Universidade Federal do ABC

Projeto 1

Jogo de Labirinto em LCD com o Kit EXTO XM852

Docente:

Germán Carlos Santos Quispe.

Discentes:

Gilmar Correia Jeronimo

Lucas Barboza Moreira Pinheiro

21 de Agosto de 2019

Universidade Federal do ABC

Projeto 1

Jogo de Labirinto em LCD com o Kit EXTO XM852

Trabalho para Avaliação na Disciplina de Aplicações de Microcontroladores da Universidade Federal do ABC.

Docente:

Germán Carlos Santos Quispe.

Discentes:

Gilmar Correia Jeronimo - 11014515

Lucas Barboza Moreira Pinheiro - 11017015

21 de Agosto de 2019

Sumário

1 Introdução **2**

2 Materiais, métodos e projeto **4**

2.1 Kit EXSTO XM 852 4

2.2 MIDE 51 4

2.3 SDCC 4

2.4 Gravador XM85X 5

2.5 Conversor Digital-Analógico 5

2.6 Auto-falante 5

3 Código e Lógica Desenvolvidas **6**

3.1 Configuração do *LCD* 6

3.2 Configuração da porta serial 10

3.3 Temporizadores 11

3.4 DAC e Auto-falante 13

3.5 Função Main 15

4 Conclusão **18**

5 Apêndice A - Códigos **20**

5.1 main.c 20

5.2 LCD.h 23

5.3 LCD.c 24

5.4 SERIAL.h 35

5.5 SERIAL.c 35

5.6 TIMER.h 39

5.7 TIMER.c 39

5.8 Math.h 41

5.9 Math.c 41

5.10 DAC.h 41

5.11 DAC.c 45

5.12 Makefile 46

1 Introdução

O projeto consiste na programação de um jogo de labirinto utilizando o kit MX 852, que será exibido no display LCD do próprio kit e cujos comandos serão entrados pelo usuário por meio do teclado do computador e passados ao programa via comunicação serial. Além disso, o jogo dispõe duas músicas: uma para a tela inicial e outra tocada durante o jogo , que utilizam o auto-falante e o DAC.

O objetivo do jogo é movimentar o jogador (representado por um ponto piscante no display) ao longo dos espaços em branco do labirinto e encontrar uma saída, o que significa fazer com que o jogador saia dos limites da tela pelas laterais ou pela parte superior. Quando isso ocorre, passa-se para o nível seguinte, para o qual haverá um novo labirinto, com uma progressão de dificuldade a cada nível.

O jogo se inicia com uma tela que mostra o nome do jogo ("Maze Game"), com uma música de fundo, e pede ao usuário que pressione a tecla espaço para iniciar o jogo. O jogo então exibe uma tela com instruções de como jogar e entra no seu primeiro nível. O jogador pode se mover em quatro direções (cima, baixo,esquerda,direita) e tais movimentos são controlados pelo usuário por meio do teclado do computador pelas teclas W, S, A e D respectivamente. A cada vez que o usuário passa de nível, uma mensagem aparece na tela mostrando o número do nível que virá a seguir e então um novo labirinto é gerado na tela. O jogo termina quando o usuário consegue fazer o jogador passar pela saída do último nível. Quando isso ocorre, a mensagem "Você venceu"aparece na tela.

Um fluxograma simplificado da lógica do jogo é apresentado na Figura 1.

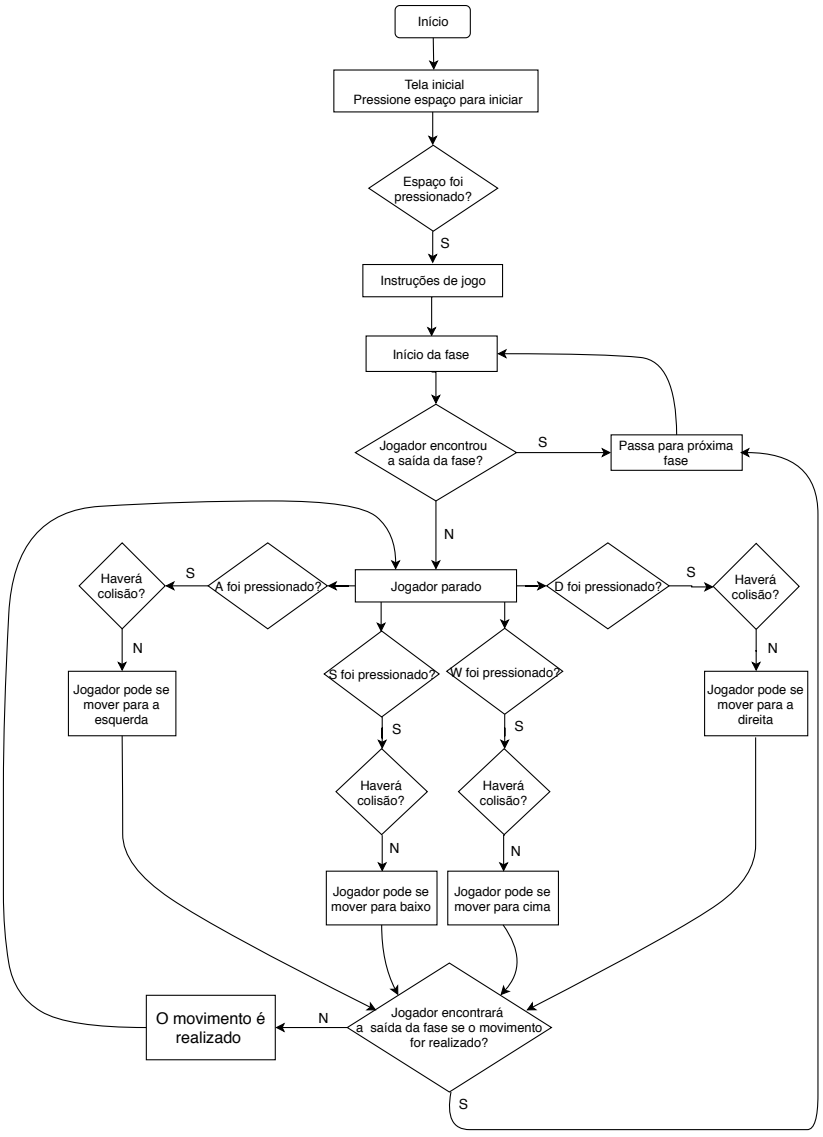


Figura 1: Fluxograma simplificado do jogo.

2 Materiais, métodos e projeto

Os equipamentos e softwares utilizados para este projeto estão descritos nesta seção.

2.1 Kit EXSTO XM 852

O kit EXSTO XM 852 é um kit educacional desenvolvido pela Exsto Tecnologias para disciplinas de arquitetura de computadores e tem como principal foco o uso da arquitetura 8051 em modo processador. O kit possui memórias externas EEPROM e RAM e diversas aplicações em portais mapeados em memória, além de alguns recursos conectados diretamente aos terminais do microcontrolador, como aplicações temos conversor A/D e D/A, motor de passo, display de 7 segmentos e display LCD.

O microcontrolador 8051 teve sua produção iniciada pela Intel em 1981 e atualmente é um dos mais populares no mercado, possuindo forte apelo didático devido à sua estrutura interna de fácil compreensão. Dentre suas principais características pode-se citar: frequência de clock de 12 MHz, dois temporizadores/ contadores de 16 bits, um canal de comunicação serial, cinco fontes de interrupção (dois timers, dois pinos externos e o canal de comunicação serial) e um oscilador de clock interno.

2.2 MIDE 51

MIDE-51 é um ambiente de desenvolvimento integrado (IDE) para o microcontrolador MCS-51. Seu pacote completo inclui o compilador Assembler, o Compilador C para Dispositivos Pequenos (SDCC), o Emulador TS Controls 8051 e o Simulador JSIM-51.

É por meio desta IDE que os programas que serão gravados na placa do kit são escritos, compilados e construídos (operação "build", responsável por gerar as extensões de arquivos que serão gravadas na memória do microcontrolador). Utiliza linguagem C.

2.3 SDCC

O Compilador C para Dispositivos Pequenos (Small Device C Compiler ou SDCC) é uma suíte de compiladores padronizada em C, reprogramável e otimizada direcionada para os processadores baseados no MCS-51 da Intel, originalmente desenvolvida pela *Sandeep Dutta*. Dentre os recursos oferecidos pelo

compilador SDCC estão: uma gama completa de tipos de variáveis (*bool*,*int*,*char*,*short*,*long*,*long long* e *float*), um conjunto de otimizações padrão (eliminação de sub-expressões globais, loop invariante, eliminação de código "morto", entre outras) e testes de regressão automatizados.

2.4 Gravador XM85X

O software XM85X é responsável pela gravação do arquivo construído pelo compilador na memória no microcontrolador, para a sua execução no kit. Para tanto, o arquivo com extensão .hex gerado pelo compilador deve ser carregado no gravador,deve-se configurar a porta do computador que será conectada ao kit, estabelecer a conexão entre ambos, pressionar o botão reset para apagar o programa que estiver em execução no dispositivo e então gravar o programa desejado.

2.5 Conversor Digital-Analógico

O conversor digital-analógico presente na placa tem resolução de 8 bits e sua tensão de saída varia de 0 a 5V aproximadamente. Este conversor será utilizado para converter o sinal digital produzido pelo programa em um sinal analógico para ser utilizado no auto-falante.

2.6 Auto-falante

O auto-falante utilizado para reproduzir as músicas de fundo do jogo será aquele presente no próprio kit.

3 Código e Lógica Desenvolvidas

O projeto desenvolvido utiliza os dois temporizadores (*timers*), o módulo de *LCD*, a porta serial, o conversor digital-analógico (DAC) e o auto-falante do kit XM852.

As configurações da porta serial e do *LCD* determinam a lógica do jogo e são atualizadas a todo o instante, enquanto o auto-falante e consequentemente o DAC, necessário para seu funcionamento, atuam apenas nos momentos específicos.

O código está dividido em seis módulos, a saber: main.c, DAC.c, LCD.c, SERIAL.c , TIMER.c e Math.c.

3.1 Configuração do *LCD*

O LCD para ser configurado utiliza alguns endereços para escrever e ler instruções ou dados. O arquivo LCD.h contém todos os endereços e funções necessárias para a configuração do mesmo, como mostrado a seguir:

```
#ifndef LCD_H_INCLUDED
#define LCD_H_INCLUDED

#define clearDisp 0x01  // Limpa o display (slide:10 do 07_LCD.pdf de sistmicro )
#define cursorHomeL1 0x80        // Colocar o cursor no primeiro segmento
#define cursorHomeL2 0xC0        // Colocar o cursor no segundo segmento
#define cursorHomeL3 0x90        // Colocar o cursor no terceiro segmento
#define cursorHomeL4 0xD0        // Colocar o cursor no quarto segmento

#define entryModeShift 0x06 // Habilitar deslocamento para a direita
#define configFunc 0x3F         // Habilitar (0 0 0 0 1 DL N F x x) no qual DL = (1 - 8 bits ou 0 - 4bits)
                                //  N = (1 - 2 linhas ou 0 - 1 linha)
                                    //  F = (1 - 5x10 pontos ou 0 - 5x7 pontos)
#define onoffControl 0x0C         // Habilitar Displays, Cursor e Blink  (0 0 0 0 1 D C B)
#define setCgramAddress 0x40// Habilitar configuração na CGRam

#define dispWriteInst 0xFFC2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, C - A7 e A6,
 ↪  0 - é o endereço CS_DISPLAY)
#define dispWriteData 0xFFD2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, D - A7 e A6,
 ↪  0 - é o endereço CS_DISPLAY)
#define dispReadInst 0xFFE2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, E - A7 e A6, 0
 ↪  - é o endereço CS_DISPLAY)
#define dispReadData 0xFFF2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, F - A7 e A6, 0
 ↪  - é o endereço CS_DISPLAY)
```

```
21
22  #define LEFT 0x01
23  #define CENTER 0x02
24  #define RIGHT 0x03
25
26  static unsigned char __far __at dispWriteInst winstLCD;//  atribui valor ao endereço de instrucao de escrita
27  static unsigned char __far __at dispWriteData wdataLCD;//  atribui valor ao endereço de dados de escrita
28  static unsigned char __far __at dispReadInst rinstLCD;//  atribui valor ao endereço de instrucao de leitura
29  static unsigned char __far __at dispReadData rdataLCD;//  atribui valor ao endereço de dados de leitura
30
31  struct SChar{
32          unsigned char adds[8];
33  };
34
35  struct Map{
36          char schar;
37  };
38
39  static struct SChar point;
40  struct SChar SCmap[8];
41
42  struct Map map[4][16];
43
44  void LCDconfig();
45
46  unsigned char configMap(unsigned char pline, unsigned char pcol);
47
48  void setMap1CGram();
49  void setMap2CGram();
50  void setMap3CGram();
51  void setMap4CGram();
52  void setMap5CGram();
53
54  void LCD_putTextAt(char* text, unsigned char line, unsigned char alignment);
55
56  unsigned char LCD_putText(char* text, unsigned char line, unsigned int time);
57
58  void LCD_putCharAt(char chr, unsigned char line, unsigned char col);
59
60  void LCD_putSCharAt(unsigned char sline, unsigned char scol, unsigned char line, unsigned char col);
61
62  void clearLCD();
63
64  void printMap();
65
66  void printMapAt(unsigned char row, unsigned char col);
67
68  void setCursorHomeAtLine(unsigned char line);
69  #endif
```

Duas *structs* são usadas para configurar o jogo., as *structs* de *SChar* e a *Map*. A *SChar* armazena os oito (8) endereços de carácteres especiais, uma variável de *array* nomeada de SCmap guarda os endereços de todos os caracteres especiais utilizados. A *struct Map* possui um tamanho de 16 por 4 guardando a referência ao carácter especial utilizado na respectiva posição do LCD.

O modulo *LCD.c* contém cinco funções responsáveis por configurar cada um dos níveis, nomeadas como *setMap1CGRAM*, *setMap2CGRAM*, *setMap3CGRAM*, *setMap4CGRAM* e *setMap5CGRAM*. Dentro de cada uma destas funções, são configurados os endereços dos sete caracteres especiais que serão usados para desenhar o labirinto e um oitavo para configurar a posição do personagem.

Em seguida, são enviados os endereços dos caracteres especiais para o LCD para que estes possam ser criados e armazenados na CGRAM. Assim, são configuradas as posições corretas em que tais caracteres devem ser colocados a fim de compor o labirinto. Por fim, o mapa é impresso por meio da função *printMap* e os caracteres especiais são escritos no LCD nas posições definidas. As posições e os caracteres especiais variam de nível para nível, como mostrado o mapa do nível 5 na Figura 2.

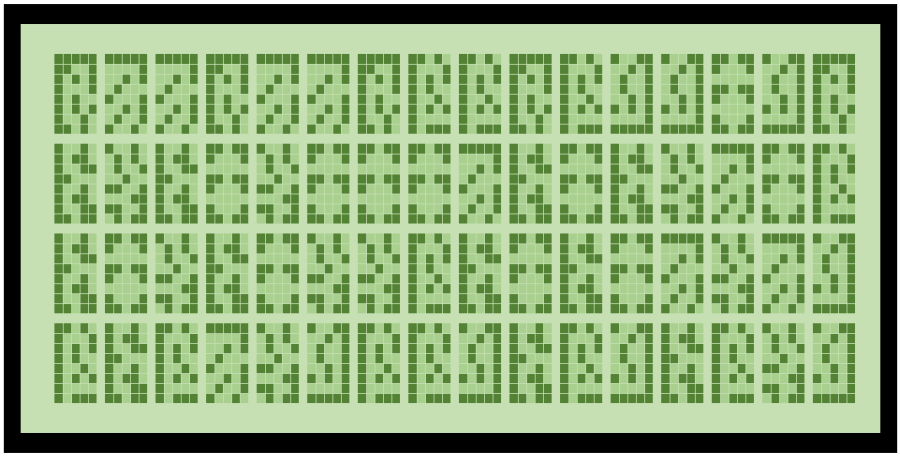


Figura 2: Representação Esquemática do Mapa do Nível 5.

A função *configMap* é a principal função do módulo LCD.c e tem a função de imprimir a nova posição do jogador de acordo com as modificações realizadas pela função *serialControl*. Essa função leva em conta que o único carácter que deve ser atualizado é o que contém o jogador. Ficando piscando pra identificar sua posição, a solução de cada nível deve ser encontrada pelo jogador possibilitando somente saídas para direita, esquerda e acima, como mostrado uma possível solução na Figura 3.

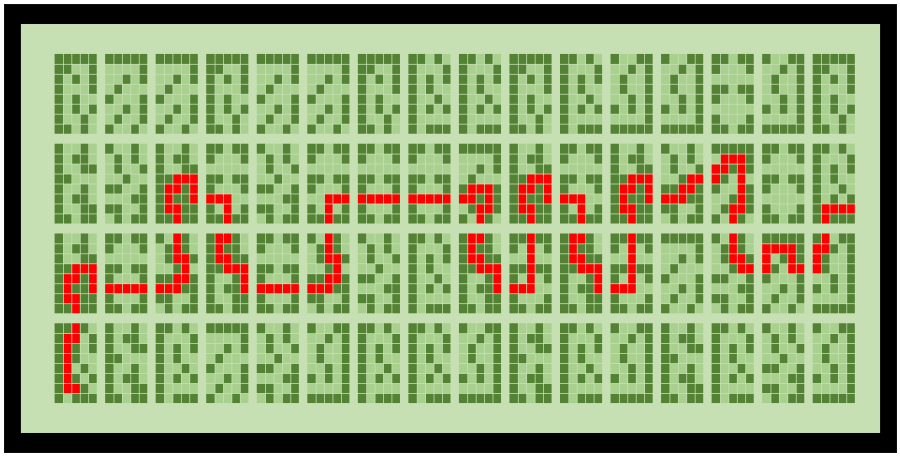


Figura 3: Possível Solução para o Nível 5.

Para controlar o jogador a seguinte referência foi utilizada, como mostra na Figura 4. O jogador possuí uma quarta ordenada (line,col,sline,scol) contendo a posição da Linha do LCD (line), como 1, 2, 3 ou 4, a Coluna do LCD (col), que varia de 1 até 16, a linha dentro do carácter (sline), indo de 1 até 8 e a coluna do caráter (scol), indo de 1 até 5.

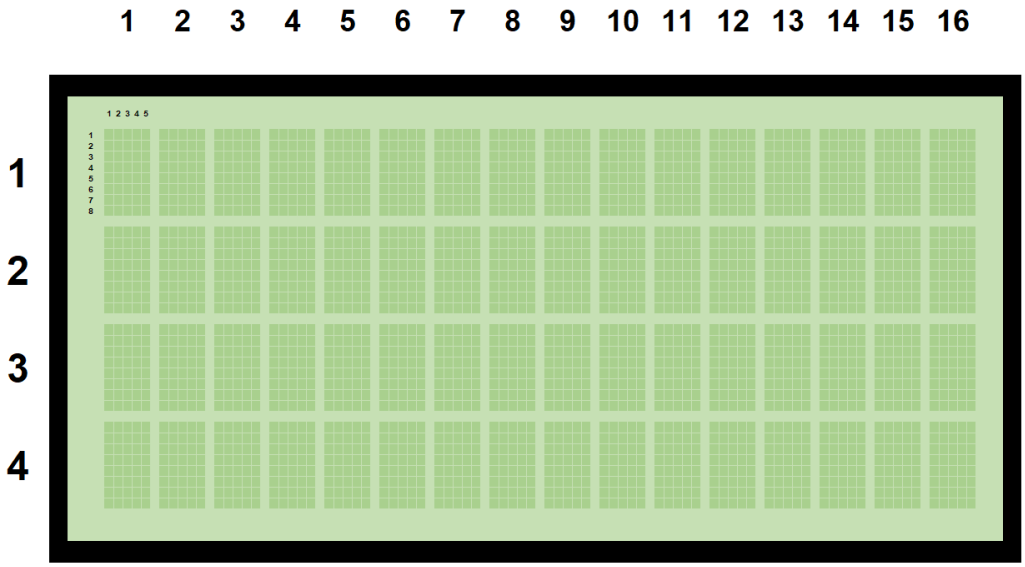


Figura 4: Referência de Posições do Jogador Dentro do LCD.

Algumas outras funções úteis utilizadas para melhor configuração de textos no LCD foram as funções *setCursorAt*, *setChar*, *LCD_putSCharAt*, *LCD_putTextAt* e *LCD_putText*.

A função *setCursorAt* descreve como colocar algum carácter numa posição específica do *display* LCD. A função *setChar* imprime o carácter desejado de acordo com a tabela ASCII. A função *LCD_putSCharAt* faz o controle a carácter do jogador.

A função *LCD_putTextAt* recebe três parâmetros principais, o texto que se deseja escreve, a correspondente linha e se o texto estará alinhado a esquerda, direita ou centralizado, parâmetro LEFT, RIGHT e CENTER. Por fim, a função *LCD_putText* coloca um texto maior que dezesseis (16) caracteres no LCD, fazendo a movimentação dos caracteres para mostrar o texto inteiro disponível.

3.2 Configuração da porta serial

A função principal do módulo SERIAL.c é a função *serialControl*, responsável por modificar ou não a posição do jogador depois de receber um comando do teclado via serial e realizar os testes de colisão do jogador com as paredes do labirinto.

O carácter recebido via serial é armazenado na variável *rxMsg* e então submetido duas condicionais para verificar:

- **A direção do movimento:** Os caracteres 'W', 'A', 'S' e 'D' comandam movimentos respectivamente para cima, para a esquerda, para baixo e para a direita.
- **O próximo movimento:** Se o jogador está em uma linha ou coluna tal que o movimento comandado faria o jogador mudar de quadrado. Exemplo: o usuário quer mover o jogador para a esquerda estando na primeira coluna de um dos quadrados de carácter especial.
- **Saída do Mapa:** Se o jogador está na 5ª coluna da 16ª coluna de caracteres do LCD e quer se movimentar para a direita), então a função retornaria 0, o que significa que o jogador encontrou uma saída do mapa jogado.

Após isso, é realizado o teste de colisão com as paredes do mapa. Realiza-se uma operação *AND* entre o endereço da linha ou coluna do carácter para a qual o jogador irá se movimentar e o endereço correspondente à posição atual do jogador. A variável *logic_op* armazena o resultado desta operação e indica se o jogador pode ou não se movimentar. Esta é uma condição necessária definir se a posição do jogador será ou não alterada posteriormente. Um exemplo é mostrado para a condição de andar para cima:

```
1 logic_op = scmap[(map[player->line-1][player->col-1].schar) - 1].adds[player->sline-2] &
   ↪  (pow(2,5-player->scol));        // Verifica se andar para cima é válido, ou seja, se não possui nenhuma parede:
2
3 // Como a lógica funciona:
4 // Pega o carácter da posição atual do player (scmap[(map[player->line-1][player->col-1].schar) - 1])
5 // Verifica se o endereço da linha de cima (.adds[player->sline-2])
```

```
// Realiza a operação binária AND para ver se o jogador pode subir para a linha de cima.
// Ex: adds[linha 1] = 1 1 1 0 1
//     player pos    = 0 0 0 1 0
// Se o operando logic_op for igual 0 significa que o jogador poderá se movimentar para linha de cima, se for 1, não
↪  poderá.
```

Caso o jogador possa se movimentar, verifica-se novamente se ele está em uma linha ou coluna tal que o movimento comandado faria o jogador mudar de quadrado. Em caso negativo, incrementa-se ou decrementa-se a coluna ou a linha do jogador, a depender do sentido do movimento. Em caso afirmativo, incrementa-se ou decrementa-se a linha ou coluna de caracteres, fazendo o jogador mudar de quadrado e é setado a linha ou coluna do jogador. Por exemplo, se o jogador está na 1ª linha do carácter especial, na 3ª linha do LCD e 'W' é pressionado, ele vai para a 8ª linha carácter especial da 2ª linha do LCD. Em seguida, o mapa é impresso novamente.

No próximo ciclo, a função $LCD\_putSCharAt$ então substitui o caractere especial no qual o jogador está por um novo caractere especial para o qual o ponto correspondente ao jogador estará na nova posição.

3.3 Temporizadores

A temporização do programa foi implementada por meio dos dois *timers* do microcontrolador, o *timer* 0 e *timer* 1.

O *timer* 0 está associado à função *delay* e é utilizado para realizar toda a temporização do programa. A função *delay* recebe como parâmetros a quantidade de tempo e uma *flag* que indica se a unidade é milissegundos ($milisseconds = 1$) ou microssegundos ($milisseconds = 0$).

Quando o tempo recebido está em milissegundos, a função calcula a quantidade de ciclos do *timer* e a quantidade de contagens do último ciclo correspondentes ao tempo passado como parâmetro (aproximando um ciclo a 65 ms) e os armazena nas *flags* *Timer0.cycles* e *Timer0.lastClock* respectivamente, e inicializa o timer 0 em seguida. Como indicado no código a seguir:

```
void delay(unsigned int time, unsigned char miliseconds){

        Timer0.flag=1;

        if(!Timer0.finishMili && miliseconds){
                Timer0.cycles = time/65;
```

```
                Timer0.lastClock = 65535 - ((time % 65)*1000);

                TR0 = 1;
                Timer0.finishMili = 1;

                while(Timer0.finishMili);
        }
        else if(!Timer0.finishMili){
                Timer0.cycles = 1;
                Timer0.lastClock = 65535 - time;
                TR0 = 1;

                while(!Timer0.finishMicro);

                Timer0.finishMicro = 0;
        }
}
```

Quando o tempo recebido está em microssegundos, a função seta o número de ciclos para 1 e calcula o número de contagens até a condição de *overflow*, que ocorre em 65535 *bits*.

Nos dois casos, quando ocorre a interrupção, a função *timer* é responsável por reinicializar o *timer* e implementar a lógica responsável por verificar se o tempo decorrido desde sua inicialização é igual ao tempo recebido como parâmetro pela função *timerDelay*. A lógica da interrupção é dada por:

```
/* FUNÇÃO timer:
*                Função que controla a interrupção do TIMER0
*/
void volatile timer() __interrupt 1{                        //(slide aplicmicro 11_C.pdf pg.13)

        TH0 = 0;                                                                                    //
         ↪  Zerar os bits mais significativos do contador
        TL0 = 0;                                                                                    //
         ↪  Zerar os bits menos significativos do contador
        TF0 = 0;                                                                                   // Zero
         ↪  a flag do contador
        TR0 = 0;                                                                                   // Paro
         ↪  timer0

        if(Timer0.cycles > 1){
        Timer0.cycles--;                                                                  // decrementa cycles
                TR0 = 1;                                                                           //
                 ↪  inicia contador
        }
        else if(Timer0.cycles == 1){
```

```
16                  TH0 = Timer0.lastClock & 0xFF ;                                    // Atribui ao ultimo
                     ↪  timer o valor dos bits mais significativos
17                  TL0 = Timer0.lastClock >> 8;                                     // Atribui ao ultimo timer
                     ↪  o valor dos bits menos significativos
18          Timer0.cycles--;                                                          // decrementa cycles
19
20                  Timer0.finishMicro = 1;
21
22                  TR0 = 1;                                                                    //
                     ↪  inicia contador
23          }
24          else if(Timer0.cycles == 0){
25          Timer0.finishMili = 0;                                                   // finish é uma flag que
             ↪  indica que o tempo acabou
26          }
27  }
```

O *timer* 1 está associado à temporização da porta serial e a *flag* TI é utilizada para evitar a sobreposição no envio de mensagens para o *buffer*.

3.4 DAC e Auto-falante

A biblioteca DAC.h contém uma lista de constantes que representam as notas musicais e cujos valores correspondem à frequência da onda quadrada utilizada para gerá-las através do DAC. A correspondência entre a frequência do sinal colocado no DAC e a frequência sonora medida foi feita por meio de um ajuste de curva experimental que pode ser verificado abaixo na Figura 5. Através de um simples *for* foi realizado a coleta de pontos experimentais. Dado um valor para a variável *delay* era obtida uma correspondente frequência em um aplicativo de celular para afinação de instrumentos musicais. O código utilizado para gerar a onda quadrada é o seguinte:

```
1      unsigned int n, delay = ??;
2
3      dacWrite = 255;
4      for(n = 0;n < delay;n++);
5      dacWrite = 0;
6      for(n = 0;n < delay;n++);
```

A função *for* foi utilizada por apresentar um tempo de ciclo menor se comparado com a função de

delay desenvolvida na Subseção 3.3. Com alguns pontos obtidos, foi possível traçar um ajuste de curva que melhor se adequava aos dados experimentais.

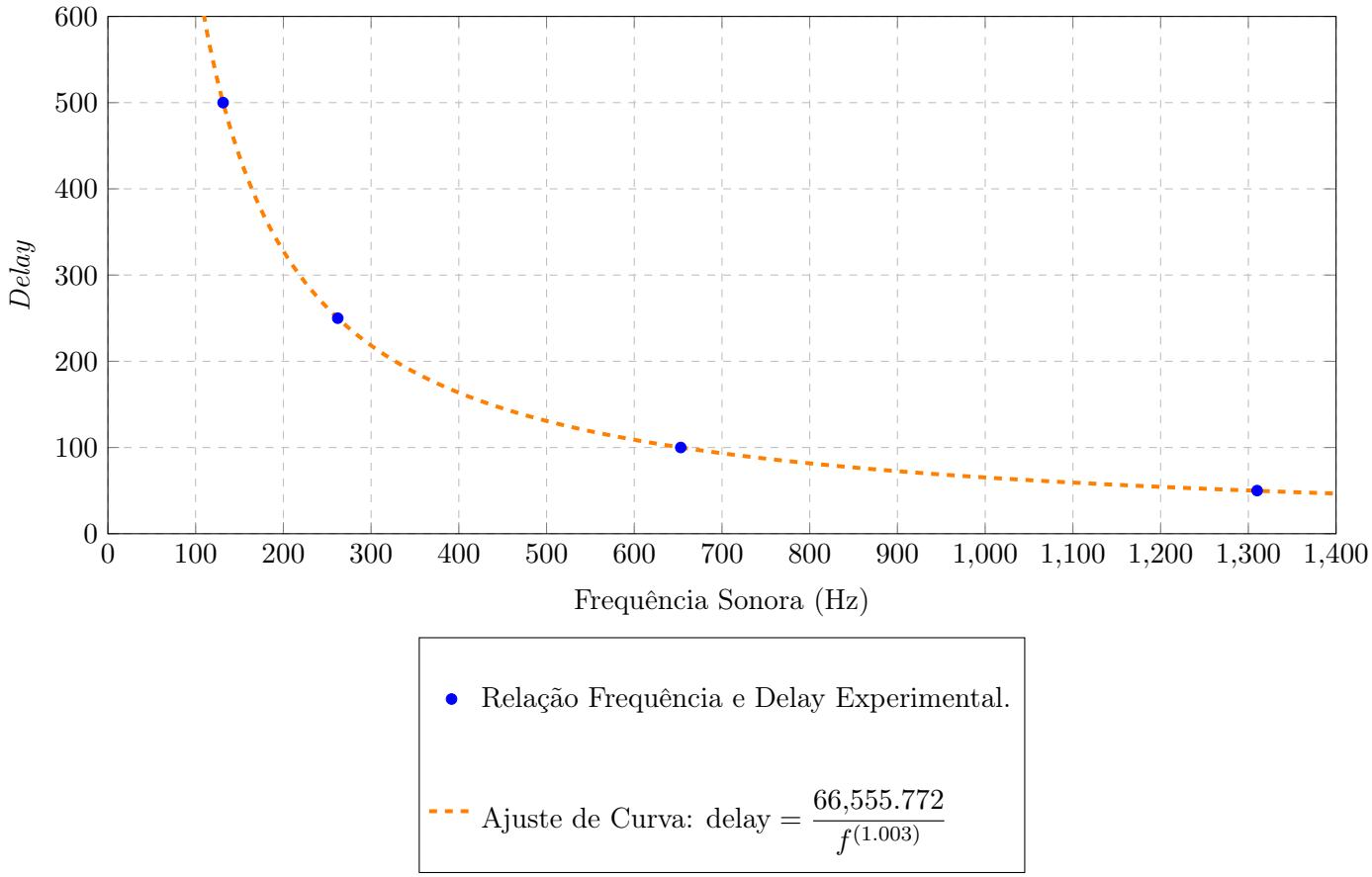
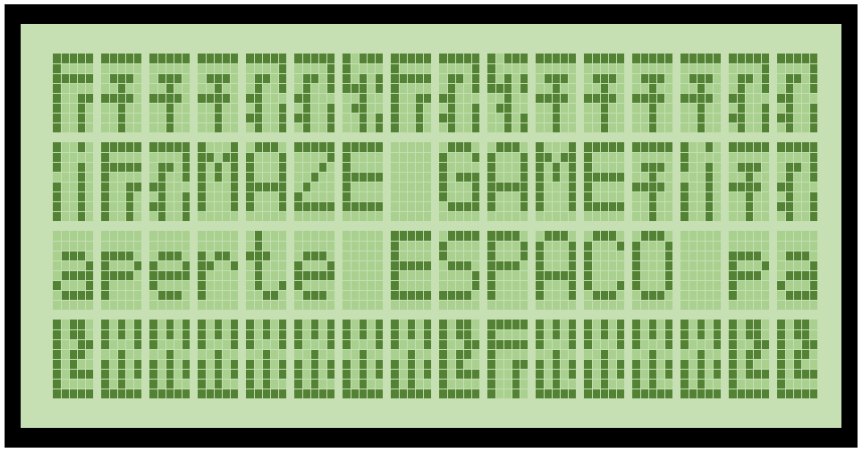


Figura 5: Gráfico Experimental Entre o Número de Ciclos Correspondente a Meio Período da Onda Quadrada Utilizada e a Frequência Sonora Correspondente Produzida no Auto-Falante.

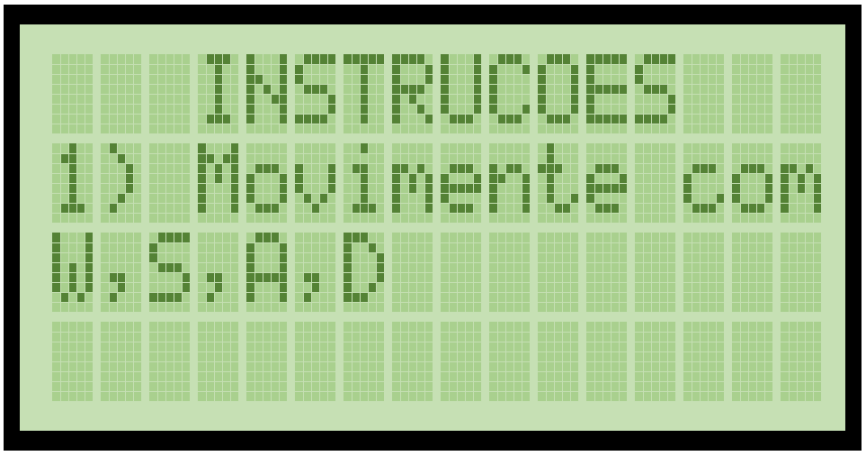
Com esse ajuste boa parte das notas puderam ser alcançadas com a afinação correta, exceto para notas mais agudas, como as sextas, sétimas e oitavas.

Além disso, a biblioteca contém três exemplos de melodias a serem implementadas no auto-falante, de forma que para cada melodia existe um vetor correspondente às cifras e outro correspondente aos tempos da mesma. O código foi baseado de (Usinainfo, 2016).

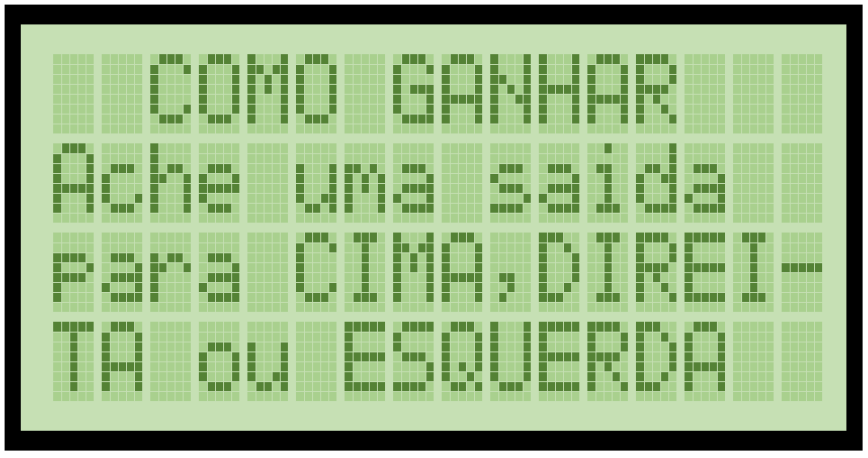
No módulo DAC.c, a função *speaker* recebe a frequência desejada e a duração desejada da nota e cria uma onda quadrada com estas características. A função *squarewave*, por sua vez, controla os tempos em que a nota é tocada e o tempo de pausa da mesma. Por fim, função *sing* recebe o número correspondente à melodia que será tocada e é responsável por chamar a função *squarewave*, passando para a mesma as notas (frequências) e os tempos de cada nota correspondentes à melodia escolhida.



(a)



(b)



(c)

Figura 6: Tela Inicial do Jogo em (a). Em (b) e (c) as Instruções de Jogo.

3.5 Função Main

Na função *main*, inicialmente são configuradas as interrupções, os timers, o LCD e a serial. Em seguida, é chamada uma função que configura a tela inicial, "MAZE GAME" e a instrução "Aperte ESPACO

para iniciar". Após o usuário pressionar espaço, uma outra função configura a tela de instruções de jogo, que é exibida por aproximadamente 3.5 s, sendo mostrado as instruções do jogo, como mostrado na Figura 6 a, b e c.

Após esse período, a função responsável por configurar o primeiro nível é chamada. Dentro dela, primeiramente é exibida a mensagem "NIVEL 1" por 3 s, como na Figura 7.

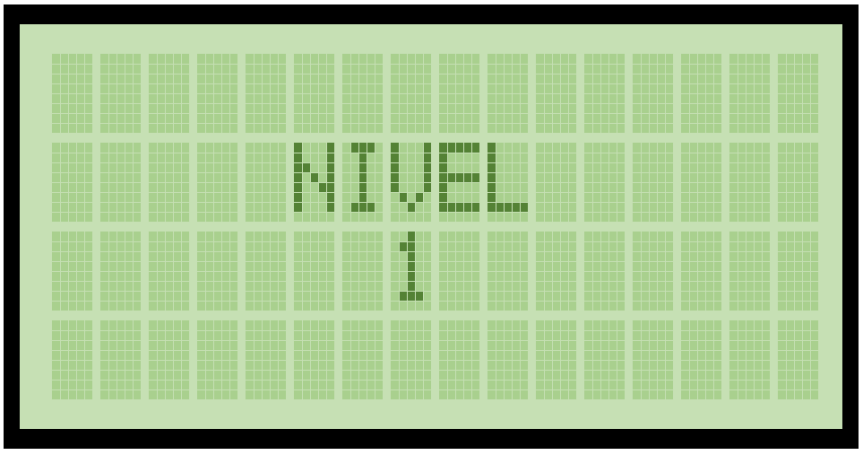


Figura 7: Tela para Nível 1.

Depois, é setada a posição inicial do jogador e impresso o mapa por meio da função *printMap*. Então, é chamada a função *configMap*, que permanece sendo executada até que a variável *control* assuma o valor 0, isto é, até que o jogador encontre a saída do labirinto do primeiro nível. Então, as funções responsáveis por setar os demais níveis são chamadas e podem de forma análoga, até que o último nível seja concluído. Todos os mapas são mostrados na Figura 8.

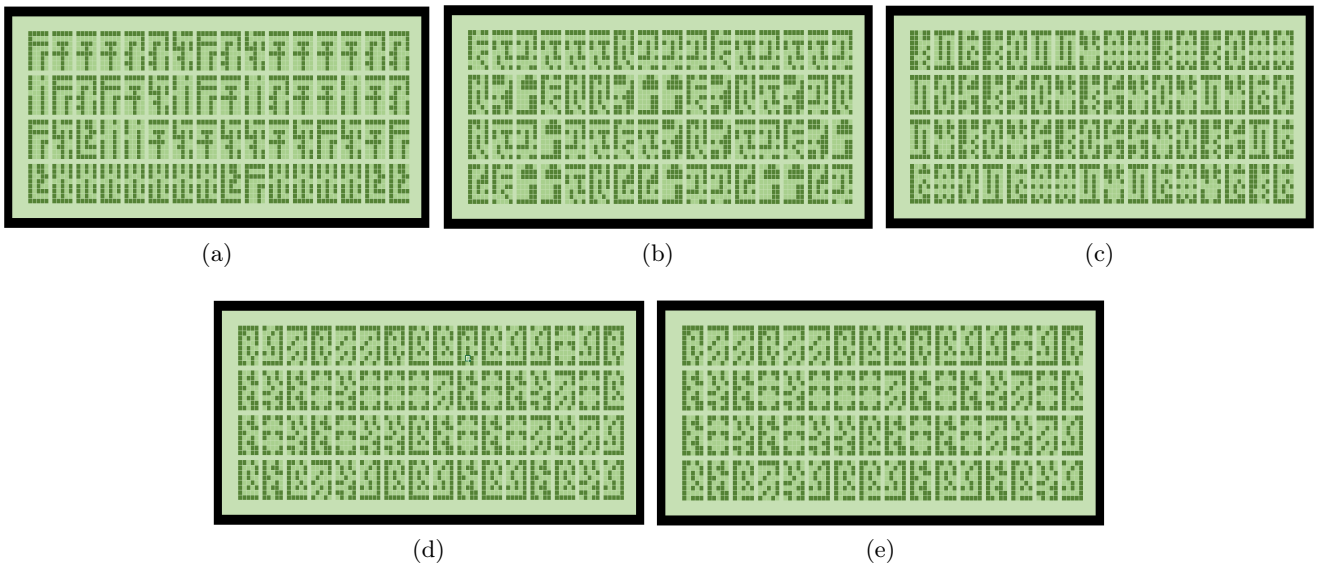


Figura 8: Telas do Mapa no Nível (a) 1, (b) 2, (c) 3, (d) 4 e (e) 5.

Quando isso ocorre, entra a função *setFinish*, que exibe a mensagem "VOCE VENCEU" na tela por 3 s e o programa é finalizado, como na Figura 9

Figura 9: Jogo Concluído.

4 Conclusão

O projeto do jogo em LCD foi concluído para a entrega final, de forma que todas as funções exigidas funcionaram corretamente. Utilizando o *display* de LCD com mais de 10 caracteres especiais, o *Speaker* e a comunicação Serial.

Um dos maiores problemas verificados durante o desenvolvimento do projeto foi achar as frequências corretas para tocar no *Speaker*, sendo necessário coletar dados experimentais para verificar a relação entre o tempo de ciclo do *kit* com a frequência ouvida.

Outra dificuldade foi tornar o código modular, para fazer isso foi necessário compilar o código utilizando o próprio SDCC pelo terminal permitindo distribuir mais memória para as operações desejadas.

Como trabalhos futuros, vê a necessidade da otimização do código para a música tocar em paralelo com o jogo. Como o tempo de ciclo para verificação da comunicação serial demanda bastante processamento deveria se otimizar esse processo afim de tornar o jogo mais rápido.

Referências

Usinainfo. **Tocando o Tema do Super Mário com Buzzer no Arduíno.** 2016. <https://www.usinainfo.com.br/blog/tocando-tema-do-super-mario-com-buzzer-e-arduino/>. Accessed: 2019-08-17.

5 Apêndice A - Códigos

5.1 main.c

```
/*
================================================================================
=====================================PROJETO 2==================================
================================================================================
Nomes:
        Gilmar Correia Jeronimo         - 11014515
        Lucas Barboza Moreira Pinheiro  - 11017015
*/

#include <mcs51/8051.h>
#include <stdio.h>
#include "LCD.h"
#include "TIMER.h"
#include "SERIAL.h"
#include "DAC.h"

volatile struct position *player;

void interruptConfig(){
        IE = 0x82;                                                      // Habilitando interrupções, Serial e timer0 (slide sistmicro
        ↪   03_Interrupções.pdf pg.9)
        IP = 0x08;                                                     // Prioridade de interrupção Serial
}

void configs(){
        interruptConfig();                                          // Habilitando configuracoes
        timerConfig();
        LCDconfig();
        serialConfig(1,player);
}


void setInitialScreen(){

        unsigned char control = 1;

        setMap1CGram();
        LCD_putTextAt("MAZE GAME",2,CENTER);

        while(control)
                control = LCD_putText(" aperte ESPACO para iniciar",3, 250);

        clearLCD();

}

void setInstructions(){

        LCD_putTextAt("INSTRUCOES",1,CENTER);
        LCD_putTextAt("1) Movimente com",2,LEFT);
        LCD_putTextAt("W,S,A,D ",3,CENTER);
        delay(3500,1);
        clearLCD();

        LCD_putTextAt("COMO GANHAR",1,CENTER);
        LCD_putTextAt("Ache uma saida ",2,LEFT);
        LCD_putTextAt("para CIMA,DIREI-",3,LEFT);
        LCD_putTextAt("TA ou ESQUERDA",4,LEFT);
        delay(3500,1);
```

```
59          clearLCD();
60  }
61
62
63
64  void setLevel1(){
65          unsigned char control = 1;
66
67          LCD_putTextAt("NIVEL",2,CENTER);
68          LCD_putTextAt("1",3,CENTER);
69          delay(3000,1);
70          clearLCD();
71
72          player->sline = 7;
73          player->scol = 3;
74          player->line = 4;
75          player->col = 10;
76
77          printMap();
78
79          while(control)
80                  control = configMap(player->line, player->col, 1);
81
82          clearLCD();
83  }
84
85  void setLevel2(){
86          unsigned char control = 1;
87
88          LCD_putTextAt("NIVEL",2,CENTER);
89          LCD_putTextAt("2",3,CENTER);
90          delay(3000,1);
91          clearLCD();
92
93          setMap2CGram();
94
95          player->sline = 7;
96          player->scol = 2;
97          player->line = 4;
98          player->col = 2;
99
100         while(control)
101                 control = configMap(player->line, player->col, 2);
102
103         clearLCD();
104 }
105
106 void setLevel3(){
107         unsigned char control = 1;
108
109         LCD_putTextAt("NIVEL",2,CENTER);
110         LCD_putTextAt("3",3,CENTER);
111         delay(3000,1);
112         clearLCD();
113
114         setMap3CGram();
115
116         player->sline = 5;
117         player->scol = 5;
118         player->line = 4;
119         player->col = 10;
120
121         while(control)
122                 control = configMap(player->line, player->col,3);
123
124         clearLCD();
125 }
```

```
126
127 void setLevel4(){
128         unsigned char control = 1;
129
130         LCD_putTextAt("NIVEL",2,CENTER);
131         LCD_putTextAt("4",3,CENTER);
132         delay(3000,1);
133         clearLCD();
134
135         setMap4CGram();
136
137         player->sline = 3;
138         player->scol = 4;
139         player->line = 4;
140         player->col = 16;
141
142         while(control)
143                 control = configMap(player->line, player->col,2);
144
145         clearLCD();
146 }
147
148 void setLevel5(){
149         unsigned char control = 1;
150
151         LCD_putTextAt("NIVEL",2,CENTER);
152         LCD_putTextAt("5",3,CENTER);
153         delay(3000,1);
154         clearLCD();
155
156         setMap5CGram();
157
158         player->sline = 7;
159         player->scol = 5;
160         player->line = 4;
161         player->col = 1;
162
163         while(control)
164                 control = configMap(player->line, player->col,2);
165
166         clearLCD();
167 }
168 void setFinish(){
169         LCD_putTextAt("VOCE",2,CENTER);
170         LCD_putTextAt("VENCEU",3,CENTER);
171         delay(3000,1);
172 }
173
174 void main(void){
175
176     configs();
177
178         while(1){
179                 setInitialScreen();
180
181                 setInstructions();
182
183                 setLevel1();
184
185                 setLevel2();
186
187                 setLevel3();
188
189                 setLevel4();
190
191                 setLevel5();
192
```

```
193                 setFinish();
194         }
195
196 }
```

5.2 LCD.h

```
1   #ifndef LCD_H_INCLUDED
2   #define LCD_H_INCLUDED
3
4   #define clearDisp 0x01                // Limpa o display (slide:10 do 07_LCD.pdf de sistmicro )
5   #define cursorHomeL1 0x80         // Colocar o cursor no primeiro segmento
6   #define cursorHomeL2 0xC0         // Colocar o cursor no segundo segmento
7   #define cursorHomeL3 0x90         // Colocar o cursor no terceiro segmento
8   #define cursorHomeL4 0xD0         // Colocar o cursor no quarto segmento
9
10  #define entryModeShift 0x06         // Habilitar deslocamento para a direita
11  #define configFunc 0x3F                 // Habilitar (0 0 0 0 1 DL N F x x) no qual DL = (1 - 8 bits ou 0 - 4bits)
12                                                                      //  N = (1 - 2 linhas ou 0 - 1 linha)
13                                                                      //  F = (1 - 5x10 pontos ou 0 - 5x7 pontos)
14  #define onoffControl 0x0C          // Habilitar Displays, Cursor e Blink  (0 0 0 0 1 D C B)
15  #define setCgramAddress 0x40// Habilitar configuração na CGRam
16
17  #define dispWriteInst 0xFFC2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, C - A7 e A6, 0 - é o endereço CS_DISPLAY)
18  #define dispWriteData 0xFFD2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, D - A7 e A6, 0 - é o endereço CS_DISPLAY)
19  #define dispReadInst 0xFFE2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, E - A7 e A6, 0 - é o endereço CS_DISPLAY)
20  #define dispReadData 0xFFF2 // Ativação do decodificador de endereços (F - A15 até A12, F - A11 até A8, F - A7 e A6, 0 - é o endereço CS_DISPLAY)
21
22  #define LEFT 0x01
23  #define CENTER 0x02
24  #define RIGHT 0x03
25
26
27  static unsigned char __far __at dispWriteInst winstLCD;         //  atribui valor ao endereço de instrucao de escrita
28  static unsigned char __far __at dispWriteData wdataLCD;         //  atribui valor ao endereço de dados de escrita
29  static unsigned char __far __at dispReadInst rinstLCD;                  //  atribui valor ao endereço de instrucao de leitura
30  static unsigned char __far __at dispReadData rdataLCD;                  //  atribui valor ao endereço de dados de leitura
31
32  struct SChar{
33          unsigned char adds[8];
34  };
35
36  struct Map{
37          char schar;
38  };
39
40  static struct SChar point;
41  struct SChar SCmap[8];
42
43  struct Map map[4][16];
44
45  void LCDconfig();
46
47  unsigned char configMap(unsigned char pline, unsigned char pcol, unsigned char song);
48
49  void setMap1CGram();
50
51  void setMap2CGram();
52
53  void setMap3CGram();
54
```

```
55  void setMap4CGram();
56
57  void setMap5CGram();
58
59  void LCD_putTextAt(char* text, unsigned char line, unsigned char alignment);
60
61  unsigned char LCD_putText(char* text, unsigned char line, unsigned int time);
62
63  void LCD_putCharAt(char chr, unsigned char line, unsigned char col);
64
65  void LCD_putSCharAt(unsigned char sline, unsigned char scol, unsigned char line, unsigned char col);
66
67  void clearLCD();
68
69  void printMap();
70
71  void printMapAt(unsigned char row, unsigned char col);
72
73  void setCursorHomeAtLine(unsigned char line);
74  #endif
```

5.3 LCD.c

```
1   #include <mcs51/8051.h>
2   #include "LCD.h"
3   #include "TIMER.h"
4   #include "Math.h"
5   #include "DAC.h"
6   #include "SERIAL.h"
7
8   void LCDconfig(){
9       winstLCD = clearDisp;                                   // Atribuindo instrucao
10          delay(10,0);                                                // Delay 10 microsegundos
11          winstLCD = configFunc;                                  // Atribuindo instrucao
12          delay(10,0);
13          winstLCD = entryModeShift;                              // Atribuindo instrucao
14          delay(10,0);
15          winstLCD = onoffControl;                                // Atribuindo instrucao
16          delay(10,0);
17  }
18
19  void setCursorAt(unsigned char line, unsigned char col){
20      if(line == 1)
21          winstLCD = cursorHomeL1 + (col-1);
22      else if(line == 2)
23          winstLCD = cursorHomeL2 + (col-1);
24      else if(line == 3)
25          winstLCD = cursorHomeL3 + (col-1);
26      else if(line == 4)
27          winstLCD = cursorHomeL4 + (col-1);
28
29      delay(10,0);
30  }
31
32  void setChar(char chr){
33      wdataLCD = chr;
34      delay(10,0);
35  }
36
37  void printMapAt(unsigned char row, unsigned char col){
38          setCursorAt(row, col);
```

```
        setChar((map[row-1][col-1].schar) - 1);
}

void printMap(){
        unsigned char row, col;

        for(row = 0; row < 4; row++){
                for(col = 0 ;col<16;col++){
                        setCursorAt(row+1, col+1);
                        setChar((map[row][col].schar) - 1);
                }
        }
}

void setMap1CGram(){
        unsigned char n,m;

        unsigned char c0[] =  {0,0,0,0,0,0,0,0};
        unsigned char c1[] =  {0x1F,0x10,0x1F,0x10,0x13,0x12,0x12,0x12};
        unsigned char c2[] =  {0x1F,0x00,0x0E,0x04,0x1E,0x04,0x04,0x04};
        unsigned char c3[] =  {0x1F,0x01,0x0D,0x09,0x18,0x09,0x19,0x09};
        unsigned char c4[] =  {0x17,0x10,0x15,0x1C,0x04,0x0C,0x05,0x05};
        unsigned char c5[] =  {0x12,0x10,0x12,0x02,0x12,0x12,0x12,0x12};
        unsigned char c6[] =  {0x16,0x16,0x13,0x16,0x14,0x17,0x10,0x1F};
        unsigned char c7[] =  {0x15,0x15,0x11,0x04,0x15,0x15,0x14,0x1F};

        for(n = 0; n < 8 ;n++){
                SCmap[0].adds[n] = c0[n];
                SCmap[1].adds[n] = c1[n];
                SCmap[2].adds[n] = c2[n];
                SCmap[3].adds[n] = c3[n];
                SCmap[4].adds[n] = c4[n];
                SCmap[5].adds[n] = c5[n];
                SCmap[6].adds[n] = c6[n];
                SCmap[7].adds[n] = c7[n];
        }

        winstLCD = setCgramAddress;                                     // Atribuindo primeiro endereço da CGRAM
    delay(10,0);

        for(m = 0;m< 8; m++){
                for(n = 0; n < 8 ;n++){
                        wdataLCD = SCmap[m].adds[n];                // Atribuindo escrita
                        delay(10,0);
                }
        }

        map[0][0].schar = 2;
        map[0][1].schar = 3;
        map[0][2].schar = 3;
        map[0][3].schar = 3;
        map[0][4].schar = 4;
        map[0][5].schar = 4;
        map[0][6].schar = 5;
        map[0][7].schar = 2;
        map[0][8].schar = 4;
        map[0][9].schar = 5;
        map[0][10].schar = 3;
        map[0][11].schar = 3;
        map[0][12].schar = 3;
        map[0][13].schar = 3;
        map[0][14].schar = 4;
        map[0][15].schar = 4;

        map[1][0].schar = 6;
        map[1][1].schar = 2;
        map[1][2].schar = 4;
```

```
106         map[1][3].schar = 2;
107         map[1][4].schar = 3;
108         map[1][5].schar = 5;
109         map[1][6].schar = 6;
110         map[1][7].schar = 2;
111         map[1][8].schar = 3;
112         map[1][9].schar = 6;
113         map[1][10].schar = 4;
114         map[1][11].schar = 3;
115         map[1][12].schar = 3;
116         map[1][13].schar = 6;
117         map[1][14].schar = 3;
118         map[1][15].schar = 4;
119 
120         map[2][0].schar = 2;
121         map[2][1].schar = 5;
122         map[2][2].schar = 7;
123         map[2][3].schar = 6;
124         map[2][4].schar = 6;
125         map[2][5].schar = 3;
126         map[2][6].schar = 5;
127         map[2][7].schar = 3;
128         map[2][8].schar = 5;
129         map[2][9].schar = 5;
130         map[2][10].schar = 3;
131         map[2][11].schar = 5;
132         map[2][12].schar = 2;
133         map[2][13].schar = 5;
134         map[2][14].schar = 3;
135         map[2][15].schar = 2;
136 
137         map[3][0].schar = 7;
138         map[3][1].schar = 8;
139         map[3][2].schar = 8;
140         map[3][3].schar = 8;
141         map[3][4].schar = 8;
142         map[3][5].schar = 8;
143         map[3][6].schar = 8;
144         map[3][7].schar = 8;
145         map[3][8].schar = 7;
146         map[3][9].schar = 2;
147         map[3][10].schar = 8;
148         map[3][11].schar = 8;
149         map[3][12].schar = 8;
150         map[3][13].schar = 8;
151         map[3][14].schar = 7;
152         map[3][15].schar = 7;
153 
154         printMap();
155 
156 }
157 
158 void setMap2CGram(){
159         unsigned char n,m;
160 
161         unsigned char c0[] =  {0,0,0,0,0,0,0,0};
162         unsigned char c1[] =  {0x1B,0x10,0x17,0x10,0x13,0x14,0x02,0x11};
163         unsigned char c2[] =  {0x1F,0x00,0x17,0x12,0x02,0x0B,0x08,0x05};
164         unsigned char c3[] =  {0x1F,0x01,0x15,0x05,0x04,0x1D,0x00,0x13};
165         unsigned char c4[] =  {0x1D,0x15,0x11,0x15,0x15,0x14,0x12,0x19};
166         unsigned char c5[] =  {0x1C,0x1D,0x01,0x03,0x17,0x03,0x1B,0x1A};
167         unsigned char c6[] =  {0x1D,0x11,0x13,0x17,0x14,0x15,0x10,0x1F};
168         unsigned char c7[] =  {0x06,0x0F,0x0F,0x00,0x03,0x03,0x03,0x1F};
169 
170         for(n = 0; n < 8 ;n++){
171                 SCmap[0].adds[n] = c0[n];
172                 SCmap[1].adds[n] = c1[n];
```

```
173             SCmap[2].adds[n] = c2[n];
174             SCmap[3].adds[n] = c3[n];
175             SCmap[4].adds[n] = c4[n];
176             SCmap[5].adds[n] = c5[n];
177             SCmap[6].adds[n] = c6[n];
178             SCmap[7].adds[n] = c7[n];
179         }
180
181         winstLCD = setCgramAddress;                                 // Atribuindo primeiro endereço da CGRAM
182     delay(10,0);
183
184         for(m = 0;m< 8; m++){
185             for(n = 0; n < 8 ;n++){
186                 wdataLCD = SCmap[m].adds[n];                // Atribuindo escrita
187                 delay(10,0);
188             }
189         }
190
191         map[0][0].schar = 2;
192         map[0][1].schar = 3;
193         map[0][2].schar = 4;
194         map[0][3].schar = 3;
195         map[0][4].schar = 3;
196         map[0][5].schar = 3;
197         map[0][6].schar = 5;
198         map[0][7].schar = 4;
199         map[0][8].schar = 4;
200         map[0][9].schar = 4;
201         map[0][10].schar = 2;
202         map[0][11].schar = 3;
203         map[0][12].schar = 4;
204         map[0][13].schar = 3;
205         map[0][14].schar = 3;
206         map[0][15].schar = 4;
207
208         map[1][0].schar = 5;
209         map[1][1].schar = 6;
210         map[1][2].schar = 8;
211         map[1][3].schar = 2;
212         map[1][4].schar = 5;
213         map[1][5].schar = 5;
214         map[1][6].schar = 6;
215         map[1][7].schar = 8;
216         map[1][8].schar = 8;
217         map[1][9].schar = 2;
218         map[1][10].schar = 6;
219         map[1][11].schar = 5;
220         map[1][12].schar = 3;
221         map[1][13].schar = 6;
222         map[1][14].schar = 4;
223         map[1][15].schar = 5;
224
225         map[2][0].schar = 5;
226         map[2][1].schar = 3;
227         map[2][2].schar = 4;
228         map[2][3].schar = 8;
229         map[2][4].schar = 4;
230         map[2][5].schar = 3;
231         map[2][6].schar = 2;
232         map[2][7].schar = 3;
233         map[2][8].schar = 6;
234         map[2][9].schar = 5;
235         map[2][10].schar = 6;
236         map[2][11].schar = 3;
237         map[2][12].schar = 4;
238         map[2][13].schar = 2;
239         map[2][14].schar = 6;
```

```
240         map[2][15].schar = 8;
241
242         map[3][0].schar = 7;
243         map[3][1].schar = 2;
244         map[3][2].schar = 8;
245         map[3][3].schar = 8;
246         map[3][4].schar = 4;
247         map[3][5].schar = 6;
248         map[3][6].schar = 7;
249         map[3][7].schar = 7;
250         map[3][8].schar = 8;
251         map[3][9].schar = 7;
252         map[3][10].schar = 4;
253         map[3][11].schar = 7;
254         map[3][12].schar = 8;
255         map[3][13].schar = 8;
256         map[3][14].schar = 7;
257         map[3][15].schar = 4;
258
259         printMap();
260
261 }
262
263 void setMap3CGram(){
264         unsigned char n,m;
265
266         unsigned char c0[] =  {0,0,0,0,0,0,0,0};
267         unsigned char c1[] =  {0x1B,0x1B,0x18,0x19,0x1A,0x11,0x1A,0x19};
268         unsigned char c2[] =  {0x1F,0x08,0x0A,0x0A,0x0A,0x0A,0x02,0x1F};
269         unsigned char c3[] =  {0x1B,0x11,0x15,0x05,0x14,0x15,0x11,0x1B};
270         unsigned char c4[] =  {0x19,0x13,0x17,0x10,0x03,0x1B,0x19,0x13};
271         unsigned char c5[] =  {0x12,0x1B,0x1A,0x09,0x00,0x11,0x11,0x1B};
272         unsigned char c6[] =  {0x12,0x17,0x10,0x12,0x15,0x14,0x13,0x1F};
273         unsigned char c7[] =  {0x1B,0x11,0x15,0x00,0x15,0x00,0x15,0x1B};
274
275
276         for(n = 0; n < 8 ;n++){
277                 SCmap[0].adds[n] = c0[n];
278                 SCmap[1].adds[n] = c1[n];
279                 SCmap[2].adds[n] = c2[n];
280                 SCmap[3].adds[n] = c3[n];
281                 SCmap[4].adds[n] = c4[n];
282                 SCmap[5].adds[n] = c5[n];
283                 SCmap[6].adds[n] = c6[n];
284                 SCmap[7].adds[n] = c7[n];
285         }
286
287         winstLCD = setCgramAddress;                                     // Atribuindo primeiro endereço da CGRAM
288     delay(10,0);
289
290         for(m = 0;m< 8; m++){
291                 for(n = 0; n < 8 ;n++){
292                         wdataLCD = SCmap[m].adds[n];                // Atribuindo escrita
293                         delay(10,0);
294                 }
295         }
296
297         map[0][0].schar = 2;
298         map[0][1].schar = 3;
299         map[0][2].schar = 7;
300         map[0][3].schar = 2;
301         map[0][4].schar = 4;
302         map[0][5].schar = 3;
303         map[0][6].schar = 3;
304         map[0][7].schar = 6;
305         map[0][8].schar = 8;
306         map[0][9].schar = 8;
```

```
307         map[0][10].schar = 2;
308         map[0][11].schar = 8;
309         map[0][12].schar = 2;
310         map[0][13].schar = 4;
311         map[0][14].schar = 8;
312         map[0][15].schar = 8;
313 
314         map[1][0].schar = 3;
315         map[1][1].schar = 4;
316         map[1][2].schar = 5;
317         map[1][3].schar = 2;
318         map[1][4].schar = 5;
319         map[1][5].schar = 4;
320         map[1][6].schar = 6;
321         map[1][7].schar = 2;
322         map[1][8].schar = 5;
323         map[1][9].schar = 6;
324         map[1][10].schar = 4;
325         map[1][11].schar = 6;
326         map[1][12].schar = 3;
327         map[1][13].schar = 6;
328         map[1][14].schar = 7;
329         map[1][15].schar = 4;
330 
331         map[2][0].schar = 3;
332         map[2][1].schar = 6;
333         map[2][2].schar = 2;
334         map[2][3].schar = 4;
335         map[2][4].schar = 6;
336         map[2][5].schar = 5;
337         map[2][6].schar = 2;
338         map[2][7].schar = 4;
339         map[2][8].schar = 5;
340         map[2][9].schar = 5;
341         map[2][10].schar = 6;
342         map[2][11].schar = 4;
343         map[2][12].schar = 2;
344         map[2][13].schar = 5;
345         map[2][14].schar = 3;
346         map[2][15].schar = 7;
347 
348         map[3][0].schar = 7;
349         map[3][1].schar = 8;
350         map[3][2].schar = 4;
351         map[3][3].schar = 3;
352         map[3][4].schar = 7;
353         map[3][5].schar = 8;
354         map[3][6].schar = 8;
355         map[3][7].schar = 3;
356         map[3][8].schar = 6;
357         map[3][9].schar = 3;
358         map[3][10].schar = 7;
359         map[3][11].schar = 8;
360         map[3][12].schar = 6;
361         map[3][13].schar = 7;
362         map[3][14].schar = 2;
363         map[3][15].schar = 7;
364 
365         printMap();
366 
367 }
368 
369 void setMap4CGram(){
370         unsigned char n,m;
371 
372         unsigned char c0[] =  {0,0,0,0,0,0,0,0};
373         unsigned char c1[] =  {0x1F,0x19,0x15,0x11,0x14,0x15,0x12,0x1A};
```

```
374         unsigned char c2[] =  {0x1F,0x01,0x05,0x08,0x11,0x05,0x09,0x12};
375         unsigned char c3[] =  {0x1B,0x11,0x00,0x1B,0x11,0x00,0x11,0x1B};
376         unsigned char c4[] =  {0x12,0x0A,0x09,0x04,0x19,0x03,0x19,0x0B};
377         unsigned char c5[] =  {0x12,0x16,0x13,0x18,0x12,0x16,0x13,0x1B};
378         unsigned char c6[] =  {0x1A,0x11,0x15,0x14,0x12,0x15,0x10,0x1F};
379         unsigned char c7[] =  {0x13,0x05,0x09,0x09,0x05,0x15,0x01,0x1F};
380 
381         for(n = 0; n < 8 ;n++){
382                 SCmap[0].adds[n] = c0[n];
383                 SCmap[1].adds[n] = c1[n];
384                 SCmap[2].adds[n] = c2[n];
385                 SCmap[3].adds[n] = c3[n];
386                 SCmap[4].adds[n] = c4[n];
387                 SCmap[5].adds[n] = c5[n];
388                 SCmap[6].adds[n] = c6[n];
389                 SCmap[7].adds[n] = c7[n];
390         }
391 
392         winstLCD = setCgramAddress;                                     // Atribuindo primeiro endereço da CGRAM
393     delay(10,0);
394 
395         for(m = 0;m< 8; m++){
396                 for(n = 0; n < 8 ;n++){
397                         wdataLCD = SCmap[m].adds[n];                // Atribuindo escrita
398                         delay(10,0);
399                 }
400         }
401 
402         map[0][0].schar = 2;
403         map[0][1].schar = 8;
404         map[0][2].schar = 3;
405         map[0][3].schar = 2;
406         map[0][4].schar = 3;
407         map[0][5].schar = 3;
408         map[0][6].schar = 2;
409         map[0][7].schar = 7;
410         map[0][8].schar = 7;
411         map[0][9].schar = 2;
412         map[0][10].schar = 7;
413         map[0][11].schar = 8;
414         map[0][12].schar = 8;
415         map[0][13].schar = 4;
416         map[0][14].schar = 8;
417         map[0][15].schar = 2;
418 
419         map[1][0].schar = 6;
420         map[1][1].schar = 5;
421         map[1][2].schar = 6;
422         map[1][3].schar = 4;
423         map[1][4].schar = 5;
424         map[1][5].schar = 4;
425         map[1][6].schar = 4;
426         map[1][7].schar = 4;
427         map[1][8].schar = 3;
428         map[1][9].schar = 6;
429         map[1][10].schar = 4;
430         map[1][11].schar = 6;
431         map[1][12].schar = 5;
432         map[1][13].schar = 3;
433         map[1][14].schar = 4;
434         map[1][15].schar = 7;
435 
436         map[2][0].schar = 6;
437         map[2][1].schar = 4;
438         map[2][2].schar = 5;
439         map[2][3].schar = 6;
440         map[2][4].schar = 4;
```

```
        map[2][5].schar = 5;
        map[2][6].schar = 5;
        map[2][7].schar = 7;
        map[2][8].schar = 6;
        map[2][9].schar = 4;
        map[2][10].schar = 6;
        map[2][11].schar = 4;
        map[2][12].schar = 3;
        map[2][13].schar = 5;
        map[2][14].schar = 3;
        map[2][15].schar = 8;

        map[3][0].schar = 7;
        map[3][1].schar = 6;
        map[3][2].schar = 7;
        map[3][3].schar = 3;
        map[3][4].schar = 5;
        map[3][5].schar = 8;
        map[3][6].schar = 7;
        map[3][7].schar = 7;
        map[3][8].schar = 8;
        map[3][9].schar = 6;
        map[3][10].schar = 7;
        map[3][11].schar = 8;
        map[3][12].schar = 6;
        map[3][13].schar = 7;
        map[3][14].schar = 5;
        map[3][15].schar = 8;

        printMap();

}

void setMap5CGram(){
        unsigned char n,m;

        unsigned char c0[] =  {0,0,0,0,0,0,0,0};
        unsigned char c1[] =  {0x1F,0x19,0x15,0x11,0x14,0x15,0x12,0x1A};
        unsigned char c2[] =  {0x1F,0x01,0x05,0x08,0x11,0x05,0x09,0x12};
        unsigned char c3[] =  {0x1B,0x11,0x00,0x1B,0x11,0x00,0x11,0x1B};
        unsigned char c4[] =  {0x12,0x0A,0x09,0x04,0x19,0x03,0x19,0x0B};
        unsigned char c5[] =  {0x12,0x16,0x13,0x18,0x12,0x16,0x13,0x1B};
        unsigned char c6[] =  {0x1A,0x11,0x15,0x14,0x12,0x15,0x10,0x17};
        unsigned char c7[] =  {0x13,0x05,0x09,0x09,0x05,0x15,0x01,0x1F};

        for(n = 0; n < 8 ;n++){
                SCmap[0].adds[n] = c0[n];
                SCmap[1].adds[n] = c1[n];
                SCmap[2].adds[n] = c2[n];
                SCmap[3].adds[n] = c3[n];
                SCmap[4].adds[n] = c4[n];
                SCmap[5].adds[n] = c5[n];
                SCmap[6].adds[n] = c6[n];
                SCmap[7].adds[n] = c7[n];
        }

        winstLCD = setCgramAddress;                                     // Atribuindo primeiro endereço da CGRAM
    delay(10,0);

        for(m = 0;m< 8; m++){
                for(n = 0; n < 8 ;n++){
                        wdataLCD = SCmap[m].adds[n];                // Atribuindo escrita
                        delay(10,0);
                }
        }

        map[0][0].schar = 2;
```

```
508         map[0][1].schar = 3;
509         map[0][2].schar = 3;
510         map[0][3].schar = 2;
511         map[0][4].schar = 3;
512         map[0][5].schar = 3;
513         map[0][6].schar = 2;
514         map[0][7].schar = 7;
515         map[0][8].schar = 7;
516         map[0][9].schar = 2;
517         map[0][10].schar = 7;
518         map[0][11].schar = 8;
519         map[0][12].schar = 8;
520         map[0][13].schar = 4;
521         map[0][14].schar = 8;
522         map[0][15].schar = 2;
523 
524         map[1][0].schar = 6;
525         map[1][1].schar = 5;
526         map[1][2].schar = 6;
527         map[1][3].schar = 4;
528         map[1][4].schar = 5;
529         map[1][5].schar = 4;
530         map[1][6].schar = 4;
531         map[1][7].schar = 4;
532         map[1][8].schar = 3;
533         map[1][9].schar = 6;
534         map[1][10].schar = 4;
535         map[1][11].schar = 6;
536         map[1][12].schar = 5;
537         map[1][13].schar = 3;
538         map[1][14].schar = 4;
539         map[1][15].schar = 7;
540 
541         map[2][0].schar = 6;
542         map[2][1].schar = 4;
543         map[2][2].schar = 5;
544         map[2][3].schar = 6;
545         map[2][4].schar = 4;
546         map[2][5].schar = 5;
547         map[2][6].schar = 5;
548         map[2][7].schar = 7;
549         map[2][8].schar = 6;
550         map[2][9].schar = 4;
551         map[2][10].schar = 6;
552         map[2][11].schar = 4;
553         map[2][12].schar = 3;
554         map[2][13].schar = 5;
555         map[2][14].schar = 3;
556         map[2][15].schar = 8;
557 
558         map[3][0].schar = 7;
559         map[3][1].schar = 6;
560         map[3][2].schar = 7;
561         map[3][3].schar = 3;
562         map[3][4].schar = 5;
563         map[3][5].schar = 8;
564         map[3][6].schar = 7;
565         map[3][7].schar = 7;
566         map[3][8].schar = 8;
567         map[3][9].schar = 6;
568         map[3][10].schar = 7;
569         map[3][11].schar = 8;
570         map[3][12].schar = 6;
571         map[3][13].schar = 7;
572         map[3][14].schar = 5;
573         map[3][15].schar = 8;
574 
```

```
575         printMap();
576 
577 }
578 
579 unsigned char configMap(unsigned char pline, unsigned char pcol, unsigned char song){
580 
581         unsigned char control = serialControl(SCmap, map);
582         delay(100,0);
583         //sing(3);
584         setCursorAt(pline, pcol);
585         setChar((map[pline-1][pcol-1].schar) - 1);
586         sing(song);
587         //delay(100,0);
588 
589         return control;
590 }
591 
592 
593 
594 void setCursorHomeAtLine(unsigned char line){
595     if(line == 1)
596         winstLCD = cursorHomeL1;
597     else if(line == 2)
598         winstLCD = cursorHomeL2;
599     else if(line == 3)
600         winstLCD = cursorHomeL3;
601     else if(line == 4)
602         winstLCD = cursorHomeL4;
603 
604     delay(10,0);
605 }
606 
607 
608 
609 void setPlayerCursor(unsigned char pline, unsigned char pcol, unsigned char sline, unsigned char scol){
610 
611         unsigned char n;
612 
613         for(n = 0; n < 8 ;n++)
614                 point.adds[n]= SCmap[map[pline-1][pcol-1].schar-1].adds[n];
615 
616         point.adds[sline-1] += pow(2,5-scol);
617 
618         winstLCD = 0x40;                                    // Atribuindo primeiro endereço da CGRAM
619     delay(10,0);
620 
621 
622         for(n = 0; n < 8 ;n++){
623                 wdataLCD = point.adds[n];               // Atribuindo escrita
624         delay(10,0);
625     }
626 }
627 
628 void clearLCD(){
629         winstLCD = clearDisp;                                       // Atribuindo instrucao
630         delay(10,0);                                                       // Delay 10 microsegundos
631         winstLCD = configFunc;                                      // Atribuindo instrucao
632         delay(10,0);
633         winstLCD = entryModeShift;                                 // Atribuindo instrucao
634         delay(10,0);
635         winstLCD = onoffControl;                                   // Atribuindo instrucao
636         delay(10,0);
637 }
638 
639 void LCD_putTextAt(char* text, unsigned char line, unsigned char alignment){
640 
641         unsigned char n, col = 0, size;
```

```
642         char txt[17] = {0};
643 
644         for(n = 0;text[n] != '\0';n++)
645         txt[n] = text[n];
646 
647         size = n;
648 
649         if (alignment == LEFT)
650                 col = 1;
651         else if(alignment == CENTER)
652                 col = (16-size)/2 + 1;
653         else if(alignment == RIGHT)
654                 col = (16-size);
655 
656         setCursorAt(line, col);
657 
658         for(n = 0; n<size; n++)
659                 setChar(txt[n]);
660 
661 }
662 
663 unsigned char LCD_putText(char* text, unsigned char line, unsigned int time){
664 
665         unsigned char i = 0, j = 0, k =0,size, control = 1;
666     char chr;
667         char txt[50] = {0};
668 
669     for(i = 0;text[i] != '\0';i++)
670         txt[i] = text[i];
671 
672         txt[i] = text[i];
673     size = i;
674 
675     for(i=0;i < size && control ==1 ;i++){
676 
677                 setCursorHomeAtLine(line);
678 
679                 for(j = 0; j<16 && control ==1;j++)
680                 setChar(txt[j]);                                // Atribuindo escrita
681 
682 
683                 sing(1);
684                 delay(time-100,0);
685 
686         chr = txt[0];
687 
688         for(k = 1; k< size && control ==1;k++)
689                         txt[k-1] = txt[k];
690 
691                 txt[size-1] = chr;
692 
693                 control = serialBegin();
694 
695                 if(control == 0)
696                         break;
697     }
698 
699         return control;
700 }
701 
702 void LCD_putCharAt(char chr, unsigned char line, unsigned char col){
703         setCursorAt(line, col);
704         setChar(chr);
705 }
706 
707 void LCD_putSCharAt(unsigned char sline, unsigned char scol, unsigned char line, unsigned char col){
708         setPlayerCursor(line,col,sline,scol);
```

```
709         setCursorAt(line, col);
710         setChar(0);
711 }
```

5.4 SERIAL.h

```
1  #ifndef SERIAL_H_INCLUDED
2  #define SERIAL_H_INCLUDED
3
4  static char rxMsg = 0;
5
6  struct position{
7          unsigned char sline;
8          unsigned char scol;
9          unsigned char line;
10         unsigned char col;
11 };
12
13 void transmitter(unsigned char message);
14
15 char receiver();
16
17 void serialConfig(unsigned char read,struct position *gamer);
18
19 unsigned char serialBegin();
20
21 unsigned char serialControl(struct SChar scmap[8], struct Map map[4][16]);
22 #endif
```

5.5 SERIAL.c

```
1  #include "SERIAL.h"
2
3  #include <mcs51/8051.h>
4
5  #include "LCD.h"
6  #include "TIMER.h"
7  #include "DAC.h"
8  #include "Math.h"
9
10 struct position *player;
11
12 /* FUNÇÃO transmitter:
13 *                 Transmite um char para a serial
14 */
15 void transmitter(unsigned char message){
16         if(! TI){
17                 SBUF = message;                                 // Manda a mensagem para o buffer
18
19                 while(TI == 0);                                 // Esperando a flag TI parar a transmissão
20                         TI = 0;                                         // Zerando a flag
21         }
22 }
23
```

```
/* FUNÇÃO transNumber:
*                Converte um número para char e transmite pela serial
*/
void transNumber(unsigned char number){
        transmitter(number/10 + '0');
        transmitter(number%10 + '0');
        transmitter('\n');
}

/* FUNÇÃO receiver:
*                recebe um char da serial
*/
char receiver(){

        return SBUF;                                            // Recebe a mensagem do buffer
}


/* FUNÇÃO receivedChar:
*                retorna o último char recebido pela Serial
*/
char receivedChar() {

        return rxMsg;
}

/* FUNÇÃO serialConfig:
*                Habilita a comunicação serial do projeto recebendo como parâmetro um variável que indicará se a serial é de Leitura ou de Escrita, através da variável read. Além de receber um struct position que comutará a posição do jogador durante o jogo
*/
void serialConfig(unsigned char read, struct position *gamer){
        player = gamer;

        if(read)
                SCON = 0x50;                                            // Configurando o modo 1 para serial (05 Serial.pdf de sistmicro)
        else
                SCON = 0x40;
}


/* FUNÇÃO serialBegin:
*                Inicializa a recepção de mensagens através da serial, salvando o caractér no rxMsg. Essa função em específico só função para a tela de inicialização - Se apertar ' ' (ESPAÇO), o jogo se inicia;
*/
unsigned char serialBegin(){

        if(RI){
                rxMsg = receiver();

                if(rxMsg == ' ')
                        return 0;
        }

        return 1;
}


/* FUNÇÃO serialControl:
*                Controla o caractér do jogador alterando a sua posição
*/

unsigned char serialControl(struct SChar scmap[8],struct Map map[4][16]){

        unsigned char logic_op = 1;

        LCD_putSCharAt(player->sline,player->scol, player->line,player->col);                           // Coloca o caractér do jogador na posição correta
```

```
88
89      if(RI){                             //Se receber uma mensagem
90
91              printMapAt(player->line,player->col);
92
93              rxMsg = receiver();                                     //Salva o char recebido
94
95              if(rxMsg == 'w' || rxMsg ==
                 ↪    'W'){
                 ↪    Se a mensagem for W
96
97                      // ========================================== Lógica da Colisão para Cima ==========================================
98                      // Verifica se o player pode continuar andando para Cima
99                      if((player->sline - 1) >0) // Se o jogador não estiver na primeira linha de cada quadrado do LCD
100                             logic_op = scmap[(map[player->line-1][player->col-1].schar) - 1].adds[player->sline-2] & (pow(2,5-player->scol));        //
                                 ↪    Verifica se andar para cima é válido, ou seja, se não possui nenhuma parede:
101                             // Como a lógica funciona:
102                             //                 Pega o caractér da posição atual do player (scmap[(map[player->line-1][player->col-1].schar) - 1])
103                             //                Verifica se o endereço da linha de cima (.adds[player->sline-2])
104                             //                 Realiza a operação binária & para ver se o jogador pode subir para a linha de cima.
105                             //                        Ex:               adds[linha 1] = 1 1 1 0 1
106                             //                                        player pos          = 0 0 0 1 0
107                             // Se o operando logic_op for igual 0 significa que o jogador poderá se movimentar para linha de cima, se for 1, não poderá.
108
109                     else if(player->sline == 1 && player->line > 1)// Se o jogador estiver na primeira linha de cada quadrado do LCD
110                             logic_op = scmap[(map[player->line-2][player->col-1].schar) - 1].adds[7] & (pow(2,5-player->scol)); // Verifica se a linha de
                                 ↪    quadrados do LCD uma posição acima permite transição ou se possuí parede.
111                     else if(player->sline == 1 && player->line == 1 )// Se o jogador estiver na última linha do LCD
112                             return 0; // O jogador achou uma saída do mapa
113                     // ========================================== Lógica da Colisão para Cima ==========================================
114
115                     if(logic_op == 0){                                      // Se o jogador puder se movimentar
116
117                             if(player->sline != 1)
118                                     player->sline--;                        //decrementa a posição no mapa
119                             else{
120                                     if(player->line != 1){
121                                             player->sline = 8;              //coloca no último segmento do quadrado do LCD na linha acima
122                                             player->line--;                         //decrementa a posição no mapa
123                                     }
124                             }
125                     }
126
127             }
128             else if(rxMsg == 's' || rxMsg == 'S'){
129
130                     // ========================================== Lógica da Colisão para Baixo ==========================================
131                     // Verifica se o player pode continuar andando para Baixo
132                     if((player->sline-1) < 7) // Se o jogador não estiver na última linha de cada quadrado do LCD
133                             logic_op = scmap[(map[player->line-1][player->col-1].schar) - 1].adds[player->sline] & (pow(2,5-player->scol)); // Verifica se
                                 ↪    andar para cima é válido.
134                     else // Se o jogador estiver na última linha de cada quadrado do LCD
135                             logic_op = scmap[(map[player->line][player->col-1].schar) - 1].adds[0] & (pow(2,5-player->scol)); // Verifica se a linha de
                                 ↪    quadrados do LCD uma posição abaixo permite transição ou se possuí parede.
136                     // ========================================== Lógica da Colisão para Baixo ==========================================
137
138                     if(logic_op == 0){                                      // Se o jogador puder se movimentar
139
140                             if(player->sline != 8)
141                                     player->sline++;                        //incrementa a posição no mapa
142                             else{
143                                     if(player->line != 4){
144                                             player->sline = 1;              //coloca no primeiro segmento do quadrado do LCD na linha abaixo
145                                             player->line++;                         //incrementa a posição no mapa
146                                     }
147                             }
148                     }
```

```
149
150             }
151             else if(rxMsg == 'd' || rxMsg == 'D'){
152
153                     // ========================================== Lógica da Colisão para a Direita ==========================================
154                     // Verifica se o player pode continuar andando para Direita
155                     if((player->scol+1) < 6) // Se o jogador não estiver na última coluna de cada quadrado do LCD
156                             logic_op = scmap[(map[player->line-1][player->col-1].schar) - 1].adds[player->sline-1] & (pow(2,5-(player->scol+1))); //
                                ↪    Verifica se andar para direita é válido.
157                     else if(player->scol == 5 && player->col < 16)         // Se o jogador estiver na última coluna de cada quadrado do LCD
158                             logic_op = scmap[(map[player->line-1][player->col].schar) - 1].adds[player->sline-1] & (pow(2,5-(1)));        // Verifica se a
                                ↪    coluna de quadrados do LCD uma posição para direita permite transição ou se possuí parede.
159                     else if(player->scol == 5 && player->col == 16) // Se o jogador estiver na última coluna do LCD
160                             return 0;        // O jogador achou uma saída do mapa
161                     // ========================================== Lógica da Colisão para a Direita ==========================================
162
163                     if(logic_op == 0){                                              // Se o jogador puder se movimentar
164                             if(player->scol != 5)
165                                     player->scol++;                                         //incrementa a posição no mapa
166                             else{
167                                     if(player->col != 16){
168                                             player->scol = 1;                       //coloca no primeiro segmento de coluna do quadrado do LCD na coluna
                                                ↪    à direita
169                                             player->col++;                                  //incrementa a posição no mapa
170                                     }
171                             }
172                     }
173             }
174             else if(rxMsg == 'a' || rxMsg == 'A'){
175
176                     // ========================================== Lógica da Colisão para a Esquerda ==========================================
177                     // Verifica se o player pode continuar andando para Esquerda
178                     if((player->scol-2)>-1)  // Se o jogador não estiver na primeira coluna de cada quadrado do LCD
179                             logic_op = scmap[(map[player->line-1][player->col-1].schar) - 1].adds[player->sline-1] & (pow(2,5-(player->scol-1)));  //
                                ↪    Verifica se andar para esquerda é válido.
180                     else if(player->scol==1 && player->col >1)        // Se o jogador estiver na primeira coluna de cada quadrado do LCD
181                             logic_op = scmap[(map[player->line-1][player->col-2].schar) - 1].adds[player->sline-1] & (pow(2,5-(5)));// Verifica se a
                                ↪    coluna de quadrados do LCD uma posição para esquerda permite transição ou se possuí parede.
182                     else if(player->scol==1 && player->col == 1) // Se o jogador estiver na primeira coluna do LCD
183                             return 0; // O jogador achou uma saída do mapa
184                     // ========================================== Lógica da Colisão para a Esquerda ==========================================
185
186                     if(logic_op == 0){                                              // Se o jogador puder se movimentar
187                             if(player->scol != 1)
188                                     player->scol--;                                         //decrementa a posição no mapa
189                             else{
190                                     if(player->col != 1){
191                                             player->scol = 5;                       //coloca no último segmento de coluna do quadrado do LCD na coluna à
                                                ↪    esquerda
192                                             player->col--;                                  //decrementa a posição no mapa
193                                     }
194                             }
195                     }
196             }
197
198             //printMap();                                                                   // Imprime o mapa do jogo novamente
199             RI = 0;                                                                         // Reinicializa o flag de receber mensagens pela Serial
200         }
201
202         return 1;                                                                   // O jogador ainda não achou uma saída
203 }
```

5.6 TIMER.h

```
#ifndef TIMER_H_INCLUDED
#define TIMER_H_INCLUDED

/* A struct timer guarda as variáveis relacionada aos timers usados
*        TIMER -> Delay Microseconds e Milisseconds
*
*         Como a memória do timer interno dá overflow em 65535 e cada contagem acontece em 1 microsegundo, o timer dá um overflow a cada 65.535 milisegundos,
 ↪    portanto:
*        Cicles    = (tempo * 1000)/ 65535 (DIVISÃO INTEIRA)
*         lastClock = (tempo * 1000)% 65535 (RESTO DA DIVISÃO)
*   finishMili   -> flag para informar que a contagem miliseconds acabou
*   finishMicro   -> flag para informar que a contagem microseconds acabou
*   flag     -> Controla se o timer está temporizando ou não
*/
struct timer{
        unsigned int cycles;
        unsigned int lastClock;
        unsigned char finishMili;
        unsigned char finishMicro;
        unsigned char flag;
};

struct timer Timer0;

void timerConfig();

void delay(unsigned int time, unsigned char miliseconds);

void volatile timer() __interrupt 1;

#endif
```

5.7 TIMER.c

```
#include "TIMER.h"
#include <mcs51/8051.h>
#include <time.h>
#include <stdio.h>

void timerConfig(){
        TH0 = 0;                                        // Zerar os bits mais significativos do temporizador TIMER0
        TL0 = 0;                                        // Zerar os bits menos significativos do temporizador TIMER0

        TMOD = 0x21;                                 // Habilitando modo 8-bits recarga automática (TIMER1) e contagem modo 16-bits (TIMER0) (slide sistmicro
         ↪    04_Timer.pdf pg.8)

        TH1 = 0xFD;                                 // Zerar os bits mais significativos do temporizador - HABILITANDO COMUNICAÇÃO 9600 bps(usando 8 bits)
        TL1 = 0xFD;                                 // Zerar os bits menos significativos do temporizador - HABILITANDO COMUNICAÇÃO 9600 bps (usando 8 bits)

    TR1 = 1;                                // Iniciando o T1

        P1 = 0x00;

        Timer0.finishMicro = 0;
        Timer0.finishMili = 0;
}

```

```
23  /* FUNÇÃO delay:
24  *                Funcao para delay em milisegundos ou microsegundos que recebe como parâmetro a quantidade de tempo desejado
25  *       Se selecionado o modo milisegundos = 1 realiza-se a temporização em milisegundos, se miliseconds = 0 a temporização será de microsegundos.
26  */
27  void delay(unsigned int time, unsigned char miliseconds){
28
29          Timer0.flag=1;
30
31          if(!Timer0.finishMili && miliseconds){
32                  Timer0.cycles = time/65;
33                  Timer0.lastClock = 65535 - ((time % 65)*1000);
34
35                  TR0 = 1;
36                  Timer0.finishMili = 1;
37
38                  while(Timer0.finishMili);
39          }
40          else if(!Timer0.finishMili){
41                  Timer0.cycles = 1;
42                  Timer0.lastClock = 65535 - time;
43                  TR0 = 1;
44
45                  while(!Timer0.finishMicro);
46
47                  Timer0.finishMicro = 0;
48          }
49
50
51
52  }
53
54  /* FUNÇÃO timer:
55  *                Função que controla a interrupção do TIMER0
56  */
57  void volatile timer() __interrupt 1{                        //(slide aplicmicro 11_C.pdf pg.13)
58
59          TH0 = 0;                                                                                                    // Zerar os bits mais significativos do
           ↪    contador
60          TL0 = 0;                                                                                                    // Zerar os bits menos significativos do
           ↪    contador
61          TF0 = 0;                                                                                                   // Zero a flag do contador
62          TR0 = 0;                                                                                                   // Paro timer0
63
64          if(Timer0.cycles > 1){
65          Timer0.cycles--;                                                                                                // decrementa cycles
66                  TR0 = 1;                                                                                           // inicia contador
67          }
68          else if(Timer0.cycles == 1){
69                  TH0 = Timer0.lastClock & 0xFF ;                                                                  // Atribui ao ultimo timer o valor dos bits mais significativos
70                  TL0 = Timer0.lastClock >> 8;                                                                    // Atribui ao ultimo timer o valor dos bits menos significativos
71          Timer0.cycles--;                                                                                                // decrementa cycles
72
73                  Timer0.finishMicro = 1;
74
75                  TR0 = 1;                                                                                           // inicia contador
76          }
77          else if(Timer0.cycles == 0){
78          Timer0.finishMili = 0;                                                                                    // finish é uma flag que indica que o tempo acabou
79          }
80  }
```

5.8 Math.h

```
#ifndef Math_H_INCLUDED
#define Math_H_INCLUDED

unsigned char pow(unsigned char base, unsigned char power);

#endif
```

5.9 Math.c

```
#include<Math.h>

/*        Função POW:
                - Realiza a potência de um número na base dada
*/
unsigned char pow(unsigned char base, unsigned char power){
        unsigned char n, total = 1;

        for(n = 0; n < power; n++){
                total*=base;
        }

        return total;
}
```

5.10 DAC.h

```
#ifndef DAC_H_INCLUDED
#define DAC_H_INCLUDED

#define dacValue 0xFFE4


/*
Após coletado alguns valores de frequência para alguns valores de delay no código:
        dacWrite = 255;
        for(n=0;n < delay; n++);
        dacWrite = 0;
        for(n=0;n < delay; n++);

        Foi obtido uma ajuste de curva em relação freq x delay resultando na respectiva equação
                y(x) = 66007/((x)^1.001)

                sendo x -> frequência
                        e y -> delay

        Obtendo assim os delay para as notas desejadas.
*/
#define C8 16
#define B7 17
#define AS7 18
```

```
#define A7 19
#define GS7 20
#define G7 21
#define FS7 22
#define F7 23
#define E7 25
#define DS7 26
#define D7 28
#define CS7 30
#define C7 31
#define B6 33
#define AS6 35
#define A6 37
#define GS6 39
#define G6 42
#define FS6 44
#define F6 47
#define E6 50
#define DS6 53
#define D6 56
#define CS6 59
#define C6 63
#define B5 66
#define AS5 70
#define A5 75
#define GS5 79
#define G5 84
#define FS5 89
#define F5 94
#define E5 99
#define DS5 105
#define D5 112
#define CS5 118
#define C5 125
#define B4 133
#define AS4 141
#define A4 149
#define GS4 158
#define G4 167
#define FS4 177
#define F4 188
#define E4 199
#define DS4 211
#define D4 223
#define CS4 237
#define C4 251
#define B3 266
#define AS3 282
#define A3 298
#define GS3 316
#define G3 335
#define FS3 355
#define F3 376
#define E3 398
#define DS3 422
#define D3 447
#define CS3 474
#define C3 502
#define B2 532
#define AS2 564
#define A2 597
#define GS2 633
#define G2 670
#define FS2 710
#define F2 753
#define E2 797
#define DS2 845
```

```
#define D2 895
#define CS2 949
#define C2 1005
#define B1 1065
#define AS1 1128
#define A1 1195
#define GS1 1266
#define G1 1342
#define FS1 1422
#define F1 1506
#define E1 1596
#define DS1 1691
#define D1 1792
#define CS1 1898
#define C1 2011
#define B0 2131
#define AS0 2258
#define A0 2392
#define PAUSE 0


__xdata __at dacValue unsigned char dacWrite;

unsigned int mainMario_melody[] = {
  E5, E5, PAUSE, E5,
  PAUSE, C5, E5, PAUSE,
  G5, PAUSE, PAUSE,  PAUSE,
  G4, PAUSE, PAUSE, PAUSE,

  C5, PAUSE, PAUSE, G4,
  PAUSE, PAUSE, E4, PAUSE,
  PAUSE, A4, PAUSE, B4,
  PAUSE, AS4, A4, PAUSE,

  G4, E5, G5,
  A5, PAUSE, F5, G5,
  PAUSE, E5, PAUSE, C5,
  D5, B4, PAUSE, PAUSE,

  C5, PAUSE, PAUSE, G4,
  PAUSE, PAUSE, E4, PAUSE,
  PAUSE, A4, PAUSE, B4,
  PAUSE, AS4, A4, PAUSE,

  G4, E5, G5,
  A5, PAUSE, F5, G5,
  PAUSE, E5, PAUSE, C5,
  D5, B4, PAUSE, PAUSE
};

unsigned char mainMario_noteTime[] = {
  15, 15, 15, 15,
  15, 15, 15, 15,
  15, 15, 15, 15,
  15, 15, 15, 15,

  15, 15, 15, 15,
  15, 15, 15, 15,
  15, 15, 15, 15,
  15, 15, 15, 15,

  12, 12, 12,
  15, 15, 15, 15,
  15, 15, 15, 15,
  15, 15, 15, 15,

  15, 15, 15, 15,
```

```
  15, 15, 15, 15,
  15, 15, 15, 15,
  15, 15, 15, 15,

  12, 12, 12,
  15, 15, 15, 15,
  15, 15, 15, 15,
  15, 15, 15, 15,
};

unsigned int underworld_melody[] = {
  C4, C5, A3, A4, AS3, AS4, PAUSE,
  PAUSE,
  C4, C5, A3, A4,
  AS3, AS4, PAUSE,
  PAUSE,
  F3, F4, D3, D4,
  DS3, DS4, PAUSE,
  PAUSE,
  F3, F4, D3, D4,
  DS3, DS4, PAUSE,
  PAUSE, DS4, CS4, D4,
  CS4, DS4,
  DS4, GS3,
  G3, CS4,
  C4, FS4, F4, E3, AS4, A4,
  GS4, DS4, B3,
  AS3, A3, GS3,
  PAUSE, PAUSE, PAUSE
};

unsigned char underworld_noteTime[] = {
  12, 12, 12, 12,
  12, 12, 6,
  3,
  12, 12, 12, 12,
  12, 12, 6,
  3,
  12, 12, 12, 12,
  12, 12, 6,
  3,
  12, 12, 12, 12,
  12, 12, 6,
  6, 18, 18, 18,
  6, 6,
  6, 6,
  6, 6,
  18, 18, 18, 18, 18, 18,
  10, 10, 10,
  10, 10, 10,
  3, 3, 3
};

unsigned int adobe_melody[] = {
  B4, B4, PAUSE, B4,
  D5, D5, PAUSE, D5,
  A4, A4, PAUSE, A4,
  B4, B4, PAUSE, PAUSE
};
unsigned char adobe_noteTime[] = {
  6,6,6,6,
  6,6,6,6,
  6,6,6,6,
  6,6,6,6
};

unsigned int beep1_melody[] = {
```

```
  C4,PAUSE
};
unsigned char beep1_noteTime[] = {
  6,12
};

unsigned int beep2_melody[] = {
  CS4,PAUSE
};
unsigned char beep2_noteTime[] = {
  6,12
};

void speaker(unsigned int note, unsigned int noteDuration);

void squareWave(unsigned int note, unsigned int tempo);

void sing(unsigned char song);

#endif
```

5.11 DAC.c

```
#include "DAC.h"
#include "TIMER.h"

unsigned int i = 0;

/*
        Baseado no código de: https://www.usinainfo.com.br/blog/tocando-tema-do-super-mario-com-buzzer-e-arduino/
*/
void speaker(unsigned int note, unsigned int noteDuration){
        long numCycles = note * noteDuration / 1000;
        long i;
        unsigned int n;

        for (i = 0; i < numCycles; i++) {
                dacWrite = 255;
                for(n = 0;n< note ;n++);
                dacWrite = 0;
                for(n = 0;n< note;n++);
        }
}

/*
        Baseado no código de: https://www.usinainfo.com.br/blog/tocando-tema-do-super-mario-com-buzzer-e-arduino/
*/
void squareWave(unsigned int note, unsigned int tempo){

        unsigned int noteDuration = 1000 / tempo;
        unsigned int pauseBetweenNotes;

        speaker(note,noteDuration);

        //pauseBetweenNotes = noteDuration * 1.30;
        //delay(pauseBetweenNotes,1);

        speaker(PAUSE,noteDuration);
}

/*
```

```
        Baseado no código de: https://www.usinainfo.com.br/blog/tocando-tema-do-super-mario-com-buzzer-e-arduino/
*/
void sing(unsigned char song) {



        if(song == 1){
                //for( i=0;i<(sizeof(adobe_melody)/sizeof(int));i++)
                squareWave(adobe_melody[i],adobe_noteTime[i]);

                if(i+1<(sizeof(adobe_melody)/sizeof(unsigned int)))
                        i++;
                else
                        i = 0;

        }
        else if(song == 2){
                squareWave(underworld_melody[i],underworld_noteTime[i]);

                if(i+1<(sizeof(underworld_melody)/sizeof(unsigned int)))
                        i++;
                else
                        i = 0;
        }
        else if(song == 3){
                squareWave(mainMario_melody[i],mainMario_noteTime[i]);

                if(i+1<(sizeof(mainMario_melody)/sizeof(unsigned int)))
                        i++;
                else
                        i = 0;
        }
        else if(song == 4){
                for( i =0;i<(sizeof(beep1_melody)/sizeof(int));i++)
                        squareWave(beep1_melody[i],beep1_noteTime[i]);
        }
        else if(song == 5){
                for( i =0;i<(sizeof(beep2_melody)/sizeof(int));i++)
                        squareWave(beep2_melody[i],beep2_noteTime[i]);
        }
}
```

5.12 Makefile

```
sdcc --model-large -c Math.c
sdcc --model-large -c LCD.c
sdcc --model-large -c TIMER.c
sdcc --model-large -c DAC.c
sdcc --model-large -c SERIAL.c

sdcc --model-large main.c LCD.rel TIMER.rel DAC.rel SERIAL.rel Math.rel

packihx main.ihx >main.hex
```