Biblioteca LCD para a Caixa Preta

Versão 1.0, 13/04/2020

Funções

void	lcd_gps_data	(byte lin, byte col)
void	lcd_gps_hora	(byte lin, byte col)
void	lcd_gps_lat	(byte lin, byte col)
void	lcd_gps_long	(byte lin, byte col)
void	lcd_gps_vel_kph	(byte lin, byte col)
void	lcd_gps_dado	(byte lin, byte col, byte qual)
void	lcd_float	(byte lin, byte col, float fx, byte prec)
void	lcd_dec32	(byte lin, byte col, long dt)
void	lcd_dec32u	(byte lin, byte col, long dt)
void	lcd_dec32nz	(byte lin, byte col, long dt)
void	lcd_dec32unz	(byte lin, byte col, long dt)
void	lcd_hex32	(byte lin, byte col, long dt)
void	lcd_dec16	(byte lin, byte col, int dt)
void	lcd_dec16u	(byte lin, byte col, word dt)
void	lcd_dec16nz	(byte lin, byte col, int dt)
void	lcd_dec16unz	(byte lin, byte col, word dt)
void	lcd_hex16	(byte lin, byte col, byte dt)
void	lcd_dec8	(byte lin, byte col, byte dt)
void	lcd_dec8u	(byte lin, byte col, byte dt)
void	lcd_dec8nz	(byte lin, byte col, byte dt)
void	lcd_dec8unz	(byte lin, byte col, byte dt)
void	lcd_hex8	(byte lin, byte col, byte dt)
void	lcd_spc	(byte lin, byte col, byte qtd)
void	lcd_str	(byte lin, byte col, char *pt)
void	lcd_char	(byte lin, byte col, char dt)
void	lcd_apaga_lin	(byte lin)
void	lcd_apaga	(void)
void	lcd_config	(void)
void	lcd_cmdo	(byte dt)
void	lcd_busy_hold	(void)
byte	lcd_status	(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 **lcd_gps_data** (byte lin, byte col)
  Imprimir data (DD/MM/AA) do GPS no LCD a partir de (lin,col).

- void **lcd_gps_hora** (byte lin, byte col)
  Imprimir HH:MM:SS no LCD a partir de (lin,col).

- void **lcd_gps_lat** (byte lin, byte col)
  Imprimir Latitude DD MM.MMMMM N/S no LCD a partir de (lin,col).

- void **lcd_gps_long** (byte lin, byte col)
  Imprimir Longitude DDD MM.MMMMM E/W no LCD a partir de (lin,col).

- void **lcd_gps_vel_kph** (byte lin, byte col)
  Imprimir velocidade kph a partir de (lin,col).

- void **lcd_gps_dado** (byte lin, byte col, byte qual)
  Imprimir um determinado dado do GPS. Argumento qual indica a posição no vetor **gps_dados**.

- void **lcd_float** (byte lin, byte col, float f, byte prec)
  No Arduino, double e float têm a mesma precisão
  Escreve no LCD (posição lin,col) 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** (byte lin, byte col, long dt)
  Escrever no LCD (posição lin,col) 32 bits em decimal, com sinal e com zeros à esquerda.

- void **lcd_dec32u** (byte lin, byte col, long dt)
  Escrever no LCD (posição lin,col) 32 bits em decimal, sem sinal e com zeros à esquerda.

- void **lcd_dec32nz** (byte lin, byte col, long dt)
  Escrever no LCD (posição lin,col) 32 bits em decimal, com sinal e sem zeros à esquerda.

- void **lcd_dec32unz** (byte lin, byte col, long dt)
  Escrever no LCD 32 (posição lin,col) bits em decimal, sem sinal e sem zeros à esquerda.

- void **lcd_hex32** (byte lin, byte col, long dt)
  Escrever no LCD (posição lin,col) 32 bits em hexadecimal.

- void **lcd_dec16** (byte lin, byte col, word dt)
  Escrever no LCD (posição lin,col) 16 bits em decimal, com sinal e com zeros à esquerda.

- void **lcd_dec16u** (byte lin, byte col, word dt)

Escrever no LCD (posição lin,col) 16 bits em decimal, sem sinal e com zeros à esquerda.

- void **lcd_dec16nz** (byte lin, byte col, word dt)
  Escrever no LCD (posição lin,col) 16 bits em decimal, com sinal e sem zeros à esquerda.

- void **lcd_dec16unz** (byte lin, byte col, word dt)
  Escrever no LCD (posição lin,col) 16 bits em decimal, sem sinal e sem zeros à esquerda.

- void **lcd_hex16** (byte lin, byte col, word dt)
  Escrever no LCD (posição lin,col) 16 bits em hexadecimal.

- void **lcd_dec8** (byte lin, byte col, byte dt)
  Escrever no LCD (posição lin,col) 8 bits em decimal, com sinal e com zeros à esquerda.

- void **lcd_dec8u** (byte lin, byte col, byte dt)
  Escrever no LCD (posição lin,col) 8 bits em decimal, sem sinal e com zeros à esquerda.

- void **lcd_dec8nz** (byte lin, byte col, byte dt)
  Escrever no LCD (posição lin,col) 8 bits em decimal, com sinal e sem zeros à esquerda.

- void **lcd_dec8unz** (byte lin, byte col, byte dt)
  Escrever no LCD (posição lin,col) 8 bits em decimal, sem sinal e sem zeros à esquerda.

- void **lcd_hex8** (byte lin, byte col, byte dt)
  Escrever no LCD (posição lin,col) 8 bits em hexadecimal.

- 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_apaga_lin** (byte lin)
  Escrever brancos na linha indicada.

- void **lcdb_apaga** (void)
  Apagar todo o LCD - Escreve brancos. Por segurança, acerta os caractes para posicionar cursor.

- void **lcdb_config** (void)
  Configurar o buffer para o LCD

- 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_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.

---- Funções para 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

---------- Sobre o LCD ----------

PA7 = D7

PA6 = D6

PA5 = D5

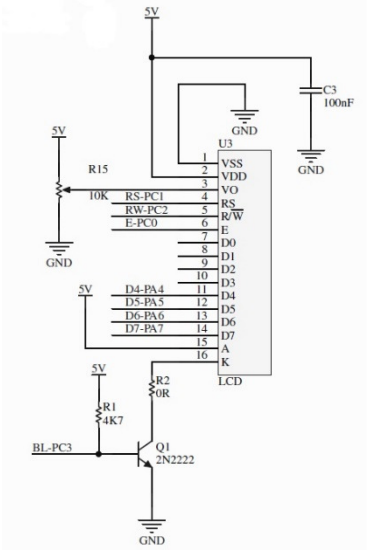
PA4 = D4

PC3 = Back Light

PC2 = R/#W

PC1 = RS

PC0 = E



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 a interrupção do Timer 2, para atualizar uma linha de cada vez. As linhas são independentes, ou seja, a escrita para quando chega ao fim da linha.

Os bits da variável **lcd_mudou** indicam quais linhas necessitam ser atualizadas. Assim, se **lcd_mudou** = 0, significa que nenhuma linha precisa de alteração.
Lcd_mudou = 0000 0000 → nenhuma linha para atualizar;
Lcd_mudou = 0000 0010 → atualizar a linha 1
Lcd_mudou = 0000 0110 → atualizar as linhas 1 e 2.

O Timer 1 sempre consulta **lcd_mudou** para saber se deve habilitar a interrupção do Timer 2. Se for ligado, o Timer 2 atualiza a linha a partir do endereço indicado na variável global **lcd_lin**.

Para facilitar o trabalho, foi criada a função **lcd_linha_alterou(lcd_rr)**, onde **lcd__rr** = 0, 1, 2 ou 3 indica a linha a ser verificada, na verdade, qual bit de **lcd_mudou** deve ser consultado. Por exemplo, **lcd_linha_alterou(2)** vai consultar o bit 2 da variável **lcd_mudou**.

A função **lcd_linha_alterou(lcd_rr)** faz as seguintes operações, caso o bit indicado esteja em 1:
1) Atualiza **lcd_lin** com o endereço da linha a ser transferida para o LCD,
2) Apaga, na variável **lcd_mudou**, o bit que foi testado e
3) Retorna TRUE

Existe ainda variável **lcd_busy** que indica que uma atualização do LCD está acontecendo que uma nova não deve ser iniciada.

A interrupção do Timer 1, (se **lcd_busy = FALSE**) chama a função **lcd_linha_alterou(lcd_rr)** indicando qual linha consultar e caso retorne TRUE, habilita a interrupção do Timer 2.

Numa versão, a consulta era sempre feita da linha 0 até a última. Acontecia que se a linha 0 era alterada com uma frequência muito grande, as demais linhas nunca eram atualizadas. Para corrigir esse problema, a ordem de verificação é circular.

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
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.