

```
1  /*****
2  * Arduino library for the Devantech LCD03/LCD05 with 3x4 keypad - V2.3 (ANSI C) *
3  * https://www.robot-electronics.co.uk/htm/Lcd03tech.htm *
4  * © Copyright 2019 Filippo Pardini - filippo@robotica.eng.br *
5  * http://blog.robotica.eng.br/ *
6  * *
7  * This program is free software: you can redistribute it and/or modify it *
8  * under the terms of the GNU Lesser General Public License as published by *
9  * the Free Software Foundation, either version 3 of the License, or any *
10 * later version. *
11 * *
12 * This program is distributed in the hope that it will be useful, *
13 * but WITHOUT ANY WARRANTY; without even the implied warranty of *
14 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the *
15 * GNU Lesser General Public License for more details. *
16 * *
17 * You should have received a copy of the GNU Lesser General Public License *
18 * along with this program. If not, see <http://www.gnu.org/licenses/>. *
19 *****/
20 * *
21 * Remember that the TWI bus must have pull-up resistors on SDA and SCL. *
22 * As LCD03/LCD05 are 5V, if the microcontroller is 3.3V a logic level *
23 * converter must be used. I recommend 1.8K or 4.7K resistors. *
24 * *
25 * This version was tested with Arduino 1.8.9 IDE and ESP-WROOM-32 *
26 * DOIT ESP32 DEVKIT V1 board. Some boards, as that, already has pull-up *
27 * resistors included at the I2C default pins (in this case SDA 21 SCL 22) *
28 * *
29 *****/
30
31 #ifndef LCD03_N1_h
32 #define LCD03_N1_h
33
34 // _____
35
36 char endereco; // Endereço LCD03
```

```
37  unsigned long tespera;                                     // Tempo de espera máximo no teclado (seg)
38
39  // -----
40
41  #define cursor_home envia_comando(1);
42  #define hide_cursor envia_comando(4);
43  #define show_underline_cursor envia_comando(5);
44  #define show_blinking_cursor envia_comando(6);
45  #define backspace envia_comando(8);
46  #define horizontal_tab envia_comando(9);
47  #define smart_line_feed envia_comando(10);
48  #define vertical_tab envia_comando(11);
49  #define clear_screen envia_comando(12);
50  #define clear_line(li) display_texto_lc("                ",li,1);
51  #define carriage_return envia_comando(13);
52  #define clear_column envia_comando(17);
53  #define tab_set envia_comando(18);
54  #define backlight_on envia_comando(19);
55  #define backlight_off envia_comando(20);
56  #define disable_startup_message envia_comando(21);
57  #define enable_startup_message envia_comando(22);
58  #define double_keypad_scan_rate envia_comando(28);
59  #define normal_keypad_scan_rate envia_comando(29);
60  #define change_address(adr) muda_endereco(adr);          // adr => 0xC6 ou 0xC8 ou 0xCA ou 0xCC ou 0xCE
61
62  // -----
63
64  void set_cursor(byte pos);                                 // Posiciona cursor 1 a 80
65  void set_cursor_lc(byte linha,byte coluna);               // Posiciona cursor na linha e coluna
66  void envia_comando(byte cmd);                             // Envia um comando
67  void muda_endereco(byte ender);                           // Muda o endereço do LCD03
68  byte read_keypad(void);                                    // Le o caractere do teclado
69  void lcd_registers_read(byte *fifo,byte *lsb,byte *msb,byte *ver); // Le os 4 registros do LCD03
70  char convert_keypad(byte tipo,byte car);                  // Converte os bits do teclado para caracteres
71  bool entrada_teclado(char *str);                          // Processa a entrada no teclado
72  void LCD03_begin(char ender,unsigned long tes);           // Inicializa o LCD03
```

```
73 void Serial_print(char *vetor,size_t inicio,size_t cp); // Imprime um vetor char no monitor serial
74 void display_float_lc(float flt,uint8_t lin,uint8_t col); // Display de um float na linha e coluna
75 void display_float(float flt); // Display de um float onde está o cursor
76 void display_texto_lc(const char *txt,uint8_t lin,uint8_t col); // Display de um texto na linha e coluna
77 void display_texto(const char *txt); // Display de um texto onde está o cursor
78 void display_vectorchar_lc(char *data,uint8_t lin,uint8_t col); // Display de um vetor char na linha e coluna
79 void display_vectorchar(char *data); // Display de um vetor char onde está o cursor
80 void display_char_lc(char val,uint8_t lin,uint8_t col); // Display de um caractere na linha e coluna
81 void display_char(char val); // Display de um caractere onde está o cursor
82 void display_string_lc(String *stri,uint8_t lin,uint8_t col); // Display de um String na linha e coluna
83 void display_string(String *stri); // Display de um String onde está o cursor
84
85 // -----
86
87 void LCD03_begin(char ender,unsigned long tes)
88 {
89 // Inicializa o LCD03
90
91 byte buf;
92 byte lsb;
93 byte msb;
94 byte ver;
95
96 endereco = ender >> 1;
97 tespera = tes * 1000;
98 lcd_registers_read(&buf,&lsb,&msb,&ver); // Lê os 4 registros
99 Serial.print("\nLCD03 versao ");
100 Serial.print(ver,DEC);
101 Serial.print("\nFIFO buffer = ");
102 Serial.print(buf,DEC);
103 Serial.print(" Bytes\n");
104 }
105
106 // -----
107
108 void set_cursor(byte pos)
```

```
109 {
110     // Posiciona o cursor 1 <= pos <= 80
111
112     byte dad[3];
113
114     dad[0] = 0;
115     dad[1] = 2;
116     dad[2] = pos;
117     Wire.beginTransaction(endereco);
118     Wire.write(dad,3);
119     Wire.endTransmission();
120 }
121
122 // -----
123
124 void set_cursor_lc(byte linha,byte coluna)
125 {
126     // Posiciona o cursor na linha e coluna
127
128     byte dad[4];
129
130     dad[0] = 0;
131     dad[1] = 3;
132     dad[2] = linha;
133     dad[3] = coluna;
134     Wire.beginTransaction(endereco);
135     Wire.write(dad,4);
136     Wire.endTransmission();
137 }
138
139 // -----
140
141 void envia_comando(byte cmd)
142 {
143     // Envia um comando para o registro 0
144
```

```
145     byte dad[2];
146
147     dad[0] = 0;
148     dad[1] = cmd;
149     Wire.beginTransmission(endereco);
150     Wire.write(dad,2);
151     Wire.endTransmission();
152 }
153
154 // -----
155
156 void muda_endereco(byte ender)
157 {
158     // Muda endereço I2C
159
160     byte dad[7];
161
162     dad[0] = 0;
163     dad[1] = 25;
164     dad[2] = 0x19;
165     dad[3] = 0xA0;
166     dad[4] = 0xAA;
167     dad[5] = 0xA5;
168     dad[6] = ender; // 0xC6 ou 0xC8 ou 0xCA ou 0xCC ou 0xCE
169     Wire.beginTransmission(endereco);
170     Wire.write(dad,7);
171     Wire.endTransmission();
172 }
173
174 // -----
175
176 void lcd_registers_read(byte *fifo,byte *lsb,byte *msb,byte *ver)
177 {
178     // Lê os 4 registros do LCD03
179
180     Wire.requestFrom(endereco,4);
```

```
181     *fifo = Wire.read(); // Registro 0 - fifo buffer
182     *lsb = Wire.read(); // Registro 1 - lsb teclado
183     *msb = Wire.read(); // Registro 2 - msb teclado
184     *ver = Wire.read(); // Registro 3 - Versão
185 }
186
187 // -----
188
189 bool entrada_teclado(char *str)
190 {
191     // Lê o teclado e recebe um vetor de caracteres. Nao é testado o espaco livre no buffer FIFO de 64 bytes
192     // A leitura é encerrada quando é digitado #
193
194     byte cod; // Variável para ler caractere do teclado
195     uint8_t i; // Indice
196     unsigned long t; // Contador de tempo
197
198     i = 1;
199     cod = 0; // Valor inicial inválido
200
201     // Espera no máximo "tespera" milisegundos para teclar
202     t = millis(); // Inicia contagem de tempo
203     while((cod == 0) && ((millis() - t) < tespera)) cod = read_keypad();
204     if (cod == 0) return false; // Esgotou o tempo
205     else // Tem caractere
206     {
207         str[i] = cod; // Armazena no string
208         i++; // Incrementa indice
209         while(1) // Até entrar com #
210         {
211             while(read_keypad() != 0); // Espera reset dos registros para 0
212             while((cod = read_keypad()) == 0); // Espera proxima digitação
213             if(cod == 35) // Se for #
214             {
215                 str[0] = 32; // Primeiro elemento sempre ' '
216                 str[i] = (byte)'\0'; // Encerra o string

```

```
217     return true; // Volta OK
218 }
219 else // Não é #, é outro caractere
220 {
221     str[i] = cod; // Armazena no string
222     i++; // Incrementa indice
223 }
224 }
225 }
226 }
227
228 // -----
229
230 byte read_keypad(void)
231 {
232     // Le o caractere digitado no teclado
233
234     byte fifo,lsb,msb,ver;
235     byte caractere;
236
237     lcd_registers_read(&fifo,&lsb,&msb,&ver);
238     if (lsb != 0) caractere = convert_keypad(1,lsb);
239     else if (msb != 0) caractere = convert_keypad(2,msb);
240     else caractere = 0;
241     return caractere;
242 }
243
244 // -----
245
246 char convert_keypad(byte tipo,byte car)
247 {
248     // Converte os bits do teclado para caracteres
249
250     byte baite;
251
252     baite = 0; // Inválido
```

```
253     switch (tipo)
254     {
255         case 1:          // É LSB
256         {
257             switch (car)
258             {
259                 case 1:    {baite = 49; break;}    // ASCII 1
260                 case 2:    {baite = 50; break;}    // ASCII 2
261                 case 4:    {baite = 51; break;}    // ASCII 3
262                 case 8:    {baite = 52; break;}    // ASCII 4
263                 case 16:   {baite = 53; break;}    // ASCII 5
264                 case 32:   {baite = 54; break;}    // ASCII 6
265                 case 64:   {baite = 55; break;}    // ASCII 7
266                 case 128:  {baite = 56; break;}    // ASCII 8
267                 default:   break;
268             }
269             break;
270         }
271
272         case 2 :        // É MSB
273         {
274             switch (car)
275             {
276                 case 1:    {baite = 57; break;}    // ASCII 9
277                 case 2:    {baite = 42; break;}    // ASCII *
278                 case 4:    {baite = 48; break;}    // ASCII 0
279                 case 8:    {baite = 35; break;}    // ASCII #
280                 default:   break;
281             }
282             break;
283         }
284         default:        break;
285     }
286     return baite;
287 }
288
```

```
289 // -----
290
291 void Serial_print(char *vetor, size_t inicio, size_t cp)
292 {
293     // Imprime no monitor serial um vetor de bytes iniciando na posição inicio por cp posições
294
295     size_t i, k;
296     char crt;
297
298     k = inicio + cp;
299     for(i=inicio; i<k; i++)
300     {
301         switch (vetor[i])
302         {
303             case 49: {crt = '1'; break;} // ASCII 1
304             case 50: {crt = '2'; break;} // ASCII 2
305             case 51: {crt = '3'; break;} // ASCII 3
306             case 52: {crt = '4'; break;} // ASCII 4
307             case 53: {crt = '5'; break;} // ASCII 5
308             case 54: {crt = '6'; break;} // ASCII 6
309             case 55: {crt = '7'; break;} // ASCII 7
310             case 56: {crt = '8'; break;} // ASCII 8
311             case 57: {crt = '9'; break;} // ASCII 9
312             case 42: {crt = '*'; break;} // ASCII *
313             case 48: {crt = '0'; break;} // ASCII 0
314             case 35: {crt = '#'; break;} // ASCII #
315         }
316         Serial.print(crt);
317     }
318 }
319
320 // -----
321
322 void display_float_lc(float flt, uint8_t lin, uint8_t col)
323 {
324     // Faz o display de um float com 2 decimais a partir da linha lin coluna col
```

```
325
326     String stri;
327
328     stri = String(flt,2);
329     set_cursor_lc(lin,col);
330     Wire.beginTransmission(endereco);
331     Wire.write(stri.c_str());
332     Wire.endTransmission();
333 }
334
335 // -----
336
337 void display_float(float flt)
338 {
339     // Faz o display de um float com 2 decimais a partir da posição do cursor
340
341     String stri;
342
343     Wire.beginTransmission(endereco);
344     Wire.write(stri.c_str());
345     Wire.endTransmission();
346 }
347
348 // -----
349
350 void display_texto_lc(const char *txt,uint8_t lin,uint8_t col)
351 {
352     // Faz o display de um texto a partir da linha lin coluna col
353
354     set_cursor_lc(lin,col);
355     Wire.beginTransmission(endereco);
356     Wire.write(txt);
357     Wire.endTransmission();
358 }
359
360 // -----
```

```
361
362 void display_texto(const char *txt)
363 {
364     // Faz o display de um texto a partir da posição do cursor
365
366     Wire.beginTransaction(endereco);
367     Wire.write(txt);
368     Wire.endTransmission();
369 }
370
371 // -----
372
373 void display_vectorchar_lc(char *data, uint8_t lin, uint8_t col)
374 {
375     // Faz o display de um vetor de caracteres a partir da linha lin coluna col
376
377     String stri;
378
379     stri = String(*data);
380     set_cursor_lc(lin, col);
381     Wire.beginTransaction(endereco);
382     Wire.write(stri.c_str());
383     Wire.endTransmission();
384 }
385
386 // -----
387
388 void display_vectorchar(char *data)
389 {
390     // Faz o display de um vetor de caracteres a partir da posição do cursor
391
392     String stri;
393
394     stri = String(*data);
395     Wire.beginTransaction(endereco);
396     Wire.write(stri.c_str());
```

```
397     Wire.endTransmission();
398 }
399
400 // -----
401
402 void display_char_lc(char val,uint8_t lin,uint8_t col)
403 {
404     // Faz o display de um caractere a partir da linha lin coluna col
405
406     set_cursor_lc(lin,col);
407     Wire.beginTransmission(endereco);
408     Wire.write(val);
409     Wire.endTransmission();
410 }
411
412 // -----
413
414 void display_char(char val)
415 {
416     // Faz o display de um caractere a partir da posição do cursor
417
418     Wire.beginTransmission(endereco);
419     Wire.write(val);
420     Wire.endTransmission();
421 }
422
423 // -----
424
425 void display_string_lc(String *stri,uint8_t lin,uint8_t col)
426 {
427     // Faz o display de um String a partir da linha lin coluna col
428
429     set_cursor_lc(lin,col);
430     Wire.beginTransmission(endereco);
431     Wire.write(stri->c_str());
432     Wire.endTransmission();
```

```
433 }
434
435 // -----
436
437 void display_string(String *stri)
438 {
439     // Faz o display de um String a partir da posição do cursor
440
441     Wire.beginTransaction(endereco);
442     Wire.write(stri->c_str());
443     Wire.endTransmission();
444 }
445
446 // -----
447
448
449 #endif
450
```