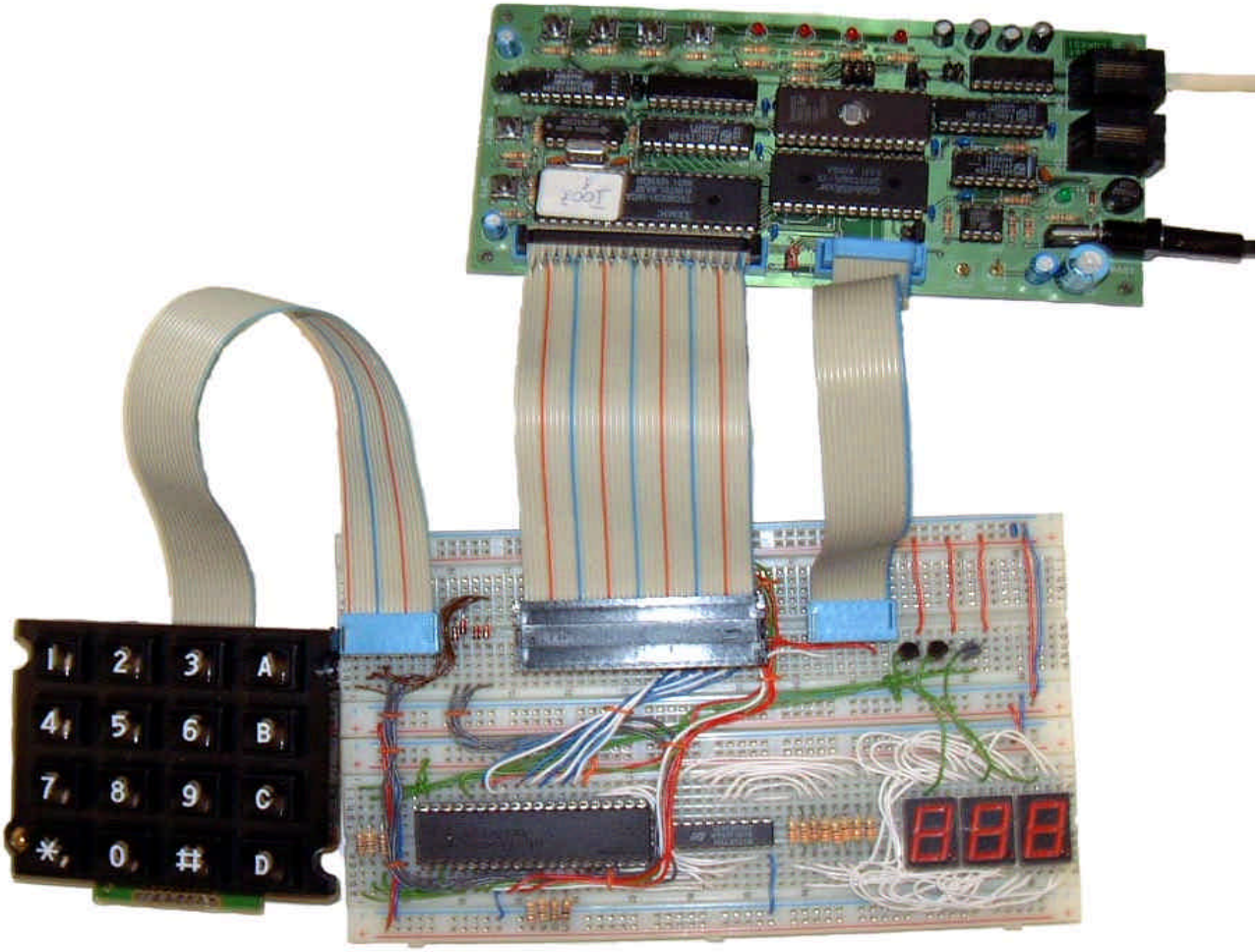
FEUP Universidade do Porto
Faculdade de Engenharia

Relatório do trabalho

Fechadura Electrónica

Sistemas de Microcontroladores – Suc



www.fe.up.pt/~ee98055/suc/fechadura

Pedro Leal **Rui Sousa** **Daniel Duarte**

3º Ano – LEEC

Turma 7, Grupo 4

Docente: Américo Azevedo

Junho de 2002

Índice:

1. Objectivos

O objectivo principal deste trabalho é projectar e implementar um sistema para uma fechadura electrónica, nas vertentes de hardware e software, baseado num microprocessador da série 8051 e na placa de desenvolvimento CORE 51.

Este sistema deve possuir como entrada um teclado e, como saídas, 3 displays de 7 segmentos, um LED verde, um vermelho e um bit para controlo da fechadura.

O funcionamento deveria ser o seguinte:

- Em repouso, o sistema tinha os leds e o display de 7 segmentos desligados.
- A introdução do código de utilizador de 3 dígitos alfanuméricos permitiria a abertura da fechadura durante 1 segundo em simultâneo com a activação do LED verde. Cada dígito do código acenderia, sequencialmente o segmento central de cada display. Códigos não completamente inseridos teriam de ser automaticamente apagados ao fim de 10 segundos. Existiriam também duas teclas extra # e * que permitiriam, respectivamente, confirmar e cancelar a instrução.
- A introdução de um código errado bloqueava o sistema durante 5 segundos, durante os quais este devia ligar o LED vermelho e ignorar possíveis introduções de dados no teclado.
- Deveria também existir um modo de utilizador especial, em conjunto com o código de acesso respectivo, que permitiria alterar o código de utilizador e o código de utilizador especial. Neste modo o LED verde deve piscar e os visualizadores deveriam mostrar as letras pressionadas.

Ao longo de todo o processo de desenvolvimento tivemos também sempre como objectivo a obtenção de um sistema facilmente evolutivo, tanto no hardware como no software. Isto levou-nos a alterar ligeiramente as especificações iniciais do seu funcionamento, de modo a torná-lo modular, criando quase um sistema de menús e menus e submenus para a sua configuração. Outra vantagem resultante da alteração do modo de funcionamento do ponto de vista do utilizador, foi a sua compatibilização com o modo de funcionamento dos sistemas típicos de alarme residenciais existentes no mercado.

Pretendeu-se também tentar garantir ao máximo a robustez e fiabilidade do sistema tentado, tanto ao nível do software como do hardware, dota-lo de procedimentos, rotinas, e soluções em que limitasse ao máximo os riscos de falha.

2. Projecto e Implementação do Hardware

Ao nível do hardware era necessário ligar ao microprocessador:

- 2 bits de saída para LEDs
- 1 bit para a actuação da fechadura
- 3 displays de 7 segmentos
- 1 teclado matricial de 16 teclas, 4 linhas x 4 colunas

2.1 Funcionamento do Display:

Para activar os 3 displays, escolhemos multiplexa-los e activar cada um sequencialmente. Era impensável usa-los separadamente, pois isso iria implicar 8 bits para cada um. Ao todo iria ser necessário 3 portas de 8 bits só para os displays. Assim, usamos só uma porta ligada em paralelo a todos os displays e, para activar cada um deles separadamente, é activado um bit de uma outra porta que, ao estar ligado ao ponto comum de cada display, funciona como “chip select”. Assim são necessários apenas 3 bits para controlo mais oito bits de dados. O princípio de funcionamento baseia-se em que cada um dos display só pode estar aceso de cada vez. Se a frequência de refrescamento for suficientemente alta, então todos os displays parecem, ao olho humano, estar activos em simultâneo. Isto foi realizado através da utilização de uma interrupção associada a um timer. Periodicamente, e independentemente do programa principal, o micro processador vem executar a rotina de interrupção em que:

- desliga o display actual
- carrega o novo valor destinado a aparecer no display seguinte
- liga o novo display, e sai fora

O sinal de enable dos displays é o representado na figura 1, em que as três formas de onda representam os 3 sinais de enable de cada display.

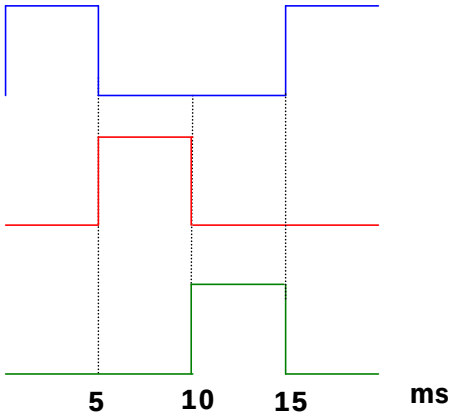


Figura 1

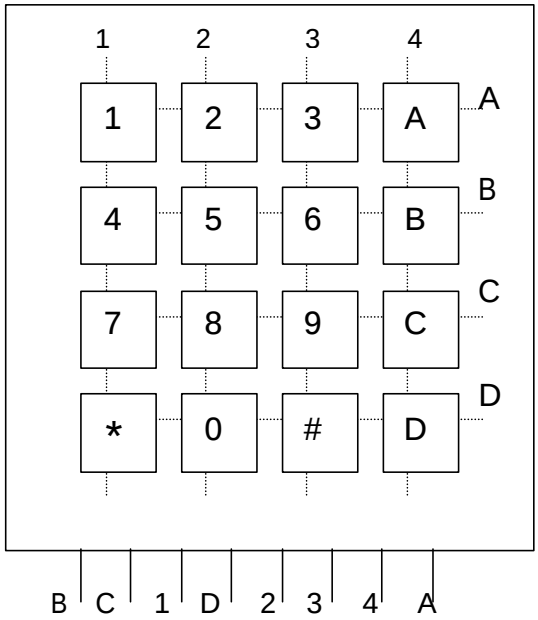


Figura 2

2.2 Funcionamento do Teclado:

O funcionamento do teclado matricial (Figura 2) é também relativamente simples. Existem 4 linhas e 4 colunas. Quando se pressiona um botão está-se a fazer a ligação eléctrica entre a linha e a coluna respectiva. Para se detectar qual o botão premido, o que o sistema faz é (tendo por exemplo as linhas ligadas como saídas e as colunas como entradas):

- Activa uma linha de cada vez e verifica se esse bit aparece em alguma das colunas.
- Caso esse bit apareça numa coluna então tem-se a informação de qual a linha e coluna associada ao botão pressionado.

Dado que do ponto de vista de um micro processador o processo de se carregar num botão demora bastante tempo, então é possível associar uma rotina que lê o teclado, a uma interrupção e a um timer. Assim, periodicamente o microprocessador vem ler o teclado verificando se existe alguma tecla pressionada, guardando esse valor e voltando posteriormente ao programa principal.

2.3 Funcionamento da PPI

Para implementar todo o sistema seriam então necessários 19 bits. Embora fosse possível usando directamente as portas do micro processador disponíveis na CORE e posições de memória externa através do uso de latches, esta técnica iria ter alguns inconvenientes:

- iria implicar a utilização de bastante lógica discreta para multiplexar/desmultiplexar externamente sinais do micro processador.
- Iria impossibilitar qualquer crescimento posterior do hardware do sistema. Por exemplo se se pretendesse utilizar 4 dígitos de código em vez de 3, essa evolução não poderia ser acompanhada pelo aumento de 3 para 4 displays.

Por estes motivos, e também com a vantagem de aumentar a simplicidade do sistema ao nível do hardware, preferimos usar uma PPI. Este periférico permitiu-nos obter 3 portas adicionais, além das já existentes no micro, usando para isso 4 posições da memória externa. Assim apenas com um circuito integrado obtemos um sistema facilmente escalonável, simples de projectar/montar e, fundamentalmente, simples de utilizar. Outra vantagem prende-se com o facto de todas as portas da PPI permitirem receber ou fornecer 2,5mA de corrente, enquanto que o microprocessador está limitado a valores inferiores.

A PPI usada, 80c55a, apresenta bastantes possibilidades de configuração, tanto no modo de funcionamento (tipo de dados), como na selecção de entrada ou saída para cada porta. Como era apenas pretendido usar entradas e saídas simples, então escolhemos o modo 0 como modo de funcionamento. Visto que a porta C apresenta a característica de poder ser configurada para ter 4 bits como entrada e os outros 4 como saída, optamos por ligar o teclado a essa porta. Ficamos então com a porta A e B para os displays. Optamos por usar a porta B com saída “de dados” e a porta A como “enable” de cada displays. Assim ficamos ainda com mais 5 bits da porta A livres para expansão de outras saídas (como por exemplo mais displays).

2.4 Implementação do Display:

Já sabemos como ligar os displays fisicamente no entanto há um aspecto que não pode ser descuidado. Trata-se dos valores de corrente associados. Visto que cada LED do displays consome um valor típico de 10mA então nem o bit de enable nem os oito bits de "dados" poderiam ser ligados directamente à PPI. O bit de enable iria ser necessário ter uma corrente máxima de 8 x 10mA = 80mA, sendo por isso necessário implementar um sistema capaz de fornecer este valor. Para este bit escolhemos usar um transístor PNP BC557, em que a base é ligada à saída da porta A da PPI, o emissor ao Vcc e o colector ligado ao ponto comum dos displays. Assim este transístor vai funcionar como interruptor controlado pela tensão na base. A corrente de base é sempre muito baixa, e o elevado valor de corrente necessário para activar cada um dos displays passa apenas no emissor e no colector.

A activação do sistema processa-se do seguinte modo:

- Na base os transístores têm sempre uma tensão típica de 4.3V, que se deve à queda de tensão V_{BE} entre o Vcc e a base.
- A corrente do emissor é aproximadamente igual à corrente de colector do transístor. Por sua vez esta é proporcional à corrente de base do transístor, sendo a razão de proporcionalidade o seu ganho (β). Da folha de características verificamos que pode variar entre 100 e 800.
- A corrente de base vai ser quanto maior quanto a diferença de tensão entre a base e o nível de tensão da porta PA. Esse valor é imposto pela resistência de base.

Assim sendo, escrever para PA o nível lógico alto implica ter uma corrente de emissor mínima, e escrever o nível lógico baixo uma corrente de emissor máxima. Por isso para activar os displays é necessário colocar o bit da porta PA correspondente no nível lógico 0.

Para o valor da resistência de base usamos a seguinte fórmula:

$$R_B = \frac{V_{CC} - 0.7 - V_{OL}}{8 \times 10mA/b}$$ **Equação 1**

Deu-nos um valor aproximado de 1KΩ.

A ligação dos displays à porta B da PPI também não pode ser feita directamente porque a PPI não pode absorver os 10mA de cada led individual. Usamos por isso um driver de corrente ULN2803. Este inverte o sinal introduzido pela PPI e tem capacidade para absorver bastante mais que os 10mA necessários. Preferimos a utilização deste integrado em detrimento de transístores e simples inversores por dois motivos: Em relação aos transístores permitia não ter mais 8 componentes discretos e respectivas resistências; Em relação a um inversor da série 74HCT porque contem já 8 portas, enquanto que os inversores têm menos, o que implicaria o uso de mais do que um integrado.

Para limitar a corrente nos LEDs usamos uma resistência calculada da seguinte forma:

$$R_d = \frac{V_{CC} - V_{CE} - V_D - V_{OL\_ULN2803}}{I_{Pico}}$$ **Equação 2**

Para calcular a I_{Pico}, seguimos este raciocínio:

- apenas um display está aceso de cada vez. Em cada interrupção é aceso o display seguinte e apagado o anterior, o que significa que cada display está apagado 2 períodos e aceso 1.
- vamos usar interrupções periódicas em cada 5ms, ou seja temos um período de refrescamento de 15ms e um período em que está ligado de 5ms.
- Temos 3 displays, e queremos uma corrente média em cada um de 10mA

$$I_{med} = I_{pico} \times \frac{T_{on}}{T_{off}} = 10 \times \frac{15}{5} = 30mA$$ **Equação 3**

Durante a fase de montagem, seleccionamos experimentalmente resistências de 330Ω por estas garantirem uma luminosidade adequada ao display.

Ficamos assim com um sistema em que cada display tem um enable activo baixo accionando o bit correspondente da porta A, e em que os dados são introduzidos na porta B com um sinal activo alto.

2.5 Implementação do Teclado:

Em relação ao teclado escolhemos para bits de saída os 4 menos significativos da porta C e para entrada os 4 mais significativos. Para garantir que as entradas nunca ficavam desligadas de nenhum nível lógico, incluímos resistências entre as 4 entradas e Vcc. Assim no estado em que não há nenhum botão pressionado as entradas estão garantidamente a 1. Quando se pressiona um botão qualquer garante-se que o valor na porta de entrada é igual ao valor na saída, e nunca se chega a gerar nenhum fenómeno crítico como se não fosse colocada a resistência, criando um curto circuito entre alguns bits da porta C e Vcc.

Um outro problema possível poderia acontecer quando se pressionassem dois botões da mesma coluna ao mesmo tempo. Caso isso acontecesse, no momento em que o microprocessador colocasse todas as saídas excepto uma a 1, ia-se gerar um curto circuito entre as duas delas. Para evitar isto incluímos também díodos entre as portas de saída da PPI e as entradas de linha do teclado, com o ânodo virado para a PPI. Aqui usamos díodos genéricos 1N4148.

2.6 Implementação dos LEDS

Como este sistema não ia ser fisicamente ligado a uma fechadura eléctrica, optámos por associar esse sinal de controlo a um LED. Assim sempre que esse sinal fosse activado esse LED seria aceso.

Iríamos assim precisar de 3 LEDs. Escolhemos, também para simplificar o projecto, utilizar os LEDS já existentes na CORE51.

2.7 Implementação da PPI

Já sabemos agora que a PPI vai funcionar no modo 0, tem de ter as portas A e B configuradas como saída, a parte alta da porta C também como saída e a parte baixa configurada como entrada. A esta configuração corresponde a palavra de controlo 0x88H.

Em relação às entradas da PPI era necessário liga-las à CORE, e atribuir-lhe um endereço de memória. Para isso ligamos directamente o barramento de dados do conector principal da CORE ao barramento da PPI. Os sinais de /RD, RST e /WR também foram ligados directamente entre o conector principal da CORE e a PPI.

Do conector secundário usamos os bits A0, A1 e AUX. Os primeiros dois correspondem aos bits menos significativos do barramento de endereços já desmultiplexado e o segundo é um sinal activo baixo quando o microprocessador acede a toda a área de memória externa desde 0xE000 até 0xFFFF.

Como a PPI é o único periférico que vamos usar, podemos "reservar" toda a área auxiliar para este periférico. Assim, ligamos o sinal /AUX ao /CS da PPI e os sinais A0 e A1 directamente aos sinais A0 e A1 da PPI. Neste caso vamos ter para endereços base da PPI:

- Palavra de Controlo 0xE000
- Porta A 0xE001
- Porta B 0xE002
- Porta C 0xE003

Estes endereços repetem-se periodicamente no espaço desde 0xE000 até 0xFFFF, no entanto basta-nos usar um dos conjuntos possíveis.

2.8 Configuração da CORE 51

Do ponto de vista do hardware o único ponto que falta descrever é a configuração da CORE 51. Esta configuração é feita através de jumpers, e foram usados as seguintes configurações:

JP1	ON	Controla o sinal /EA
JP2, JP3	ON	Selecciona o mapa de descodificação 0 da PAL
JP4, JP5	OFF, ON	Configuram o tipo de memória de programas
JP6, JP7	ON, OFF	Configuram o tipo de memória de dados
JP8, JP9	Indiferente	Permitem ligar a tecla INTR a /INT0 ou T0
JP10, JP11	Indiferente	Permitem ligar o segundo canal série aos bits P1.2 e P1.3
JP12	ON	Activa os periféricos existentes na placa (leds e botões)
JP13	ON	Liga a tensão de alimentação da placa aos conectores de expansão

A core usada estava equipada com um RAM de 8kb e com um micro processador da série 8031.

3. Projecto e Implementação do Software:

O software de suporte à fechadura electrónica foi feito na linguagem C e não em assembly. Tomamos esta opção pelas seguintes razões:

- C é mais intuitivo do que assembly por ser uma linguagem de alto nível.
- É modular, permitindo reutilização de código
- C permite também estruturar todo o código fonte de uma forma simples, que devido à sua modularidade é perfeitamente compatível com futuros upgrades ao sistema, tanto no software como no hardware

3.1 Controlo do hardware

No desenvolvimento do software optou-se por tentar criar uma estrutura que permitisse acesso ao hardware de uma forma simples e transparente para o programa principal. Isto principalmente em três áreas distintas:

- acesso ao display
- leitura do teclado
- controlo de tempo

Isto foi conseguido através da utilização da rotina de interrupção do timer 0, configurando-o para funcionamento como contador. Sempre que o contador chega a 0 é executada a rotina de interrupção. Esta vai "tratar" destas três áreas distintas. Isto é executado de uma forma automática e perfeitamente transparente para o resto do programa. Assim o programador do programa principal apenas tem de utilizar as funções já existentes de escrita nos displays, leitura do teclado e controlo de tempo. Ainda que num nível muito restrito, a rotina de interrupção e funções associadas funcionam quase como um mini-sistema operativo responsável pela interacção com o hardware, libertando o programador desse trabalho sempre que tal for necessário.

Assim, criaram-se diversas funções que não se destinam a ser usadas pelo programa principal, mas apenas pela função de interrupção. Todas as funções podem assim ser distinguidas quanto à sua disponibilidade ou não para serem usadas pelo programa principal.

3.2 Organização do Código Fonte:

Do ponto de vista da organização do código fonte, optamos por criar módulos tanto quanto possível independentes uns dos outros. Cada módulo é responsável por uma área, por exemplo: display, teclado, interrupção, testes, main, etc. Isto permitiu-nos ter as diversas funções separadas em ficheiros diferentes conforme a sua finalidade/ área de actuação.

Os módulos dividem-se fundamentalmente em duas áreas. Uma área é o módulo (main.c) que implementa o programa principal e funções relacionadas. A outra são todos os outros módulos, que implementam funções de suporte ao primeiro.

3.2.1 Ficheiro de include (fechadu.h)

Comum a todos os módulos encontra-se um ficheiro de include onde estão concentradas todas as definições, protótipos de funções e mesmo includes relativos ao microprocessador. Esse ficheiro é fechadu.h, e a sua constituição é a seguinte:

1. includes de ficheiros necessários
2. definições de labels
3. definições de variáveis externas, comuns a todos os módulos
4. protótipos externos de todas as funções do sistema

Optámos por criar apenas um ficheiro de include e não vários (um para cada módulo) porque consideramos que a dimensão e complexidade do programa não o justificava. Ainda assim o facto de todo o tipo de definições do programa estar concentrado num só ficheiro permite um muito simples controlo, gestão e modificação.

3.3 Descrição dos módulos de suporte:

3.3.1 - Rotina de Interrupção (interpt.c):

A rotina de interrupção é responsável pela implementação das funções descritas anteriormente. Adicionalmente também implementa uma funcionalidade do programa principal, sendo esta o piscar do LED verde quando o sistema está no modo de utilizador especial. Os algoritmos de cada uma das secções da rotina de interrupção estão representados nos três diagramas.

No módulo da rotina de interrupção está também presente a função de inicialização do timer0. Antes de mais, optámos por usar apenas o timer 0 de modo a deixar o timer 1 livre, permitindo que este pudesse ser utilizado pelo canal série, e assim interagir com o programa de monitorização no computador. A inicialização do timer configura-o para modo 1, e carrega o valor a decrementar para os registos apropriados.

3.3.2 - Leitura do Teclado (keyb.c):

A leitura do teclado está também montada de uma forma transparente para o utilizador. Periodicamente, o teclado é lido, é avaliada a validade ou não da leitura e é guardada essa informação num buffer. Esta fase é implementada pela rotina de interrupção.

O sistema implementado para guardar os dados do teclado é um buffer que implementa uma estrutura de FIFO (first in first out). Para implementar esta estrutura estão presentes três funções: wr_keyb_buf, rd_keyb_bf e clr_keyb_buf. A primeira é usada pela rotina de interrupção e as últimas são usadas pelo programa principal.

Dado que a rotina de interrupção é executada cada 5ms, consideramos que este era um tempo excessivamente curto para se ler o teclado periodicamente, por isso associamos esta leitura a um contador que vai permitir contabilizar um tempo que ronda os 40ms.

A função de inicialização do teclado é responsável por duas áreas distintas. Por um lado garante que todas as posições do buffer do teclado têm o caracter nulo (‘N’), por outro inicializa o apontador para o primeiro endereço a ser escrito do buffer.

Funções do teclado:

void init_keyb(void);	Função das inicializações do teclado
void wr_keyb_buf (uchar valor);	Recebe um valor de tecla e guarda-o no buffer do teclado. É usado pela rotina de interrupção
uchar rd_keyb_buf (void);	Função que retorna o 1º valor introduzido no buffer do teclado. Retorna o caracter ‘n’ caso esteja vazio. É usada pelo programa principal.
void clr_keyb_buf(void);	Apaga os valores contidos no buffer do teclado. É usada pelo programa principal
uchar keyb_scan(void);	Função que percorre o teclado e retorna o valor da tecla premida. Usa o vector teclas[i] para retornar esse valor. É usada pela rotina de interrupção.
uchar key_validate (uchar new_key);	Recebe o valor da nova tecla e implementa o algoritmo de verificação da tecla premida. Usa a variável global old_key para comparação. Retorna o valor da nova tecla. É usada pelo programa principal
void wr_keys (uchar tecla, uchar display)	Recebe o valor de uma tecla e o nº de display e usa as funções key_converter() e wr_disp() para as escrever no display seleccionado. É usada pelo programa principal

3.3.3 - Acesso ao Display (display.c):

Tal como já foi referido na descrição do hardware, os displays estão ligados à porta A e B da PPI, sendo a porta B para recepção do valor a escrever no display e a porta A o enable de cada display individual. Todo o projecto foi feito de modo a que apenas um display estivesse aceso de cada vez. Esta técnica de multiplexagem permite ter valores diferentes escritos em cada display, sendo a informação renovada em cada interrupção. Para implementar esta funcionalidade criaram-se várias funções de controlo dos displays, e parte da rotina de interrupção gere também esta funcionalidade: O timer que origina a rotina de interrupção é periodicamente carregado com o valor correspondente a 5ms. Este tempo foi determinado experimentalmente como o tempo máximo necessário para que não se visse cintilação nos displays, dando a sensação que todos estão acesos em simultâneo.

Funções do display:

void en_disp (uchar display);	Recebe como parâmetro o numero de display e activa esse display. 0 desactiva todos. É usada pela função de interrupção
void wr_disp (uchar valor, uchar disp);	Recebe o valor a escrever e o número do display. Guarda o valor na posição correspondente do buffer do display. Disp 9 corresponde a todos. É usada pelo programa principal
uchar key_converter (uchar tecla);	Recebe um valor de tecla e converte-o no correspondente valor requerido para escrever para os displays, retornando esse valor. Esta função é disponibilizada ao programa principal.

3.3.5 - Controlo de Tempo (time.c):

Sempre que o programa principal tem necessidade de contabilizar um período de tempo específico, este controlo pode ser feito pelas funções criadas para este efeito. O princípio é muito simples: existe uma função que recebe o valor do tempo que se pretende controlar e carrega o valor necessário para uma variável global, que vai funcionar como flag. Quando essa flag tiver chegado a 0 então é porque o tempo já passou.

Funções de controlo de tempo

void time_ct (uint tempo);	Recebe um valor em ms e configura o contador para o número de interrupções correspondentes. O contador é variável global timer0_ct, tornando-se 0 quando o tempo tiver passado. É usada pelo programa principal.
void delay (void);	Mantém o microprocessador parado enquanto não passar todo o tempo seleccionado (contador de tempo não for 0). É usada pelo programa principal.

3.3.6 - Testes ao Sistema (testes.c):

As funções de teste permitem testar as quatro áreas principais de interacção com o hardware. Acedem a esses recursos da mesma forma que o programa principal o faz, levando a que o bom funcionamento destes testes seja garantia de que o hardware e o software de suporte está bem.

Funções de teste:

void teste_7seg (void);	Escreve sequencialmente em cada display vários algarismos.
void teste_7seg2 (void);	Acende durante um período de tempo todos os dígitos do três displays
void teste_leds (void);	Acende sequencialmente cada um dos leds
void teste_keyb (void);	Escreve nos 3 displays o valor da tecla pressionada. Sai fora da função quando se pressionar em #

Todas as funções de teste são usadas pelo programa principal.

3.3.7 - Inicializações (init.c):

A função init_hardw() é responsável por inicializar todas as questões relativas ao hardware. Assim está sob a sua alçada:

- activar as interrupções e a interrupção do timer 0;
- inicializar os apontadores da memória externa com o valor conveniente;
- inicializar a PPI com a palavra de controlo adequada, e garantir que as suas saídas estão a 1;
- Garantir que todos os leds e displays estão desligados;
- Correr a rotina de inicialização do timer e do teclado;

Todas as chamadas de outras funções de inicialização do hardware (timer, teclado, etc) estão já inseridas na função init_hardw(), de modo a que o programa principal tenha apenas de chamar esta função.

3.4 Programa Principal (main.c)

No desenvolvimento do programa principal optou-se por seguir uma estrutura de máquina de estados. Esta estrutura permite simplificar bastante todo o desenvolvimento do programa principal, assim como garantir que o sistema está sempre num estado determinado.

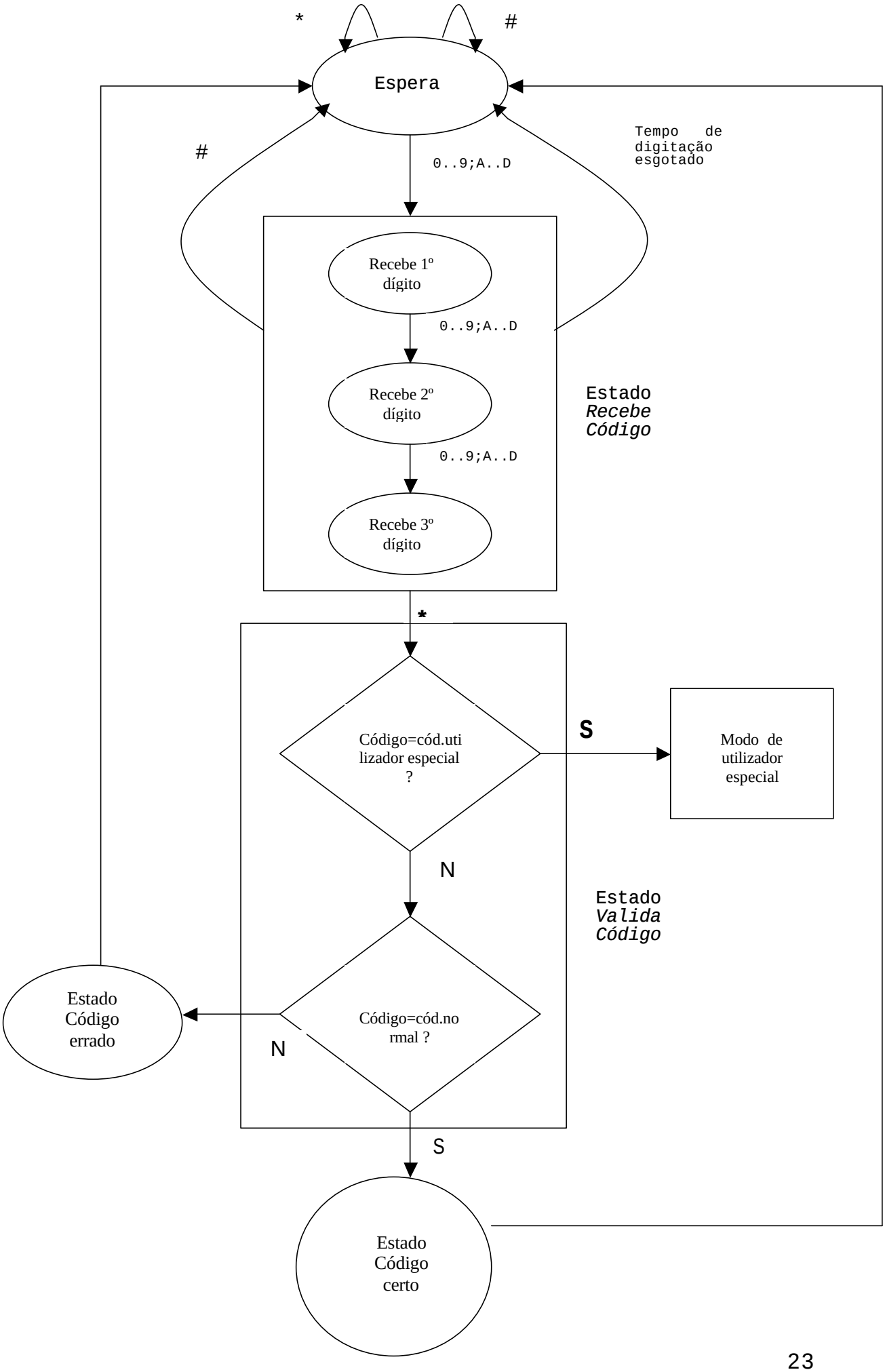
Foi nesta fase de projecto do sistema que se optou por não seguir à risca o modo de funcionamento do sistema sugerido. Isto aconteceu para permitir retirar uma série de vantagens já referidas anteriormente.

Tentou-se sempre que o programa obedecesse às seguintes características:

- o sistema nunca deve ficar “bloqueado”
- o tempo de resposta a uma instrução deve ser rápido
- a tecla # deve funcionar em todo o programa como tecla de cancelamento / correcção.
- A tecla * deve ser sempre a tecla de activação da instrução.
- Em todos os estados do programa principal deve existir um período máximo para introdução dos dados, garantindo que ao fim de x tempo desde que se pressionou a última tecla o sistema volta ao estado de espera.
- Todas as condições de transição de estado devem estar explicitamente declaradas, sendo estas as únicas que podem levar as transições a acontecerem.

A implementação de todas estas características em todos os estados do sistema tornou um código simples num bastante complicado e de difícil leitura/interpretação.

Criou-se também um fluxograma contendo todos os estados e respectivas condições de transição.


*
#
Espera
#
0..9;A..D
Tempo de digitação esgotado
Recebe 1º dígito
0..9;A..D
Recebe 2º dígito
0..9;A..D
Recebe 3º dígito
Estado Recebe Código
*
Código=cód.utilizador especial ?
S
Modo de utilizador especial
N
Estado Valida Código
Estado Código errado
Código=cód.normal ?
N
S
Estado Código certo

Implementaram-se cada um dos estados tendo em conta as suas condições particulares. O mais complicado foi o estado de utilizador especial, que contem vários sub-estados.

Esta estrutura permite uma análise e tratamento modular que facilmente pode crescer com novas características.

O programa principal chama várias funções que se destinam a tornar o código mais simples e intuitivo. Algumas delas são:

Funções do programa principal (main.c)

uchar compara_codigos(void);	Compara o código temporário (temp_code) com o código de utilizador especial (special_code) e com o de utilizador normal (user_code). Retorna 2 se for igual ao código especial, 1 se código normal e 0 se diferente de ambos. É usada pela função principal.
void reset_codes(void);	Muda os códigos actuais para os códigos iniciais
void change_user_code(void);	Muda o código de utilizador
void change_special_code(void);	Muda o código de utilizador especial
void main(void);	Implementa os vários estados do programa principal, e respectivas condições de transição

3.4.1 Descrição dos estados

Estado de ESPERA:

Neste estado o sistema tem todas as saídas desligadas. Os displays ficam com um ponto a piscar continuamente. Quando se pressionar uma tecla diferente de * ou # ele passa para o estado recebe_codigo. Ao passar envia como parâmetro para este estado a informação da tecla premida, que vai funcionar como primeiro dígito do código.

Estado RECEBE_CODIGO:

Neste estado sempre que é premida uma tecla é acendido o segmento central de cada um dos displays. Guarda os dígitos do código no array temp_code[];

Estado VALIDA_CODIGO:

Este estado não interage directamente com o utilizador. Limita-se a especificar qual o próximo estado em função dos valores existentes nos arrays temp_code, user_code, e special_code.

Estado CODIGO_CERTO:

Neste estado é aceso o LED verde e o TRINCO. Dura 1s e após esse tempo ele ignora todos os valores introduzidos no buffer do teclado entretanto e volta ao estado de espera.

Estado CODIGO_ERRADO:

Neste estado acendem-se todos os LEDs de todos os displays, assim como o LED vermelho durante 5s. Após este estado ele volta ao estado de espera e ignora os valores introduzidos no teclado entretanto.

Estado MODO_UTILIZADOR_ESPECIAL

Neste estado o LED verde fica a piscar. Isto é conseguido através da rotina de interrupção. Ao pressionar-se uma tecla ela aparece no primeiro display, indicando que se escolheu aquele submenu. Para se entrar nele pressiona-se no *, levando o sistema a executar a função pretendida. Após a execução dessa função volta ao modo de utilizador especial.

4. Problemas Encontrados

Durante o projecto e montagem do sistema deparamo-nos com diversos problemas. Fica aqui uma descrição sucinta de vários:

- Dificuldades ao aceder à PPI

Numa fase inicial não se conseguia ler nem escrever da PPI fazendo o include do ficheiro que define o acesso à memória externa. Embora este não seja motivo para o sistema não funcionar, conseguiu-se superar o problema quando se começou a aceder através de apontadores, ex.:

```
unsigned char *PPI_CONTROL;
PPI_CONTROL=0xE003;
*PPI_CONTROL=0x88;
```

- Dificuldades na leitura do teclado

Numa fase posterior tivemos dificuldades em ler o teclado. O problema estava na parte da função keybscan() que faz circular um 0 entre 4 saídas de dados da porta C da PPI. Este problema foi resolvido deixando de se tentar rodar esse 0 pelos vários bits e atribuindo-o directamente a cada um dos bits.

- Jumpers errados

Como as placas CORE andavam constantemente a rodar entre vários grupos, acontecia por vezes o sistema não funcionar por a CORE ter algum componente estragado ou estar mal configurada. Estes problemas eram tipicamente resolvidos fazendo o auto-teste da CORE e corrigindo os seus Jumpers.

- Problemas na configuração do compilador

Se o compilador não estiver bem configurado todo o software ou partes não correm. São especialmente críticas as secções que configuram o modelo de memória, o tamanho da ROM de código, e o nível de optimização do código.

- Problemas no envio do ficheiro .hex

Só se começou a ter sucesso no funcionamento do sistema quando se passou a enviar o ficheiro para a RAM da CORE directamente através de um programa de terminal como o TeraTerm e não pelo μVision 1 ou 2.

- Problemas por falta de disponibilidade de material

Também aconteceu com alguma frequência haver dificuldade em se ter acesso ao laboratório por este estar demasiado cheio, ou por o tempo de permanência nele estar limitado a 2 horas.

5. Evolução do sistema

A primeira e mais simples evolução do sistema é passar para códigos de 4 dígitos, assim como para 4 displays. Esta alteração é muito simples, e requer alterações mínimas no código. A principal é nas definições globais NUM_DIGITOS_CODIGO e NUM_DISPLAYS.

Uma alteração também bastante interessante era associar uma buzina ao sistema. Esta permitia ouvir um tom de aceitação quando se está a abrir a porta, um tom de erro quando o código está errado, um tom de erro quando o buffer do teclado estivesse cheio e um tom de aceitação quando se pressionasse numa tecla.

O código fonte do programa principal está um pouco confuso e podia ser bastante melhorado.

Uma outra alteração bastante útil e simples é o aumento das configurações no modo de utilizador especial. Algumas destas configurações possíveis, e bastante fáceis de implementar, são:

- configurar o valor dos temporizadores de espera;
- configurar o tempo do sistema em que a fechadura está aberta;
- configurar o tempo em que o sistema fica bloqueado com um código errado;
- utilização ou não de teclas alfanuméricas nos códigos;

Ao mudar o código não existe nenhum algoritmo de verificação se o novo código é igual, ou não, ao outro código já existente. Esta situação não é critica porque caso o sistema tenha os dois códigos iguais, ao introduzi-lo ele vai automaticamente para o modo de utilizador especial. No entanto podia ser implementado um sistema que rejeite o novo código caso ele fosse igual ao outro.

Outra funcionalidade bastante interessante seria o bloqueio da abertura de porta após x tentativas de código erradas. Após este bloqueio apenas seria possível o desbloqueio com o código de supervisor ou com um código diferente, criado para este efeito com bastantes mais dígitos. O número de tentativas máximas poderia ser configurado no menu de supervisor, assim como a activação ou não desta funcionalidade.

Temos consciência que o código fonte não está optimizado do ponto de vista do espaço que ocupa. Durante todo o processo de projecto esta nunca foi uma prioridade desde que não se atingisse o limite máximo disponível (8Kb de RAM). Uma melhoria bastante interessante seria a optimização do código para este efeito, especialmente ao nível das várias variáveis/arrays que se encontram espalhadas pelas várias funções.

As funções change_user_code() e change_special_code() são completamente idênticas, tirando uma linha que as diferencia. Torna-se assim um desperdício de espaço. Era também interessante torna-las numa só, em que ao receber um parâmetro fazia a diferença entre o código de utilizador e o código especial.

6. Conclusões

Todo o desenrolar do trabalho correu como esperado e, apesar de alguns contratempos e atrasos, conseguiu-se finalizar o projecto de acordo com todas as especificações requeridas. Obtivemos assim, baseado na nossa aprendizagem de sistemas digitais microprocessadores e da actual disciplina de suc, um sistema electrónico de fechadura de porta, pouco complexo e perfeitamente capaz de ser utilizado por pessoas sem quaisquer conhecimentos nesta área.

Permitiu-nos também aperfeiçoar os nossos conhecimentos e métodos de trabalho ao nível do projecto e implementação de sistemas de microcontroladores.

7. Bibliografia

C51 Primer, Hitex (UK), 1996

Mathew Chapman, *The Final Word on the 8051*, 1994

Thomas W. Schultz, *C and the 8051*, Prentice Hall, 1998

Sencer Yeralan, *Programming and interfacing the 8051*, Addison-Wesley Publishing Company, 1995

Mário Águas, *O microcontrolodor 8051*, CINEL, 1996

Brian Kernigham, Dennis Ritchie, *The C Programming Language - Second Edition*, Prentice Hall Software Series, 1988

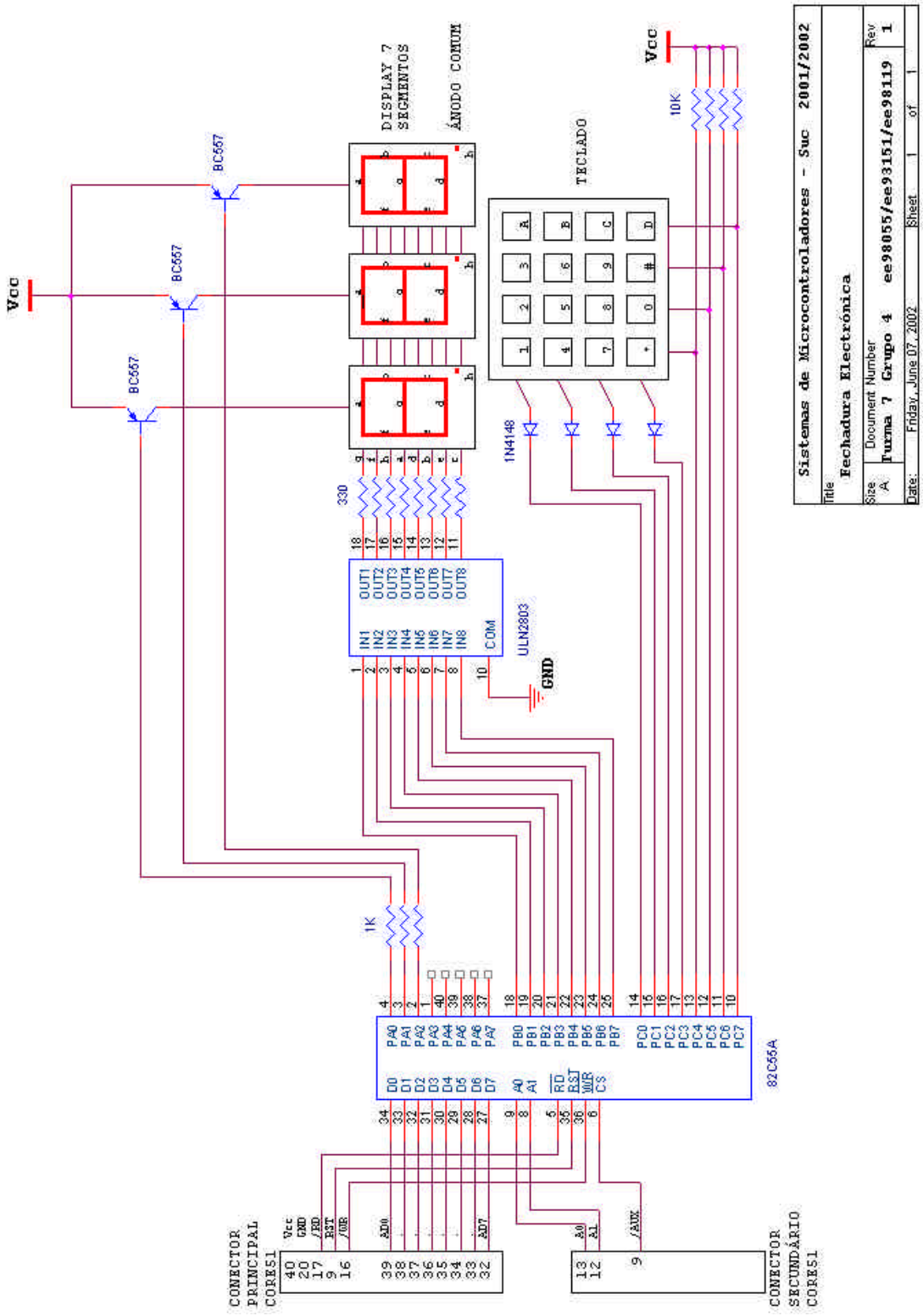
João Paulo Sousa, *CORE51*, 2ª Edição, Departamento de Engenharia Electrotécnica e de Computadores FEUP, 2001

Documentação do programa µVision 2

Datasheets várias do micro processador 8031 e do periférico 82C55A, e do ULN2803

Anexos

A.1 Esquema da montagem



A.2 Fotografia da montagem

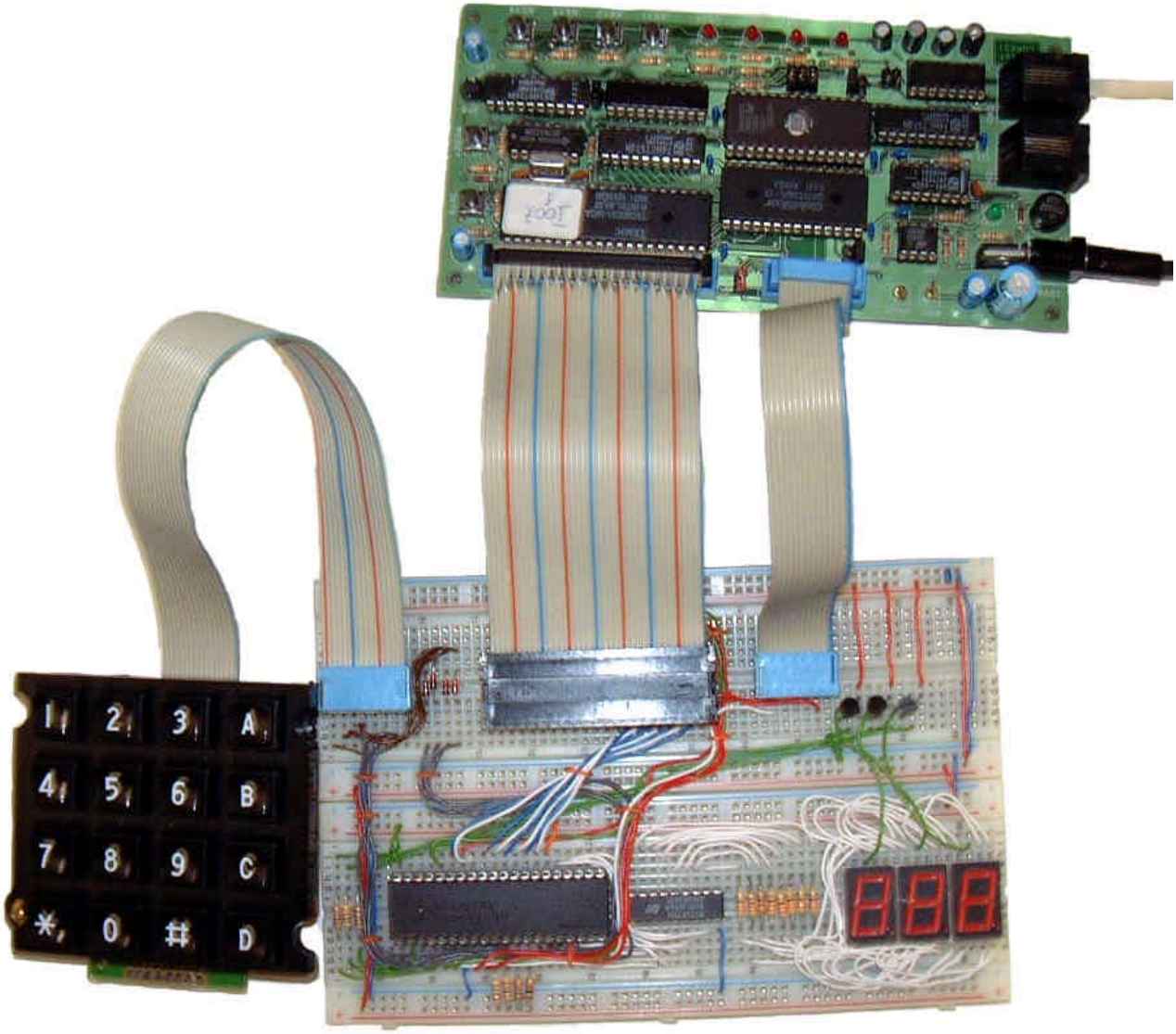


Figura 3

A.3 Diagramas das funções principais do programa

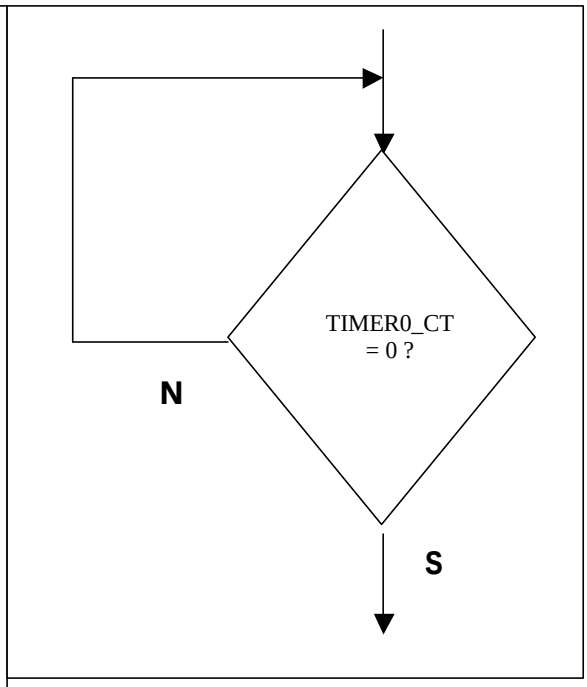


Figura 4, função delay()

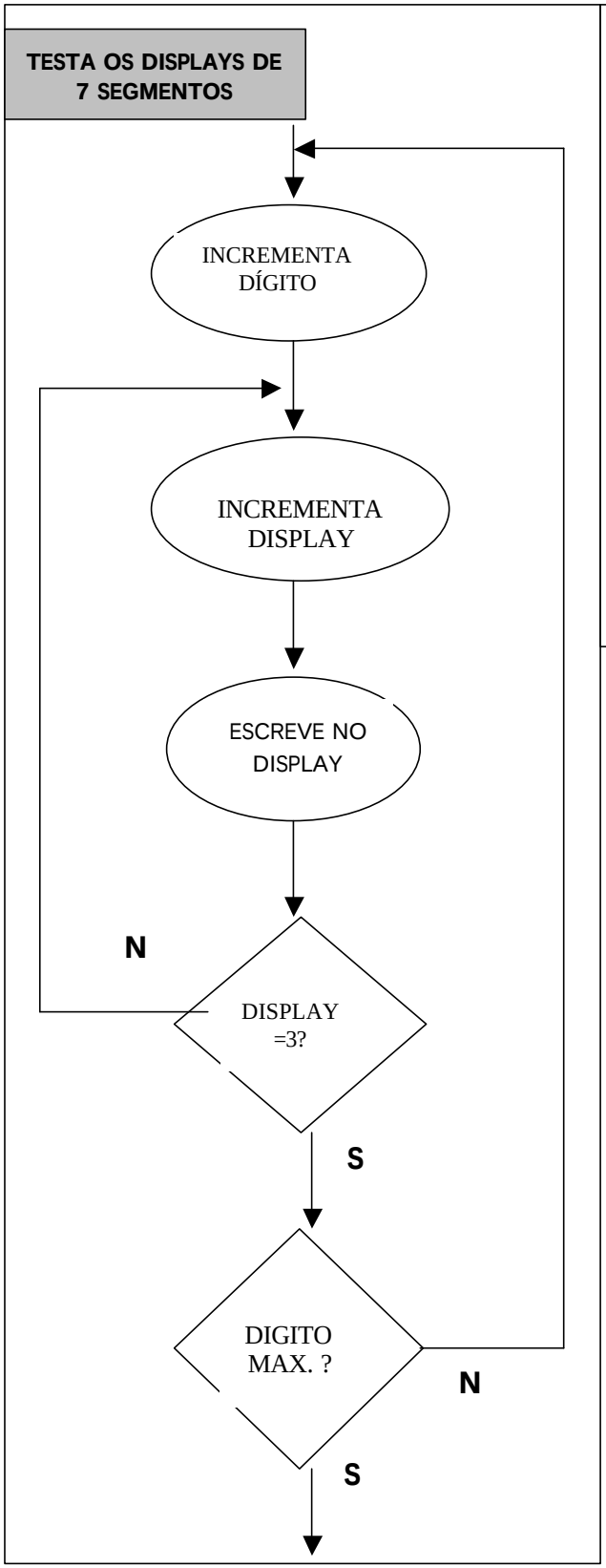


Figura 5, função teste_7seg()

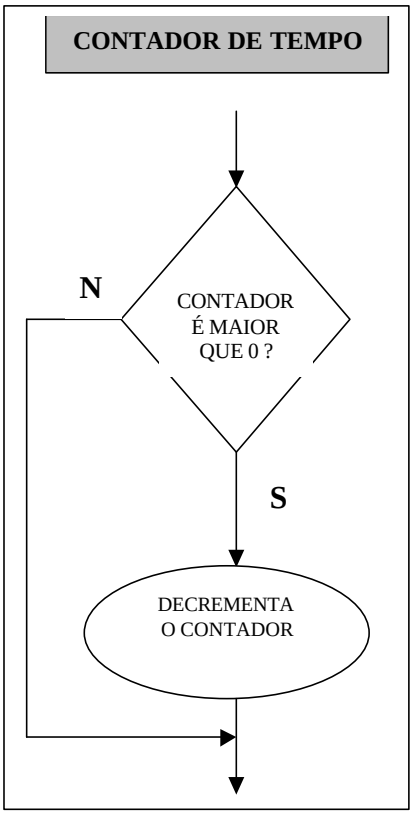


Figura 7, tempo em timer0_int()

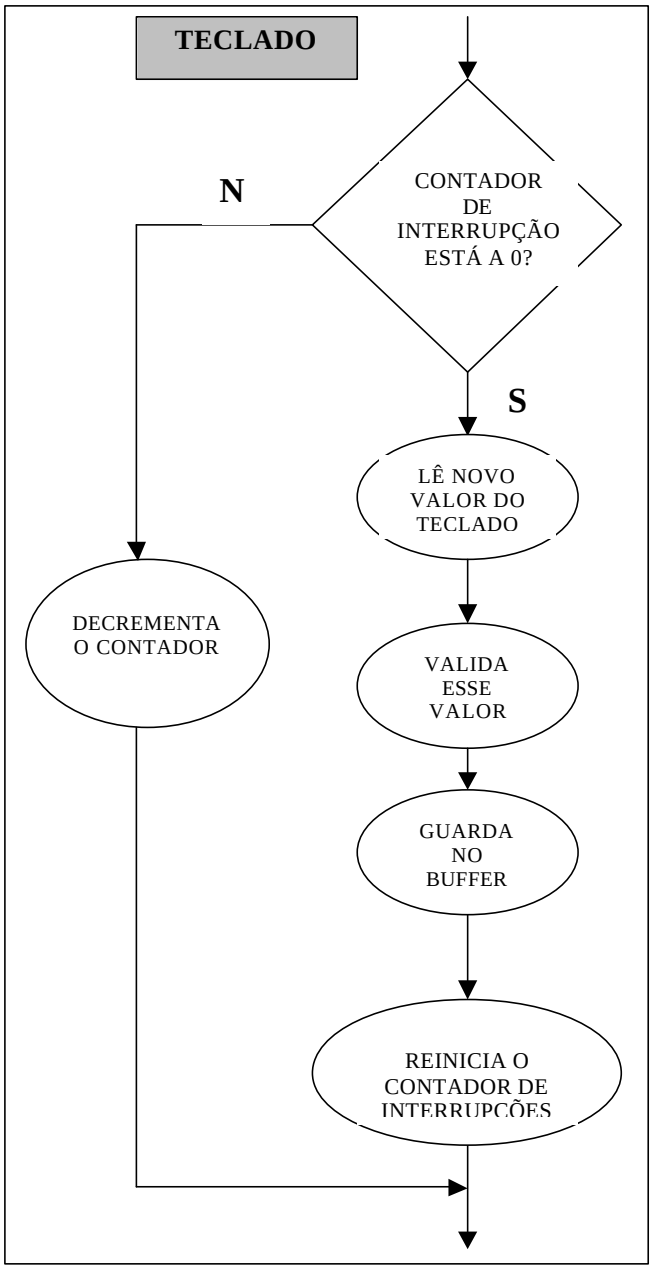


Figura 6, parte do teclado da função timer0_int()

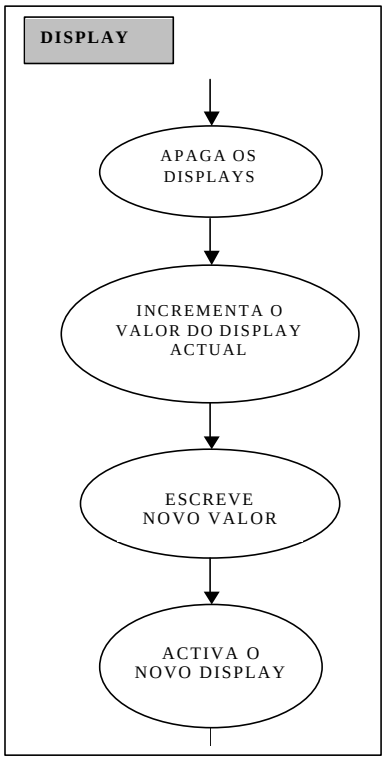


Figura 8, parte do display da funcão timer0 int

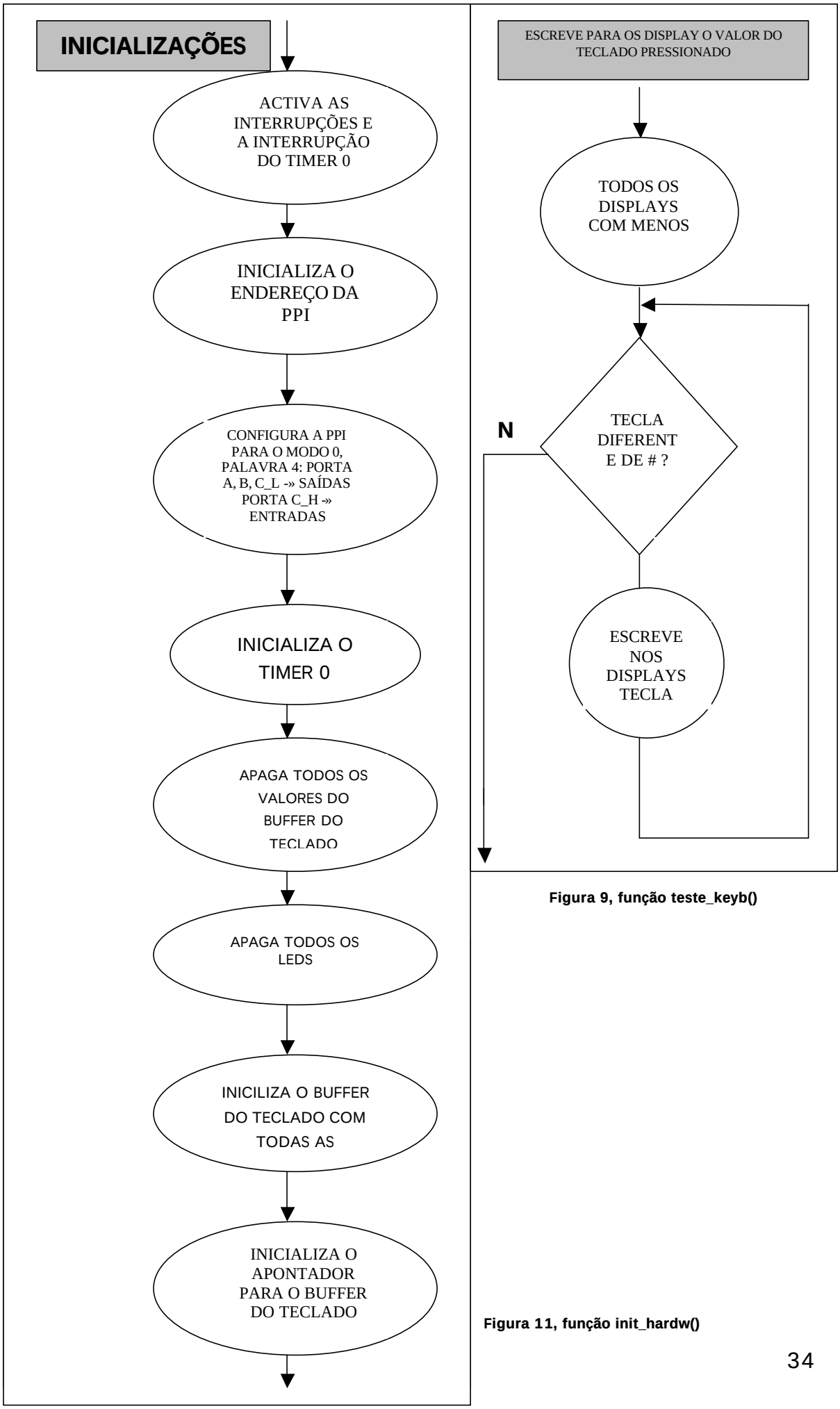


Figura 9, função teste_keyb()

Figura 11, função init_hardw()

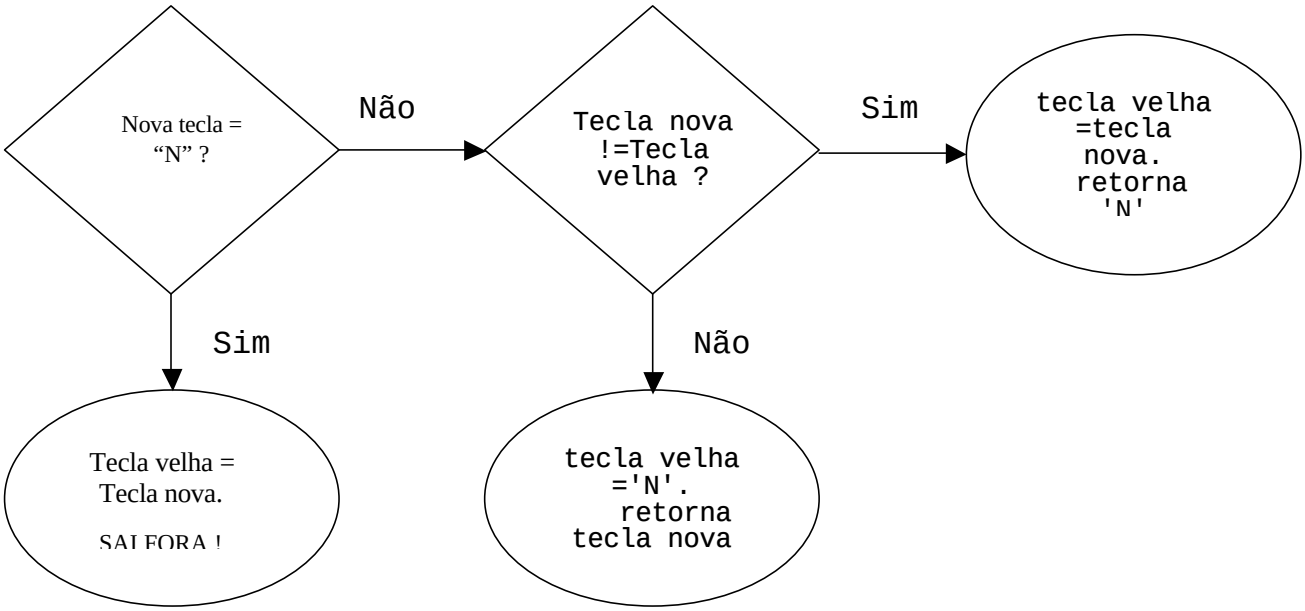


Figura 13, função key_validate()

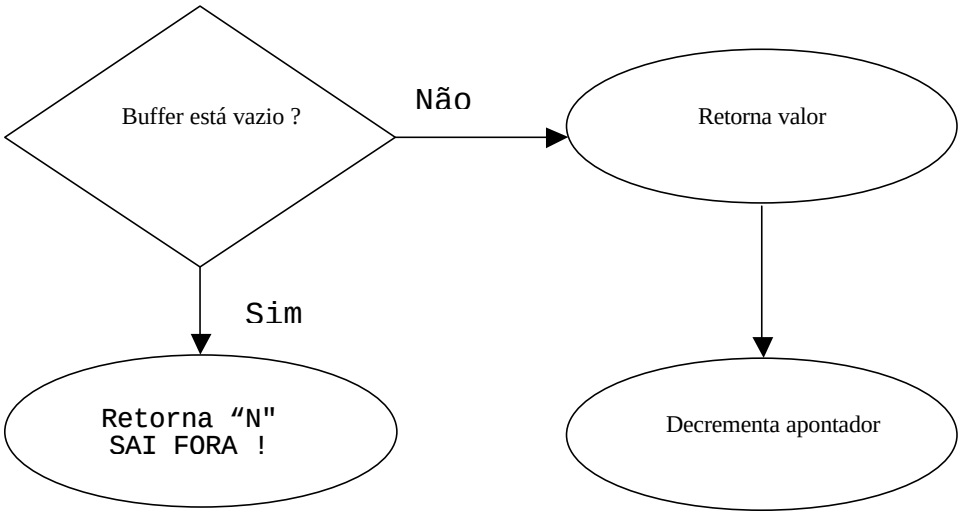


Figura 14, função rd_keyb_buf()

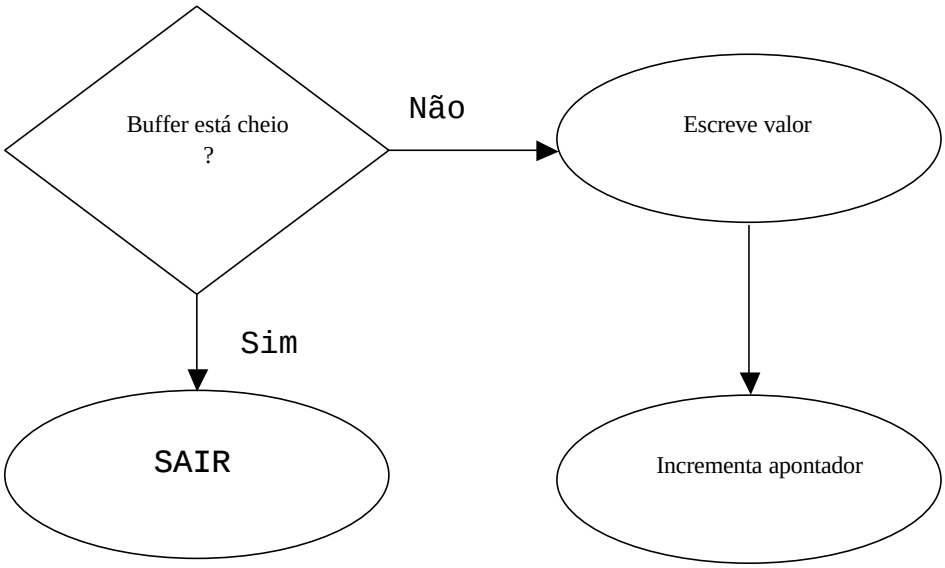


Figura 15, função wr_keyb_bf

A.4 Listagem do Código

fechadu.h

```
/*************************************************************
                      FECHADURA ELECTRÓNICA

                   FEUP * LEEC * SUC * 2001/02

               Rui Sousa * Pedro Leal * Daniel Duarte
*************************************************************/
/*************************************************************
                      MÓDULO: fechadu.h
       Concentra num só ficheiro todas as definições
       de todo o programa
*************************************************************/
/*-------------------------------------------------------------
                      Includes
-------------------------------------------------------------*/
#include <atmel\at89x51.h>                  // define os registros do 8051
#include <intrins.h>                        // define as funções intrinsecas do
compilador

/*-------------------------------------------------------------
                      Defines
-------------------------------------------------------------*/
#define uchar  unsigned char
#define uint   unsigned int


// definições do timer
#define TIMER0_COUNT   60928                // define o número a contar pelo timer0
para 5ms:65536-5000*11.0592/12
#define TIMER0_H       TIMER0_COUNT >> 8     // define o valor alto do timer0,
fazendo 1 shift para a direita
#define TIMER0_L       TIMER0_COUNT & 0x00ff  // define o valor baixo do timer0,
limpando os 8 bits menos significativos


#define NUM_DISPLAYS  3
// definições do display
#define ZERO   0xFA    // valor binário a escrever para um display de 7 segmentos de
modo a que seja escrito o valor correspondente
#define UM     0xA0
#define DOIS   0x79
#define TREZ   0xB9
#define QUATRO 0xA3
#define CINCO  0x9B
#define SEIS   0xDB
#define SETE   0xA8
#define OITO   0xFB
#define NOVE   0xBB
#define LETRA_A        0xEB
#define LETRA_C        0x5A
#define LETRA_B        0xD3
#define LETRA_D        0xF1
#define MENOS  0x01
#define PONTO  0x04
#define NADA   0x00
#define OITOP  0xFF
#define ASTE   0xC1
#define CARDI  0xD0

//definições dos LEDS
#define LEDVERM P1_4                // no core LED1
#define LEDVERD        P1_5         // LED2
#define TRINCO P1_6          // LED3
#define PISCA_LED_CT 100     // numero de interrupções antes de trocar o valor do
led

// definições do teclado
#define KEYB_BUF_SIZE 5             // pode ir desde 1 até ...
```

```
#define NUM_KEYB_DELAY 12     /* 10 é bom para quem pressiona rapidamente o teclado e
15 para pressionar lentamente. 12 é um meio termo bastante satisfatório*/

// definições do estado da programa principal
#define ESPERA                0
#define RECEBE_CODIGO         1
#define VALIDA_CODIGO         2
#define CODIGO_CERTO          3
#define CODIGO_ERRADO         4
#define MODO_UTILIZADOR_ESP   5

#define NUM_DIGITOS_CODIGO    3


#define ON     0
#define OFF    1

/*-------------------------------------------------------------
                      Variáveis Externas
-------------------------------------------------------------*/
// apontadores para os endereços da PPI
extern unsigned char xdata *DISPLAY;
extern unsigned char xdata *DISPLAY_EN;
extern unsigned char xdata *KEYB;
extern unsigned char xdata *PPI_CONTROL;


/*-------------------------------------------------------------
Protótipos Externos de Todas as Funções dos vários Módulos
-------------------------------------------------------------*/
extern void init_keyb(void);          // inicializa o teclado
extern void init_hardw (void);                // initializa todo o hardware
extern uchar key_converter (uchar tecla);     // converte teclas em valores do display
extern uchar key_validate(uchar new_key);     // valida tecla lida
extern uchar keyb_scan (void);                // faz o scan do teclado
extern uchar rd_keyb_buf (void);      // le o buffer do teclado
extern void clr_keyb_buf(void);               // apaga o buffer do teclado
extern void delay (void);             // função de espera
extern void en_disp (uchar display);  // activa o display selecionado. 0 desativa
todos
extern void teste_7seg (void);                // faz o teste dos displays
extern void teste_7seg2 (void);               // faz outro testes aos displays
extern void teste_keyb (void);                // testa o teclado
extern void teste_leds (void);                // testa os leds
extern void time_ct (uint valor);     // recebe o valor em ms para contar.timer0_ct
vem a 0 qd o tempo tiver passado
extern void timer0_init (void);               // inicializa o timer 0
extern void timer0_int (void);                // função de interrupção do timer 0
extern void wr_disp (uchar valor, uchar disp); // escreve no buffer do display disp o
valor valor. se disp==9 escreve em todos
extern void wr_keyb_buf (uchar valor);        // escreve no buffer do teclado
extern void wr_keys (uchar tecla, uchar display);    // escreve no display o valor da
tecla
```

init.c

```
/*************************************************************
                      FECHADURA ELECTRÓNICA

                   FEUP * LEEC * SUC * 2001/02

               Rui Sousa * Pedro Leal * Daniel Duarte
*************************************************************/



/*************************************************************
                      MÓDULO: init.c
       Módulo responsável pela inicializaçã´do hardware
*************************************************************/



/*------------------------------------------------------------
                      Include
------------------------------------------------------------*/
#include "fechadu.h"



/*------------------------------------------------------------
               Declaração de Variáveis Globais
------------------------------------------------------------*/
unsigned char xdata *PPI_CONTROL;     // apontadores para os endereços da PPI
unsigned char xdata *DISPLAY;
unsigned char xdata *DISPLAY_EN;
unsigned char xdata *KEYB;



/*------------------------------------------------------------
                      Protótipos de Funções
------------------------------------------------------------*/
void init_hardw (void);                      // initializações do hardware



/*------------------------------------------------------------
                      Funções
------------------------------------------------------------*/
void init_hardw (void){

       // inicializações

       EA = ET0 = 1;                 // activa as interrupções e a interrupção do
TIMER0

       DISPLAY_EN = 0xE000;          // initializa o endereço da PPI
       DISPLAY = 0xE001;
       KEYB = 0xE002;
       PPI_CONTROL = 0xE003;

       *PPI_CONTROL = 0x88;                  // configura a PPI para o Modo 0,
palavra 4: Porta A, B, C_L -> saídas. Porta C_H -> entradas
       *DISPLAY_EN = *DISPLAY = *KEYB = 0XFF;       // Inicializa todas as saídas da
PPI a 1

       timer0_init();                // inticializa o timer0

       wr_disp(NADA,9);              // apaga todos os valores do buffer do teclado

       LEDVERD=LEDVERM=TRINCO=OFF;   // apaga todos os leds

       init_keyb();                  // inicializa o teclado

}
```

display.c

```
/*************************************************************
                      FECHADURA ELECTRÓNICA

                    FEUP * LEEC * SUC * 2001/02

               Rui Sousa * Pedro Leal * Daniel Duarte
*************************************************************/

/*************************************************************
                      MÓDULO: display.c
       Implementa todas as funções de gestão dos displays
*************************************************************/

/*-------------------------------------------------------------
                      Include
-------------------------------------------------------------*/
#include "fechadu.h"

/*-------------------------------------------------------------
                      Variáveis Globais
-------------------------------------------------------------*/
uchar disp_bf[NUM_DISPLAYS];  // guarda o valor a escrever em cada display

/*-------------------------------------------------------------
                      Protótipos de Funções
-------------------------------------------------------------*/
void en_disp (uchar display);        // activa o display 1,2 ou 3. 0 desativa todos
void wr_disp (uchar valor, uchar disp); // escreve no buffer do display. se disp=9
escreve em todos
uchar key_converter (uchar tecla);   // converte um valor de tecla no valor a escreer
nos displays
void wr_keys (uchar tecla, uchar display);   // escreve uma tecla mos displays

/*-------------------------------------------------------------
                      Funções
-------------------------------------------------------------*/
void en_disp (uchar display){

       // activa o valor display que se pretende acender

       switch (display){
              case 0: *DISPLAY_EN = 0xFF; break;    // com 0 desactiva os displays
              case 1: *DISPLAY_EN = 0xFE; break;
        case 2: *DISPLAY_EN = 0xFD; break;
            case 3: *DISPLAY_EN = 0xFB; break;
              default: *DISPLAY_EN = 0xF8; break;  // se isto se passar acende
todos. ficam mais claros
              }
}

void wr_disp (uchar valor, uchar disp){

       // escreve um valor na posicao display do buffer do teclado
       // se display==9 escreve para todos os displays

       uchar tmp=0;
       if (disp == 9){
              for (tmp=0; tmp<=(NUM_DISPLAYS-1); tmp++){
                     disp_bf[tmp]=valor;
              }
       }
       else {
              disp_bf[disp]=valor;
       }
}

uchar key_converter (uchar tecla){

       // converte o valor da tecla lido no valor a escrever para
       // o display, de modo a acender os respectivos leds

       switch (tecla){
              case '1': return UM; break;
```

```
                case '2': return DOIS; break;
                case '3': return TREZ; break;
                case '4': return QUATRO; break;
                case '5': return CINCO; break;
                case '6': return SEIS; break;
                case '7': return SETE; break;
                case '8': return OITO; break;
                case '9': return NOVE; break;
                case '0': return ZERO; break;
                case 'A': return LETRA_A; break;
                case 'B': return LETRA_B; break;
                case 'C': return LETRA_C; break;
                case 'D': return LETRA_D; break;
                case '*': return ASTE; break;
                case '#': return CARDI; break;
                case 'N': return NADA; break;

                // se der erro acende todos os leds:
                default: return OITOP; break;
        }
}

void wr_keys (uchar tecla, uchar display){

        // escreve a tecla lida para o display
        // se 9 escreve para todos os displays

        wr_disp(key_converter(tecla), display);

}
```

keyb.c

```
/*************************************************************
                        FECHADURA ELECTRÓNICA

                    FEUP * LEEC * SUC * 2001/02

               Rui Sousa * Pedro Leal * Daniel Duarte
*************************************************************/

/*************************************************************
                        MÓDULO: keyb.c
       Implementa todas as funções de gestão do teclado
*************************************************************/

/*-----------------------------------------------------------
                        Include
-----------------------------------------------------------*/
#include "fechadu.h"

/*-----------------------------------------------------------
                        Variáveis Globais
-----------------------------------------------------------*/
// teclado
uchar old_key = 'N';          // guarda última tecla
uchar keyb_buf[KEYB_BUF_SIZE];       // buffer do teclado
uchar *keyb_buf_pt;           // apontador para o buffer do teclado

// array com os valores do teclado. 'N' significa nada ou nenhum
unsigned char teclas[] =
{'1','2','3','A','4','5','6','B','7','8','9','C','*','0','#','D','N'};

/*-----------------------------------------------------------
                        Protótipos de Funções
-----------------------------------------------------------*/
void init_keyb(void);                // inicializa o teclado
void wr_keyb_buf (uchar valor);              // escreve para o buffer do teclado
uchar rd_keyb_buf (void);            // lê o buffer do teclado
void clr_keyb_buf(void);             // apaga o buffer do teclado
uchar keyb_scan(void);               // faz o scan do teclado
uchar key_validate(uchar new_key);   // valida ou não a nova tecla

/*-----------------------------------------------------------
                        Funções
-----------------------------------------------------------*/
void init_keyb(void){

        // função de inicialização do teclado

        uchar tmp;

        // inicializa o buffer do teclado com todas as posições vazias
        for (tmp=0; tmp<=(KEYB_BUF_SIZE-1); tmp++){
                keyb_buf[tmp]='N';
        }

        // initializa o apontador para o buffer do teclado
        keyb_buf_pt = &keyb_buf[0];
}

void wr_keyb_buf (uchar valor){

        //escreve para o buffer do teclado

        if (valor != 'N'){                                  // se valor for diferente
de nada faz
                if (keyb_buf_pt <= &keyb_buf[KEYB_BUF_SIZE-1]) {// se não estiver
cheio
                        *keyb_buf_pt = valor;               // escreve valor
                        keyb_buf_pt ++;                             // incrementa
apontador
                }                                           // se estiver cheio não
faz nada
        }
}
```

```
uchar rd_keyb_buf (void){

        // lê do buffer di teclado

        if (keyb_buf_pt > &keyb_buf[0]) {            // se não estiver vazio
                keyb_buf_pt --;                              // decrementa o apontador
                return *keyb_buf_pt;                 // retorna o valor
        }
        else {return 'N';}                           // se estiver vazio retorna NADA
}

void clr_keyb_buf(void){
        // apaga o buffer do teclado
        keyb_buf_pt=&keyb_buf[0];            // inciia apontador para a 1ª posição
}

uchar keyb_scan(void){

        // rastreia o teclado para detectar uma tecla
        uchar coluna=0x0F, i=0,linha=0xFE;

        for (linha=0; linha<=4 & coluna == 0x0F; linha++){
                // escreve cada uma das linhas a 0, uma de cada vez
                switch (linha){
                        case 0:{ *KEYB = 0xFE; break;}
                        case 1:{ *KEYB = 0xFD; break;}
                        case 2:{ *KEYB = 0xFB; break;}
                        case 3:{ *KEYB = 0xF7; break;}
                        default:{ *KEYB = 0xFE;       break;}
                }
                coluna= *KEYB;
                coluna = coluna >> 4;
        }

        switch (coluna){
                // detecta valor lido na coluna
                case (0x0E): coluna = 1; break;
                case (0x0D): coluna = 2; break;
                case (0x0B): coluna = 3; break;
                case (0x07): coluna = 4; break;
                default: coluna = 5;
        }

        // se não detectou nada então garante que retorna 'N'
        if (coluna == 5) { linha =4;}

        // retorna o valor correcto do array
        return (teclas [((linha-1)*4)+coluna-1]);
}

uchar key_validate(uchar new_key){
        // compara a nova tecla com a tecla antiga e retorna o valor correcto

        if (new_key == 'N') {old_key = 'N';return 'N';}     // se não houve uma tecla
premida actualiza old_key
        else {                                       // se houve tecla premida faz:
                if (new_key != old_key){             // se a nova tecla é diferente da antiga
faz:
                old_key = new_key;                   // actualiza tecla_antiga com novo valor
                return 'N';
                }

                else {                               // se a tecla é igual à antiga
                        old_key = 'N';               // re-inicia tecla antiga
                        return new_key;
                }
        }
}
```

intrpt.c

```
/*************************************************************
                      FECHADURA ELECTRÓNICA

                   FEUP * LEEC * SUC * 2001/02

               Rui Sousa * Pedro Leal * Daniel Duarte
*************************************************************/



/*************************************************************
                      MÓDULO: intrpt.c
       Controla as rotinas de interrupção do sistema
*************************************************************/



/*-------------------------------------------------------------
                      Include
-------------------------------------------------------------*/
#include "fechadu.h"



/*-------------------------------------------------------------
                      Variáveis Globais
-------------------------------------------------------------*/
uchar disp_actual = 1;        // número do último display escrito. varia entre 1 e
NUM_DISPLAYS
uint timer0_ct = 0;           // contador do timer 0. 0 se tempo passou. é usado por
outros módulos

uchar keyb_ct=NUM_KEYB_DELAY; // contador de interrupções para leitura do teclado
uchar led_ct=PISCA_LED_CT;    // contador de interrupções para piscar o LED VERDE



/*-------------------------------------------------------------
                      Variáveis Globais Externas
-------------------------------------------------------------*/
extern uchar nextstate;                     // próximo estado do programa principal
extern uchar disp_bf[NUM_DISPLAYS];  // buffer do display



/*-------------------------------------------------------------
                      Protótipos de Funções
-------------------------------------------------------------*/
void timer0_int(void);        // rotina de interrupção do timer0
void timer0_init (void);      // initialização do timer0




/*-------------------------------------------------------------
                      Funções
-------------------------------------------------------------*/
void timer0_int(void) interrupt 1 {

       // rotina de interrupção do timer 1

       uchar new_key = 'N';          // inicializa a nova tecla


       // trata do teclado:
       if (keyb_ct==0){                                    // se o contador de
interrupções já está a zero faz:
              wr_keyb_buf( key_validate( keyb_scan()));    // key validate escreve o
novo valor caso seja validado usando wr_keyb_bf();
              keyb_ct = NUM_KEYB_DELAY;                    // re-inicia o contador
de interrupções
       }
       else (keyb_ct--);
```

```
        //trata do display
        en_disp(0);                             // apaga os displays

        disp_actual++;                          // incrementa o valor do display actual
        if (disp_actual==(NUM_DISPLAYS+1)) disp_actual=1;

        *DISPLAY = disp_bf[disp_actual-1];      // escreve o novo valor
        en_disp(disp_actual);                   // activa o novo display


        // contador de tempo
        if (timer0_ct>0){                       // se ainda não for 0 decrementa
                timer0_ct--;
        }


        // led verde
        if (led_ct==0){                                 // se já for 0 faz:
                if (nextstate==MODO_UTILIZADOR_ESP){LEDVERD=~LEDVERD;}
                led_ct=PISCA_LED_CT;            // re-inicia contador
        }
        else {led_ct--;}                        // se não for 0 decrementa


        // timer
        timer0_init();                          // reinicia o timer0
}



void timer0_init(void){

        // initializa o timer0

        TR0 = 0;                // para o timer0
        TMOD = 0x11;            // timer 0,1 como TIMERS, controlados por TRx, modo 1
        TH0 = TIMER0_H;                 // carrega o valor inicial da contagem
        TL0 = TIMER0_L;
        TR0 = 1;                // arranca o timer0
}
```

time.c

```
/*************************************************************
                      FECHADURA ELECTRÓNICA

                   FEUP * LEEC * SUC * 2001/02

               Rui Sousa * Pedro Leal * Daniel Duarte
*************************************************************/


/*************************************************************
                      MÓDULO: time.c
       Implementa as funções de controlo do tempo
*************************************************************/


/*-------------------------------------------------------------
                      Include
-------------------------------------------------------------*/
#include "fechadu.h"



/*-------------------------------------------------------------
                      Variáveis Globais
-------------------------------------------------------------*/
extern uint timer0_ct;        // contador do timer0. 0 se tempo passou



/*-------------------------------------------------------------
                      Protótipos de Funções
-------------------------------------------------------------*/
void time_ct (uint tempo);    // carrega o contador que valor de tempo adequado
void delay (void);            // fica à espera enquanto tempo não passar



/*-------------------------------------------------------------
                      Funções
-------------------------------------------------------------*/
void time_ct (uint tempo){

/*recebe um valor inteiro em ms (maior que 5 e múltiplo de 5
       escreve no contador de tempo o valor correspondente para contar
       esse tempo com interrupções de cada 5ms.

       Está limitado a 65536 contagens, o que significa:
              65536*5ms = 327680 ms = 327s = 5.45minutos
       Se for necessário mais tempo pode-se passar de int para double
       ou usar outra variavel char para contar no máximo 256*5.45min

       tempos comuns: 5min    300000ms60000 contagens
                      1min    60000ms 12000 contagens
                      10 s    10000ms 2000 contagens
                      5 s     5000ms  1000 contagens
                      1 s     1000ms  200 contagens*/

       timer0_ct = (tempo/5);
}


void delay (void){

       // enquanto a variavel timer0_ct for 0 não faz nada

       while (timer0_ct > 0) {_nop_();}
}
```

testes.c

```
/*************************************************************
                      FECHADURA ELECTRÓNICA

                   FEUP * LEEC * SUC * 2001/02

               Rui Sousa * Pedro Leal * Daniel Duarte
*************************************************************/



/*************************************************************
                      MÓDULO: testes.c
       Implementa todas as rotinas de teste do sistema
*************************************************************/



/*-------------------------------------------------------------
                      Include
-------------------------------------------------------------*/
#include "fechadu.h"



/*-------------------------------------------------------------
                      Protótipos de Funções
-------------------------------------------------------------*/
void teste_7seg (void);              // teste dos displays
void teste_7seg2 (void);      // outro teste do display
void teste_leds (void);              // teste dos leds
void teste_keyb (void);              // teste do teclado



/*-------------------------------------------------------------
                      Funções
-------------------------------------------------------------*/
void teste_7seg (void) {

       // testa o displays de 7 segmentos

       uchar array[]= {ZERO, UM, DOIS, TREZ, QUATRO, CINCO, SEIS, SETE, OITO, NOVE,
LETRA_A, LETRA_B, LETRA_C, LETRA_D, ASTE, CARDI, PONTO, MENOS, NADA};
       uchar tmp=1, display=1, valor=0, tecla_lida='N';                          //
contadores para os for e while

       wr_disp(NADA,9);                                                 // garante que o
buffer está a 0

              for (valor=0; valor<=18; valor++){                        // permite
percorrer todos os valores do array
              if (tecla_lida=='#') {break;}

                     for (display=0; display < NUM_DISPLAYS; display++){  // permite
percorrer os 3 displays

                            for (tmp=0; tmp < NUM_DISPLAYS; tmp++){           //
escreve no buffer dos displays

                                   if (tmp==display) wr_disp(array[valor],display);
                                   else wr_disp(NADA,tmp);

                            }

                            time_ct (500);                                    //
inicializa o contador do timer 0 para 1000ms
                            delay ();                                         // fica à
espera durante 2s
                            tecla_lida=rd_keyb_buf();
                            if (tecla_lida=='#') {break;}
                     }
```

```
                        wr_disp (array[valor],9);                               // escreve
todos iguais
                        time_ct (500);                                          //
inicializa o contador do timer 0 para 2000ms
                        delay ();                                               // fica à
espera durante 2s
                }


        if (tecla_lida=='#') {return;}
        teste_7seg2();                                          // chama o teste 2
}



void teste_7seg2(void){

        // acende durante um perído todos os displays
        wr_keys ('8',9);              // escreve 8 no display

        time_ct (2000);                       // inicializa o timer 0 para 2000ms
        delay ();                     // fica à espera

        wr_disp (NADA,9);             // apaga o display
}



void teste_leds(void){

        // escreve alternadamente para todos os leds

        LEDVERM = ON;
        LEDVERD = TRINCO = OFF;
        time_ct (500);
        delay ();

        LEDVERM = TRINCO = OFF;
        LEDVERD=ON;
        time_ct (500);
        delay();

        LEDVERM=LEDVERD=1;
        TRINCO=ON;
        time_ct (500);
        delay();

        LEDVERM=LEDVERD=TRINCO=OFF;
}



void teste_keyb(void){

        //escreve para os displays o valor da tecla pressionada

        uchar tmp='N';

        wr_disp(MENOS,9);

                while (tmp != '#'){
                        tmp=rd_keyb_buf();
                        if (tmp!='N'){
                                wr_keys (tmp,9);
//                              LEDVERM=~LEDVERM;     //Usado para teste
                        }
                }
}
```

main.c

```
/************************************************************
                        FECHADURA ELECTRÓNICA

                    FEUP * LEEC * SUC * 2001/02

                Rui Sousa * Pedro Leal * Daniel Duarte
************************************************************/


/************************************************************
                        MÓDULO: main.c
        Implementa todo o programa principal.
************************************************************/


/*-----------------------------------------------------------
                        Include
-----------------------------------------------------------*/
#include "fechadu.h"



/*-----------------------------------------------------------
                        Variáveis Globais
-----------------------------------------------------------*/
//timer
extern uint timer0_ct;        // flag que assinala que o tempo passou

// variaveis do main
uchar nextstate=ESPERA;              // próximo estado
uchar nextsubstate=UM;        // próximo sub-estado

uchar const default_user_code[]={'1','2','3'};
uchar const default_special_code[]={'9','8','7'};
uchar user_code[NUM_DIGITOS_CODIGO];
uchar special_code[NUM_DIGITOS_CODIGO];
uchar temp_code[NUM_DIGITOS_CODIGO];




/*-----------------------------------------------------------
                        Protótipos de Funções
-----------------------------------------------------------*/
uchar compara_codigos(void);
void reset_codes(void);
void change_user_code(void);
void change_special_code(void);
void main(void);



/*-----------------------------------------------------------
                        Funções
-----------------------------------------------------------*/
uchar compara_codigos(void){

/*      Comapara as variáveis globais temp_code[] com special_code[]
        e user_code[] e retorna:
                2 se for igual a special code
                1 se for igual a user code
                0 nas outras situações */

        uchar i,u_code=1,s_code=1;

        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){
                if (user_code[i]!=temp_code[i] | u_code!=1){
                        u_code=0;       // nesta situação esta flag fica a 0
                }

                if (special_code[i]!=temp_code[i] | s_code!=1){
                        s_code=0;
                }
```

```
        }
        // código especial == codigo normal retorna modo especial
        if (s_code==1) {return 2;}    // retorna 2 se igual a special code
        if (u_code==1) {return 1;}    // retorna 1 se igual a user code
        else {return 0;}              // retorna 0 nas outras situações

}



void reset_codes(void){

        // coloca todos os códigos iguais ao seu valor default

        uchar i;
        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){
                user_code[i]=default_user_code[i];
                special_code[i]=default_special_code[i];
        }
}



void change_user_code(void){

        /* muda o código de utilizador para o valor que vai recebendo
        do teclado. Só activa a alteração quando se pressiona em * no fim*/

        uchar i, tecla_lida='N';

        wr_disp(MENOS,9);

        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){
                do{tecla_lida=rd_keyb_buf();}
                while( tecla_lida=='N' | tecla_lida=='*');

                if (tecla_lida=='#') {break;}
                else{

                        temp_code[i]=tecla_lida;
                        wr_keys(tecla_lida,i);
                }

        }

        if (tecla_lida=='#') {return;}
        while (tecla_lida!='*' & tecla_lida!='#'){
                tecla_lida=rd_keyb_buf();
        }
        if (tecla_lida=='#') {return;}


        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){
                user_code[i]=temp_code[i];
                temp_code[i]='N';
        }

        wr_disp(MENOS,9);
         tecla_lida='N';
}



void change_special_code(void){

        /* muda o código de utilizador especial para o valor que vai
        recebendo do teclado. Só activa a alteração quando se
        pressiona em * no fim */

        uchar i, tecla_lida='N';

        wr_disp(MENOS,9);

        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){
                do{tecla_lida=rd_keyb_buf();}
```

```
                while( tecla_lida=='N' | tecla_lida=='*');

                if (tecla_lida=='#') {break;}
                else{

                        temp_code[i]=tecla_lida;
                        wr_keys(tecla_lida,i);
                }

        }

        if (tecla_lida=='#') {return;}
        while (tecla_lida!='*' & tecla_lida!='#'){
                tecla_lida=rd_keyb_buf();
        }
        if (tecla_lida=='#') {return;}


        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){
                special_code[i]=temp_code[i];
                temp_code[i]='N';
        }

        wr_disp(MENOS,9);
         tecla_lida='N';
}



void main(void){
        uchar i, tecla_lida='N';

        init_hardw();  // inicializa o hardware


        // inicializa o software
        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){                                          //
initializa os códigos iguais ao default
                user_code[i]=default_user_code[i];
                special_code[i]=default_special_code[i];
                temp_code[i]='N';
        }

        tecla_lida='N';


        while (1){
                LEDVERM=LEDVERD=TRINCO=OFF;
                wr_disp(NADA,9);


                switch (nextstate){
                        case ESPERA:{

                                // para garantir que a sistema não fica "pendurado" foi
necessário incluir todos estes ciclos
                                while (tecla_lida=='N' | tecla_lida=='#' |
tecla_lida=='*'){
                                        wr_disp(PONTO,9);
                                        time_ct(500);
                                        while (timer0_ct>0 & (tecla_lida=='N' |
tecla_lida=='#' | tecla_lida=='*')){
                                                tecla_lida=rd_keyb_buf();
                                                _nop_();
                                        }
                                        if (tecla_lida!='N' & tecla_lida!='#') {break;}
                                        else {
                                                wr_disp(NADA,9);
                                                time_ct(500);
                                                while (timer0_ct>0 & (tecla_lida=='N' |
tecla_lida=='#' | tecla_lida=='*')){
                                                        tecla_lida=rd_keyb_buf();
                                                        _nop_();
                                                }
                                        }
                                }
```

```
                                nextstate=RECEBE_CODIGO;
                                break;
                        }

                        case RECEBE_CODIGO:{


                                /* Recebe os 3 valores do código para o array
temporário
                                - ao fim de 60 segundos sem se introduzir nenhum valor
sai fora;
                                - # manda o sistema para o estado anterior
                                - * não é aceite como código
                                - só aceita os primeiros 3 dígitos introduzidos
                                */


                                wr_disp(NADA,9);                        // apaga os
displays


                                time_ct(60000);                                 // inicia
o timer com o tempo máximo de espera

                                temp_code[0]=tecla_lida;                // escreve no
array temporário a tecla lida
                                wr_disp(MENOS,0);                       // escreve no
display o primeiro traço
//                              wr_keys(tecla_lida,0);


                                for (i=1;i<=NUM_DIGITOS_CODIGO-1 & timer0_ct>0  &
tecla_lida!='#';){      // ciclo para os números do código que faltam
                                        do { tecla_lida=rd_keyb_buf(); }
                                        while (tecla_lida=='N' & timer0_ct>0);

                                        if (tecla_lida!='*' & tecla_lida!='#'){
                                                temp_code[i]=tecla_lida;
                                                wr_disp(MENOS,i);
//                                              wr_keys(tecla_lida,i);
                                                i++;
                                        }

                                }

                                // espera por 1 *
                                if (tecla_lida!='#' & timer0_ct>0){
                                        do {tecla_lida=rd_keyb_buf(); }
                                        while (tecla_lida != '*' & timer0_ct>0 &
tecla_lida != '#');
                                }

                                if (timer0_ct == 0 | tecla_lida=='#') {
                                        nextstate=ESPERA;
                                        break;
                                }

                                if (timer0_ct > 0 & tecla_lida=='*') {
                                        nextstate=VALIDA_CODIGO;
                                        break;
                                }

                                else {
                                        nextstate=ESPERA;
                                        break;
                                }


                        case VALIDA_CODIGO:{
                                // Compara valores recebidos com os códigos actuais
                                i=compara_codigos();   // a fução compara codigos
compara cod_tmp com codigo_utilizador

                                if (i==1){nextstate=CODIGO_CERTO;}
                                else { if (i==2){nextstate=MODO_UTILIZADOR_ESP;}
                                        else {nextstate=CODIGO_ERRADO;}
                                }
```

```
                        for (i=0;i<=NUM_DIGITOS_CODIGO-1;i++){
                // re-inicia o código temporário
                                temp_code[i]='N';
                        }

                        break;
                }


                }


                case CODIGO_CERTO:{
                        LEDVERD = ON;           // acende led verde
                        TRINCO = ON;            // activa o trinco
                        time_ct(1000);          // activa a espera durante 1s
                        delay();                // fica à espera que esse segundo
passe
                        LEDVERD=TRINCO=OFF;     // apaga os leds
                        nextstate=ESPERA;       // no próximo estado volta ao
estado de espera
                        clr_keyb_buf();                 // apaga todas as teclas
lidas -> ignora esses inputs
                        break;                  // sai fora
                }

                case CODIGO_ERRADO:{
                        LEDVERM = ON;           // acende o led vermelho
                        wr_disp(OITOP,9);       // acendo todos os leds de todos
os displays
                        time_ct(5000);          // activa o contador para 5s
                        delay();                // fica à espera durante esse
tempo
                        wr_disp(NADA,9);        // apaga o buffer dos displays
                        LEDVERM = OFF;          // apaga o led vermelho
                        nextstate=ESPERA;       // o próximo estado é o de espera
                        clr_keyb_buf();                 // apaga todos os valores
no buffer do teclado
                        break;                  // salta fora para recomeçar
                }
                case MODO_UTILIZADOR_ESP:{

                        // fica à espera de uma tecla
                        time_ct(60000);
        // inicia o contador de tempo
                        do {tecla_lida=rd_keyb_buf(); }
                        while ((tecla_lida=='N'|tecla_lida=='*') & timer0_ct
>0);

                        if (tecla_lida=='#' | timer0_ct==0) {nextstate=ESPERA;
break;} // se tecla encontrada é # ou o tempo esgotou, sai fora

                        // converte o valor da tecla
                        tecla_lida=key_converter(tecla_lida);               //
converte o valor da tecla para o valor escrevivel no ecra
                        nextsubstate=tecla_lida;                            //
próximo estado depende da tecla pressionada
                        wr_disp(tecla_lida,0);                              //
escreve no display 0 esse valor


                        time_ct(60000);
                // initializa o contador de tempo

                        // fica à epera de um *, # ou que o tempo esgote
                        do { tecla_lida = rd_keyb_buf();}
                        while (tecla_lida != '*' & tecla_lida != '#' &
timer0_ct > 0 );        // enquanto não se carregar no *, no # ou o tempo não
esgotar lê o buffer do teclado


                        if (timer0_ct==0) {nextstate=ESPERA; break;}
        // se acabou o tempo então volta ao estado inicial

                        nextstate=MODO_UTILIZADOR_ESP;
```

```
                        if (tecla_lida == '#') {break;}      // se se carrega
no # então volta ao menu anterior, e re-inicia o menu de utilizador especial
                        if (tecla_lida == '*') {      // se se carregou no *
entra nos sub-estados

                                time_ct(250);
                                delay();

                                switch (nextsubstate){
                                        case UM:       teste_leds();  break;
                                        case DOIS:     teste_7seg();  break;
                                        case TREZ:     teste_keyb();  break;
                                        case SETE:     change_user_code();
        break;
                                        case OITO:     change_special_code();
        break;
                                        case NOVE:     reset_codes(); break;
                                        default:{
                                                wr_disp(OITOP,9);
                                                time_ct(2000);
                                                delay();
                                                break;
                                        }
                                }
                                wr_disp(NADA,9);
                        }
                        break;
                }
        }
    }
}
```

A.5 Configurações do Compilador

O compilador usado foi o µVision 2 versão V2.23. As configurações necessárias para o bom funcionamento do programa são as seguintes:

Target:

Memory Model: Small
Code Rom Size: Large
Operating System: None

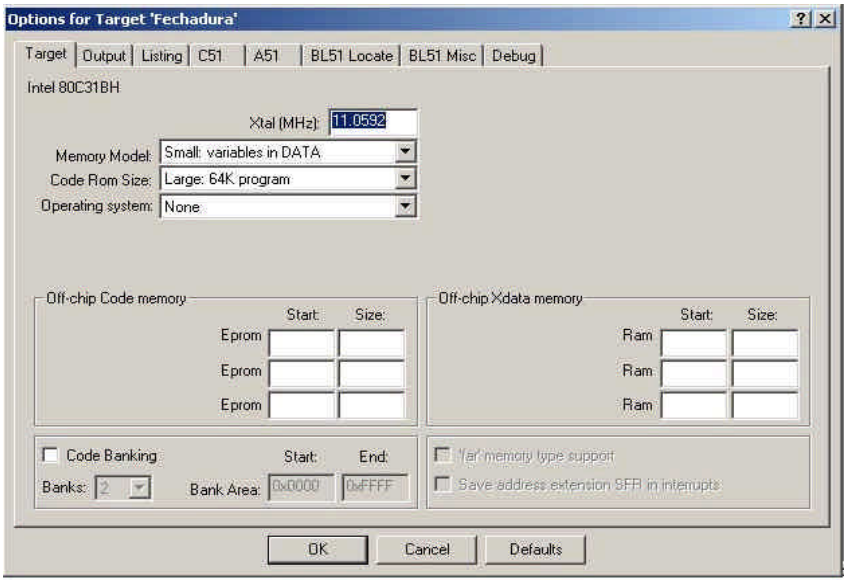


Figura 16

C51:

Level: 6: Loop Rotation
Emphasis: Favor Speed

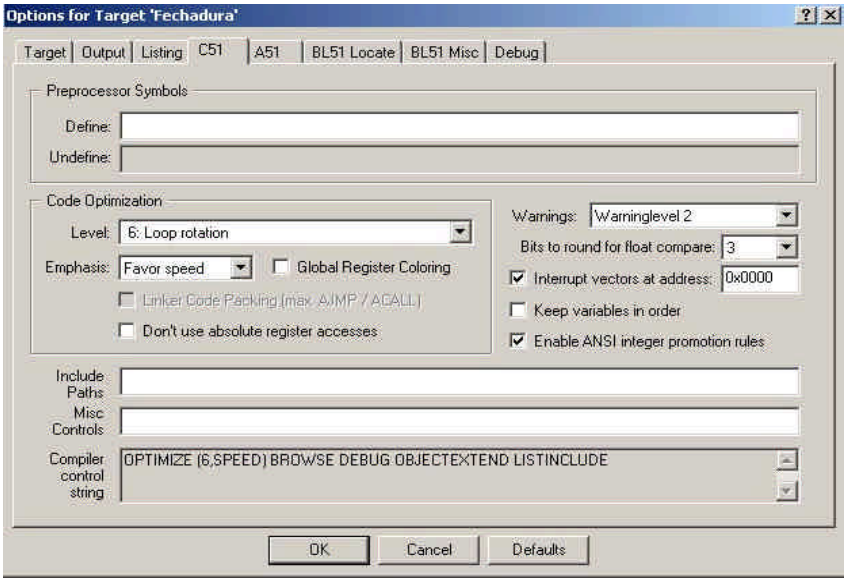


Figura 17

BL51 Locate:

Xdata:1400

Com uma memória de 8Kb este valor permite ter um código até 5120bytes, e um espaço para Xdata até 2880bytes.

Caso se pretenda também usar as funções de execução passo a passo e de paragem do programa é necessário incluir:

Code: 0030

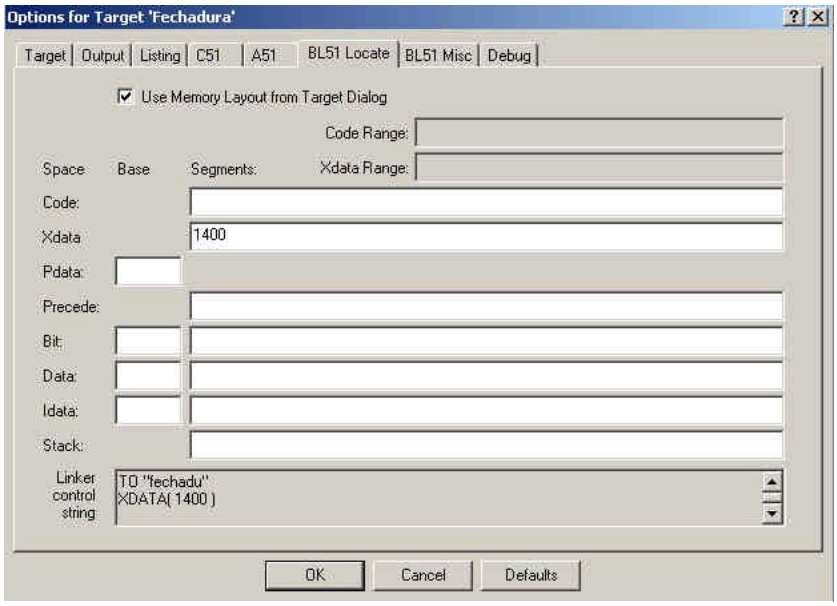


Figura 18

A.6 Manual do Utilizador

Para utilização desta fechadura automática deixamos aqui o seu manual de utilização.

Assim, considerámos as principais teclas o "*" que nos permite validar qualquer opção e o "#" que a qualquer momento nos concede o cancelamento (saída).

O estado espera ou inicial encontra-se registado com indicação de um ponto a piscar "." em todos os displays.

Os códigos pré definidos são "123" para o utilizador e "987" para o supervisor.

A introdução do código de utilizador seguida da sua validação por “*” permite abrir a porta caso o código esteja certo ou bloquear o sistema durante 5 segundos caso esteja errado. Na primeira situação acende-se o LED verde enquanto a porta é aberta e o sistema fica parado 1 segundo, na segunda acende-se o LED vermelho, os displays e o sistema fica parado 5 segundos.

Caso se introduza o código de supervisor entra-se no modo de configuração. Neste modo o LED verde está sempre a piscar, e estão disponíveis as seguintes opções:

1 seguido de * para teste dos leds:

Neste modo os 3 leds piscam alternadamente.

2 seguido de * para teste dos displays;

É escrito em todos os displays sequencialmente vários caracteres.

3 seguido de * para teste do teclado;

Neste teste sempre que se pressiona uma tecla o valor correspondente aparece escrito nos 3 displays.

7 seguido de * para alterar o código de utilizador

Marcar: código novo e novamente " * ", para alteração do código do utilizador;

8 seguido de * para alterar o código de supervisor

Marcar código novo e novamente " * ", para alteração do código do supervisor;

9 seguido de * ;

Permite reiniciar todos os códigos deixando-os com o seu valor por defeito

Em todos os estados (excepto quando o sistema está bloqueado devido a um código errado) é possível marcar “#” para cancelar os dígitos introduzidos ou sair para o estado anterior.

Existe também um temporizador que limita o tempo máximo para a introdução de dados. Caso o utilizador não pressione nenhuma tecla durante 1 minuto o sistema é automaticamente reenviado para o estado de espera.