**Biblioteca LCD para o Arduino**

|  |  |
| --- | --- |
| PA7 = D7  PA6 = D6  PA5 = D5  PA4 = D4  PC3 = Back Light  PC2 = R/#W  PC1 = RS  PC0 = E | Uma imagem contendo texto  Descrição gerada automaticamente |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Port A - Dados | | | | Port C - Controle | | | |
| PA7 | PA6 | PA5 | PA4 | PC3 | PC2 | PC1 | PC0 |
| D7 | D6 | D5 | D4 | BL | R/#W | RS | E |

#define NRL 4 //Quantidade de linhas do LCD, numeradas de 0, 1, 2, 3

#define NRC 20 //Quantidade de colunas do LCD, numeradas de 0, 1, 2, ..., 20

Acesso ao LCD SEMPRE é feito via um buffer!

A escrita no LCD é lenta e há a incerteza de quanto tempo que se gasta esperando o BUSY = 0. Para ficar mais rápido e facilitar o acesso ao LCD, foi criada matriz **lcd\_buf[NRL][NRC]**. As escritas acontecem nesse buffer, sendo que cada escrita marca, na variável **lcd\_mudou**, a linha do LCD que foi alterada. O Timer 1 (100 Hz) monitora essa variável e habilita Timer 2, para atualizar uma linha de cada vez. As linhas são independentes, ou seja, a escrita interrompe quando chega ao fim da linha.

O Timer 1 usa a variável **lcd\_rr** para monitorar as linhas dando prioridade rotativa (?Round Robin). A ordem de busca muda a cada interrupção. A função **lcd\_linha\_alterou(lcd\_rr)** retorna TRUE se a linha precisa ser atualizada.

Na 1a vez examina na sequência: 0,1,2,3

Na 2a vez examina na sequência: 1,2,3,0

Na 3a vez examina na sequência: 2,3,0,1

Na 4a vez examina na sequência: 3, 0,1,2

Abaixo está o trecho da rotina de interrupção do Timer 1 que busca por uma linha do LCD a ser atualizada. Ela sempre altera a sequência de busca. Notar o **for()** com NRL repetições, sendo **lcd\_rr** incrementado a cada repetição. Ao sair do **for**, **lcd\_rr** é novamente incrementada.

Isso não foi feito na versão anterior. Essa versão sempre busca na ordem 0,1,2,3. A linha 0 tinha a maior prioridade. Assim, se a linha 0 mudava com muita frequência, as demais linhas não eram mostradas.

if (lcd\_mudou){

if (lcd\_busy==FALSE){

for (i=0; i<NRL; i++){

if (**lcd\_linha\_alterou(lcd\_rr)==TRUE**){

lcd\_fase=0;

lcd\_ix=0;

lcd\_busy=TRUE;

TIFR2 |= (1<<OCF2A); //Apagar interrup pendente

TIMSK2 |= (1<<OCIE2A); //Habilitar interrup CTC (ORC1A)

**break;**

}

**lcd\_rr = (lcd\_rr+1)&3;**

}

}

}

**lcd\_rr = (lcd\_rr+1)&3;**

Variáveis usadas:

**Byte lcd\_buf [NRL] [NRC+1];** 🡪 matriz que funciona como buffer do LCD. Notar que na primeira coluna de cada linha está o endereço para reposicionar o cursor. O BIT8 já está colocado em 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 (col=0) | 2 (col=0) | 3 (col=0) | … | … | 20 (col=19) |
| Linha=0 | 0x80 | A | B | C | … | … | D |
| Linha=1 | 0xC0 | E | F | G | … | … | H |
| Linha=2 | 0x94 | I | J | K | … | … | L |
| Linha=3 | 0xD4 | M | N | O | … | … | P |

**Byte lcd\_mudou;** 🡪 indicar qual linha precisa de atualização, faixa de 0, ..., 16.

* BIT0 = 1, (0000 0001) 🡪 linha 0 necessita de atualização
* BIT1 = 1, (0000 0010) 🡪 linha 1 necessita de atualização
* BIT2 = 1, (0000 0100) 🡪 linha 2 necessita de atualização
* BIT3 = 1, (0000 1000) 🡪 linha 3 necessita de atualização

O Timer 1 (10 mseg) monitora a variável lcd\_mudou. Se ela ficar diferente de zero, ele atualiza o ponteiro **lcd\_lin** e habilita a interrupção do Timer 2 que transfere toda a linha para o LCD.

* **lcd\_busy** 🡪 TRUE indica que o LCD está sendo atualizado pelas interrupções do Timer 2, o Timer 1 não pode ativar atualização de linha enquanto a atualização atual não terminar.
* **lcd\_lin** 🡪 ponteiro que aponta para o início da linha a ser escrita no LCD. Timer 1 inicializa essa variável.
* **lcd\_fase = 0;** 🡪 fase da atualização. Conta 0, 1, 2, 3, 2, 3, 2, 3.... . Fases 0 e 1 para o cursor, fases 2 e 3 (repetidadas 20 vezes) para letras. Timer 1 inicializa essa variável com 0.
* **lcd\_ix=0;** 🡪 indexador da linha da matriz. Vai de 0, 1, ..., 20. Timer 1 inicializa essa variável com 0.

O timer2 gerando uma interrupção a cada 1 mseg é o responsável por atualizar uma linha.

Funções

|  |  |  |
| --- | --- | --- |
|  |  |  |
| void | lcdb\_float | (byte lin, byte col, float fx, byte prec) |
| void | lcdb\_dec32 | (byte lin, byte col, long dt) |
| void | lcdb\_dec32u | (byte lin, byte col, long dt) |
| void | lcdb\_dec32nz | (byte lin, byte col, long dt) |
| void | lcdb\_dec32unz | (byte lin, byte col, long dt) |
| void | lcdb\_hex32 | (byte lin, byte col, long dt) |
|  |  |  |
| void | lcdb\_dec16 | (byte lin, byte col, int dt) |
| void | lcdb\_dec16u | (byte lin, byte col, word dt) |
| void | lcdb\_dec16nz | (byte lin, byte col, int dt) |
| void | lcdb\_dec16unz | (byte lin, byte col, word dt) |
| void | lcdb\_hex16 | (byte lin, byte col, byte dt) |
|  |  |  |
| void | lcdb\_dec8 | (byte lin, byte col, byte dt) |
| void | lcdb\_dec8u | (byte lin, byte col, byte dt) |
| void | lcdb\_dec8nz | (byte lin, byte col, byte dt) |
| void | lcdb\_dec8unz | (byte lin, byte col, byte dt) |
| void | lcdb\_hex8 | (byte lin, byte col, byte dt) |
|  |  |  |
| void | lcdb\_spc | (byte lin, byte col, byte qtd) |
| void | lcdb\_str | (byte lin, byte col, char \*pt) |
| void | lcdb\_char | (byte lin, byte col, char dt) |
| void | lcdb\_atualiza | (void) |
| void | lcdb\_copia | (byte linha) |
| void | lcdb\_config | (void) |
|  |  |  |
| void | lcd\_float | (float fx, byte prec) |
| void | lcd\_dec32 | (long dt) |
| void | lcd\_dec32u | (long dt) |
| void | lcd\_dec32nz | (long dt) |
| void | lcd\_dec32unz | (long dt) |
| void | lcd\_hex32 | (long dt) |
|  |  |  |
| void | lcd\_dec16 | (int dt) |
| void | lcd\_dec16u | (word dt) |
| void | lcd\_dec16nz | (int dt) |
| void | lcd\_dec16unz | (word dt) |
| void | lcd\_hex16 | (word dt) |
|  |  |  |
| void | lcd\_dec8 | (byte dt) |
| void | lcd\_dec8u | (byte dt) |
| void | lcd\_dec8nz | (byte dt) |
| void | lcd\_dec8unz | (byte dt) |
| void | lcd\_hex8 | (byte dt) |
|  |  |  |
| void | lcdb\_apaga | (void) |
| void | lcd\_str\_lc | (byte lin, byte col, char \*pt) |
| void | lcd\_str | (char \*pt) |
| void | lcd\_spc | (byte qtd) |
| void | lcd\_cursor\_lc | (byte lin, byte col) |
| void | lcd\_cursor | (byte addr) |
| void | lcd\_char | (byte dt) |
| void | lcd\_char\_nbusy | (byte dt) |
| void | lcd\_cmdo | (byte dt) |
| void | lcd\_busy\_hold | (void) |
| byte | lcd\_status | (void) |
| void | lcd\_config | (void) |
| void | lcd\_pinos | (void) |
| void | lcd\_inic | (void) |
| void | lcd\_nib\_wr | (byte nib) |
|  |  |  |
| void | lcd\_e | (void) |
| void | lcd\_E | (void) |
| void | lcd\_rs | (void) |
| void | lcd\_RS | (void) |
| void | lcd\_rw | (void) |
| void | lcd\_RW | (void) |
| void | lcd\_bl | (void) |
| void | lcd\_BL | (void) |
| void | lcd\_Bl | (void) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

* void **lcdb\_apaga** (void)

Escrever brancos em todo o buffer LCD

* void **lcdb\_spc** (byte lin, byte col, byte qtd)

Escrever uma qtd de espaços a partir da posição (lin,col)

* void **lcdb\_str** (byte lin, byte col, char \*msg)

Escrever uma string a partir da posição (lin,col)

* void **lcdb\_char** (byte lin, byte col, byte dt)

Escrever um char, na posição (lin,col)

* void **lcdb\_atualiza** (void)

Atualizar LCD com os dados do buffer

* void **lcdb\_copia** (byte linha)

Copiar uma linha do buffer para o LCD

* void **lcdb\_config** (void)

Configurar o buffer para o LCD

.

* void **lcd\_float** (float f, byte prec)

No Arduino, double e float têm a mesma precisão

Escreve no LCD o float fx com prec casas após a vírgula e apresenta o sinal.

Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])

Limite da parte inteira = 9 dígitos. Limite da parte fracionária = 12 dígitos.

Caso ultrapasse esses os limites imprime ###, ###

O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###.

Na verdade, o máximo é 999.999.999,999967. Exemplos:

999.999.999,0 🡪 imprime 999.999.936,000000 (por causa da precisão da representação)

876.543.210,123456789 🡪 imprime 876543232,000000 (por causa da precisão da representação)

* void **lcd\_dec32** (long dt)

Escrever no LCD 32 bits em decimal, com sinal e com zeros à esquerda.

* void **lcd\_dec32u** (long dt)

Escrever no LCD 32 bits em decimal, sem sinal e com zeros à esquerda.

* void **lcd\_dec32nz** (long dt)

Escrever no LCD 32 bits em decimal, com sinal e sem zeros à esquerda.

* void **lcd\_dec32unz** (long dt)

Escrever no LCD 32 bits em decimal, sem sinal e sem zeros à esquerda.

* void **lcd\_hex32** (long dt)

Escrever no LCD 32 bits em hexadecimal.

* void **lcd\_dec16** (word dt)

Escrever no LCD 16 bits em decimal, com sinal e com zeros à esquerda.

* void **lcd\_dec16u** (word dt)

Escrever no LCD 16 bits em decimal, sem sinal e com zeros à esquerda.

* void **lcd\_dec16nz** (word dt)

Escrever no LCD 16 bits em decimal, com sinal e sem zeros à esquerda.

* void **lcd\_dec16unz** (word dt)

Escrever no LCD 16 bits em decimal, sem sinal e sem zeros à esquerda.

* void **lcd\_hex16** (word dt)

Escrever no LCD 16 bits em hexadecimal.

* void **lcd\_dec8** (byte dt)

Escrever no LCD 8 bits em decimal, com sinal e com zeros à esquerda.

* void **lcd\_dec8u** (byte dt)

Escrever no LCD 8 bits em decimal, sem sinal e com zeros à esquerda.

* void **lcd\_dec8nz** (byte dt)

Escrever no LCD 8 bits em decimal, com sinal e sem zeros à esquerda.

* void **lcd\_dec8unz** (byte dt)

Escrever no LCD 8 bits em decimal, sem sinal e sem zeros à esquerda.

* void **lcd\_hex8** (byte dt)

Escrever no LCD 8 bits em hexadecimal.

* void **lcdb\_apaga** (void)

Escrever brancos em todo o LCD

* void **lcd\_spc** (byte qtd)

Escrever no LCD qtd de espaços. Máximo de 20 espaços.

* void **lcd\_cursor\_lc** (byte lin, byte col)

No LCD, posicionar cursor na posição linha=0,1, ... e coluna=0,1,2... .

É preciso definir NRC (qtd de colunas). Aqui não usa, mas é preciso definir NRL (qtd de linhas).

* void **lcd\_cursor** (byte addr)

No LCD, posicionar cursor no endereço addr da RAM de Dados do LCD.

* void **lcd\_str\_lc** (byte lin, byte col, char \*pt){

No LCD, escrever string terminada em zero na posição "lin" e "col"

* void **lcd\_str** (char \*pt)

No LCD escrever string terminada em zero na posição atual do cursor.

* void **lcd\_char** (byte dt)

Escrever no LCD um char. Espera Busy = 0. Usa lcd\_busy\_hold().

* void **lcd\_char\_nbusy** (byte dt)

Escrever no LCD um char. Não testa o Busy. Usa lcd\_nib\_wr() que tem delays.

* void **lcd\_cmdo** (byte dt)

Escrever um comando. Espera Busy = 0. Usa lcd\_busy\_hold().

* void **lcd\_busy\_hold** (void)

Só retorna com BUSY = 0.

* byte **lcd\_status** (void)

Retorna status em 8 bits BAAAAAAA (Busy + endereço).

* void **lcd\_config** (void)

Configura o LCD, usa as funções lcd\_pinos() e lcd\_inic().

* void **lcd\_pinos** (void)

Configura pinos (PC3,2,1,0 e PA7,6,5,4) para serem usados com o LCD.

* void **lcd\_inic** (void)

Executa a sequência descrita no manual para inicializar o LCD. Usa a função lcd\_nib\_wr ().

* void **lcd\_nib\_wr** (byte nib)

Escreve nibble = 0000 xxxx B no LCD. Não altera RS e RW

Não checa o Busy, por isso faz uso de delays.

Alterar estado de RW (PC2), RS (PC1) e E (PC0)- Sinais de controle do LCD

* void **lcd\_e** (void) 🡪 E=0
* void **lcd\_E** (void) 🡪 E=1
* void **lcd\_rs** (void) 🡪 RS=0 - Cmdo
* void **lcd\_RS** (void) 🡪 RS=1 - Dado
* void **lcd\_rw** (void) 🡪 RW=0 - Escrita
* **void lcd\_RW** (void) 🡪 RW=1 - Leitura

Controlar o Back Light do LCD (BL = PC3): Apagar / Acender / Inverter

* void **lcd\_bl** (void) 🡪 BL = Apagado
* void **lcd\_BL** (void) 🡪 BL = Aceso
* void **lcd\_Bl** (void) 🡪 BL = Invertido