PROJETO

DE

SISTEMAS EMBEBIDOS

SISTEMA DE GESTÃO DE ILUMINAÇÃO

REALIZADO POR

GRUPO 7D

VASCO GOUVEIA | 47659

JOÃO MARQUES | 48056

ALEXANDRE SILVA | 48195

TURMA LT51N

ANO LETIVO 22/23

ÍNDICE

PARÁGRAFO **ASSUNTO** **Nº PÁG.**

CAPÍTULO 1 – INTRODUÇÃO E OBJETIVO

CAPÍTULO 2 – HARDWARE ABSTRACTION LAYER

CAPÍTULO 3 – DRIVERS

PARÁGRAFO **ASSUNTO** **Nº PÁG.**

CAPÍTULO 4 – APLICAÇÃO

ANEXO A

MÁQUINAS DE ESTADO

BIBLIOGRAFIA

CAPÍTULO 1

INTRODUÇÃO E OBJETIVO

101. OBJETIVO.

No âmbito da Unidade Curricular de Sistemas Embebidos, pretende-se construir um sistema de iluminação controlado por um sensor de luz e um sensor de presença.

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

102. DESCRIÇÃO DO PROJETO.

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

Um *Pyroelectric Infra-Red sensor* (PIR) [AM312], que capta movimentos ao seu redor, detetando a presença de entes ao seu redor;

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

Um encoder rotativo [KY-040], que para além de um botão, permite obter informação sobre a direção da rotação do seu eixo;

Um módulo Electrically-Erasable Programmable Read-Only Memory (EEPROM) [CAT25128], que possibilita o armazenamento de dados no formato de 8 bits;

Um sensor de luminosidade [BH1750FVI], que capta informação acerca da luz ambiente podendo variar entre [1-65535] Lx.

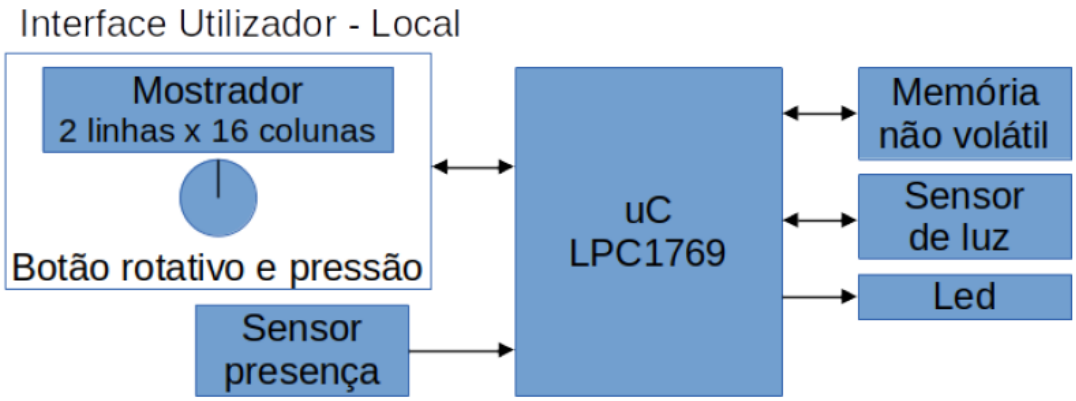

Figura 1 – Diagrama de blocos.

O funcionamento de cada um dos elementos, vai ser descrito neste trabalho, onde serão abordadas as suas características de funcionamento.

O sistema irá ter dois modos de funcionamento, o normal e o de manutenção. No modo normal, a luz acende e apaga em função da presença humana e da luz ambiente. Neste modo, o botão rotativo permite observar a hora e a data, ou alterar para o modo de manutenção.

Quando acende a luz, é efetuado um registo na EEPROM com a hora, a data e a luz ambiente.

No modo de manutenção, é possível acertar a hora e a data, verificar e apagar os registos, e modificar a sensibilidade da luz.

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

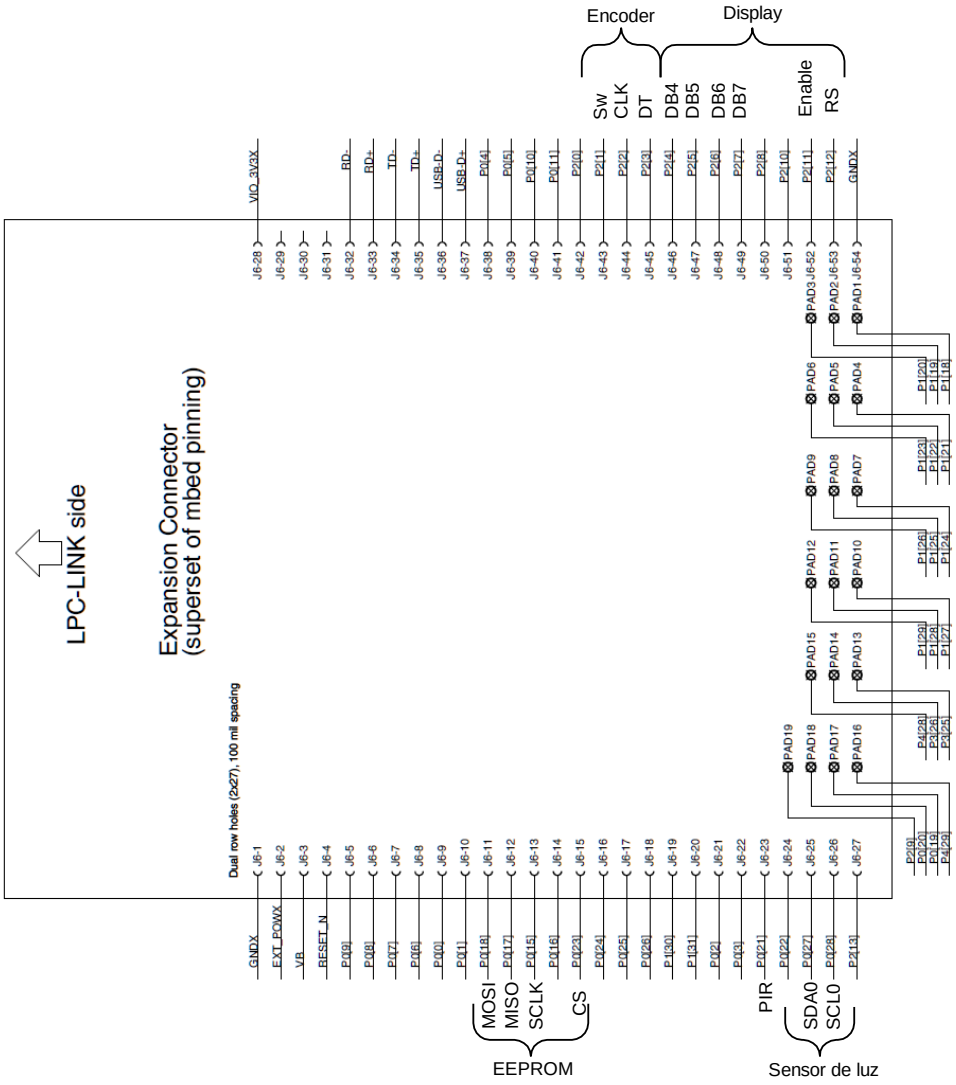

Figura 2 – Ligação da placa de desenvolvimento aos periféricos usados neste trabalho.

103. O MICROCONTROLADOR LPC1769.

O microcontrolador usado neste projeto utiliza um processamento de 32 bits. Apesar disso, ele pode processar a informação a 16 bits, ou seja, o conjunto de instruções Thumb-2; pode endereçar um espaço de 2^{32} bytes de memória, ou seja, 4Gbytes.

Apesar de disso, tem 512KB de memória flash (memória não volátil) e 64KB de RAM (memória volátil). Dentro do espaço de endereçamento encontram-se ainda os dispositivos periféricos, cada um com o seu endereço ou um conjunto de endereços.

Entre os vários dispositivos, destacam-se aqueles usados neste trabalho:

- Os pinos de entrada e saída de uso geral configuráveis ou GPIO;
- O Real Time Clock;
- Os temporizadores ou Timers;
- Inter-Integrated Circuit ou I2C;
- Serial Peripheral Interface ou SPI.

Juntamente com a sua aplicação, cada um destes dispositivos irá ser descrito mais adiante.

104. PANORÂMICA.

Este projeto é constituído quatro capítulos e um anexo, cujos conteúdos se discriminam de seguida.

a. No Capítulo 1 é feita a introdução e uma breve descrição de todo o projeto.

b. No Capítulo 2 são identificadas as funções do HAL.

c. No Capítulo 3 são identificadas as funções dos Drivers.

d. No Capítulo 4 será abordada a aplicação.

e. No anexo são identificadas todas as máquinas de estado usadas neste trabalho identificadas por **Figura A.X**.

CAPÍTULO 2

HARDWARE ABSTRACTION LAYER

201. GENERALIDADES.

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

Neste capítulo, vão ser descritas as funções dos cinco ficheiros usados e os respetivos periféricos:

- fgpio.h
  Contém todas as funções relacionadas com os pinos do microcontrolador.
- rtchall.h
  Contém todas as funções relacionadas com o Real Time Clock.
- spi.h
  Contém todas as funções relacionadas com o SPI.
- timers.h
  Contém todas as funções relacionadas com os TIMER do microcontrolador.

Neste capítulo vão ser descritas as funcionalidades do hardware fazendo sempre referência direta aos registos, mas o código utiliza o Common Microcontroller Software Interface Standard (CMSIS), fornecendo os mesmos registos dentro de estruturas de dados.

202. OS PINOS DE ENTRADA E SAÍDA.

Os pinos de entrada e saída estão organizados em portos (Port) e dentro de cada porto podem existir até 32 bits. Cada bit pode ser configurado para funcionar em uma de quatro funções, GPIO, e as restantes associadas a um dispositivo. Quando for feita a descrição do SPI e I2C, será identificada a função correspondente a cada pino.

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

Neste ponto (**202**) irá ser descrito o GPIO e as suas configurações. Cada pino pode ser configurado como entrada ou saída, e quando for entrada pode ser configurado o *pull-up*.

Para o efeito, foram construídas funções de escrita, leitura, tipo do pino, direção do pino e o *pull-up*. As operações de leitura e escrita podem ser de bit (1), byte (8), word (16) ou double word (32).

Na **figura 1** apresenta-se um esquema resumido de cada entrada/saída.

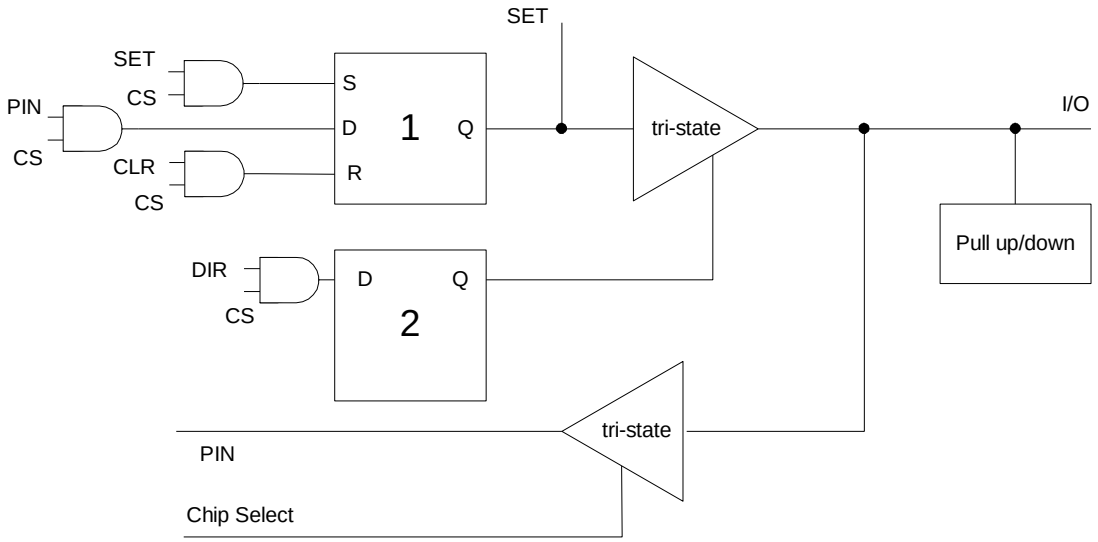

Figure 1 – Resumo de uma entrada/saída do microcontrolador.

Vai ser feita uma descrição das funções usadas, e através delas vai ser explicado o funcionamento das entradas e saídas.

- `void iotype(int port, int portbit, int type);`

Esta função define a função atribuída ao pino, primária ou GPIO, a primeira, a segunda e a terceira, conforme a **tabela 1**.

Apesar de cada função ser identificada através de dois bits, foram definidas constantes simbólicas para representar cada função do pino.

```
#define GPIO 0
#define FIRST 1
#define SECOND 2
#define THIRD 3
```

Assim, quando se chama a função é necessário fornecer o número do porto, o número do bit e a função atribuída ao pino.

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 através do registo PINSEL que se atribui as funções do pino. Como são necessários dois bits, cada registo contém 32 bits, então são necessários dois registos para cada porto.

Os registos PINSEL são seguidos, e sabendo que cada registo contém 32 bits, significa que eles estão distanciados 4 bytes. Assim, a diferença do endereço de dois registos consecutivos é igual a 4. O PINSEL0 está no endereço 0x4002 C000 e o PINSEL1 está no endereço 0x4002 C004. O PINSEL0 contém a seleção referente aos bits 0 a 15, e o PINSEL1 contém a seleção referente aos bits 16 a 31 do porto 0. Os restantes PINSEL contém a seleção referente aos outros portos.

Dentro da função, basta somar 1 ao endereço, indicando que o endereço pertence ao registo seguinte, ou seja, quatro bytes à frente.

Se for necessário passar para o porto seguinte, é necessário passar oito bytes à frente, ou seja, dois registos de 32 bits ou 8 bytes. Assim, pode-se somar o número 2, indicando que são dois registos de 32 bits.

Após o cálculo do endereço, é necessário atribuir o valor aos respetivos bits, definindo assim a função do pino.

- `void gpiodir(int port, int portbit, int dir);`

Esta função altera o registo FIODIR, um para cada porto. O primeiro registo está no endereço 0x2009 C000 e os restantes registos estão separados 0x20 ou 32 posições.

Sabendo que cada endereço contém 32 bits ou 4 bytes, basta somar 8 para passar ao registo do porto seguinte.

Para aceder aos registos, nesta função foi efetuada a diferença do endereço de dois registos FIODIR, ou seja, FIO0DIR-FIO1DIR. Usou-se a diferença para calcular o endereço dos portos 1, 2, 3 e 4. É de notar que, alternativamente, podia ter sido somado o número 8, correspondendo a $8 \times 4 = 32$.

Cada bit escrito no FIODIR corresponde ao bit que irá acionar a porta tri-state (Latch 2 da figura 1), definindo se a saída está ou não em aberto. Quando o bit é igual a um, a saída está ativada. Quando o bit é igual a 0, a saída está em aberto e o pino é entrada. Para o efeito, foram utilizadas duas constantes simbólicas, OUT e IN.

```
#define OUT 1
#define IN 0
```

- `void iopullup(int port, int portbit, int type);`

Por omissão, os pinos do microcontrolador são entradas com os *pull-up* ativados. Apesar de pouco usada, esta função é importante caso seja necessário desativar ou alterar os *pull-up*.

São usados os registos PINMODE, começando pelo PINMODE0 no endereço 0x4002 C040.

Da mesma forma que aconteceu com os registos PINSEL, estes registos estão espaçados quatro bytes. Se for necessário passar ao registo seguinte basta somar 1, significando assim que foi adicionado um conjunto de quatro bytes, e são necessários também dois registos para cada porto.

Existem dois bits para selecionar o modo, conforme a **tabela 2**.

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 – Definições dos pinos de entrada.

Por omissão, o *pull-up* está ativo, mas é possível desativar, acionar o *pull-down*, ou o repeater mode. Neste último, é possível aumentar a corrente disponibilizada pelo pino.

Para definir cada modo da tabela, foram definidas quatro constantes simbólicas.

```
#define UP 0
#define REPEAT 1
#define NONE 2
#define DOWN 3
```

- `bool gpiordpin(int port, int portbit);`

Esta função realiza a leitura do sinal existente no pino do microcontrolador. Através dos registos FIOPIN, é possível ler os 32 bits de um porto, mas nesta função é efetuada a leitura de apenas um bit.

O registo FIO0PIN está no endereço 0x2009 C014, e os restantes registos estão espaçados 0x20 ou 32 posições.

A seleção de cada um dos endereços é efetuada da mesma forma que os FIODIR. Pode ser somado o número 8, indicando que são oito conjuntos de quatro bytes, ou fazer a diferença do FIO0DIR com o FIO1DIR.

- `bool gpiordlatch(int port, int portbit);`

Esta função é semelhante à anterior, estando a diferença apenas nos registos. Através dos registos FIOSET obtém-se o resultado na saída Q da latch 1 da **figura 1**.

O endereçamento é feito de forma semelhante à função anterior. O endereço do FIO0SET é 0x2009 C018.

- `void gpiotoggle(int port, int portbit);`

Esta função recorre à função gpiordlatch. A função limita-se a escrever no pino, o contrário do que leu à saída da latch1 da **figura 1**.

- `void gpiowrbit(int port, int portbit, bool portdata);`

Esta função utiliza o registo FIOSET para ligar ou o FIOCLR para desligar o respetivo bit. O endereço do registo FIO0SET é 0x2009 C018 e do registo FIO0CLR é 0x2009 C01C.

A seleção do endereço em função do porto é realizada da mesma forma descrita na função gpiodir.

- `void gpiowr8(int port, int numbyte, int data);`

A função executa a escrita de um conjunto de oito bits no porto e no byte especificado. Através dos registos FIOxPIN0 a FIOxPIN3, é possível definir o byte utilizado.

O FIO0PIN0 está no endereço 0x2009 C014. A seleção dos registos dos restantes portos é efetuada da mesma forma que a função anterior.

- `void gpiowr16(int port, int numword, int data);`

A escrita de dezasseis bits é efetuada através dos registos FIOxPINL e FIOxPINU. Estes registos começam no endereço 0x2009 C014. A seleção dos endereços seguintes é efetua da mesma forma da função gpiodir.

- `void gpiowr32(int port, int data);`

A escrita de trinta e dois bits é efetuada através dos registos FIOxPIN. Estes registos começam no endereço 0x2009 C014. A seleção dos endereços seguintes é efetua da mesma forma da função gpiodir.

- `uint32_t gpiord32_gpio(int port);`

A leitura dos trinta e dois bits simultaneamente é efetuada a partir da latch 1 da **figura 1** pelo registo FIOPIN colocado no endereço 0x2009 C014. A seleção do endereço é efetuada da mesma forma que da função gpiodir.

203. O REAL TIME CLOCK.

O Real Time Clock é um dispositivo de muito baixo consumo, assim, ele pode ser alimentado por uma pequena pilha. Apesar disso, o bit 9 (PCRTC) no registo PCONP permite ligar o dispositivo apenas quando é necessário. Caso não seja necessário, o dispositivo mantém-se desligado reduzindo o consumo total do microcontrolador.

Na **figura 2** apresenta-se o esquema bloco deste dispositivo. Adicionalmente, ele contém ainda uma RAM de 20 bytes e um oscilador de 32KHz que irá produzir a frequência de 1Hz, necessária ao funcionamento do relógio.

As informações do relógio e do calendário encontram-se em registos diferentes. Apesar de os registos estarem consolidados, são necessárias três leituras para se obter toda a informação. Sabendo que o relógio está sempre em funcionamento, existe a probabilidade de ocorrer uma leitura errada. Isto pode acontecer se, a leitura ocorrer quando a hora for igual 23:59:59 e o dia igual 31 de dezembro. Neste caso, a forma de contornar o problema, seria efetuar duas leituras seguidas, e confirmando se são iguais.

Esta verificação seria necessária caso o projeto fosse rigorosamente dependente do relógio e do calendário.

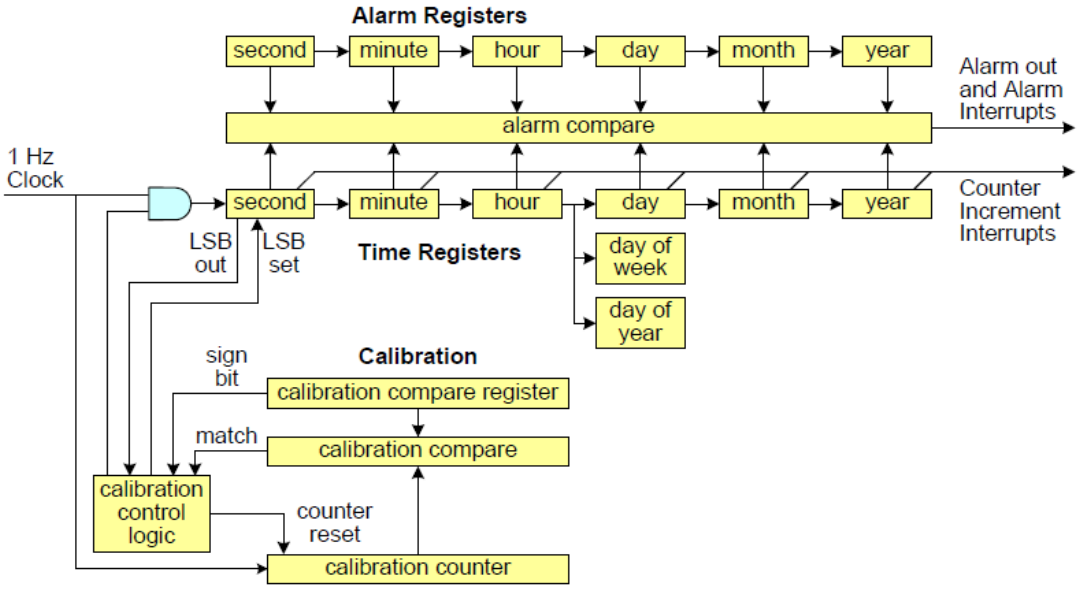

Figure 2 – Esquema bloco do Real Time Clock.

Após a alimentação estar ligada, são utilizados dois registos durante a inicialização, o CCR e o CALIBRATION, o registo de calibração.

Dentro do CCR existem três bits, o CTCRST, o CCALEN e o CLKEN.

O bit CTCRST a 1, faz reset ao divisor do oscilador interno, o oscilador que produz 1Hz a partir do oscilador de 32.768KHz.

O bit CCALEN a 1, desativa o contador de calibração.

O bit CLKEN liga ou desliga o clock do RTC.

O código utiliza uma estrutura de dados disponível no time.h. Para o efeito, é necessário transferir os dados entre as duas estruturas, a do RTC e a estrutura tm.

Quando os dados são transferidos, é necessário adaptar o valor do ano e o valor do mês.

As estruturas de dados estão representadas a seguir.

Dados do Real Time Clock:

Seconds	0 a 59
Minutes	0 a 59
Hours	0 a 23
Day of Week	0 a 6
Day of Month	1 a 31
Month	1 a 12
Year	0 a 4095
Day of Year	0 a 365

Os dados contidos no Real Time Clock foram passados para uma estrutura disponível no time.h.

struct tm {

```
  int tm_sec;     /* segundos,                  0 to 59     */
  int tm_min;     /* minutos,                   0 to 59     */
  int tm_hour;    /* horas,                     0 to 23     */
  int tm_mday;    /* dia do mês,                1 to 31     */
  int tm_mon;     /* mês,                       0 to 11     */
  int tm_year;    /* ano                        1900        */
  int tm_wday;    /* dia da semana,             0 to 6      */
  int tm_yday;    /* dia juliano,               0 to 365    */
  int tm_isdst;   /* hora de verão/inverno                  */
};
```

Funções dentro do ficheiro rtchall.h:

- `void RTC_Start(void);`

Esta função foi criada para ligar o relógio. Neste trabalho, a função é chamada após o acerto da hora ou do calendário.

Para o efeito, é ligado o bit SBIT_CLKEN dentro do registo CCR.

- `void RTC_Stop(void);`

Tal como a função anterior, esta função foi criada para desligar o relógio. Neste trabalho, a função é chamada antes de acertar a hora ou o calendário.

Para o efeito, é colocado o registo CCR igual a zero.

- `void RTC_HallInit ( void );`

Esta função é chamada pelo RTC_Init dentro do rtc.c nos DRIVERS. Ela é usada para inicializar o Real Time Clock.

Ela liga a alimentação do RTC através do registo PCONP, ativa os bits CTCRST, o CCALEN, coloca o registo CALIBRATION a zero e no fim liga o clock do RTC.

O bit CTCRST a 1, faz reset ao divisor do oscilador interno, o oscilador que produz 1Hz a partir do oscilador de 32.768KHz.

O bit CCALEN a 1, desativa o contador de calibração.

O bit CLKEN liga ou desliga o clock do RTC.

O registo CALIBRATION é usado para fazer a calibração do RTC. Neste trabalho não vai ser feita a calibração, por isso, o registo é colocado a zero.

- `void RTC_HallGetValue ( struct tm *dateTime );`

Esta função copia os dados dos registos consolidados do RTC para a estrura tm.

Faz ainda o ajuste do mês e do ano.

```
dateTime->tm_mon     = LPC_RTC->MONTH-1;
dateTime->tm_year    = LPC_RTC->YEAR-1900;
```

- `void RTC_HallSetValue ( struct tm *dateTime );`

Ao contrário da função anterior, ela copia os dados da estrutura tm para os registos do RTC.

Faz ainda o ajuste do mês e do ano.

```
LPC_RTC->MONTH =  dateTime->tm_mon+1;       // atualiza o mês
LPC_RTC->YEAR  =  dateTime->tm_year+1900;     // atualiza o ano
```

204. O SERIAL PERIPHERAL INTERFACE.

Este dispositivo utiliza quatro pinos do microcontrolador, o MOSI, o MISO, o SCLK e o SSEL.

O pino MOSI é a saída do master que vai ligar à entrada do slave.

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

O pino SCLK é a saída do master ou entrada do slave.

O pino SSEL é uma entrada do slave.

Neste projeto, o dispositivo é configurado como master e irá ligar a uma EEPROM (slave).

Ele 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.

Como master, o dispositivo deve ser capaz de selecionar o SSEL do dispositivo com quem vai comunicar. Para o efeito, foi escolhido o pino P0.23 do microcontrolador.

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.

As funções encontram-se no spi.c e o diagrama temporal do sinal SPI será apresentado no capítulo 3.

- `void SPI_Init ( void) ;`

Através dos registos PINSEL0 e PINSEL1, os pinos do SPI devem ser configurados para a terceira função. Para o efeito, usa-se a função iotype(0, 18, THIRD) em cada um dos pinos.

Sabendo que o pino SSEL é entrada do slave, e o microcontrolador funciona como master, é necessário um pino de saída capaz de acionar o pino SSEL do slave usado neste trabalho. Para o efeito, foi escolhido o pino P0.23.

É ligada a alimentação do SPI através do registo PCON, e seguidamente é configurado o clock do SPI em função do clock do processador.

Através do registo PCLOCKSEL0, configura-se uma de quatro opções, conforme a tabela 4.

PCLKSEL0 and PCLKSEL1 individual peripheral's clock select options	Function	Reset value
00	PCLK_peripheral = CCLK/4	00
01	PCLK_peripheral = CCLK	
10	PCLK_peripheral = CCLK/2	
11	PCLK_peripheral = CCLK/8, except for CAN1, CAN2, and CAN filtering when "11" selects = CCLK/6.	

Tabela 4 – Seleção da frequência.

Neste trabalho, o SPI funciona com a mesma frequência de clock ligada ao seu divisor de frequência. Sabendo que a frequência do processador é igual a 100MHz, o valor mínimo de frequência é igual a:

$$\frac{100MHz}{254} = 394{,}7KHz$$

- `void SPI_Config(int frequency, int bitData , int mode);`

Esta função configura a velocidade, o número de bits de dados e o modo de operação. O número de bits é colocado diretamente no registo SPCR nos bits 8 a 11. O modo é colocado nos bits 3 e 4 de acordo com a **tabela 3**.

A frequência é determinada pelo valor do registo SPCCR, onde é colocado o valor da divisão. Este número deve ser par e maior ou igual a 8.

O cálculo do valor do contador é efetuado da seguinte forma, considerando que o valor deve ser no máximo 255 (254 número par) e no mínimo 8:

$$\frac{\text{SystemCoreClock}}{frequência}$$

- `unsigned short SPI_Transfere(unsigned short value );`

Esta função consiste em enviar e receber um conjunto de bits através do registo SPDR, neste caso um byte.

Não existe forma para saber se a comunicação foi efetuada com sucesso.

- `int SPI_Transfer (unsigned short * txBuffer , unsigned short * rxBuffer , int lenght ) ;`

Esta função é mais elaborada do que a anterior. Ela permite enviar e receber um conjunto de bytes definido pelo parâmetro *lenght*.

A transferência de dados a enviar ou a receber é passada pelo endereço de um array ou uma estrutura.

O registo SPSR contém o estado da comunicação. Ele contém cinco bits uteis que revelam informações sobre a comunicação. O bit SPIF indica que a transferência terminou, o bit ART indica que o slave abortou a comunicação, o bit MODF indica no modo de comunicação, o bit ROVR indica que houve overrun e o bit WCOL indica que houve colisão de dados durante a escrita.

Quando o registo é lido, os bits são colocados a zero.

Para não se perder a informação, a leitura é feita para uma variável. A partir da análise da variável, o código sabe se a transmissão foi concluída ou se houve erro.

A função retorna 0 se não houver erro, ou -1 se houver erro.

Há ainda a destacar o registo SPINT. Nele existe o bit SPIF, semelhante ao bit do registo SPSR. Neste caso, o bit é ativado quando termina a comunicação, quando o bit SPIE do registo SPCR é igual a 1 e quando WCOL é igual a 1. Neste caso, seria necessário escrever 1 em cima do bit para que seja apagado.

205. O REPITITIVE INTERRUPT TIMER (RIT).

Este temporizador é constituído por um contador de 32 bits ligado à frequência de clock do processador. Ele pode ser comparado com um valor de 32 bits e pode ainda utilizar uma máscara.

São utilizados quatro registos, o RICOMPVAL, o RIMASK, o RICTRL e o RICOUNTER, visíveis na **figura 3**. A máscara não é necessária neste trabalho e fica igual a zero.

O registo RICTRL contém quatro bits que permitem configurar o temporizador:

- RITINT (bit 0)

  O bit é igual a 1 sempre o comparador indique igualdade entre o contador e RICOMPVAL. Para apagar o bit é necessário escrever 1 no próprio bit.

- RITENCLR (bit 1)

  Caso seja igual a 1, o contador é colocado a zero quando o contador é igual a RICOMPVAL. Caso seja igual a 0, o contador segue a sua contagem e não é apagado.

- RITENBR (bit 2)

  Caso seja igual a 1, o contador pára quando o processador parar.

Caso seja igual a 0, a paragem do processador não tem efeito no contador.

- RITEN (bit3)

  Este bit permite apenas ativar ou desativar o timer.

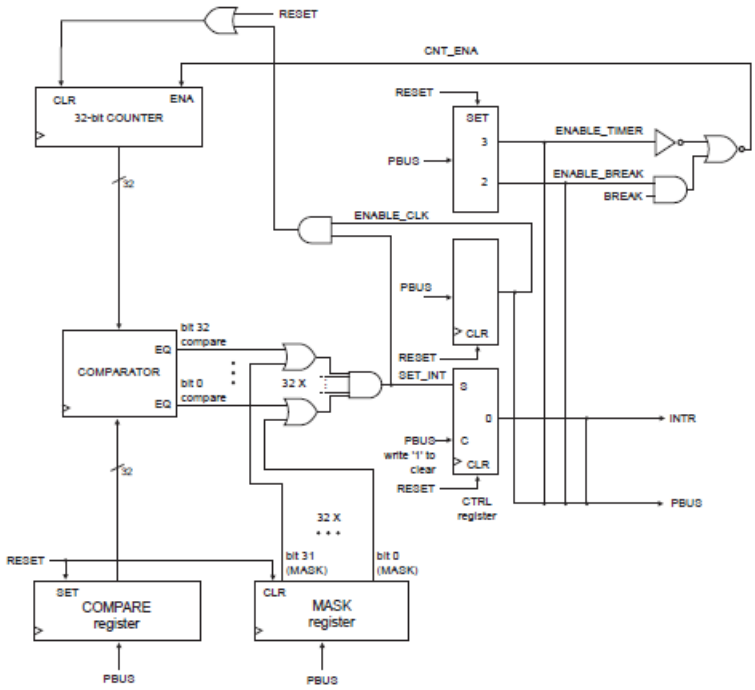

Figure 3 – Esquema bloco do RIT.

- `void Rep_timer_init(int repet);`

Tal como os outros periféricos, é necessário ligar a alimentação através do registo PCONP.

Este dispositivo utiliza um contador, como tal necessita de um clock cuja frequência é selecionada no registo PCLKSEL1. Neste caso, foi selecionada a mesma frequência de clock do processador.

Antes de atribuir qualquer valor aos registos, garantiu-se que o bit RITEN está desligado com a instrução `LPC_RIT->RICTRL &= ~(1<<RITEN);`.

O contador é colocado a zero e é atribuído o valor da divisão ao registo RICOMPVAL.

Antes de ligar o RITEN, configura-se o timer para apagar o contador quando é igual ao registo RICOMPVAL. Assim, o contador efetua uma contagem contínua entre 0 e RICOMPVAL-1.

- `void RIT_IRQACK(void);`

Esta função é chamada pelo handler dentro encoder.c nos DRIVERS, e tem a única função de apagar o bit RITINT para sinalizar o fim do pedido de interrupção e para que novos pedidos de interrupção possam ocorrer no seu devido tempo.

206. O SYSTEM TICK TIMER.

Este timer é constituído por um contador de 24 bits decrescente com a capacidade de produzir uma interrupção. Por omissão, se estiver ligado a um clock de 100MHz, produz um tempo de 10 milissegundos.

O temporizador é constituído por quatro registos, o STCTRL, o STRELOAD, o STCURR e o STCALIB, representados na **figura 4**.

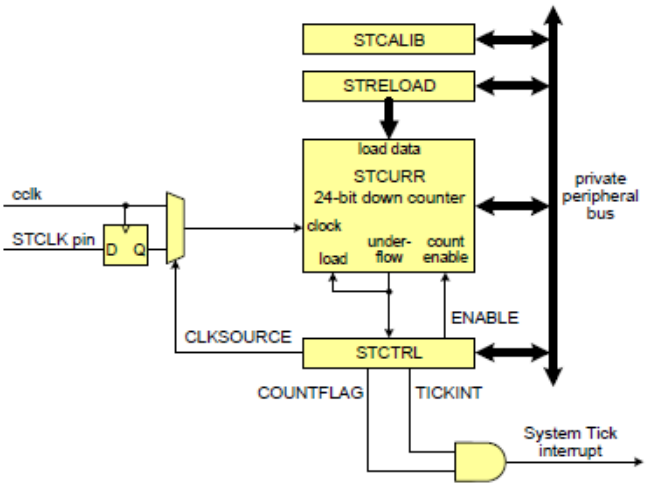

Figura 4 – Esquema bloco do SystickTimer.

- STCTRL

  Este registo contém quatro bits de controlo do timer.

  O bit ENABLE permite ou desativar o contador.

  O bit TICKINT ativa ou desativa o pedido de interrupção.

  O bit CLKSOURCE seleciona a fonte de clock. Caso seja igual a 1, utiliza o clock do CPU.

  O bit COUNTFLAG é ligado quando o contador chega a zero. É apagado ao ler o registo.

- STRELOAD

  Contém um número de 24 bits com o valor a ser colocado no contador sempre que ele chega a zero.

- STCURR

  Este registo contém o valor atual do contador, disponíveis nos bits 0 a 23.

- STCALIB

  Este registo é inicializado durante o processo de boot e contém o valor de fábrica para produzir uma interrupção a cada 10 milissegundos.

- `int WAIT_Init ( void) ;`

Foi definida uma frequência de 1000Hz para se obter o tempo de 1 milissegundo. O valor da divisão foi colocado na constante simbólica SYSTICK_FREQ.

Com esta constante simbólica, é chamada a função SysTick_Config presente no CMSIS-Core.

Caso não haja erro, a função retorna 0, caso haja erro, a função retorna -1.

- `void WAIT_Milliseconds ( uint32_t millis ) ;`

Esta função função é bloqueante e aguarda a passagem do tempo indicada por parâmetro. Durante o tempo de espera, é executado o comando __WFI(), ou seja, Wait For Interrupt. Esta instrução suspende a execução do processador até à ocorrência de uma nova interrupção.

- `uint32_t WAIT_GetElapsedMillis ( uint32_t start ) ;`

A função retorna a diferença entre o tempo passado por parâmetro e o valor atual da variável __ml.

- `void SysTick_Handler(void);`

Esta função atende o pedido de interrupção e apenas incrementa a variável __ms a cada milissegundo.

207. O TIMER 0.

O timer 0 é um dos quatro timers, 0, 1, 2 e 3. Ele é constituído por um divisor de 32 bits (Prescale Counter) e um contador de 32 bits (Timer Counter) associado a quatro registos de comparação, o MR0, o MR1, o MR2 e o MR3.

Associados a este timer estão os registos IR, TCR, TC, PR, PC, MCR, MR0, MR1, MR2, MR3, CCR, CR0, CR1, EMR e CTCR, usados na **figura 5**.

Para evitar uma exaustão, dentro de cada função, serão abordados apenas os registos usados neste trabalho.

- `void WAIT_StopwatchInit ( void) ;`

Este dispositivo necessita de ser alimentado através do registo PCONP e a origem do seu clock deve ser selecionada no registo PCLKSEL0. Neste caso, foi selecionado o mesmo valor de clock do CPU.

O registo PR contém o valor pelo qual o sinal de clock é dividido. Pretende-se uma resolução de 1 microssegundo o que equivale a 1MHz. Para o efeito, divide-se a frequência do CPU por 10^6, e o resultado menos 1 é colocado no registo PR.

Neste projeto, o valor da frequência de clock é igual a 100MHz, por isso, o valor a colocar no registo PR é igual a 99.

O registo MCR contém três bits associados a cada um dos registos MRx, o MRxI, o MRxR e o MRxS.

Neste projeto, vai ser usado o Timer 0 e os bits associados ao registo MR0.

Pretende-se construir um stopwatch, o que significa construir um contador que pára após o tempo definido na contagem. Para o efeito, são ativados os bits MR0I e MR0S, ou seja, produz um pedido de interrupção e pára.

O reset do contador é efetuado ao ligar e desligar o bit Counter Reset no registo TCR.

- `void WAIT_Stopwatch ( uint32_t waitUs ) ;`

O stopwatch baseia-se num contador que inicia a contagem a zero e termina quando chega ao MR0.

Para reiniciar a contagem, é necessário pôr o contador a zero através do bit Counter Reset no registo TCR e ligar o contador através do bit Counter Enable.

A função fica bloqueada até a contagem terminar, ou seja, quando o contador atingir o mesmo valor do registo MR0. Aqui, o contador volta a parar e produz um pedido de interrupção através do bit MR0I no registo IR.

Quando o bit MR01 é ativado, o bit é apagado, colocando 1 no próprio bit, e a função termina.

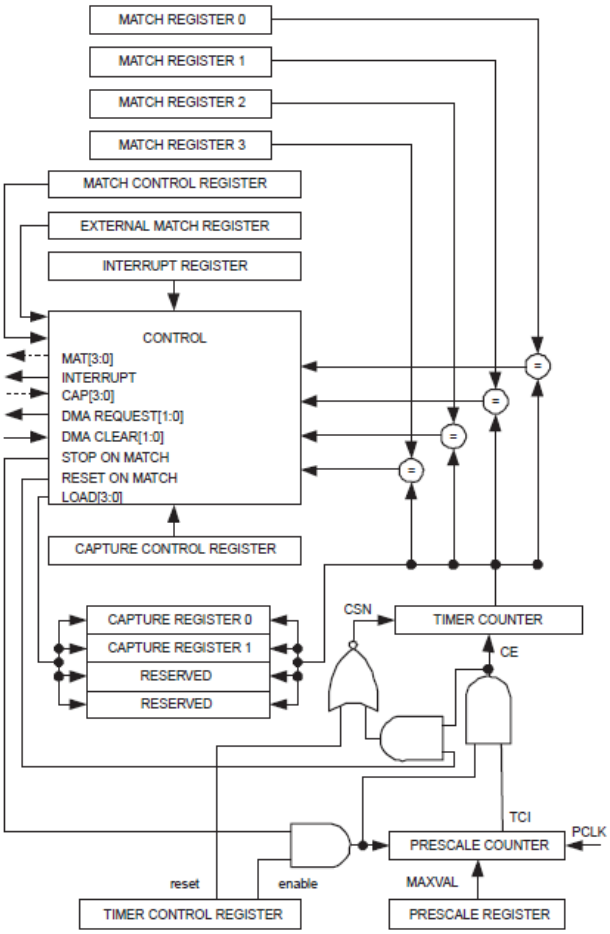

Figura 5 – Esquema bloco do Timer 0.

208. O INTER-INTEGRATED CIRCUIT (I2C).

Apesar de este trabalho usar um sensor de luz I^2C, as funções não foram desenvolvidas pelos alunos, porém, vai ser feita uma breve descrição do protocolo I^2C.

A comunicação I^2C baseia-se em dois fios, clock e dados. O protocolo I^2C pode suportar 7 ou 10 bits bits de endereçamento, dando origem a 128 ou 1024 endereços físicos. Apesar disso, alguns endereços são reservados.

Todas as comunicações são iniciadas pelo master e são feitas em conjuntos de oito bits.

Este protocolo inclui ainda um mecanismo de confirmação. Após a transmissão de cada byte, é transmitido um nono impulso de clock, a linha de dados é colocada em alta-impedância e o master aguarda uma confirmação do slave que recebeu os dados transmitidos (**Figura 6**).

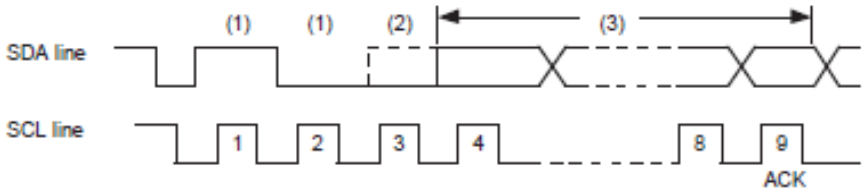

Figura 6 – Transmissão via I^2C.

CAPÍTULO 3

DRIVERS

301. GENERALIDADES.

Os drivers foram concebidos para trabalhar com as funções previamente desenvolvidas na camada HAL. Assim, a aplicação pode comunicar com os diversos dispositivos sem a necessidade de os conhecer.

As funções identificadas neste capítulo utilizam as funções desenvolvidas para a camada HAL, a fim de haver comunicação entre a aplicação e os dispositivos ligados ao microcontrolador.

Neste capítulo, vão ser abordados os seguintes drivers:

- Display LCD 16x2
- Codificador rotativo
- EEPROM CAT25128
- Led do LPCXpress
- Sensor de presença PIR
- Real Time Clock

302. DISPLAY LCD.

O display utilizado da **figura 1**, é constituído por um barramento de 8 bits de dados, um pino W/R para escrita ou leitura, um pino de Enable e um pino RS que permite diferenciar os comandos dos dados.

Ele tem ainda um pino de contraste (VEE), a alimentação do display e do Led.

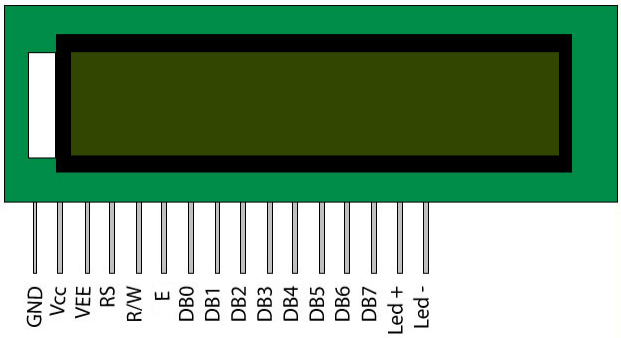

Figura 1 – Display LCD.

A comunicação vai ser realizada a quatro bits através dos pinos DB4 a DB7. A comunicação vai ser realizada apenas no modo escrita, por isso, o pino R/W é ligado a zero.

A configuração dos pinos utilizados é a descrita na **tabela 1**.

Display	L**PC1769**	Tipo
D**B7**	P2.7	GPIO out
D**B6**	P2.6	GPIO out
D**B5**	P2.5	GPIO out
DB4	P2.4	GPIO out
EN	P2.11	GPIO out
RS	P2.12	GPIO out

Tabela 1 – Ligações do LPC1769 ao display.

Pelo facto da comunicação ser feita a quatro bits, é necessário uma função capaz de transferir os dados, quatro bits de cada vez, para o exterior do LPC1769, ao mesmo tempo que ative ou desative os bits EN e RS.

Para efeito de comunicação com o display, são usados os seguintes comandos da **tabela 2**.

```
#define LCD_Clear       0b00000001 // 1,64ms todos os caracteres ASCII 'space'
#define LCD_Home        0b00000010 // 1,64ms põe o cursor no inicio da primeira linha
#define LCD_EntryMode   0b00000110 // 40µs desloca o cursor da esquerda para a direita
#define LCD_DisplayOff  0b00001000 // 40µs desliga o display
#define LCD_DisplayOn   0b00001100 // 40µs display on, cursor off, não pisca o caracter
#define LCD_Reset       0b0011     // Faz reset ao LCD
#define LCD_Set4bit     0b00100000 // 40µs 4-bits data
#define LCD_SetCursor   0b10000000 // define a posição do cursor
#define LCD_LineOne     0x00       // inicio da linha 1
#define LCD_LineTwo     0x40       // inicio da linha 2
```

Tabela 2 – Comandos do display.

A inicialização do display requer que a configuração dos pinos conforme a **tabela 1**, e que seja feita a seguinte sequência:

1. Espera no mínimo 15ms;
2. Envia o comando LCD_Reset;
3. Espera no mínimo 4,1ms;
4. Envia o comando LCD_Reset;
5. Espera no mínimo 100µs;
6. Envia o comando LCD_Reset;
7. Espera no mínimo 100µs;
8. Envia o comando LCD_Set4bit;
9. Espera no mínimo 40µs;
10. Envia o comando LCD_EntryMode;
11. Espera no mínimo 40µs;
12. Envia o comando LCD_Text_Clear;
13. Espera no mínimo 1,64ms;
14. Envia o comando LCD_DisplayOn;
15. Espera no mínimo 40µs;

Os comandos são enviados com RS igual a 0 e os dados são enviados com RS igual a 1.

Na **figura 2**, está representado o diagrama temporal da comunicação com o display.

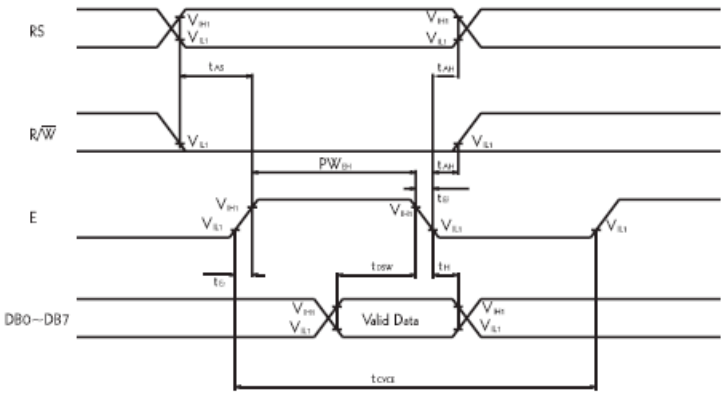

Figura 2 – Comunicação com o display.

Foram assegurados os tempos mínimos necessários apresentados na **tabela 3**. Na aplicação não existe a possibilidade de definir tempos inferiores a 1µs. Assim, o tempo mínimo atribuído é igual a 1µs.

Item		Symbol	Min.	Max.	Unit
Enable cycle time		TCYCE	500	—	ns
Enable pulse width	"High" level	PWEH	220	—	ns
Enable rise / fall time		TER, TEF	—	25	ns
Set-up time	RS, R/W, E	TAS	40	—	ns
Address hold time		TAH	10	—	ns
Data set-up time		TDSH	60	—	ns
Data delay time		TDDR	60	120	ns
Data hold time (writing)		TH	10	—	ns
Data hold time (reading)		TDHR	20	—	ns

Tabela 3 – Tempos mínimos necessários durante a comunicação.

As funções associadas ao display estão descritas a seguir:

- `void LCDText_Init(void);`

Esta função utiliza as funções da HAL para configurar os seis pinos de saída, necessários ao funcionamento do display e envia a sequência descrita anteriormente, necessária à inicialização do mesmo.

- `void LCDText_WriteChar(char ch);`

Esta função chama a função data_write e apenas envia o código ASCII do caracter desejado para o display.

- `void LCDText_WriteString ( char * str );`

Esta função recebe o endereço de uma string, enviando um código ASCII de cada vez à função LCDText_WriteChar, através da qual é enviado para o display.

- `void LCDText_SetCursor ( int row , int column ) ;`

A posição do cursor baseia-se na dimensão máxima permitida pelo LCD, ou seja, 16 colunas e 2 linhas. O valor resultante é associado ao comando LCD_SetCursor. A linha e a coluna começam com o valor 1.

Exemplo de cálculo para determinar a posição do cursor na quinta posição da segunda linha:

A posição de memória referente à primeira linha é o valor 0x00, e o valor do início da segunda linha é 0x40.

O endereço é calculado: $5 – 1 + (2 - 1) \times 0x40 = 4 + 0x40 = 0x44$

Efetua-se a operação OR com comando 1000000 e envia-se o valor final 11000100, indicando a posição de memória do display, para inserir o código ASCII do caracter desejado.

- `void LCDText_Clear (void );`

Esta função chama apenas a função cmd_write com o comando LCD_Clear.

- `void LCDText_Home(void);`

Esta função chama apenas a função cmd_write com o comando LCD_Home.

- `void LCDText_Printf (char *fmt , ... );`

Esta função é semelhante ao printf, contendo as suas funcionalidades. Para o efeito, foi usada a função vsprintf, passando os parâmetros necessários e o buffer através do qual se obtém o texto formatado.

O buffer é passado à função LCDText_WriteString, através da qual o texto é enviado ao display.

- `void nibble_write(int rs, int data);`

Apesar de a função enviar quatro bits de dados, na verdade ela manipula seis bits, os dados, o bit RS e o bit EN, conforme a **figura 2**.

Em primeiro lugar coloca os quatro bits de dados no barramento, coloca o bit RS com o devido valor, aguarda 1µs, liga o enable, aguarda 1µs, desliga o enable e aguarda outro 1µs.

Apesar de o display poder usar tempos inferiores a 1µs, a função Stopwatch tem uma resolução de apenas 1µs.

- `void byte_write(int rs, int data);`

Esta função utiliza a função niblle_write. Para o efeito, envia quatro bits de cada vez, primeiro os quatro bits mais significativos e depois os quatro bits menos significativos.

- `void cmd_write(int data);`

Esta função utiliza o byte_write, indicando o valor de oito bits e o valor de RS. Neste caso, é utilizada a constante simbólica RS_COM com o valor 0, indicando que é um comando do display.

- `void data_write(int data);`

Esta função é semelhante à anterior, mas neste caso, é utilizada a constante simbólica RS_DAT com o valor 1, indicando que são dados, ou códigos ASCII.

303. CODIFICADOR ROTATIVO.

Aqui vai ser descrito o funcionamento do codificador rotativo e o respetivo botão representado na **figura 3**. Para o efeito, vão ser construídas duas máquinas de estado que se encontram dentro do ficheiro encoder.c. A interrupção foi configurada para ser acionada cem vezes por segundo.

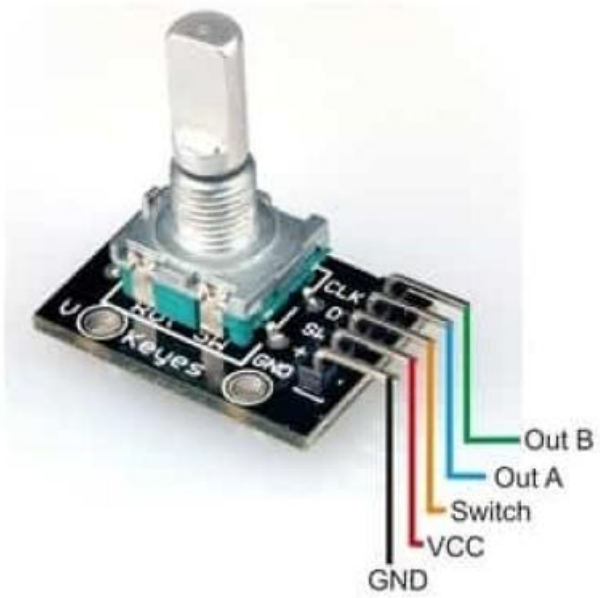

Figura 3 – Codificador rotativo.

O codificador rotativo tem dois pinos, cujo sinal está representado na **figura 4**.

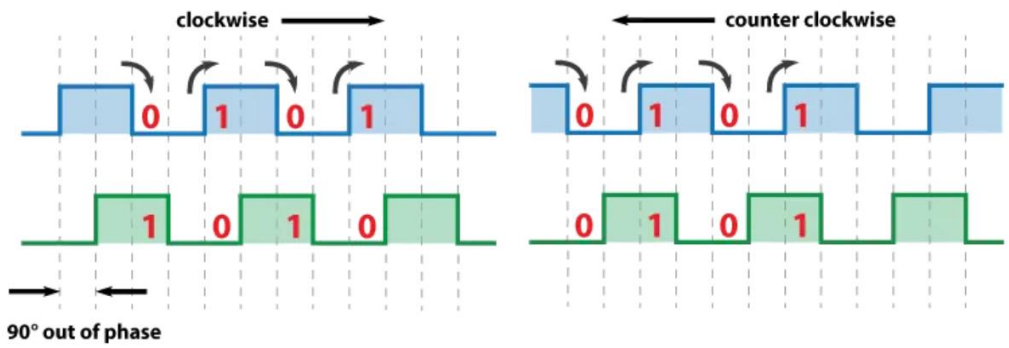

Figura 4 – Estados do codificador rotativo.

A máquina de estados foi construída para detetar as quatro mudanças de estado identificadas na **figura 4**, do 1 para o 0 ou do 2 para o 3 quando roda no sentido do relógio, e do 3 para o 2 ou do 1 para o 0 quando roda ao contrário do sentido do relógio.

A **figura A.1** apresenta a máquina de estados que está vertida em forma de código dentro do RIT_IRQHandler.

Em cada um dos quatro estados é verificado se o codificador mudou de estado, em caso afirmativo, é verificado se a relação entre o estado anterior e o estado atual corresponde a uma das quatro condições. Em função da verificação, é determinado se o codificador rodou à direita ou à esquerda.

O botão tem dois estados, ligado e desligado, mas apesar disso, foi construída uma máquina de estados com os seguintes estados:

```
BUTTON_NOTPRESSED  0 Não pressionado
BUTTON_PRESSED     1 Pressionado (transição de não pressionado para pressionado)
BUTTON_HELD        2 Mantem-se pressionado entre chamadas consecutivas
BUTTON_RELEASE     3 Libertado (transição de pressionado para não pressionado)
BUTTON_CLICKED     4 Pressionado e libertado num intervalo de tempo curto
BUTTON_DCLICKED    5 Pressionado e libertado num intervalo de tempo curto duas vezes
```

O tempo é medido através da variável button_time, incrementada em cada ocorrência de interrupção, ou seja, a cada 10 milissegundos.

Existem ainda uma constante simbólica, o TEMPO_BOTAO contendo um múltiplo de 10 milissegundos.

Foi criada a função button_pressed que retorna apenas o estado do botão, chamada dentro de cada estado.

Dentro da máquina de estados representada na **figura A.2**, verifica-se que button_time é colocado a zero sempre que o botão é pressionado.

Normalmente basta pressionar o soltar o botão para que haja mudança na máquina de estados, exceto quando o botão é pressionado duas vezes seguidas dentro do tempo estipulado em TEMPO_BOTAO. Neste caso, ocorre a mudança para BUTTON_DCLICKED.

Caso o botão fique pressionado durante um tempo superior ao TEMPO_BOTÃO, a máquina de estados muda para BUTTON_HELD.

Na aplicação, vão ser usados dois estados, o BUTTON_CLICKED e o BUTTON_DCLICKED.

O estado BUTTON_RELEASE está no código da máquina de estados, mas não foi decidido não usar por não ser necessário.

Existem ainda três funções que permitem configurar, ler e devolver o estado atual da máquina de estados, que a seguir se descrevem:

- `void ENCODER_Init ( int repet) ;`

A inicialização do codificador rotativo baseia-se em configurar o número de interrupções por segundo através da função `Rep_timer_init(int repet);` e a configuração dos três pinos de entrada, dois do codificador e um do botão.

Foram usados os pinos P2.1, P2.2 e P2.3, identificados na **figura 2** do **capítulo 1**.

- `ENCODER_ButtonValueType ENCODER_GetButton(void);`

Existem duas variáveis que contém o estado da máquina, o *state_button* e o *state_button_utilizador*. Nesta função, utiliza-se a segunda, sendo apagada após a sua leitura. Caso seja o estado BUTTON_HELD, o seu valor não é apagado na leitura, mas é apagado na máquina de estados quando o botão é solto.

- `int ENCODER_GetValue(void);`

Na máquina de estados é usada uma variável contendo a informação do sentido de rotação, que após a sua leitura é apagada. Assim, é necessário colocar o seu valor numa variável temporária a fim de ser possível retornar o seu valor após a variável *rotacao* ser apagada.

A função retorna -1 se rodar para a esquerda e retorna 1 se rodar para a direita. Retorna 0 quando o codificador não rodou.

304. EEPROM CAT25128.

Utilizou-se uma EEPROM com ligação através do SPI, descrito no capítulo 2. Para o efeito, foram utilizados quatro pinos do microcontrolador, o MOSI, o MISO, o SCLK e o CS.

Internamente, a memória é constituída pelo registo de estado e por uma EEPROM de 128Kbits, ou seja, 16Kbytes.

Existem seis comandos disponíveis, conforme a **tabela 1**.

Instruction	Opcode	Operation
WREN	0000 0110	Enable Write Operations
WRDI	0000 0100	Disable Write Operations
RDSR	0000 0101	Read Status Register
WRSR	0000 0001	Write Status Register
READ	0000 0011	Read Data from Memory
WRITE	0000 0010	Write Data to Memory

Tabela 1 – Comandos da EEPROM.

Nas funções desenvolvidas, vão ser usados principalmente os comandos WREN, RDSR, READ e WRITE.

A leitura ou a escrita na EEPROM é feita a oito bits, indicando sempre o endereço com um conjunto de dezesseis bits. Apesar disso, apenas são usados catorze bits no endereçamento da memória, ou seja, 2^{14} = 16384 bytes.

Como exemplo, na leitura da EEPROM envia-se o comando READ de oito bits, seguido do endereço de dezasseis bits, envia-se ainda um conjunto de oito bits sem significado. Durante os últimos oito bits, a memória retorna os oito bits de dados.

A visualização deste processo pode ser feita através das **figuras 5**, **6** e **7**. Foram necessárias três figuras devido à limitação de canais do osciloscópio.

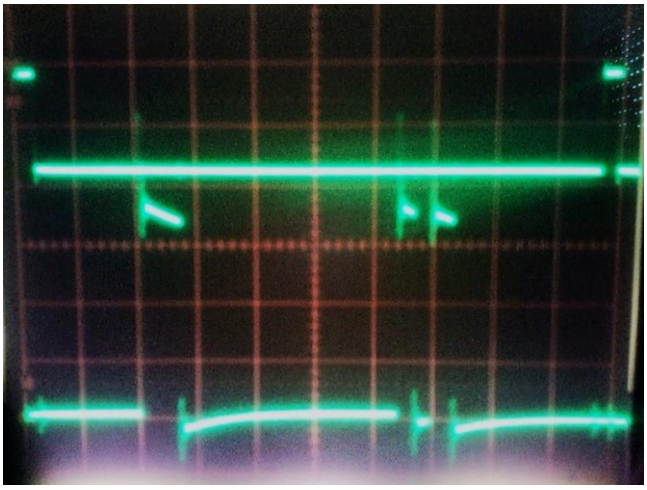

Figura 5 – Sinal CS e MOSI.

Na **figura 5,** observa-se o valor 00000011 do comando READ enviado em primeiro lugar.

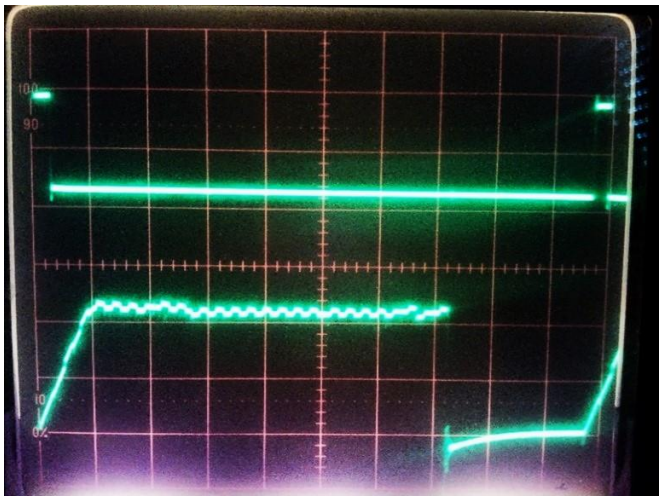

Figura 6 – Sinal CS e MISO.

Na **figura 6,** observa-se que o retorno do sinal não está bem definido no início, aparecendo vestígios do sinal de clock, consequência de estar em alta-impedância, até ao momento que retorna os valores a zero, momento no qual a EEPROM devolve o valor lido.

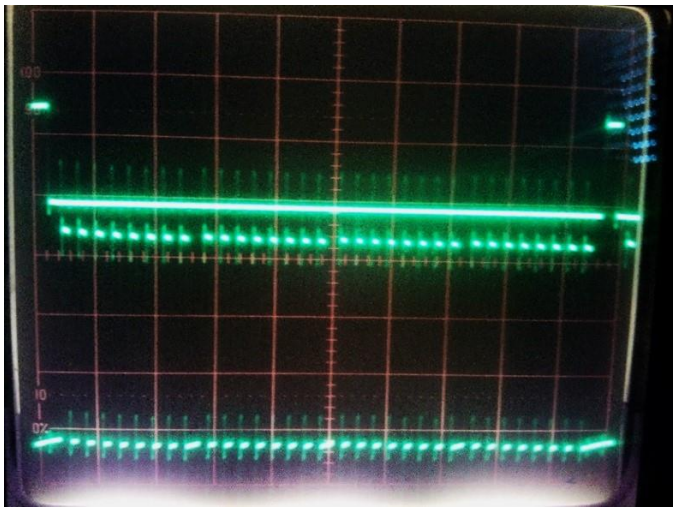

Figura 7 – Sinal CS e SCLK.

Na **figura 7,** observam-se oito ciclos de clock, repetidos quatro vezes, indicando o envio de quatro bytes. Simultaneamente com o último byte enviado, a EEPROM devolve o valor lido.

O comando WRITE produz um diagrama temporal semelhante, exceto no MISO que se mantém a alta-impedância.

Os comandos WREN e WRDI enviam apenas um byte correspondente ao comando.

Os comandos RDSR e WRSR enviam dois bytes, o comando e os dados. No comando RDSR, o segundo byte retorna pelo MISO, no comando WRSR, o MISO mantém a alta-impedância.

- `void eeprominit();`

Foi atribuído um valor como assinatura da EEPROM, definido na constante simbólica ASS_ADDR.

Nesta função, é lido o primeiro byte da EEPROM. Se o valor lido é diferente da assinatura, é escrito o valor da assinatura na EEPROM, o número de registos é colocado a zero e é definido o valor da luz por omissão.

- `int CAT25128_WriteByte (unsigned char value, unsigned int address);`

A escrita de um byte na EEPROM necessita de quatro bytes, o código WRITE, dois bytes de endereço e o byte de dados. Para isso, foi preparado um array de dimensão 4. Como o SPI recebe simultaneamente com o envio, foi preparado um array com a mesma dimensão para os dados recebidos. O array de receção vai ser descartado pelo facto de os dados recebidos terem uma informação sem qualquer significado.

Na primeira posição do array coloca-se o código WRITE, na segunda coloca-se o byte mais significativo do endereço, na terceira coloca-se o byte menos significativo do endereço e na quarta coloca-se o byte a escrever.

A operação de escrita demora 5 milissegundos, mas a verificação é feita através do bit RDY do STATUS REGISTER.

- `int CAT25128_ReadByte (unsigned char * dstValue , unsigned int address );`

A leitura é semelhante à escrita. Aqui muda o código, na quarta posição do array de transmissão coloca-se um valor qualquer e recebe-se o valor lido na quarta posição do array de receção.

- `int CAT25128_WriteBlock ( void * srcAddr , unsigned int size , unsigned int address );`

A memória apresenta a opção de escrever 64 bytes seguidos, desde que estejam dentro de um conjunto de endereços pertencentes aos primeiros seis bits, em hexadecimal, os primeiros seis bits devem conter o valor entre 0x00 e 0x3F.

Foi criado um buffer com a dimensão do número de bytes a transmitir mais três unidades para guardar o código WRITE e o endereço.

Foi utilizado o valor 0x3F para retirar os três bits menos significativos e assim calcular o número de bytes disponíveis dentro da mesma página de endereços.

Caso haja mudança de página, enviam-se os bytes da primeira página e depois enviam-se os dados da segunda página.

- `int CAT25128_ReadBlock ( void * dstAddr , unsigned int size , unsigned int address );`

Esta função funciona dentro do mesmo princípio da anterior. Se o número de bytes estiver dentro da mesma página, os bytes são lidos todos seguidos. Se houver mudança de página, são lidos os bytes da primeira página e depois são lidos os bytes da segunda página.

Os dados são guardados no array de receção.

- `void write_disable(void);`

Esta função limita-se a enviar um byte com o código WRDI, chamando a função SPI_Transfer.

- `void write_enable(void);`

Semelhante à anterior, esta função limita-se a enviar um byte com o código WREN, chamando a função SPI_Transfer.

Sempre que é realizada uma escrita na EEPROM é necessário chamar esta função.

- `void dados_write(unsigned short data, unsigned int address);`

Esta função é semelhante à função CAT25128_WriteByte, mas não retorna erro de escrita.

- `unsigned short dados_read(unsigned int address);`

Esta função é semelhante à função CAT25128 ReadByte, mas não retorna erro de escrita. A função retorna os dados lidos.

305. LED DO LPCXPRESSO

O LED está ligado diretamente ao pino P0.22, através do qual é possível verificar ou simular uma luz. Para o efeito, foram construídas algumas funções que se encontram dentro do ficheiro led.c e utilizam as funções do ficheiro fgpio.c, que a seguir se descrevem:

- `void LED_Init ( bool state);`

Esta função utiliza as funções da HAL para configurar o pino P0.22 como entrada GPIO e define um estado inicial.

- `bool LED_GetState ( void);`

Esta função chama gpiordbit e retorna o estado do respetivo bit.

- `void LED_On( void);`

Esta função chama gpiowrbit e coloca o respetivo bit a 1.

- `void LED_Off (void);`

Esta função chama gpiowrbit e coloca o respetivo bit a 0.

- `void LED_Toggle (void);`

Esta função chama gpiotoggle e inverte o respetivo bit.

306. SENSOR DE PRESENÇA PIR.

O sensor de presença baseia-se num dispositivo que fornece o valor 1 durante dois segundos, em função da presença de uma pessoa. Após dois segundos, o dispositivo volta a colocar o valor 0 na saída.

Este driver é constituído por duas funções, o PIR_Init e o PIR_GetPresence.

- `void PIR_Init (void);`

  Esta função utiliza as funções iotype e gpiodir dentro do fgpio.c, e configura o pino P0.21 como entrada GPIO.

- `bool PIR_GetPresence (void);`

  Esta função utiliza gpiordpin para ler o valor lógico do pino P0.21.

307. REAL TIME CLOCK.

O Real Time Clock faz parte do microcontrolador e tem funções especificas dentro da camada HAL, descrito no **capítulo 1**. A maioria das funções chamam as funções existentes na HAL, exceto as funções que definem ou leem os segundos, e que fazem parte da biblioteca time.h.

- `void RTC_Init ( time_t seconds);`

A inicialização do RTC é feita na camada HAL, por isso, é chamada a função RTC_HallInit, descrita no **capítulo 1**.

Esta função permite ainda definir um valor para os segundos.

- `void RTC_GetValue ( struct tm *dateTime);`

Esta função chama a função RTC_HallGetValue descrita no **capítulo 1**.

- `void RTC_SetValue ( struct tm *dateTime );`

Esta função chama a função RTC_HallSetValue descrita no **capítulo 1**.

- `void RTC_SetSeconds ( time_t seconds );`

Esta função utiliza a função time existente dentro da biblioteca time.h e define a hora de acordo com os segundos a partir do ano 1900.

A função não foi utilizada na aplicação.

- `time_t RTC_GetSeconds ( void);`

Esta função utiliza a função time existente dentro da biblioteca time.h e obtém os segundos a partir do ano 1900.

A função não foi utilizada na aplicação.

CAPÍTULO 4

APLICAÇÃO

401. GENERALIDADES.

A aplicação baseia-se no circuito da **figura 1**. O código aqui descrito é utilizado para produzir um produto com todos os componentes e gerir a utilização da iluminação em função de um detetor de presença e da luz ambiente. É utilizada uma EEPROM que armazena todas as informações necessárias.

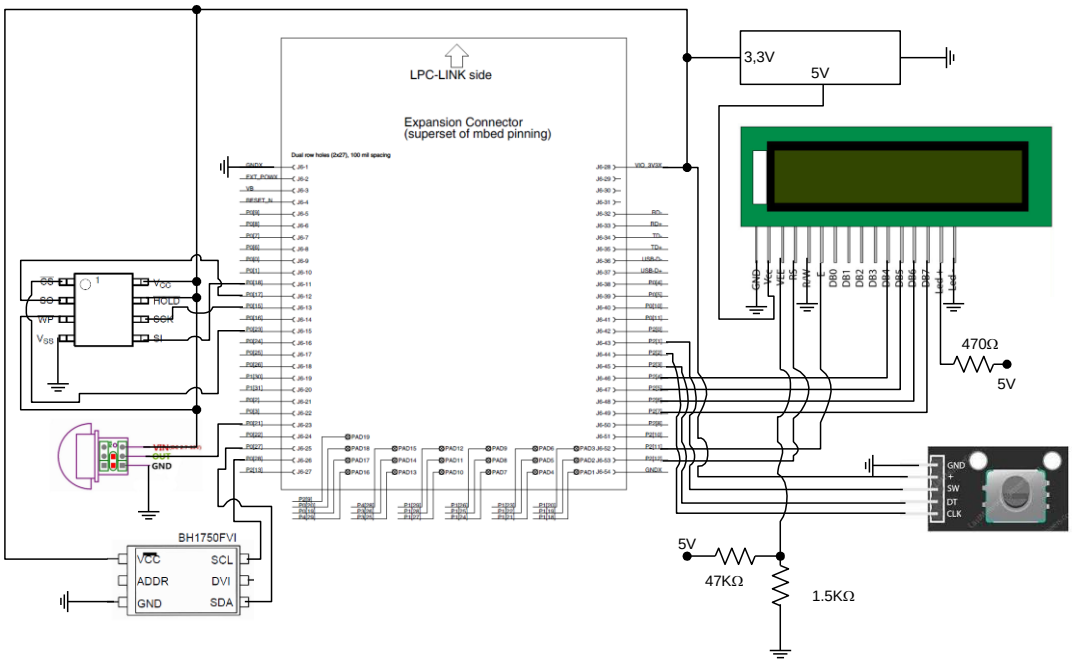

Figura 1 – Circuito da aplicação com o microcontrolador LPC1769.

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

Em cada um dos pontos, vai ser descrita a sua finalidade na aplicação e o funcionamento da respetiva máquina de estado.

Todas as máquinas de estado estão representadas no **anexo A** e são identificadas como **figura A.X**.

402. INÍCIO DA APLICAÇÃO.

O início da aplicação foi colocado no ficheiro projeto.c. Aqui são feitas todas as inicializações dos dispositivos usados neste projeto.

É aqui que se encontra a primeira e a mais simples máquina de estado, aquela que muda entre o modo normal e o modo de manutenção.

403. MODOS DE OPERAÇÃO.

Foi implementada uma máquina de dois estados, cuja função é 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, esta máquina de estados comuta imediatamente para o outro modo.

Esta máquina de estados está representada na **figura A.3**.

404. MODO NORMAL.

O modo normal é constituído por seis estados: IDLE, ACENDE_ENTRY, ACENDE, AFIXAR_ENTRY, AFIXAR e EXIT_NORMAL. O modo EXIT_NORMAL é usado com a única finalidade de sair do ciclo while e abandonar a máquina estados.

Cada um dos estados vai ser descrito a seguir, conforme a **figura A.4**:

- `IDLE`

Este é o estado inicial e aguarda a presença de uma pessoa. Caso haja presença, verifica se a luz ambiente é menor do que o valor guardado na EEPROM.

A mudança para o próximo estado e o registo na EEPROM ocorrem se as condições anteriores forem verdadeiras.

Caso haja uma resposta do botão, ele pode mostrar a hora ou sair do modo normal.

- `ACENDE_ENTRY`

Este estado acende o LED e passa imediatamente ao próximo estado.

- `ACENDE`

Neste estado, o LED encontra-se aceso e fica a aguardar uma resposta do botão ou a ausência de uma pessoa.

Caso haja uma resposta do botão, ele pode mostrar a hora ou sair do modo normal.

- `AFIXAR_ENTRY`

É um estado intermédio que permite obter o valor atual em milissegundos.

- *AFIXAR*

Este estado é obtido após um click do botão, onde é apresentada a hora e a data atual durante cinco segundos, conforme a **figura 2**.

Após os cinco segundos, é verificado se existe presença e decidido qual o próximo estado.

Figura 2 – Estado AFIXAR.

405. MODO DE MANUTENÇÃO.

Através de um menu apresentado pelo LCD, o utilizador pode selecionar uma de várias opções; o ajuste horário, o ajuste de luz, navegação pelos registos e eliminação dos mesmos, bem como a opção EXIT, que retorna ao modo normal.

Para implementar a máquina de estados do menu, optou-se por utilizar o estado seletivo e o estado de execução, como por exemplo o CLOCK e o CLOCK_EXEC.

Neste caso, o estado CLOCK_EXEC iniciará uma nova máquina de estados correspondente à execução do ajuste horário, e o CLOCK apresenta a opção no display e permite a mudança do menu.

Cada um dos estados vai ser descrito a seguir, conforme a **figura A.5**:

- *CLOCK*

Este é o primeiro estado que apresenta no display a opção de acertar o relógio, conforme a **figura 3**. Caso seja rodado o codificador, irá passar ao estado CALENDAR ou EXIT_MENU. O click do botão fará entrar no estado de execução, ou seja, CLOCK_EXEC.

Figura 3 – Estado CLOCK.

- *CLOCK_EXEC*

Este estado tem a função de chamar a função setRelogio, onde se encontra uma nova máquina de estados, a de, como o nome indica, acertar o relógio.

No retorno da função setRelogio, a máquina regressa ao estado anterior, ou seja, CLOCK.

- *CALENDAR*

Este estado apresenta no display a opção de acertar o calendário, conforme a **figura 4**, e permite mudar para o estado vizinho, ou entrar no estado que dá acesso à função que acerta o relógio, ou seja, o CALENDAR_EXEC.

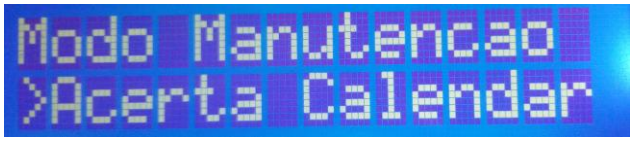

Figura 4 – Estado CALENDAR.

- *CALENDAR_EXEC*

Este estado chama diretamente a função de acertar o calendário, ou seja, o setDate.

Ao sair da função, a máquina é encaminhada imediatamente para o estado anterior, ou seja, CALENDAR.

- *LIGHT*

Este estado apresenta no display a opção de acertar a intensidade da luz ambiente, permite passar ao estado vizinho ou entrar no modo de acerto da luz.

Figura 5 – Estado LIGHT.

- *LIGHT_EXEC*

Ao entrar neste estado, é chamada a função setLight e, ao sair desta, regressa ao estado anterior, ou seja, LIGHT.

- *REGISTOS*

Neste estado, é apresentada a opção de ler os registos gravados na EEPROM, conforme a **figura 6**, sendo possível ter acesso à data e à hora de todas as vezes que a luz foi acesa.

Ao fazer click no botão, passa-se ao estado REG_EXEC.

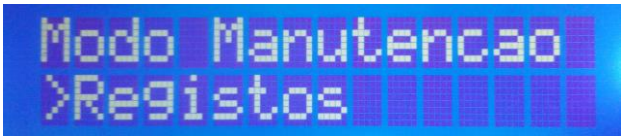

Figura 6 – Estado REGISTOS.

- *REG_EXEC*

Este estado chama a função readReg, onde se encontra outra máquina de estados que permite a visualização de cada um dos registos.

Ao retornar da função, regressa ao estado anterior, ou seja, REGISTOS:

- *CLR_REGISTO*

Este estado apresenta a opção de apagar registos, conforme a **figura 7**. Caso seja feito click no botão, passa ao estado CLR_REG_EXEC.

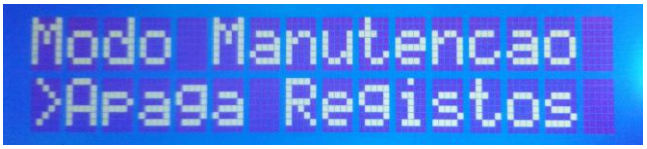

Figura 7 – Estado CLR_REGISTO.

- *CLR_REG_EXEC*

Este estado chama a função clearReg, onde é perguntado se deseja mesmo apagar os registos. No retorno da função, regressa ao estado anterior.

- *EXIT_MENU*

Este é o último estado, e com um click, permite abandonar o estado de manutenção e regressar à máquina de estados anterior.

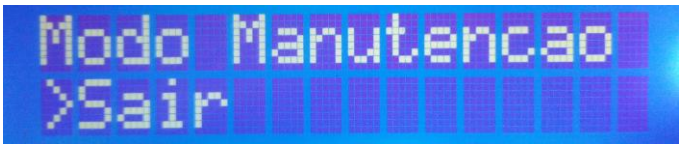

Figura 8 – Estado EXIT_MENU.

406. ACERTO DO RELÓGIO.

Dentro do ficheiro útil.c encontram-se duas máquinas de estado, setDate para o acerto da data e setRelogio para o acerto da hora.

A máquina de estados recorre à função textoRel com a finalidade de apresentar no display as informações sobre o relógio e o campo atualizado, conforme a **figura 9**.

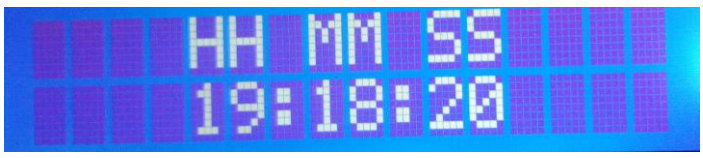

Figura 9 – Acertar o relógio.

É utilizada a variável cursor e a constante simbólica CURSOR. A primeira varia de acordo com o estado da máquina, de modo a piscar o campo que atualizado. A segunda indica o início das informações do relógio.

A variável timeout é usada para determinar a saída em caso de inatividade, e a variável blink é usada para determinar o tempo de piscar do campo atualizado. Associada à variável blink, existe a variável blinkTog que alterna o valor do campo com um espaço vazio.

Antes de começar a máquina de estados é necessário inicializar as variáveis, escrever na primeira linha do display a posição de cada campo a atualizar e parar o relógio.

Os três primeiros estados necessitam de ler o valor do codificador para decidir se incrementam ou decrementam o campo atualizado.

Cada um dos estados vai ser descrito a seguir, conforme a **figura A.6**:

- *HORA*

Dentro deste estado, é definida a posição do cursor no display e é feita a atualização do campo tm_hour em função do valor retornado pelo codificador.

É chamada a função textoRel que atualiza o display.

Após um click do botão, passa ao estado seguinte. Se houver timeout, passa ao estado UPDATE e sai.

- *MIN*

Dentro deste estado, é definida a posição do cursor no display e é feita a atualização do campo tm_min em função do valor retornado pelo codificador.

É chamada a função textoRel que atualiza o display.

Após um click do botão, passa ao estado seguinte. Se houver timeout, passa ao estado UPDATE e sai.

- *SEG*

Dentro deste estado, é definida a posição do cursor no display e é feita a atualização do campo tm_sec em função do valor retornado pelo codificador.

É chamada a função textoRel que atualiza o display.

Após um click do botão ou se houver timeout, passa ao estado UPDATE e sai.

- *UPDATE*

Neste estado, é atualizado e ligado o RTC, é apagado o display e é abandonada a máquina de estados, regressando ao menu de manutenção.

407. ACERTO DO CALENDÁRIO.

Pelo fato de esta função acertar o mesmo RTC, é semelhante à anterior, com o mesmo tipo e a mesma função de variáveis, mas voltada às informações do calendário.

Assim, no início da função é desligado o RTC, são definidas as variáveis blink, blinkTog e timeout, tal como na função anterior.

São colocadas no display as informações sobre os quatro campos atualizados, um em cada estado, conforme a **figura 10**.

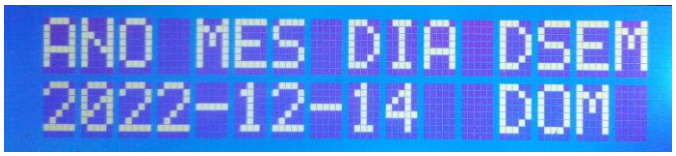

Figura 10 – Acertar o calendário.

São usadas as funções numDias e textoCal. A primeira calcula o número de dias do mês selecionado, a segunda apresenta no display a informação atualizada.

Cada um dos estados será descrito de seguida, conforme a **figura A.7**:

- *ANO*

Dentro deste estado, é definida a posição do cursor no display e é feita a atualização do campo tm_year em função do valor retornado pelo codificador.

É chamada a função textoCal que atualiza o display.

Após um click do botão, passa ao estado seguinte. Se houver timeout, passa ao estado UPDATE e sai.

- *MES*

Dentro deste estado, é definida a posição do cursor no display e é feita a atualização do campo tm_mon em função do valor retornado pelo codificador.

É chamada a função textoCal que atualiza o display.

Após um click do botão, passa ao estado seguinte. Se houver timeout, passa ao estado UPDATE e sai.

- *DIA*

Dentro deste estado, é definida a posição do cursor no display e é feita a atualização do campo tm_mday em função do valor retornado pelo codificador.

É chamada a função textoCal que atualiza o display.

Após um click do botão, passa ao estado seguinte. Se houver timeout, passa ao estado UPDATE e sai.

- *DSEMANA*

Dentro deste estado, é definida a posição do cursor no display e é feita a atualização do campo tm_wday em função do valor retornado pelo codificador.

É chamada a função textoCal que atualiza o display.

Após um click do botão ou se houver timeout, passa ao estado UPDATE e sai.

- *UPDATE*

Neste estado, é atualizado e ligado o RTC, é apagado o display e é abandonada a máquina de estados, regressando ao menu de manutenção.

408. ACERTO DA INTENSIDADE LUZ.

O acerto da intensidade da luz baseia-se na alteração do valor guardado na EEPROM. Este valor é constituído por um número de 16 bits, guardado em dois endereços consecutivos da EEPROM, neste caso, nos endereços 1 e 2, guardados na constante simbólica LUZ, conforme a **figura 14**.

Existem duas variáveis estáticas, light e sensorlight. A primeira contém o valor guardado na EEPROM e a segunda contém o valor fornecido pelo sensor de luz. Assim, elas podem ser usadas em qualquer uma das máquinas de estados.

Existem duas funções associadas à luz, getLight e setLight. A primeira obtém o valor guardado na EEPROM e coloca-o na variável light, a segunda é constituída por uma máquina de estados com a finalidade de ajustar e guardar o valor na EEPROM.

O texto é apresentado no display conforme a **figura 11**.

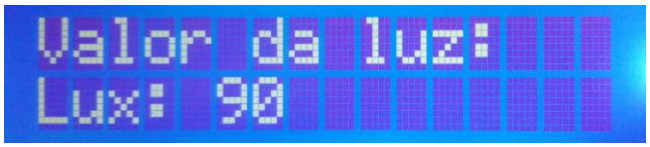

Figura 11 – Definir a intensidade da luz.

Cada um dos estados vai ser descrito a seguir, conforme a **figura A.8**:

- *ENTRY*

Este é o primeiro estado da máquina e apenas vai buscar o valor da luz à EEPROM, escreve a primeira linha de texto no display e passa ao estado seguinte.

- *ACERTA*

É neste estado que é feita a seleção do valor da intensidade da luz. Quando o codificador roda, a variável luz incrementa ou decrementa.

Após o click do botão, passa ao estado ATUALIZA.

- *ATUALIZA*

Este é o último estado da máquina de estados. Aqui é feita a atualização da EEPROM, o display é apagado e passa ao estado EXIT para que possa abandonar a máquina de estados.

409. APAGA REGISTOS.

A função chama-se clearReg e contém uma máquina de três estados, além do estado EXIT, que permite o abandono da máquina de estados.

Dentro da máquina será confirmada a intenção de apagar os registos, conforme a **figura 12**, e em caso afirmativo, é colocado o valor 0 nos endereços 3 e 4 da EEPROM, conforme a **figura 14**.

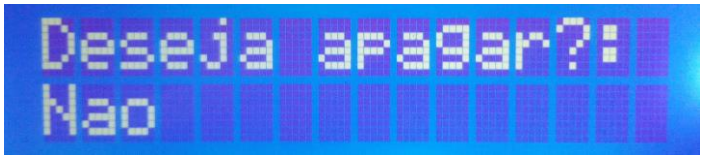

Figura 12 – Apagar registos.

Cada um dos estados vai ser descrito a seguir, conforme a **figura A.9**:

- *ENTRY*

Neste estado, é colocada a pergunta no display e passa imediatamente ao estado seguinte.

- *PERGUNTA*

Este estado, permite selecionar a respostas SIM ou NÃO. Após o click, em função da resposta selecionada, passa ao estado CLEAR ou abandona a máquina de estados.

- *CLEAR*

Neste estado, é colocado 0 nos endereços de memória 3 e 4. Apesar de a informação dos registos anteriores permanecer, é colocada a informação de que não existem registos. Quando o primeiro registo voltar a ser inserido, recomeçará a contagem.

410. MOSTRA REGISTOS.

A máquina de estados contém três estados, além do estado de saída EXIT; encontra-se dentro da função readReg.

São lidos os dois bytes contendo o valor do número de registos. O primeiro valor apresentado no display corresponde ao último registo efetuado na EEPROM.

A **figura 13** apresenta o último registo efetuado.

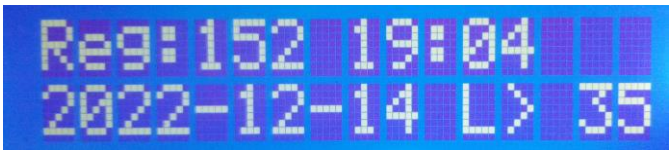

Figura 13 – Mostrar registos.

Cada um dos estados vai ser descrito a seguir, conforme a **figura A.10**:

- *ENTRY*

Este é o primeiro estado e tem a finalidade de apagar o display e verificar se o número de registos é igual a zero.

Se o valor for igual a zero, passa ao estado SEMREG. Se o valor for diferente de zero, passa ao estado PROCURA.

- *PROCURA*

Neste estado, os registos podem ser consultados, começando pelo último colocado na EEPROM.

Quando os registos são percorridos, ao atingir o último registo, ele passa para o primeiro, e vice-versa.

Para apresentar o conteúdo dos registos, é chamada a função showReg.

Após o click, passa ao estado EXIT e abandona a máquina de estados.

- *SEMREG*

Este estado é usado para apresentar no display a mensagem “Não existem registos”.

Após o click, a máquina de estados é abandonada.

411. DADOS DA EEPROM.

Para efeitos de organização dos dados na EEPROM, foi decidido guardar os dados conforme se apresentam na **figura 14**.

Em primeiro lugar, é guardada a assinatura da memória, baseada num número de oito bits.

O valor da sensibilidade da luz é guardado nos registos 0x0001 e 0x0002.

Imediatamente a seguir está guardado o número total de registos. Para o efeito, são usados dois bytes. Assim, o número pode ser maior do que 255.

Os registos são guardados a partir do endereço 0x0005 inclusive. Após o endereço 0x000D aparece o segundo registo, e assim sucessivamente.

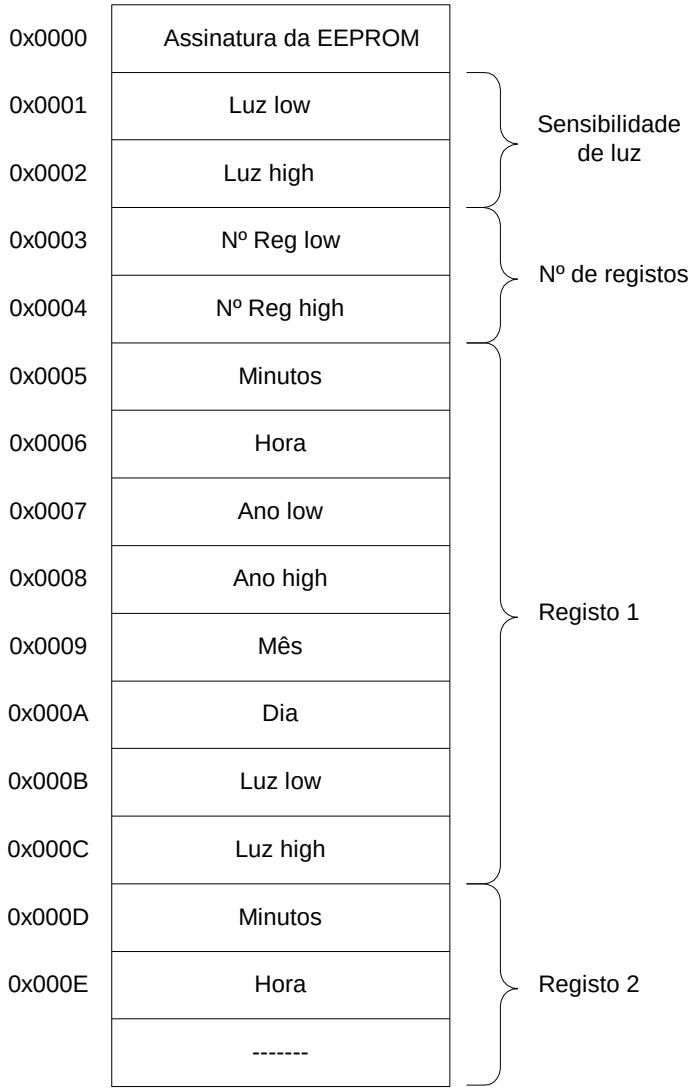

Figura 14 – Organização dos dados na EEPROM.

ANEXO A

SEPARADOR 1

ANEXO A

MÁQUINAS DE ESTADO

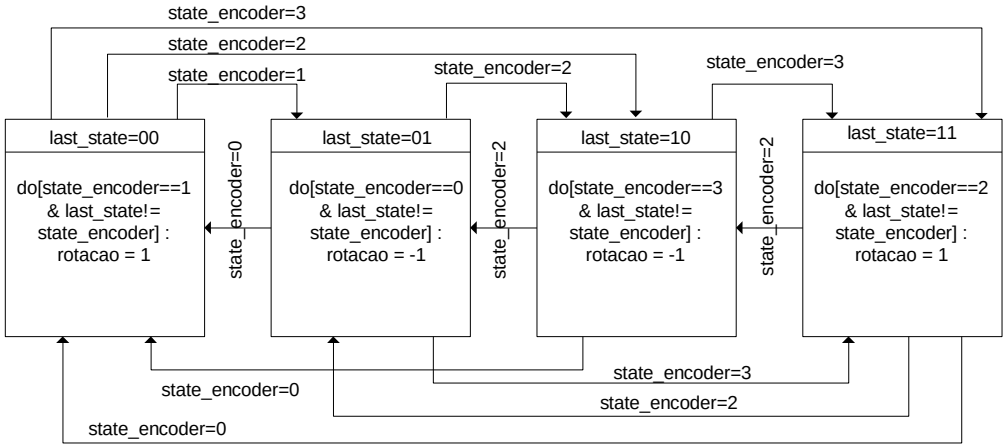

Figura A.1 - Máquina de estados do codificador rotativo.

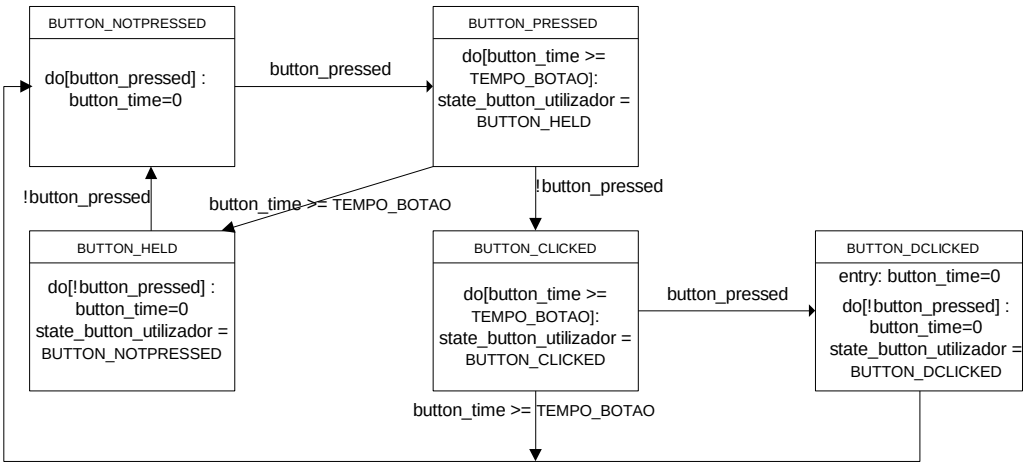

Figura A.2 - Máquina de estados do botão.

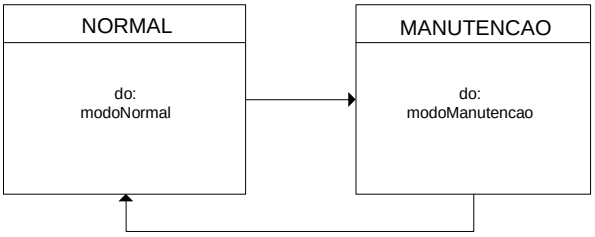

Figura A.3 - Máquina de estados normal/manutenção.

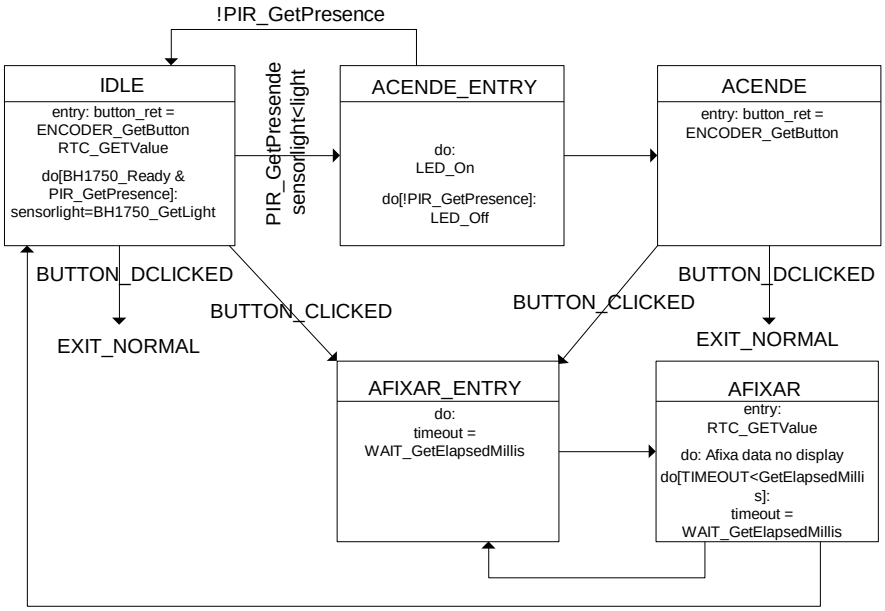

Figura A.4 - Máquina de estados do modo normal.

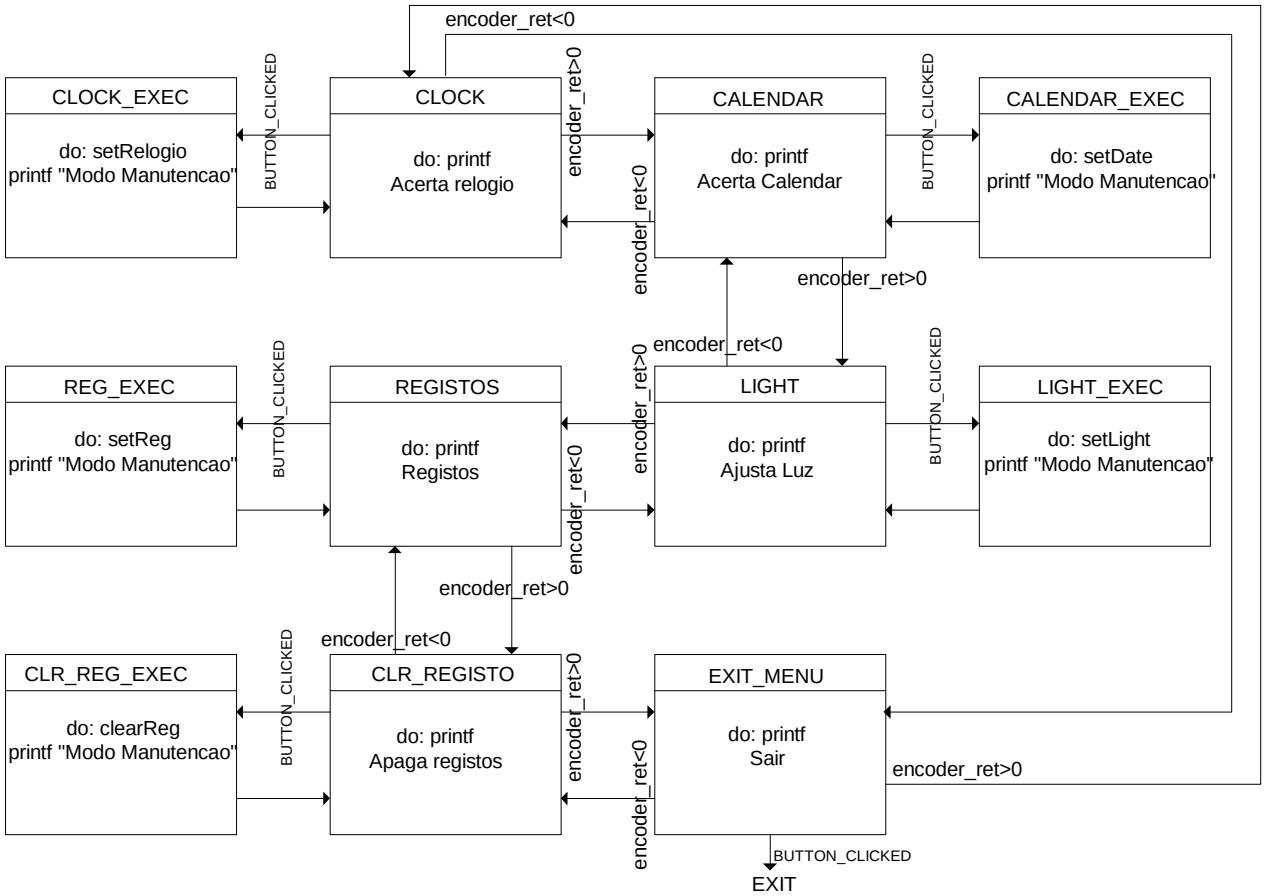

Figura A.5 - Máquina de estados do modo de manutenção.

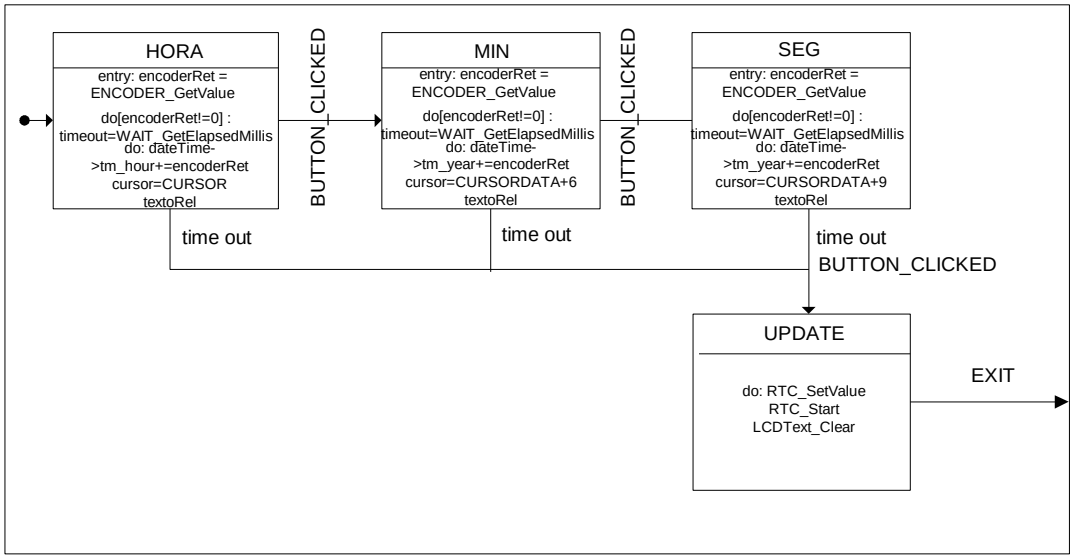

Figura A.6 - Máquina de estados de acertar a hora.

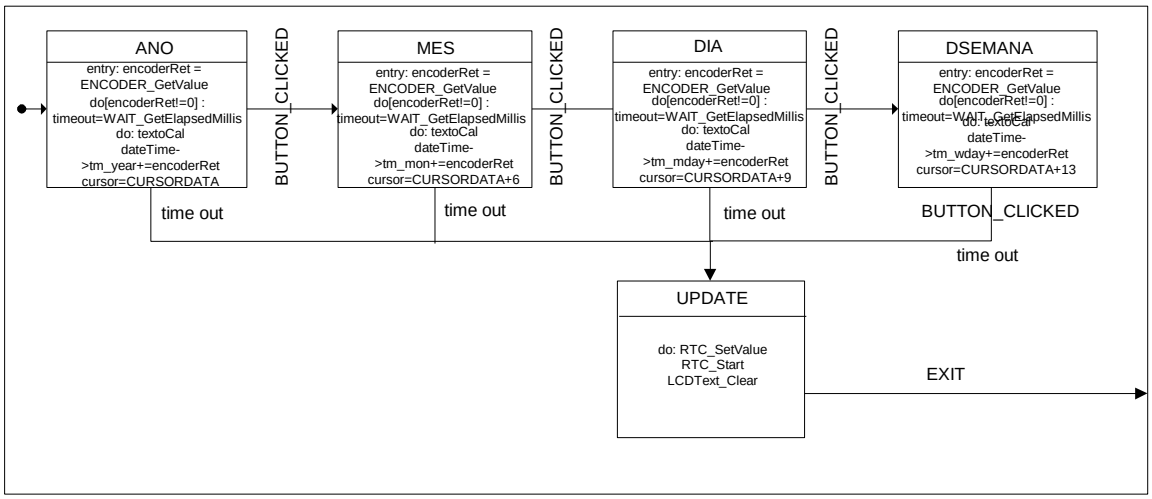

Figura A.7 - Máquina de estados de acertar o calendário.

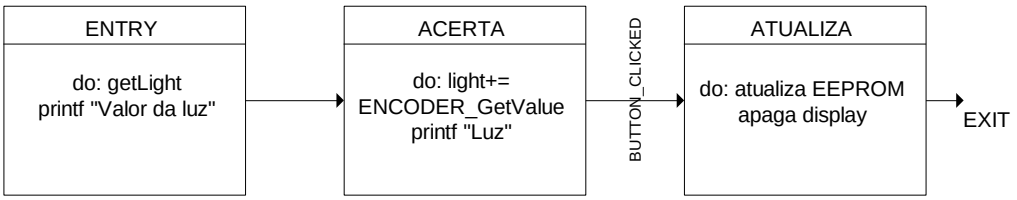

Figura A.8 - Máquina de estados de acertar a luz.

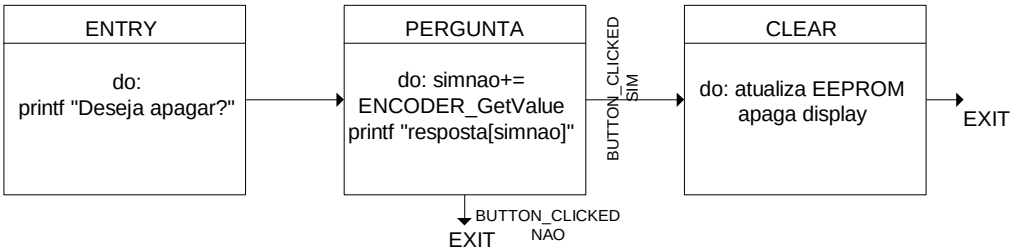

Figura A.9 - Máquina de estados de apagar registos.

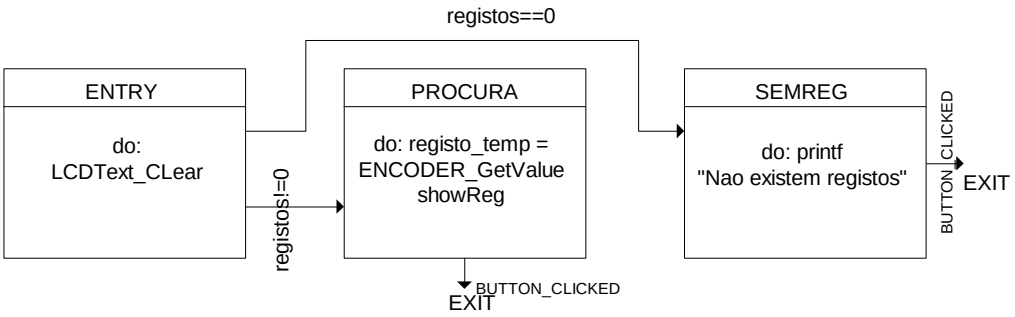

Figura A.10 - Máquina de estados de mostrar registos.

BIBLIOGRAFIA

LPC176x/5x User manual – Ver. 4. 1 – 19 December 2016

CAT25128/D datasheet – June, 2018 – Rev.9

BH1750FVI Ambient Light Sensor IC. 2011.11 – Rec.D

LCD Display (16x2) LED008.PMD