um dos registos livres para utilização integrados no *timekeeper*, que é utilizado da seguinte forma: '0' simboliza que está num estado indefinido; '1' corresponde ao horário de Verão e '2' ao horário de Inverno. Se o resultado der que a bateria entretanto se descarregou, o procedimento é colocar um '0' no DST *status*, pois o relógio fica num estado indefinido, e "limpar" a *flag* que indica que o *timekeeper* ficou sem alimentação.

As outras 4 funções deste grupo pertencem a um vector de funções. Cada função desse vector é chamada consoante o estado em que o relógio se encontra, conduzido por um contador.

A primeira função corresponde ao estado em que é periodicamente actualizado o relógio no ecrã. É onde é verificado se já passaram 60 segundos e, se for o caso, faz a leitura dos valores do *timekeeper*, verifica se há alterações do DST, processando-as, e manda actualizar o relógio no ecrã.

Na segunda função, responsável por inicializar o modo de acerto do relógio, são novamente lidos os dados do *timekeeper*, mas desta vez são tratados de forma que, através dos botões físicos, estes possam ser alterados.

Em relação à terceira função, o procedimento é o seguinte. Processa os valores alterados pelos botões e manda copiar para o *buffer* do ecrã os valores do relógio em modo de acerto (que ainda não foram alterados no *timekeeper*). Para quem está a acertar o relógio saber quais os dígitos (ou dia da semana) estão a ser alterados, estes ficam a piscar de meio em meio segundo (valor por defeito).

A quarta e última função é a que, efectivamente, altera os valores do relógio no *timekeeper*. Para tal, processa os valores que foram alterados em modo de acerto e envia para o *timekeeper*. Para além disto, actualiza também o DST *status*.

Mas como, afinal, é feita a passagem de uma função para a outra? A cada botão está associada uma rotina de interrupção que, após passar por um filtro digital, desencadeia o modo de acerto do relógio. Bem como também faz sair desse modo. Com a ajuda de temporizações, é nestas rotinas de interrupção que são feitos os filtros digitais, mas também o incremento ou decremento dos dígitos e a mudança dos dígitos a alterar (que piscam). Consoante o que o utilizador pretender, dentro destas rotinas de interrupção é alterado o valor do contador que faz chamar cada função do vector de funções referente ao relógio. Como foi esquematizado no início da descrição da camada da aplicação, a função deste vector a ser executada é chamada no ciclo principal deste programa.

5.1.5. ACTUALIZAÇÃO DOS CONTEÚDOS POR UMA USB FLASH DRIVE

É possível actualizar os conteúdos que são lidos do cartão SD/MMC e carregados logo no início do programa através de uma USB *flash drive* (ou *pen drive*), nomeadamente os pisos, respectivas descrições, etc. Para tal, está integrado no sistema um componente – o Vinculum – que, comunicando com o µC, faz a devida tradução da informação contida na *flash drive* para dados passíveis de serem lidos directamente pelo programa.

O método utilizado a nível do *software* tem traços semelhantes do caso da sub-subsecção anterior. Isto é, está presente outro vector de funções que se encarrega do processamento da actualização, se for o caso.

Mas tal como no caso anterior, há uma função que inicializa este processo, que prepara o programa para uma eventual actualização que possa ser pedida. A comunicação com o Vinculum é efectuada via USART. Portanto, na função de inicialização é configurada mais uma USART que esteja disponível. Esta configuração é definida em modo de *hardware handshaking*, pois é requerido pelo Vinculum, sem bit de paridade, 1 stop bit e 8 bits de dados. Uma nota a acrescentar: pela indicação do *datasheet* do µC, neste modo e especialmente na recepção, é imprescindível recorrer-se ao *Peripheral DMA Controller* (PDC) associado à USART. Por isso também foi configurado dentro desta função. No final ainda é feito um teste de sincronização.

A comunicação com o Vinculum é feita através de comandos. Se uma *flash drive* é inserida ou retirada do sistema ele envia uma indicação ao µC; se se pretender ler o conteúdo de um ficheiro contido na *flash drive*, tem de ser enviado um comando específico para depois o Vinculum devolver os dados presentes no interior desse ficheiro, e assim para outros casos.

Para facilitar a gestão destas trocas e respectivos processamentos, o programa tem dois modos: de comando e de dados. Isto é extremamente útil pois em modo de dados o programa espera pelo número de bytes transmitido anteriormente, e não tem de se preocupar em verificar se recebeu algum comando válido.

A primeira abordagem aos dados recebidos é efectuada pela rotina de interrupção associada ao PDC da USART. É aqui que os bytes são armazenados num vector dinâmico e verificado se são recebidos comandos válidos ou conteúdos de ficheiros. Para cada um dos dois casos é despoletado um procedimento diferente. No primeiro caso, se for recebido um conjunto de comandos esperados pelo programa, é iniciado o processo de actualização dos conteúdos, que será de seguida explicado na abordagem às funções que constituem o referido vector de funções. No caso de se tratar da recepção do conteúdo de um ficheiro, após a respectiva finalização, o processo de actualização continua para o passo seguinte.

O vector de funções que se responsabiliza, efectivamente, pela actualização dos conteúdos é composto por 7 elementos. Cada função é chamada na rotina principal do programa consoante o valor de um contador agregado a este vector.

A primeira função chama-se “inactiva”, isto é, se nenhuma *pen drive* tiver sido inserida no sistema não há qualquer processamento a este nível. Naturalmente, esta é a função que é chamada por defeito.

A próxima função que poderá ser chamada no decorrer do funcionamento do sistema é a “início da actualização”. Aqui, como o próprio nome sugere, são feitos os preparativos para a actualização, tal como “pedir” para mostrar no ecrã uma imagem a informar que o sistema está em actualização de conteúdos, não esquecendo de incrementar o contador referente ao vector de funções.

A terceira função tem como objectivo, através de envio de comandos, entrar num directório dentro da *pen drive* – por defeito é o “advert” – e listar os ficheiros presentes nesse directório. E incrementar o contador.

Na função seguinte é verificado se na lista consta o ficheiro com o nome pré-definido que contém as informações da aplicação. E é incrementado o contador.

A quinta função pede, por comandos mais uma vez, para que o Vinculum envie todo o conteúdo do ficheiro informativo e altera o modo para o modo de dados. Assim, como foi

referido acima, quando todo o conteúdo for recebido, a rotina de interrupção chama uma função que se encarrega de actualizar no cartão SD/MMC o ficheiro informativo e volta a mudar para o modo de comandos. Enquanto isso, esta quinta função fica a aguardar, sem encravar o programa, pela finalização deste processo para, agora sim, incrementar outra vez o contador depois de sair do directório “advert”.

Um parêntese: em qualquer uma destas funções anteriormente citadas, se ocorrer algum erro, é atribuído ao contador o número correspondente à sexta função (penúltima) e retornado um erro.

A sexta função “pede” para mostrar no ecrã uma imagem, que pode ser de conteúdos não actualizados ou de conteúdos actualizados, consoante tenha sido devolvido um *feedback* de ocorrência de erro ou não, respectivamente. É também activada uma temporização de 2 s para que dê tempo para o utilizador possa visualizar a informação contida na imagem, bem como o incremento do contador.

A sétima e última função testa se essa temporização já chegou ao fim. Se sim, faz outro teste: se a actualização foi bem sucedida, provoca um *reset* (através do *Watchdog Timer* (WDT)) no sistema para que este inicie já com os novos conteúdos; se ocorreu algum erro, manda colocar a imagem no ecrã que estava antes de se iniciar o processo da actualização de conteúdos. Em ambos os casos é, previsivelmente, atribuído ao contador o valor correspondente à função “inactiva”.

6. CONCLUSÕES

Ao longo deste texto foram sendo apresentadas conclusões que permitiram sustentar as opções de desenvolvimento efectuadas ao longo do projecto. Assim, neste último capítulo é realizada uma síntese das principais conclusões, consequências e relevância do trabalho realizado e perspectivados futuros desenvolvimentos.

Sob pena de repetição, é inevitável voltar a recordar a motivação que esteve na base deste trabalho. O tamanho excessivo de um computador, que é o núcleo de um sistema já implementado no mercado dos elevadores pela empresa Francisco Parracho – Electrónica Industrial, Lda., aliado ao elevado custo total do próprio sistema, fez com que esta empresa apostasse num projecto cujo intuito era tornar o produto final com dimensões e custos bem mais reduzidos face ao referenciado. A solução passou por projectar um sistema embebido para controlar ecrãs LCD, nomeadamente os TFT-LCD. Assim, direccionando as funcionalidades única e exclusivamente para este propósito, conseguiu-se cumprir o objectivo principal para este projecto.

De sublinhar que a área para o qual este produto foi desenvolvido é o mercado dos elevadores. Isto significa que este projecto foi influenciado pelas práticas habituais desta área, como por exemplo o nível da tensão de alimentação. Contudo, é perfeitamente possível que este sistema embebido seja aplicado noutra área [de negócios], desde que o

hardware seja total ou parcialmente coincidente com os requisitos. A flexibilidade do *software* desenvolvido permite esse transporte para outras situações.

Este trabalho foi dividido em duas fases. A primeira consistiu, com a ajuda de um kit de desenvolvimento, na validação dos *drivers* dos controladores e periféricos base deste projecto. Houve dificuldades nesta fase pois muitos dos *drivers* disponíveis pela ATMEL, marca do micro-controlador (µC) utilizado, não foram desenvolvidos especialmente para este kit de desenvolvimento, ou mesmo para este µC específico. Tiveram de ser devidamente alterados os *drivers* de um outro µC da ATMEL com a mesma arquitectura. A segunda fase foi responsável pela elaboração da placa de circuito impresso dedicada para este projecto, e pelo desenvolvimento da camada da aplicação cujos frutos são visíveis pelo utilizador do elevador. Os obstáculos nesta fase não foram tão acentuados como na primeira, porém, exigiu muito desenvolvimento.

Como este projecto foi patrocinado pela empresa supracitada, os requisitos do sistema foram impostos por esta. Requisitos esses que foram cumpridos e validados pelo orientador deste trabalho na empresa.

Contudo, pelas potencialidades oferecidas pelo µC utilizado, o leque de possíveis desenvolvimentos futuros é vasto. Sendo o âmbito deste projecto os elevadores, é nesta área que são perspectivados os desenvolvimentos futuros, nomeadamente os que foram discutidos com o orientador da empresa. Uma possível funcionalidade que poderia ser integrada neste sistema seria um sintetizador de voz. Seria muito útil especialmente para os utilizadores invisuais do elevador. Outra possibilidade é receber os dados de uma sonda meteorológica presente no exterior do edifício, processá-los e mostrar os respectivos valores no ecrã. Para completar o lote dos possíveis desenvolvimentos que estarão mais próximos de serem realizados, há que acrescentar a capacidade táctil que o ecrã poderia ter. Entre os outros eventuais desenvolvimentos futuros está a ligação de rede. Com isto, o sistema poderia ser controlado remotamente, porém, requeria um meio de transmissão, com todo custo e/ou dificuldade de instalação associados. Para terminar, há ainda outro desenvolvimento que poderia ser efectuado, dentro daqueles menos viáveis: transmissão no ecrã de um canal televisivo. A viabilidade é a mesma do caso anterior, até porque a consequência é a mesma.

Referências Documentais

[1] Atmel, AVR32 AP7000 datasheet, disponível em http://www.atmel.com/dyn/resources/prod_documents/doc32003.pdf, consultado em 13 Março 2011

[2] Sharp, LQ104S1DG61 Product Specifications, disponível em http://document.sharpsma.com/files/LQ104S1DG61_SS_060807a.pdf, consultado em 13 Março 2011

[3] Atmel, Mature NGW100 Network Gateway Kit, disponível em http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4102, consultado em 13 Março 2011

[4] IN-CIRCUIT GmbH, ICnova AP7000 OEM, disponível em http://www.ic-board.de/product_info.php?info=p79_ICnova-AP7000-OEM.html&XTCsid=09ebb8eee1d0a040d7252203dcbe2e7b, consultado em 13 Março 2011

[5] Portal São Francisco, Elevador, disponível em http://www.portalsaofrancisco.com.br/alfa/elevador/elevador-7.php, consultado em 13 Março 2011

[6] Prof2000, Display de 7 segmentos, disponível em http://www.prof2000.pt/users/lpa/display de 7 segmentos.ppt, consultado em 13 Março 2011

[7] Kioskea, Ecrã plano, disponível em http://pt.kioskea.net/contents/pc/ecran-plat.php3, consultado em 13 Março 2011

[8] Electrónica, Tecnologia LCD, disponível em http://www.electronica-pt.com/index.php/content/view/247/37/, consultado em 13 Março 2011

[9] Baixaki, O futuro da imagem: telas OLED, disponível em http://www.baixaki.com.br/tecnologia/2486-o-futuro-da-imagem-telas-oled.htm, consultado em 13 Março 2011

[10] TDK-Lambda, CXA-0454 Specifications, disponível em http://www.tdk-lambda.com/products/sps/catalog/eng/etl_cxa_0454.pdf, consultado em 13 Março 2011

[11] EE Herald, Basics of LCD and it's interface to a microcontroller, disponível em http://www.eeherald.com/section/design-guide/esmod17.html, consultado em 13 Março 2011

[12] Beyond Logic, Interfacing Example – 16 Character x 2 Line LCD, disponível em http://www.beyondlogic.org/parlcd/parlcd.htm, consultado em 13 Março 2011

[13] Beyond Logic, Interfacing Example – Connecting a LCD Module to the RS-232 Port, disponível em http://www.beyondlogic.org/serial/serial3.htm, consultado em 13 Março 2011

[14] fpga4fun.com, Text LCD module, disponível em http://www.fpga4fun.com/TextLCDmodule.html, consultado em 13 Março 2011

[15] UST Research, 8051 + LCD + EPROM, disponível em http://www.ustr.net/lcd001.shtml, consultado em 13 Março 2011

[16] componentkits.com, EDE702 Serial LCD Interface IC, disponível em http://www.componentkits.com/dslibrary/EDE702.pdf, consultado em 13 Março 2011

[17] Best-microcontroller-projects.com, A PIC Serial LCD project, disponível em http://www.best-microcontroller-projects.com/serial-lcd.html, consultado em 13 Março 2011

[18] Hirose Electric Co., Ltd., DF9 Series, disponível em http://www.hirose.co.jp/cataloge_hp/e54000041.pdf, consultado em 13 Março 2011

[19] Instituto Superior Técnico – Universidade Técnica de Lisboa, Visor Matricial Bidimensional de LEDs RGB, disponível em https://dspace.ist.utl.pt/bitstream/2295/575362/1/Dissertacao.pdf, consultado em 13 Março 2011

[20] Data Display Teknoloji, PrismaECO II, disponível em http://www.datadisplay.com.tr/en/Components/TFTControllerBoards/RGBConverter/PRISMAecoII.html, consultado em 13 Março 2011

[21] Data Display Teknoloji, Datasheet PrismaECO II, disponível em http://www.datadisplay.com.tr/PDF/Boards/PRISMAecoII_Rev1.7_09.12.2009.pdf, consultado em 13 Março 2011

[22] Nexus 5001 Forum, disponível em http://www.nexus5001.org/, consultado em 13 Março 2011

[23] EGS elevadores LTDA., Indicadores de posição, disponível em http://www.egselevadores.com.br/indicadoresposicao.pdf, consultado em 13 Março 2011

[24] e-motive display, LED Display – MS Series, disponível em http://www.emotivedisplay.com/ourProductsDetail.asp?productId=4, consultado em 13 Março 2011

[25] e-motive display, LED Display – CD/UD Series, disponível em http://www.emotivedisplay.com/ourProductsDetail.asp?productId=2, consultado em 13 Março 2011

[26] e-motive display, LED Display – PT Series, disponível em http://www.emotivedisplay.com/ourProductsDetail.asp?productId=3, consultado em 13 Março 2011

[27] e-motive display, Monochrome Display, disponível em http://www.emotivedisplay.com/ourProductsDetail.asp?productId=15, consultado em 13 Março 2011

[28] e-motive display, Evolution Display – Evolution S1 Series, disponível em http://www.emotivedisplay.com/ourProductsDetail.asp?productId=8, consultado em 13 Março 2011

[29] e-motive display, Evolution Display – Evolution S2 Series, disponível em http://www.emotivedisplay.com/ourProductsDetail.asp?productId=9, consultado em 13 Março 2011

[30] FTDI Chip, VNC1L – Vinculum USB Host Controller Device, disponível em http://www.ftdichip.com/Products/ICs/VNC1L.htm, consultado em 13 Março 2011

[31] FTDI Chip, UB232R Datasheet, disponível em http://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_UB232R.pdf, consultado em 13 Março 2011

[32] National Semiconductor, LM2675, disponível em http://www.national.com/ds/LM/LM2675.pdf, consultado em 13 Março 2011

[33] National Semiconductor, LM2676, disponível em http://www.national.com/ds/LM/LM2676.pdf, consultado em 13 Março 2011

[34] EURO circuits, disponível em http://www.eurocircuits.com/, consultado em 13 Março 2011

[35] Hirose Electric Co., Ltd., DF12 Series, disponível em http://www.hirose.co.jp/cataloge_hp/e53700036.pdf, consultado em 13 Março 2011

[36] National Semiconductor, LM2674, disponível em http://www.national.com/ds/LM/LM2674.pdf, consultado em 13 Março 2011

[37] DALLAS semiconductor MAXIM, DS3234, disponível em http://pdfserv.maxim-ic.com/en/ds/DS3234.pdf, consultado em 13 Março 2011

[38] Panasonic, Electric Double Layer Capacitor, disponível em http://industrial.panasonic.com/www-cgi/jvcr13pz.cgi?E+PZ+3+ABC0021+EECRG0V105H+7+WW, consultado em 13 Março 2011

[39] Texas Instruments, TPS77601, disponível em http://focus.ti.com/lit/ds/symlink/tps77601.pdf, consultado em 13 Março 2011

[40] ON Semiconductor, NTR1P02LT1, disponível em http://www.onsemi.com/pub_link/Collateral/NTR1P02LT1-D.PDF, consultado em 13 Março 2011

[41] Linear Technology, LT1785, disponível em http://cds.linear.com/docs/Datasheet/178591fc.pdf, consultado em 13 Março 2011

[42] KYOCERA ELCO, 5738 Series, disponível em http://www.kyocera-elco.com/prdct/type/memory/5738.html, consultado em 13 Março 2011

[43] FTDI Chip, VNC1L Datasheet, disponível em http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_VNC1L.pdf, consultado em 13 Março 2011

[44] FTDI Chip, FT232R, disponível em http://www.ftdichip.com/Products/ICs/FT232R.htm, consultado em 13 Março 2011

[45] ELM by ChaN, FatFs Generic FAT File System Module, disponível em http://elm-chan.org/fsw/ff/00index_e.html, consultado em 13 Março 2011

[46] Atmel, AVR32 Studio, disponível em http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4116, consultado em 13 Março 2011

[47] Atmel, AVR JTAGICE mkll, disponível em http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3353, consultado em 13 Março 2011

[48] Atmel, AVR32 GNU Toolchain, disponível em http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4118, consultado em 13 Março 2011

[49] Atmel, Application Notes, disponível em http://www.atmel.com/dyn/products/app_notes.asp?family_id=607#Application_Example_and_Algorithms, consultado em 13 Março 2011

[50] AVR freaks, “NGW100 Setup Code (Updated)”, disponível em http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=71708, consultado em 13 Março 2011

[51] AVR freaks, “disable caching, but how?”, disponível em http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=70768&highlight=cache+lcd, consultado em 13 Março 2011

[52] AngelCode, Bitmap Font Generator, disponível em http://www.angelcode.com/products/bmfont/, consultado em 13 Março 2011

Anexo A. *Printed Circuit Board* (PCB)