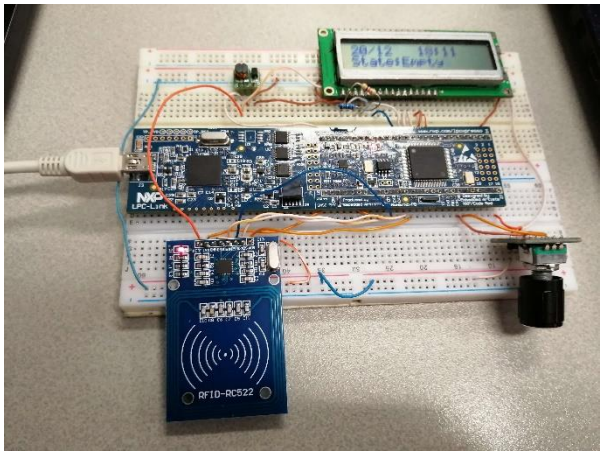
Sistemas Embebidos

Licenciatura em Engenharia Eletrónica e Telecomunicações e de Computadores

Sistema de gestão de armário com fechadura RFID



Grupo 6

44658 Rúben Castanho

47124 Carlos Ribeiro

1º Semestre letivo 2023/2024

22 de dezembro de 2023

Resumo

O sistema tinha como previsto um funcionamento autónomo com dois modos distintos de funcionamento (modo normal e de manutenção) e que fosse um sistema autónomo, isto é que funcionasse automaticamente assim que fornecesse energia.

O modo normal seria o que iria ser usado para inserir ou retirar entregas do cacifo, o modo de manutenção como o nome indica seria usado para ajustar calendário, horário e os cartões autorizados e também guarda uma coleção de registos de interações.

Na realização deste projeto todos os objetivos foram atingidos exceto a utilização completa da memória nao volátil, isto aconteceu pois quando tentámos implementar este módulo final, encontrámos erros que nos encapacitaram de testar e implementar completamente a capacidade de guardar e recolher os dados da memória, tudo isto será descrito numa secção deste relatório, no capítulo IV.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo I. Introdução

I.1. Objetivo

No âmbito da Unidade Curricular de Sistemas Embebidos, pretende-se criar um sistema autónomo de gestão de um armário para depósito e recolha de objetos com fechadura RFID.

A gestão será feita através de um microcontrolador LPC1769 associado à placa de desenvolvimento LPCXpresso.

I.2. Descrição do Projeto

O projeto é constituido por vários elementos conforme a **figura 1**;

Um leitor de cartões [MFRC522], que permite a leitura de cartões *contactless*, usando o protocolo de identificação MIFARE.

Um *Liquid Crystal Display* (LCD) [NMTC-S16205DRGHS] , que permite transmitir qualquer tipo de informação ao utilizador, dispondo de duas linhas de 16 caracteres cada;

Um *Rotary Encoder Switch* [KY-40], que não só serve de botão como permite obter informação sobre a direção da rotação do seu eixo;

O trinco é simulado com o LED2 do LPCXpress.

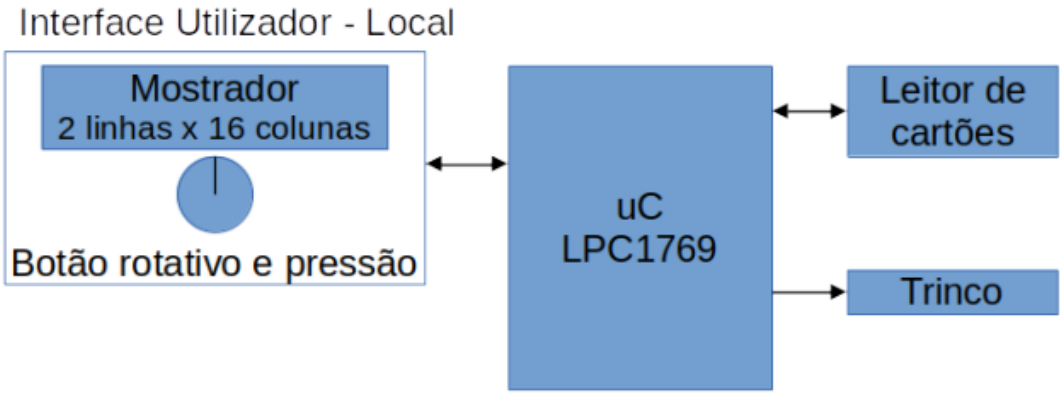


Figura 1 – Diagrama de Blocos.

O sistema irá ter dois modos de funcionamento, normal e de manutenção. No modo normal, a máquina fica a mostrar no visualizador a data e hora e o estado do armário (ocupado ou vazio), e promove a passagem de cartões, caso o cartão esteja autorizado permite a inserção ou remoção de objetos, se um cartão nao estiver na lista de autorizados, automaticamente o mostrador indica a falta de autorização. Também é verdade que só um cartão poderá ir buscar a

encomenda, então quando o armário se encontra ocupado só o cartão escolhido (da lista de autorizados) poderá abrir o armário e retirar a encomenda, neste cenário mesmo um cartão da lista de autorizados irá ser "não autorizado" caso não teja sido escolhido.
Todas estas interações são guardados nos registos que podem ser visualizados no modo de manutenção, os registos mostram a data e hora, o cartão e o estado do armário quando o registo foi feito, e dada a estrutura de dados escolhida também guardamos o estado de autorização do cartão passado.

O modo de manutenção permite a alteração da data e da hora, usando o botão rotativo, permite a adição e remoção de cartões da lista de autorizados, e permite também a visualização dos registos. Para entrarmos neste modo é usado um *doubleclick* e para sairmos do mesmo é usada a opção do menu "*Back*".

O microcontrolador está implementado numa placa de desenvolvimento, onde os seus pinos são ligados conforme assinalado na **figura 2**.

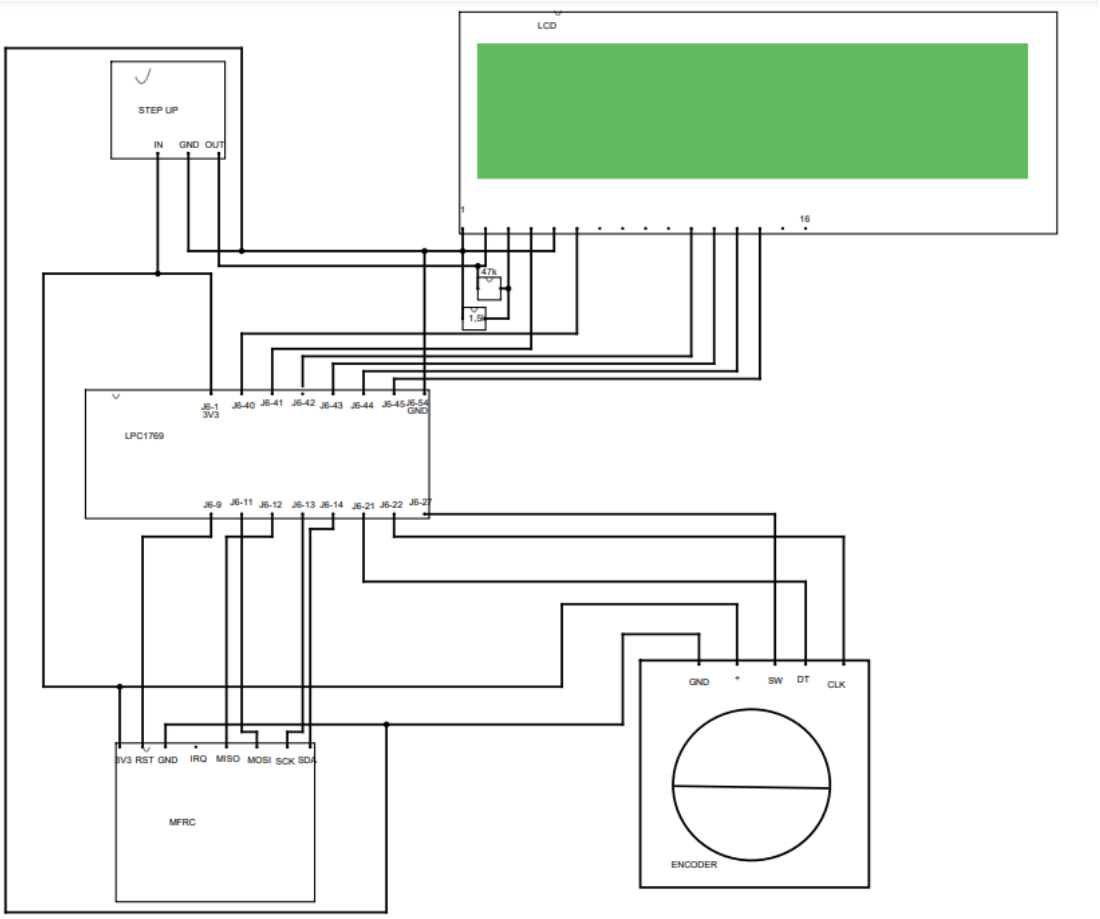


Figura 2- Esquema elétrico da ligação da placa de desenvolvimento aos periféricos usados nesta UC.

I.3. Microcontrolador LPC1769

O microcontrolador usado nesta Unidade Curricular e neste projeto têm um processamento de 32 bits. Mas também é relevante saber que pode processar informação a 16 bits, isto é, usar o conjunto de instruções Thumb-2, pode também endereçar um espaço de 2^{32} bytes de mémoria, isto é, 4 Gigabytes.

Contêm também uma memória flash (isto é memória não volátil) de 512KB e 64KB de RAM (memória volátil). Dentro ainda deste espaço de enderaçamento encontramos ainda os vários dispositivos periféricos, cada um com o seu endereço.

Entre vários dispositivos vamos apenas mencionar aqueles usados neste trabalho e nesta Unidade Curricular:

- Os pinos de entrada e saída de uso geral e configuráveis ou GPIO;
- O Real Time Clock ou RTC;
- Temporizadores ou Timers;
- Serial Peripheral Interface ou SPI.

A sua aplicação e descrição de cada um destes elementos séra feita mais adiante neste relatório.

I.4. Panorâmica

Este relatório é constituído por quatro capítulos, cujos conteúdos se discriminam de seguida.

- No Capítulo I é feita a introdução e uma breve descrição de todo o projeto.
- No Capítulo II será feita uma descrição da camada HAL.
- No Capítulo III será feita uma descrição da camada DRIVERS.
- No Capítulo IV será abordada a Aplicação.

I.5. Descrição das *Layers* do sistema embebido

Como referido na secção anterior irá ser descrito a *layer* de HAL e DRIVERS nos capítulos II e III, respetivamente. Mas antes disto é importante estabelecer a descrição das outras *layers* que compõe o sistema embebido usado e montado na duração desta UC, a **figura 3** ilustra as layers pela sua ordem.

- Aplicação – a *layer* mais "acima", isto é a layer que irá fazer uso da biblioteca MFRC522 e da biblioteca DRIVERS. É onde se assenta o código do nosso projeto.
- MFRC522 – biblioteca que faz uso de funções "preparatórias" da biblioteca DRIVERS, e lida com o periférico leitor de cartões [MFRC522].
- DRIVERS – biblioteca que contêm as funções para serem usadas na *layer* de Aplicação, esta biblioteca usará funções da HAL, esta biblioteca não deve fazer ou chamar nada que esteja dependente do microcontrolador a ser usado, pois a biblioteca DRIVERS é suposta ser "universal", isto é, se mudássemos de microcontrolador bastaria mudar a HAL e tudo o resto funcionaria.
- HAL – biblioteca que lida com o *hardware* esta biblioteca faz uso da biblioteca CMSIS, esta biblioteca é a unica que deveria depender do microcontrolador a ser usado no momento.
- CMSIS – *Common Microcontroller Software Interface Standard*, conjunto de bibliotecas e APIs que visa simplificar e unificar o desenvolvimento de software para microcontroladores *Cortex*.

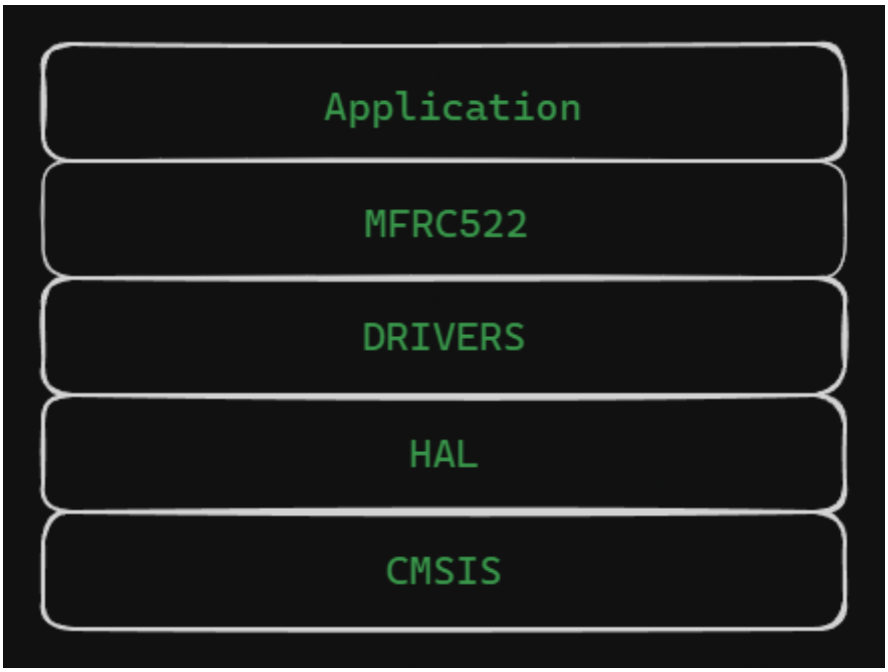


Figura 3 - Esquema das *layers* do sistema embebido

Capítulo II. Hardware Abstraction Layer

II.1. Generalidades

A função desta camada é permitir a utilização de todo o *hardware* através de funções, as quais permitem a abstração do funcionamento e configuração do *hardware*.

Neste capítulo não iremos descrever as funções feitas para usar os vários periféricos mas sim descrever os periféricos.

Apesar de não constar a documentação destas funções neste relatório aqui estão os ficheiros usados na HAL:

- pin.h

  Contém todas as funções relacionadas com atribuição de valores ou direções aos pinos.
- pinconfig.h

  Contém todas as funções relacionadas com a configuração de pinos.
- SYSTICK.h

  Contém todas as funções relacionadas com o *System Tick Timer*.
- timer.h

  Contém todas as funções relacionadas com os *TIMER* do microcontrolador
- rtcHal.h

  Contém todas as funções relacionadas com o *Real Time Clock.*
- spi.h

  Contém todas as funções relacionadas com o *Serial Peripheral Interface*.
- flash.h

  Contém todas as funções relacionadas com a mémoria não volátil.

II.2. Pinos de entrada e saída ou GPIO

Os pinos de entrada e saída estão organizados em portos, em que cada porto é um registo de 32 bits. Cada pino pode ser configurado para funcionar em um de quatro modos, GPIO (*General-Purpose Input/Output*), e as restantes associadas a um dispositivo. Na secção do SPI será identificada a função correspondente a cada pino.

O porto 0 é constituído por 31 bits[30:0], o porto 1 por 32 bits [31:90], o porto 2 por 14 bits [13:0], o porto 3 por 2 bits [26:25], e o porto 4 por 2 bits [29:28].

Cada pino pode ser configurado como entrada ou saída, e quando for entrada pode ser configurado o *pull-up*.

Na **figura 4** apresenta-se um esquema resumido um pino.

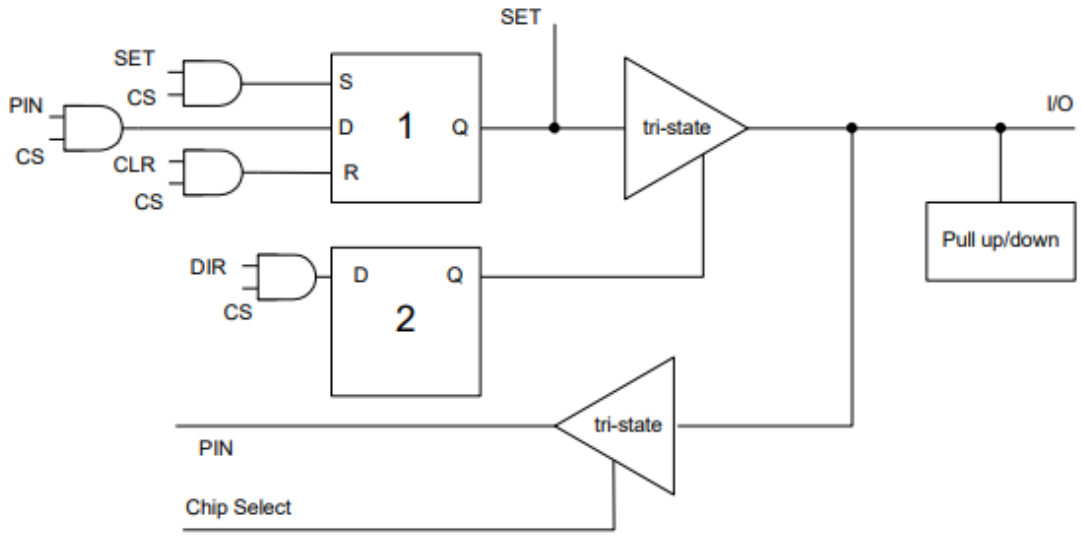


Figura 4 - Resumo de um pino do microcontrolador

Temos na **tabela 1** as funções que podem ser atribuídas aos pinos.

PINSEL0 to PINSEL9 Values	Function	Value after Reset
00	Primary (default) function, typically GPIO port	00
01	First alternate function	
10	Second alternate function	
11	Third alternate function	

Tabela 1 - Funções atribuídas aos pinos

É de constar que é através do registo PINSEL (dos quais existem vários) que se atribui funções aos pinos, como temos 4 funções precisamos de 2 bits.

Também se deve mencionar que os pinos têm modo, que para os quais são usados os registo PINMODE, que tal como acontece nos registos PINSEL é necessário 2 bits para configurar cada pino, como a **tabela 2** mostra.

PINMODE0 to PINMODE9 Values	Function	Value after Reset
00	Pin has an on-chip pull-up resistor enabled.	00
01	Repeater mode (see text below).	
10	Pin has neither pull-up nor pull-down resistor enabled.	
11	Pin has an on-chip pull-down resistor enabled.	

Tabela 2 - Modos dos pinos de entrada

Por omissão o *pull-up* está ativo, mas é possível desativar, ligar o *pull-down* ou até o *repeater mode*.

II.3. *Real Time Clock*

O *Real Time Clock* é um dispositivo de baixo consumo, como tal é alimentado por uma pilha. Mesmo tendo isto em conta é possível desligá-lo usando o bit 9 do registo PCONP.

Na **figura 5** apresenta-se o esquema de blocos do RTC. Também pode ser dito que contém uma RAM de 20 *bytes* e um oscilador de 32KHz que irá produzir uma frequência de 1Hz, que permite o funcionamento do relógio.

As informações do relógio e do calendário encontram-se em registo diferentes. Apesar dos registos estarem consolidados, são sempre necessárias três leituras para se obter toda a informação do RTC e sabendo que o relógio não para, existe a improbabilidade de uma leitura errada. Isto pode acontecer em casos extremos como por exemplo se a leitura for feita a 31 de dezembro às 23:59:59. Para contornar este problema duas leituras teriam de ser efetuadas, para confirmar a correta leitura.

Para um funcionamento rigoroso de um projeto que dependesse muito do relógio esta verificação seria necessária.

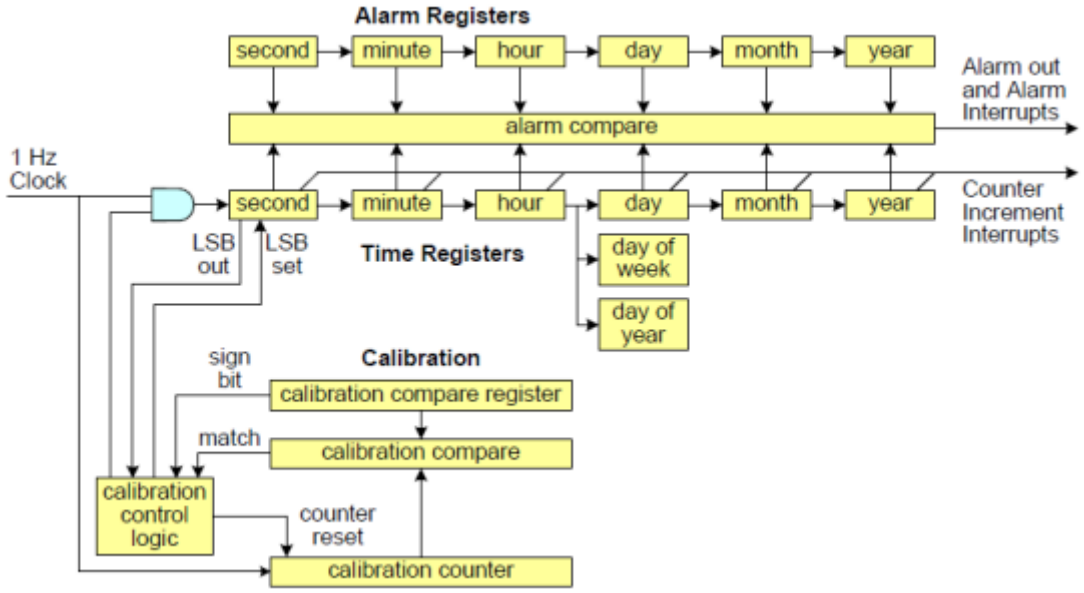


Figura 5 - Esquema de blocos do RTC

Após a alimentação, são utilizados dois registos para inicializar o RTC, o CCR e o CALIBRATION.

II.4. *Serial Peripheral Interface*

SPI é um *de-facto standard* para *synchronous serial communication*, usa uma arquitetura *master-slave*.

Para realizar esta comunicação precisamos de 4 pinos do microcontrolador, um pino MOSI, que é a saída do *master* que vai ligar à entrada do *slave*.

Um pino MISO que é a entrada do *master* que vai ligar à saída do *slave*.

Um pino SCLK que é a saída do *master* ou a entrada do *slave*.

Um pino SSEL é uma entrada do *slave*.

Neste projeto, o dispositivo é configurado como *master* e irá ligar ao leitor de cartões.

O SPI permite a comunicação de dados em *full-duplex*, com quatro modos de operação, conforme a **tabela 3**.

CPOL and CPHA settings	When the first data bit is driven	When all other data bits are driven	When data is sampled
CPOL = 0, CPHA = 0	Prior to first SCK rising edge	SCK falling edge	SCK rising edge
CPOL = 0, CPHA = 1	First SCK rising edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 0	Prior to first SCK falling edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 1	First SCK falling edge	SCK falling edge	SCK rising edge

Tabela 3 - Modos de operação SPI

Como master, o dispositivo deve ser capaz de selecionar o SSEL do dispositivo com quem vai comunicar.

Ele recebe o clock do processador que é dividido pelo valor colocado no registo SPCCR. Sendo um número de oito bits, o valor máximo da divisão é igual a 255.

II.5. *System Tick Timer*

Este timer é constituido por um contador de 24 bits decrescente com a capacidade de produzir uma interrupção. Por *default*, se estiver ligado a um *clock* de 100MHz (frequência do microcontrolador), produz um tempo de 10 milissegundos.

O temporizado é constituido por quatro registos, o STCTRL, o STCURR, o STCALIB e o STRELOAD, como representa a **figura 6**.

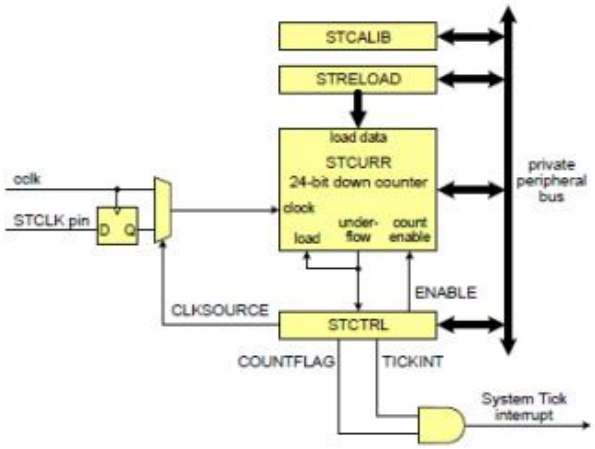


Figura 6 - Esquema de blocos do Systick

II.6. *Repetitive Interrupt Timer*

O RIT é composto por um contador de 32 bits ligado à frequência do *clock* do LPC1769. Pode ser comparado com o valor de 32 bits e pode ainda ser utilizado uma máscara.

O RIT usa quatro registos, o RIMASK, o RICTRL, o RICOUNTER e o RICOMPVAL, expostos na **figura 7**.

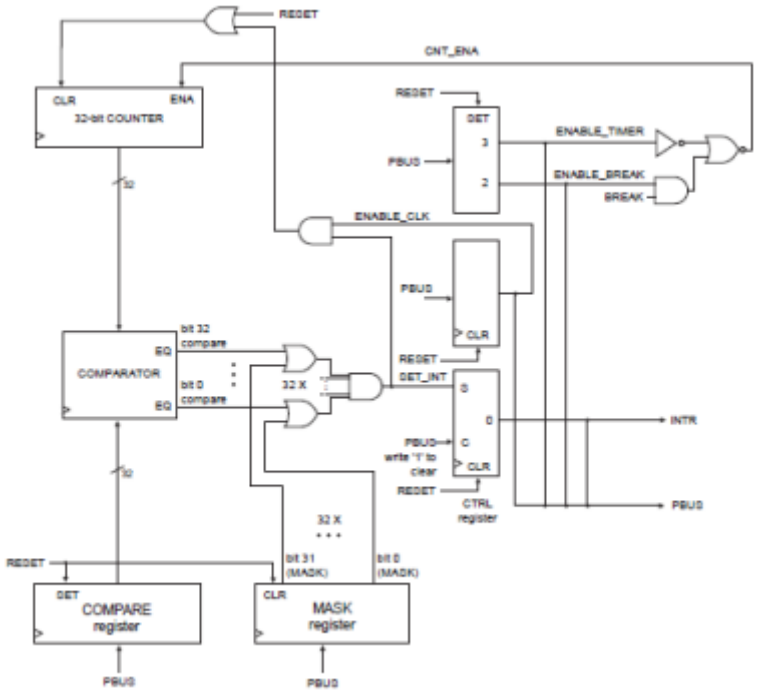


Figura 7 - Esquema de blocos do RIT

O registo RICTRL contém quatro bits que são usados para configurar o RIT:

- RITINT

  O bit é igual a 1 sempre que o comparador diga que o contador e o RICOMPVAL são iguais. Para meter este bit a 0 é necessário escrever 1.

- RITENCLR

  Caso seja 0, o contador continua a contagem sem ser apagado.

  Se for 1, o contador é colodado a zero quando chegar ao valor em RICOMPVAL.

- RITENBR

  Se for igual a 1, o contador pára se o processador parar.

  Se for igual a 0, ignora o estado do processador.

- RITEN

  Este bit ativa ou desativa o *timer*.

II.7. *Timer0*

O *timer* 0 é um de quatro *timers*, é constituído por um divisor de 32 bits (*Prescale Counter*) e um contador de 32 bits associado a quatro registos de comparação, o MR0, o MR1, o MR2 e o MR3.

Associados a este *timer* estão os vários registos representados na **figura 8**.

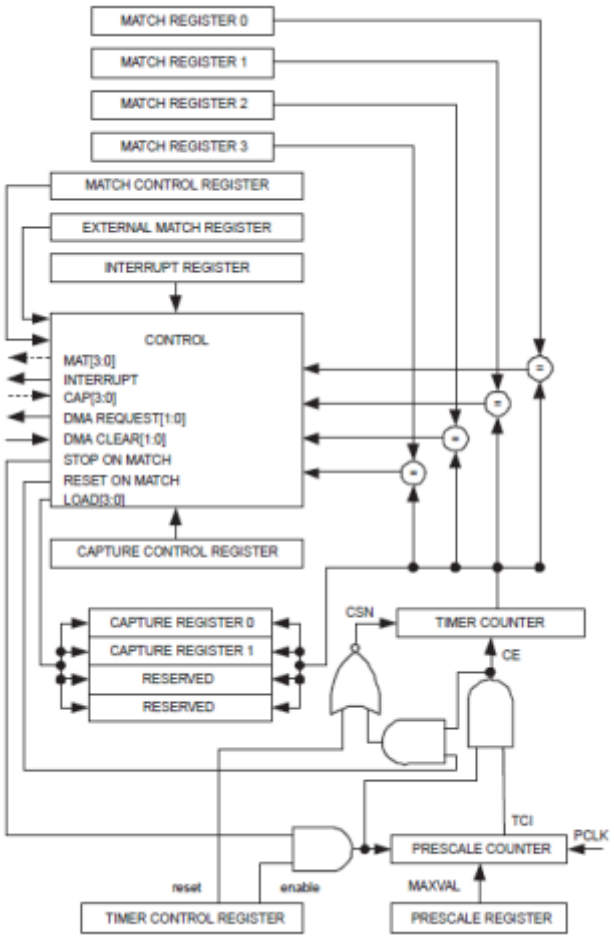


Figura 8 - Esquema de blocos do Timer0

II.8. *Flash*

A memória não volátil do LPC1769 permite ISP (*in-system programming*), o que permite utilizadores darem *update* ao programa sem remover o microcontrolador. É uma *feature* que beneficia a *maintenance* e simplifica o processo de atualizar o *firmware*.

É também relevante mencionar que a *flash* está dividida em setores como a seguinte **tabela 4** ilusta. Isto é útil pois permite a utilização de cada sector independentemente dos outros, podendo apagar só um sector ou alterar só um sector, minimizando a probabilidade de *data corruption*.

Sector Number	Sector Size [kB]	Start Address	End Address	32 kB Part	64 kB Part	128 kB Part	256 kB Part	512 kB Part
0	4	0X0000 0000	0X0000 0FFF	x	x	x	x	x
1	4	0X0000 1000	0X0000 1FFF	x	x	x	x	x
2	4	0X0000 2000	0X0000 2FFF	x	x	x	x	x
3	4	0X0000 3000	0X0000 3FFF	x	x	x	x	x
4	4	0X0000 4000	0X0000 4FFF	x	x	x	x	x
5	4	0X0000 5000	0X0000 5FFF	x	x	x	x	x
6	4	0X0000 6000	0X0000 6FFF	x	x	x	x	x
7	4	0X0000 7000	0X0000 7FFF	x	x	x	x	x
8	4	0x0000 8000	0X0000 8FFF		x	x	x	x
9	4	0x0000 9000	0X0000 9FFF		x	x	x	x
10 (0x0A)	4	0x0000 A000	0X0000 AFFF		x	x	x	x
11 (0x0B)	4	0x0000 B000	0X0000 BFFF		x	x	x	x
12 (0x0C)	4	0x0000 C000	0X0000 CFFF		x	x	x	x
13 (0x0D)	4	0x0000 D000	0X0000 DFFF		x	x	x	x
14 (0X0E)	4	0x0000 E000	0X0000 EFFF		x	x	x	x
15 (0x0F)	4	0x0000 F000	0X0000 FFFF		x	x	x	x
16 (0x10)	32	0x0001 0000	0x0001 7FFF			x	x	x
17 (0x11)	32	0x0001 8000	0x0001 FFFF			x	x	x
18 (0x12)	32	0x0002 0000	0x0002 7FFF				x	x
19 (0x13)	32	0x0002 8000	0x0002 FFFF				x	x
20 (0x14)	32	0x0003 0000	0x0003 7FFF				x	x
21 (0x15)	32	0x0003 8000	0x0003 FFFF				x	x
22 (0x16)	32	0x0004 0000	0x0004 7FFF					x
23 (0x17)	32	0x0004 8000	0x0004 FFFF					x
24 (0x18)	32	0x0005 0000	0x0005 7FFF					x
25 (0x19)	32	0x0005 8000	0x0005 FFFF					x
26 (0x1A)	32	0x0006 0000	0x0006 7FFF					x
27 (0x1B)	32	0x0006 8000	0x0006 FFFF					x
28 (0x1C)	32	0x0007 0000	0x0007 7FFF					x
29 (0x1D)	32	0x0007 8000	0x0007 FFFF					x

Tabela 4 - Setores do LPC1769

Neste projeto foi utilizado meramente o setor 29.

Capítulo III. Drivers

III.1. Generalidades

A biblioteca *DRIVERS*, foi concebida para trabalhar com a HAL previamente descrita. Sendo assim, a aplicação poderá comunicar com os diversos dispositivos sem a necessidade de os conhecer.

Neste capítulo vão ser abordados os seguintes *drivers*:

- Display LCD 16x2;
- Botão Rotativo;
- Led do LPCXpress;
- Leitor de cartões;
- Restantes drivers que servem de abstração à camada de abstração.

III.2. *Liquid Crystal Display*

O *display* utilizado da **figura 9**, é constituído por um barramento de 8 bits, um pino W/R, um pino de *enable* e um RS que permite identificar se o que está a ser transmitido é um comando ou dados.

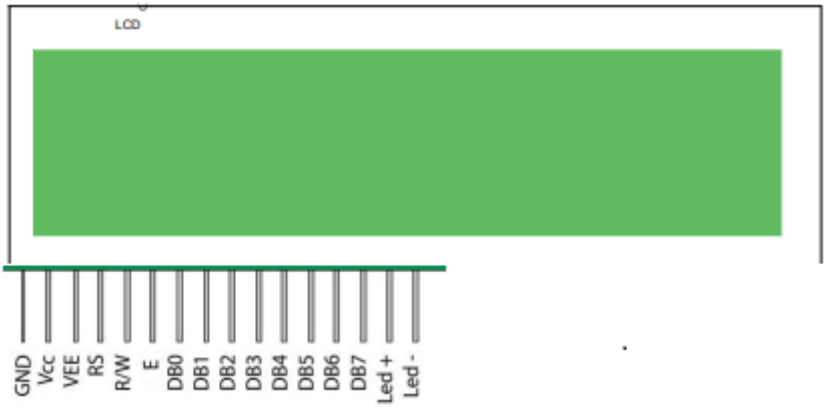


Figura 9 - Display LCD

A comunicação será realizada através dos pinos DB4 a DB7. Apenas iremos usar a escrita portanto o pino de escrita e leitura será ligado ao zero.

A configuração dos pinos é descrita na **tabela 5**.

```
/* Montagem do LCD ao micro
 * DB0.. DB3   | Not connected
 * DB4 .. DB7  | P2.0 .. P2.3
 * E           | P0.10
 * RS          | P0.11
 * WR          |GND
 */
```

Tabela 5 - Montagem do display ao LPC1769

A inicialização do *display* requer que se siga uma dada sequência ilustrada pela figura 10:

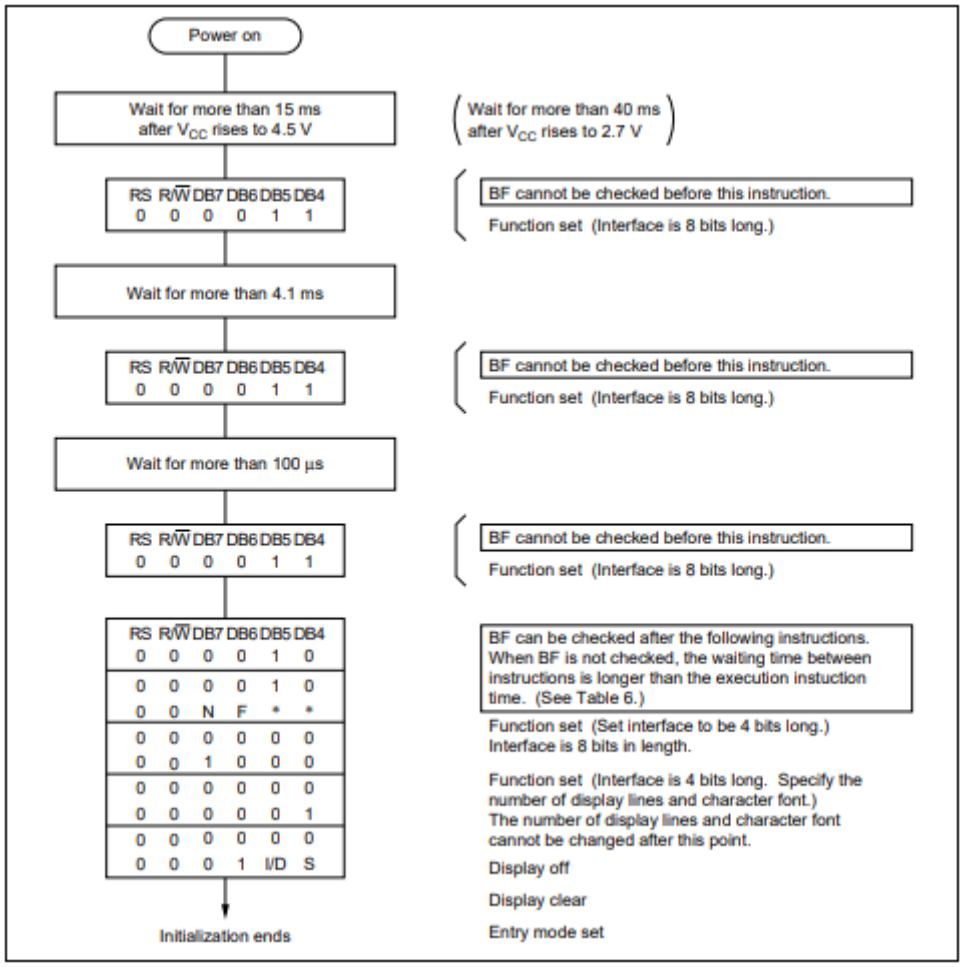


Figura 10 - Sequência para inicializar o LCD com uma *interface* a 4 bits

Os comandos acontecem quando RS igual a 0 e a transmissão de dados quando RS igual a 1.

Na **figura 11** está ilustrada o *timing* da operação de escrita:

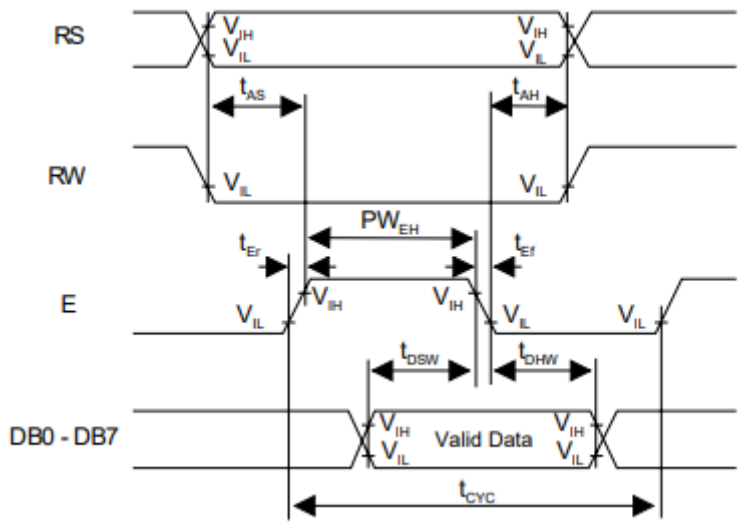


Figura 11 - *Timing* da operação de escrita

Para a correta utilização do *display* foi assegurada os tempos necessários como apresentados na **tabela 6**.

Parameter	Symbol	Conditions	Min.	Max.	Units
Enable Cycle Time	t_{CYC}	Fig.1, 2	500	--	ns
Enable Pulse Width	PW_{EH}	Fig.1, 2	230	--	ns
Enable Rise/Fall Time	t_{Er}, t_{Ef}	Fig.1, 2	--	20	ns
Address Setup Time	t_{AS}	Fig.1, 2	40	--	ns
Address Hold Time	t_{AH}	Fig.1, 2	10	--	ns
Write Data Setup Time	t_{DSW}	Fig.1	80	--	ns
Write Data Hold Time	t_{DHW}	Fig.1	10	--	ns
Read Data Delay Time	t_{DDR}	Fig.2	--	120	ns
Read Data Hold Time	t_{DHR}	Fig.2	5	--	ns

Tabela 6 - Tabela de *timings* mínimos

III.3. *Rotary Encoder Switch*

Nesta secção iremos descrever como funciona o botão rotativo que pode ser visualizado na **figura 12**.

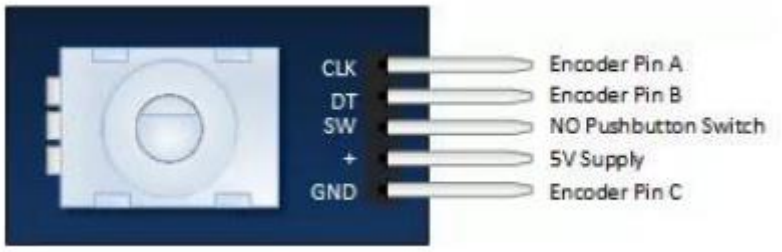


Figura 12 - Ilustração do botão e dos seus pinos

O codificador rotativo tem dois pinos, cujo o sinal é representado na **figura 13**.

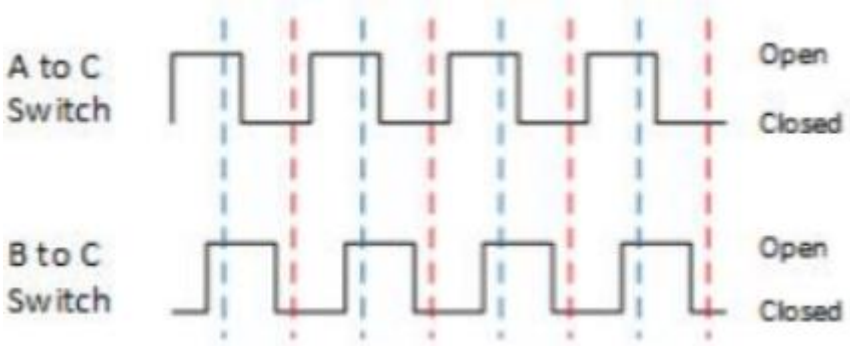


Figura 13 - Estados do codificador rotativo

Para ler a rotação tivemos de ter em conta os sinais em cima ilustrados. Quanto ao *click* do botão um *click* normal mudaria o estado do botão de *NOTPRESSED* para *CLICKED* exceto quando o botão é pressionado duas vezes dentro do tempo estipulado no ENCODER.h, neste caso o botão fica no estado *DCLICKED*.

Tanto a rotação como os dois estados de *click* serão essenciais na aplicação.

Por último na **tabela 7** encontramos as ligações usadas para ligar o botão ao LPC1769.

```
/* Montagem do ENCODER ao micro
 * CLK | P0.03
 * SW  | P2.13
 * DT  | P0.02
 */
```

Tabela 7 - Ligações do botão ao LPC1769

III.4. LED do LPCXpress

O LED está diretamente ligado ao pino 0.22, através do qual podemos verificar ou simular o trinco. O LED usa funções do pin.c e do pinconfig.c para o seu funcionamento.

III.5. MFRC522

O nosso leitor de cartões usa o periférico MFRC522 do qual existe uma biblioteca auxiliar que executa tudo o que precisamos para a aplicação, mas para esta biblioteca funcionar tivemos de criar funções que essa biblioteca usa para iniciar e configurar a ligação entre o periférico e o LPC1769, são estas que se encontram neste .c da biblioteca DRIVERS.

O MFRC522 funciona usando o *standard* SPI, estando ligado ao microcontrolador como *slave*, na tabela 8 podemos ver as ligações ao LPC1769, podemos também ver o MFRC522 ilustrado na **figura 14**.

```
/* PINOS SPI USADOS
 * MOSI |P0.18
 * MISO | P0.17
 * SCK  | P0.15
 * SS   | P0.16
 * RST  | P0.0
 */
```

Tabela 8 - Ligações usadas entre o leitor de cartões e o LPC1769

Figura 14 - *Pinout* do MFRC522

Com a interface de SPI podemos transmitir dados até 10 Mbit/s, ilustrado na **figura 15** está um diagrama de blocos da coneção SPI ao *host* do ponto de vista do MFRC522:

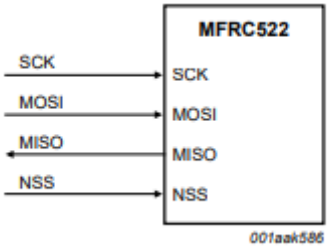


Figura 15 - Ligação SPI ao host

É importante denotar que o sinal de *clock* SCK têm de ser gerado pelo *master*, a transmissão de dados do *master* para o *slave* é feita pela linha MOSI, a transmissão inversa pela linha MISO. *Bytes* de dados são enviados com o MSB primeiro. Dados em ambas as linhas devem ser estáveis na *falling edge* pois é durante esta que os dados são transmitidos pelo MFRC522, e é estável no *rising edge*. Na tabela 9 é vista a ordem de *bytes* da leitura e na tabela 10 a ordem dos *bytes* na escrita:

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	address 1	address 2	...	address n	00
MISO	X[1]	data 0	data 1	...	data n – 1	data n

[1] X = Do not care.

Remark: The MSB must be sent first.

Tabela 9 - Ordem da leitura dos *bytes* no SPI

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	data 0	data 1	...	data n – 1	data n
MISO	X[1]	X[1]	X[1]	...	X[1]	X[1]

[1] X = Do not care.

Remark: The MSB must be sent first.

Tabela 10 - Ordem da escrita dos *bytes* no SPI

III.6. Restantes

Os restantes .c na biblioteca DRIVERS (DELAY, RTC e DRIVERS_FLASH) tratam-se meramente de funções que chamam as funções ja feitas na biblioteca HAL, isto é feito para a aplicação não ter de conhecer o *hardware*. A exceção sendo umas funções dentro do RTC.c que que usam a biblioteca time.h.

Capítulo IV. Aplicação

IV.1. Generalidades

A aplicação baseia-se no circuito da **figura 16**. O código descrito é utilizado para produzir um produto com todos os componentes e gerir a utilização de um armário com fechadura RFID.

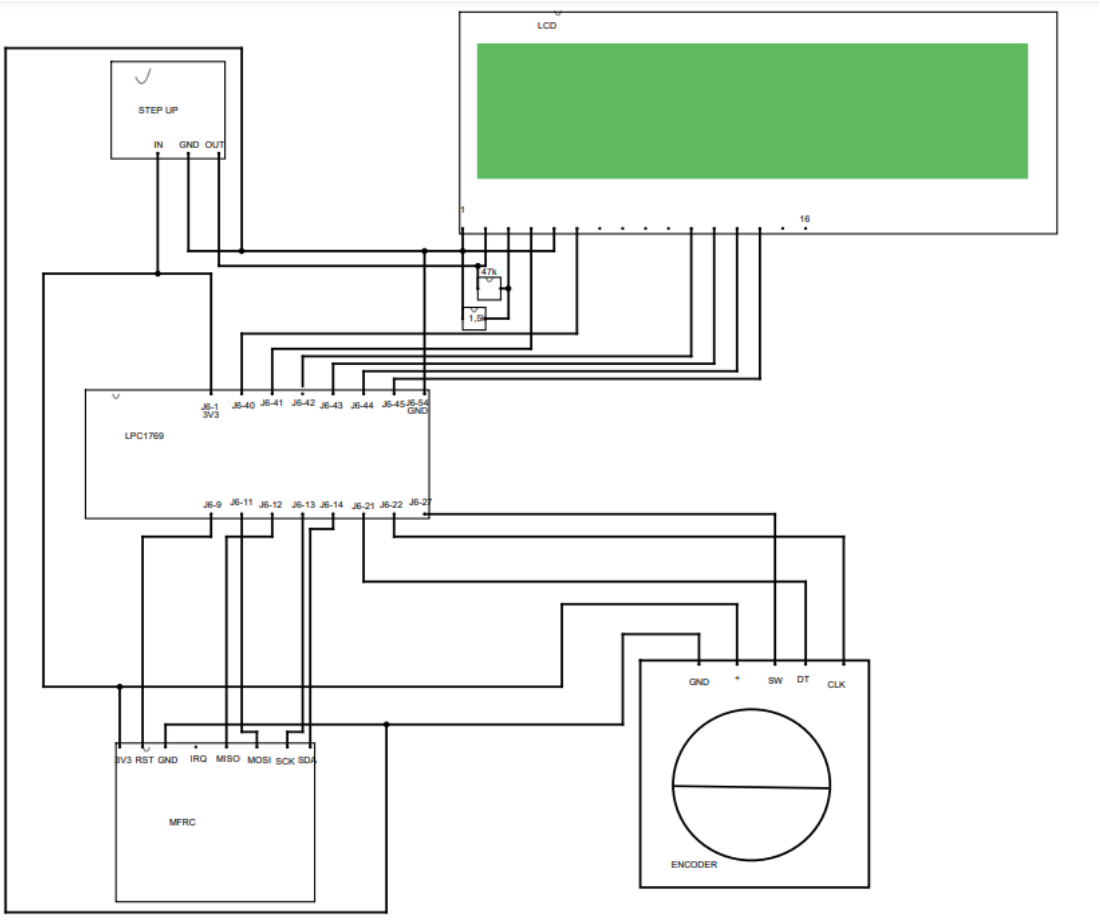


Figura 16 - Circuito da aplicação com o microcontrolador LPC1769

Para implementar a aplicação, foram utilizadas várias máquinas de estado.

Em cada uma das secções, vai ser descrita a sua finalidade na aplicação e o funcionamento da máquina de estado respetiva.

De denotar também é que o início da aplicação está colocado trab.c. Aqui são feitas as inicializações dos dispositivos usados neste projeto.

É neste código que é implementado a máquina de estados mais básica, a troca de modo normal para modo de manutenção.

IV.2. Modos de Operação

Foi implementada uma máquina de estados com dois estados, que meramente serve para chamar a máquina de estados do modo normal ou a máquina de estados do modo de manutenção.

Sempre que a aplicação sai de um modo, a máquina de estados comuta automaticamente para o outro modo.

Esta máquina de estados é descrita pela pelos dois *superstates* vistos na **figura 17**.

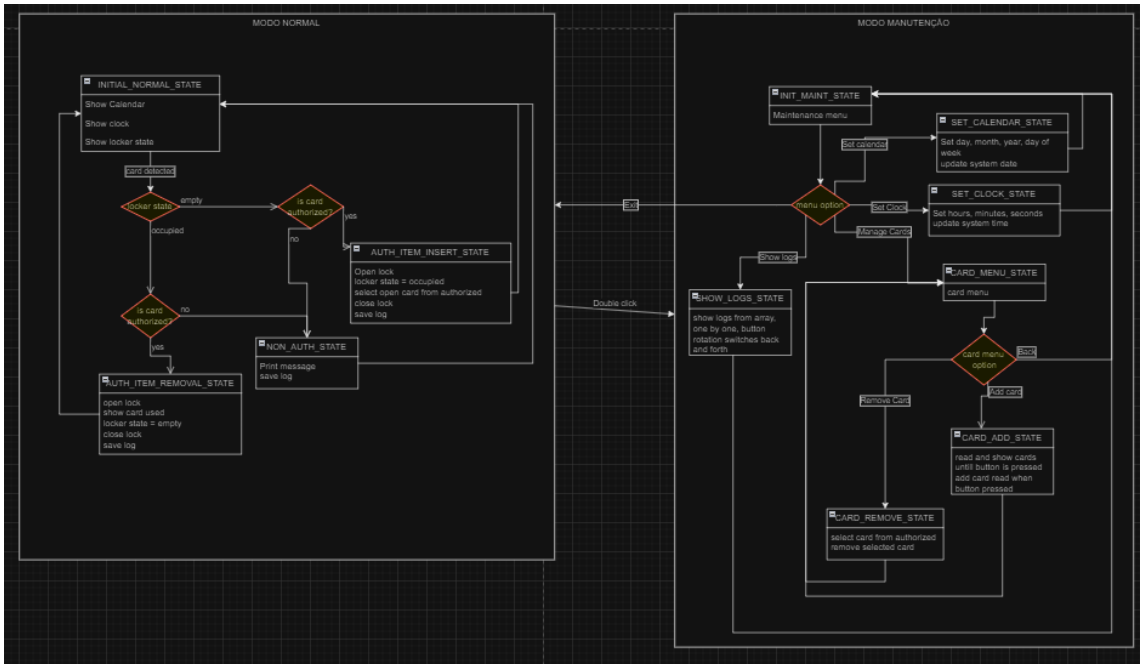


Figura 17 - Máquina de estados da aplicação com os dois *superstates*

IV.3. Modo Normal

O modo normal é constituido por quatro estados: INITIAL_NORMAL_STATE, AUTH_ITEM_INSERT_STATE, NON_AUTH_STATE, AUTH_ITEM_REMOVAL_STATE.

É importante realçar que no código encontramos um modo EXIT_NORMAL que serve meramente para acabar o *while* do código da função do modo normal, e como tal não aparece na máquina de estados.

Cada um dos estados vai ser descrito a seguir, conforme a **figura 18**.

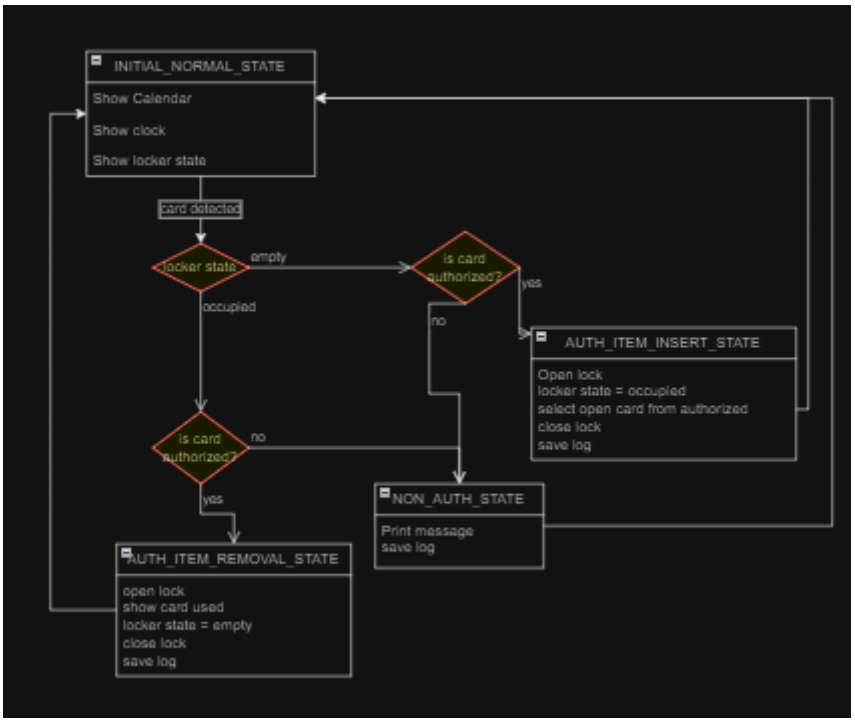


Figura 18 - Máquina de estados do modo normal

- **INITIAL_NORMAL_STATE**

Este é o estado inicial que aguarda a apresentação do cartão. Caso apareça um cartão iremos verificar se o *locker* está ocupado ou vazio, se estiver vazio iremos ver se o cartão apresentado está na lista de cartões autorizados se estiver iremos para o AUTH_ITEM_INSERT_STATE.

Se o *locker* se encontrar ocupado iremos fazer uma comparação do cartão apresentado com o cartão que a pessoa que meteu a entrega decidiu que podia ir buscar a entrega.

Em ambos os casos se o cartão não obter uma comparação favorável iremos para o estado NON_AUTH_STATE.

Na **figura 19** vemos máquina de estados deste estado.

Devemos também denotar que enquanto estivermos no *loop* deste estado continuamos a dar *update* à data relógio e ao estado do *locker*.

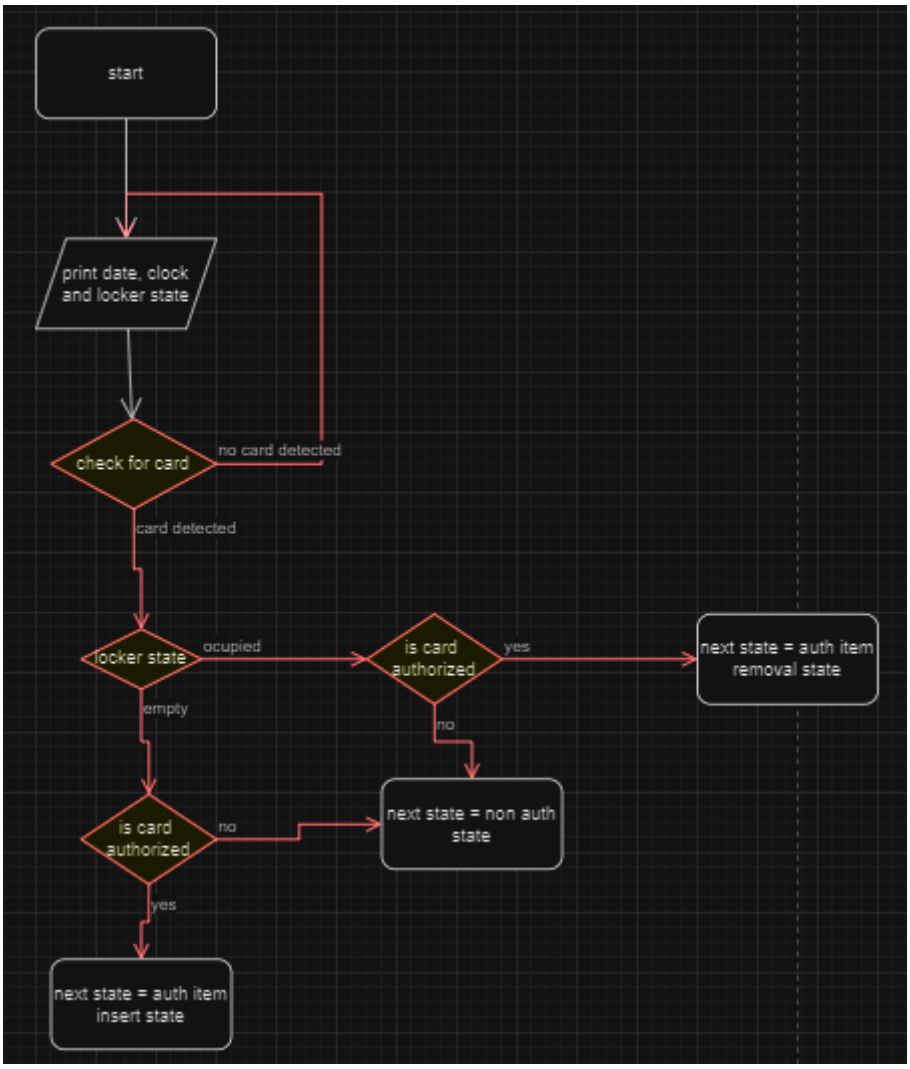


Figura 19 - Máquina de estados do modo INITIAL_NORMAL_STATE

- **AUTH_ITEM_INSERT_STATE**

Neste estado abrimos o trinco, mudamos o estado do armário de vazio para ocupado, ficamos num *loop* à espera que um dos cartões presentes na lista de autorizados seja escolhido para puder ir buscar a entrega, fechamos o trinco após a escolha e guardamos esta interação em forma de *log*.

Na **figura 20** podemos ver a máquina de estados deste estado.

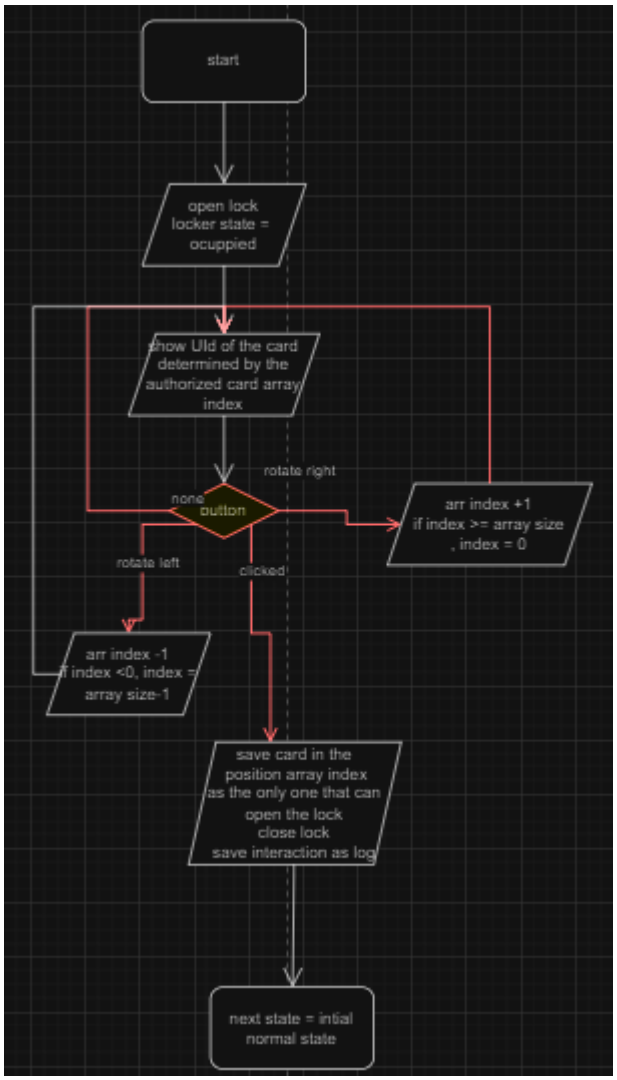


Figura 20- Máquina de estados do AUTH_ITEM_INSERT_STATE

- **NON_AUTH_STATE**

Neste estado, um dos mais simples do projeto, promovemos o aparecimento de uma mensagem a dizer “Cartão não autorizado” no *display* e guardamos esta interação na forma de *log*.

Na **figura 21** podemos ver a máquina de estados deste estado.

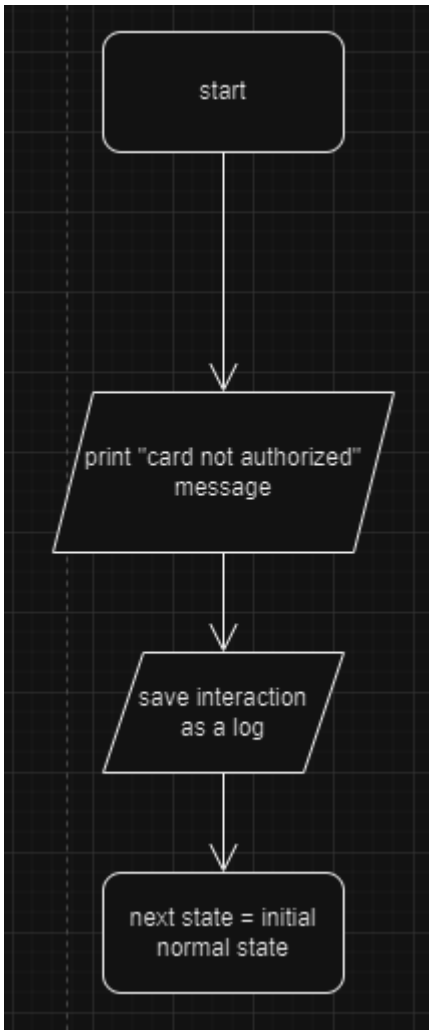


Figura 21 - Máquina de estados do NON_AUTH_STATE

- **AUTH_ITEM_REMOVAL_STATE**

Ao entrarmos neste estado abrimos o trinco, dando 5 segundos para retirar a encomenda, no *display* apresenta-se o cartão que foi escolhido para puder levantar a encomenda, fechamos o trinco e salvamos o *log* da interação.

Na **figura 22** podemos ver a máquina de estados deste estado.

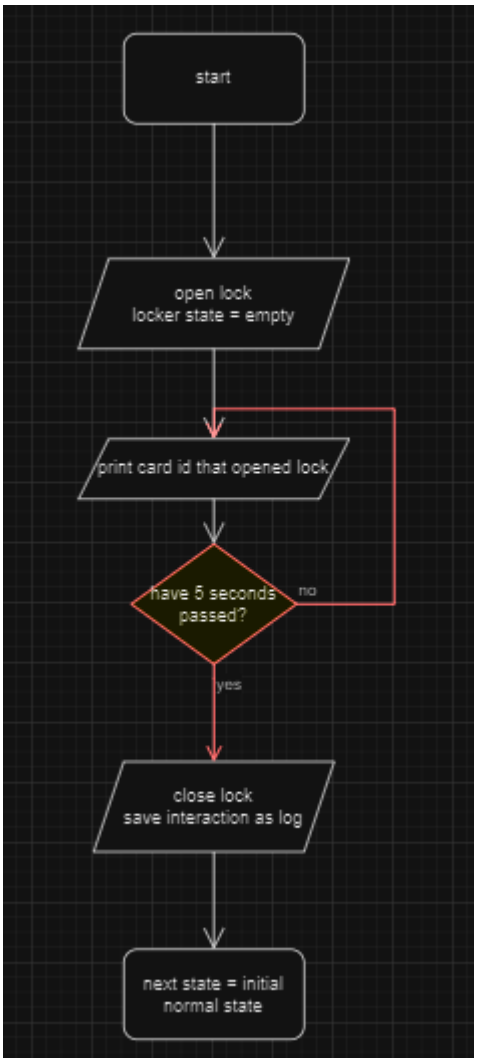


Figura 22 - Máquina de estados do AUTH_ITEM_REMOVAL_STATE

IV.4. Modo Manutenção

Ao entrarmos no modo de manutenção é apresentado um menu no LCD, o utlizador pode selecionar uma de várias opções; o ajuste do horário, o ajuste de luz, mostrar os *logs*, ou saltar para outro menu, através da escolha "*Manage cards*", onde nesse menu temos a escolha de remover ou adicionar cartões. Para sair da manutenção podemos escolher a opção "*Exit*. No menu de "*Manage cards*" temos a opção de "*Back*" para puder retroceder para o menu do modo inicial de manutenção.

A máquina de estados do modo de manutenção contêm sete modos: INIT_MAINT_STATE, SET_CALENDAR_STATE, SET_CLOCK_STATE, CARD_MENU_STATE, CARD_ADD_STATE, CARD_REMOVE_STATE e SHOW_LOGS_STATE.

A máquina de estados do modo de manutenção encontra-se na **figura 23**.

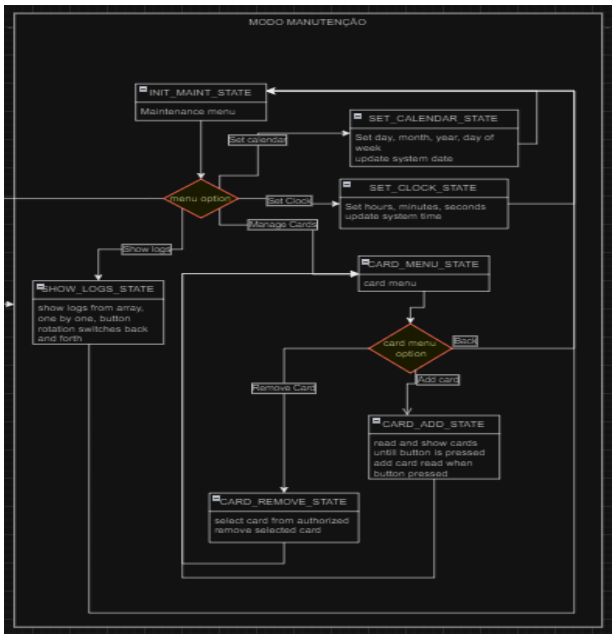


Figura 23 - Máquina de estados do modo manutenção

- **INIT_MAINT_STATE**

Estado utilizado para percorrer o menu de manutenção e fazer a escolha de para que estado ir a seguir.

A máquina de estados deste estado pode ser vista na **figura 24**.

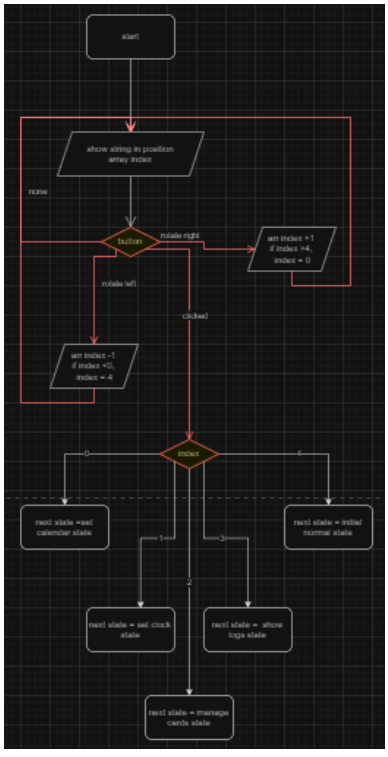


Figura 24 - Máquina de estados do INIT_MAINT_STATE

- **SET_CALENDAR_STATE**

Estado onde vamos fazer o acerto do calendário, utlizaça funções realizadas no utils.c

A máquina de estados encontra-se em anexo à entrega do relatório.

- **SET_CLOCK_STATE**

Estado onde vamos fazer o acerto do relógio, utlizaça funções realizadas no utils.c

A máquina de estados encontra-se em anexo à entrega do relatório.

- **CARD_MENU_STATE**

Estado de menu adicional, onde iremos escolher se queremos adicionar ou remover cartões faz uso da mesma "arquitetura" do menu do estado de INIT_MAINT_STATE.

A máquina de estados encontra-se na **figura 25**.

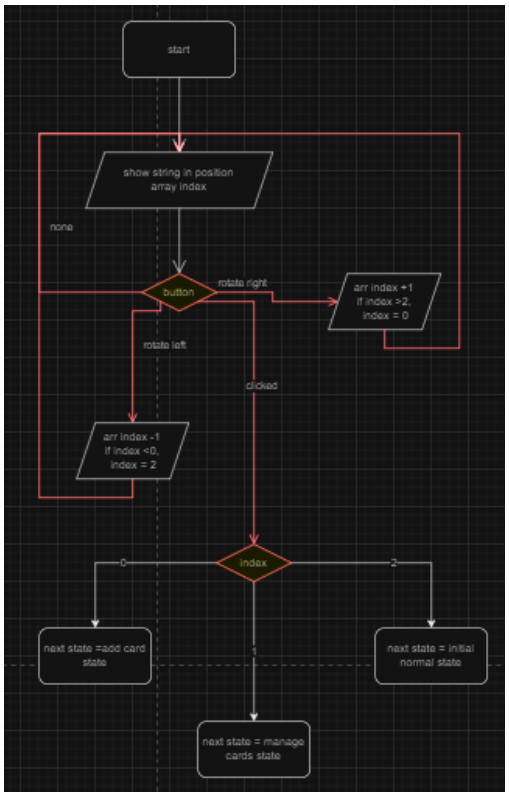


Figura 25 - Máquina de estados do CARD_MENU_STATE

- **CARD_ADD_STATE**

Ao entrarmos neste estado ficamos numa *loop* à espera que seja lido um cartão se um cartão tiver sido lido com sucesso apresentamos a identificação deste cartão no *display* e durante 5 segundos se houver *click* este cartão é adicionado à lista de cartões autorizados, após disto retornamos ao INIT_MAINT_STATE. Também é possível sair deste estado com um *double-click*.

A máquina de estados está ilustrada na **figura 26**.

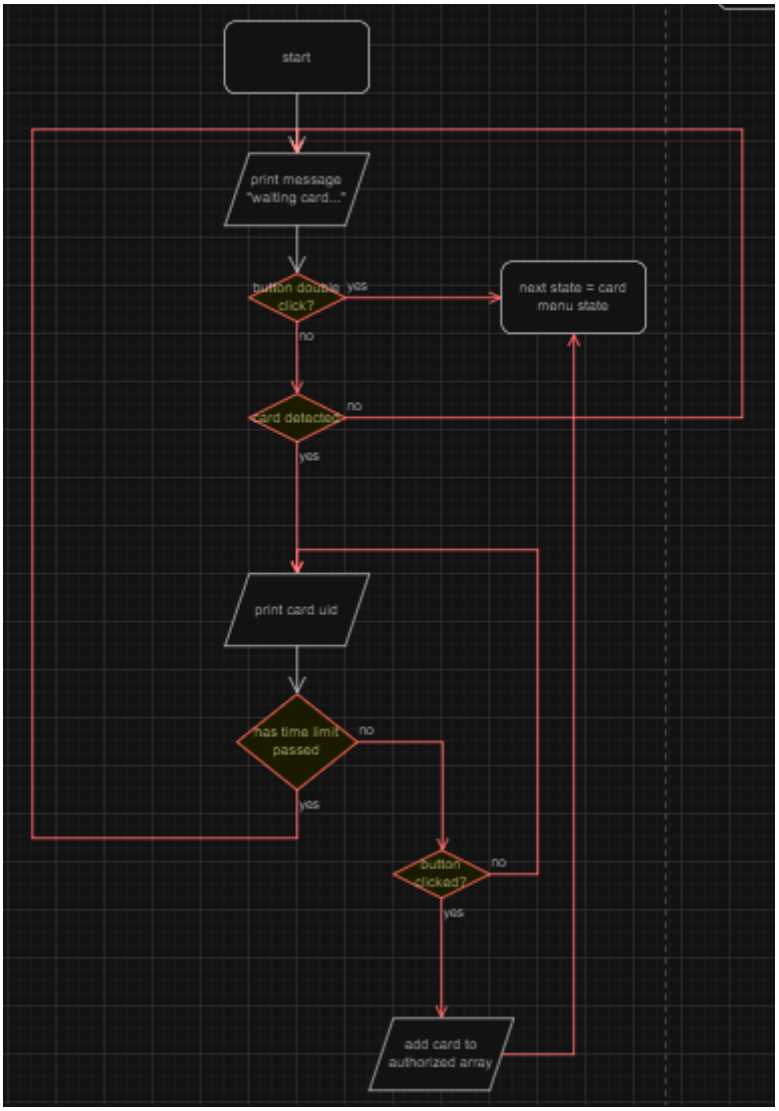


Figura 26 - Máquina de estados do CARD_ADD_STATE

- **CARD_REMOVE_STATE**

Entrando neste modo ficamos num *loop* à espera que haja um *click*, o botão rotativo promove a visualização da lista de cartões autorizados sendo apresentado no LCD o cartão neste momento selecionado, quando ocorre o *click* o cartão é apagado da lista.

A máquina de estados encontra-se na **figura 27**.

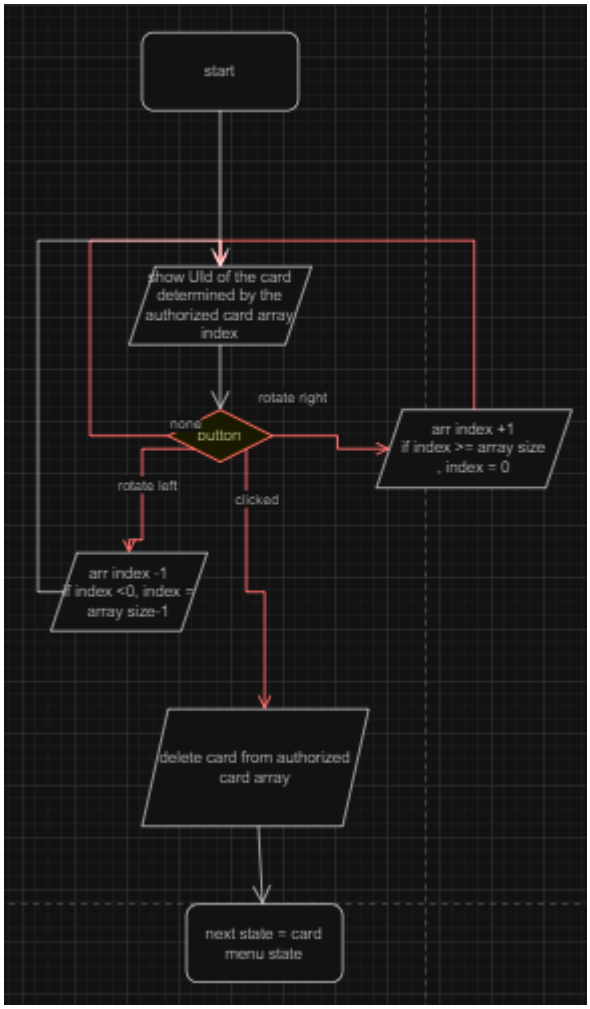


Figura 27 - Máquina de estados do CARD_REMOVE_STATE

- **SHOW_LOGS_STATE**

Por fim temos este estado, que percorre a lista de *logs* usando o botão rotativo, ao *click* voltamos ao INIT_MAINT_STATE.

A máquina de estados encontra-se na **figura 28**.

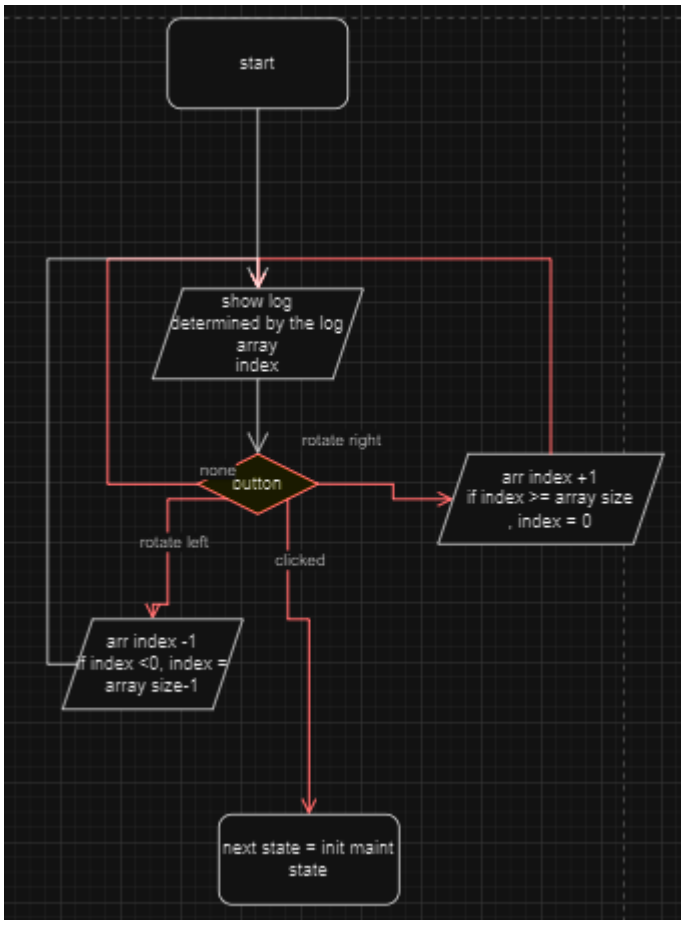


Figura 28 - Máquina de estados do SHOW_LOGS_STATE

IV.5. Sobre a *Flash* e a escolha das estruturas de armazenamento de dados

Como o a *feature* de guardar e recolher dados da *flash* ficou para o final, ao fazermos estas funções deparamo-nos com um erro de *hard fault*, e como tal não nos foi permitido acabar esta capacidade do sistema. Apesar disto, protótipos e até uma versão final de algumas funções foram criadas.

O nosso sistema iria guardar e recolher 3 *structs*, uma de *startup*, e duas com os dados dos cartões autorizados e dos *logs*. As funções de salvar e recolher o *startup* foram finalizadas mas foi aqui que nos deparámos com o *hard fault* o que parou a finalização das funções de *load* e *save* das *structs* de cartões autorizados e *logs*, mesmo assim protótipos das mesmas encontram-se comentadas no utils.c.

Para estas funções funcionarem na atual arquitetura da aplicação iriamos colocá-las no modes.c e tirar os retornos e parametros de entrada, e iriamos ter outras modificações devido ao facto que os protótipos foram feitos quando as *structs* tinham um formato diferente, mas em geral as funções finais iriam seguir a metodologia dos protótipos.

Por fim iremos apenas comentar a escolha das 3 *structs* de armazenamento de dados começando primeiro pela de *startup*:

- ***Startup***

```
typedef struct {
    uint8_t    allowedCard[10];
    uint8_t    size;
    int lockerState;
    int CardsCount;
    int LogsCount;
    uint8_t    padding[512 - 22];
} StartupStateRestoreType;
```

Figura 29 - *Struct* do *Startup*

O objetivo desta *struct* era retornar o sistema ao estado igual a que se encontrava antes de perder a energia como tal guardavamos as informações do cartão selecionado para abrir a encomenda, o estado do trinco, e os números de cartões autorizados e *logs*.

- ***AuthCards***

```
typedef struct {
    uint8_t    uidByte[10];
    uint8_t    size;
    uint8_t    padding[256 - 11];
} AuthCardsType;
```

Figura 30 - *Struct* para armazenar os cartões autorizados

O objetivo desta *struct* é simplesmente guardar as informações do cartão, os atributos são meramente aqueles da estrutura UID (da biblioteca MFRC522) que eram essenciais.

- ***Logs***

```
typedef struct {
    uint8_t    uidByte[10];
    uint8_t    size;
    int day;
    int month;
    int year;
    int hour;
    int minute;
    int lockerState;
    int authStatus;
    uint8_t    padding[512 - 34];
} LogsType;
```

Figura 31 - *Struct* dos *logs*

O objetivo desta *struct* é guardar a informação importantes para manter um registo das interações de que o armário foi alvo, daí guardarmos o estado do trinco, se o cartão passado era autorizado, a data e hora, e por fim o cartão lido.

Todas as *structs* contêm *padding* pois a *flash* para a escrita ser um sucesso, necessita que o que vai ser escrito tenha um dado tamanho (256 *bytes*, 512 *bytes*, 1024 *bytes* ou 4096 *bytes*), para as nossas *structs* foi escolhido o valor mais perto possível para a escrita ser possível.

Referências

[1] LPC176x/5x User manual

[2] LPCXpresso LPC1769 Schematics

[3] LCD Datasheet

[4] HD44780 Dot Matrix Liquid Crystal Display Controller/Driver

[5] MFRC522 Standard performance MIFARE and NTAG frontend

[6] RFID-MFRC522 Pinout

[7] Step Up - 3.3V to 5V DC-DC Boost Converter Module Board

[8] KY-040 Rotary Encoder Switch Module Board