```
 1: /*****************************************************************************
 2: * ATxmega library for the Devantech LCD03/LCD05 with 3x4 keypad - V2.0       *
 3: * © Copyright 2012-2019 Filippo Pardini - filippo@robotica.eng.br            *
 4: *                                                                           *
 5: * This program is free software: you can redistribute it and/or modify it   *
 6: * under the terms of the GNU Lesser General Public License as published by  *
 7: * the Free Software Foundation, either version 3 of the License, or any     *
 8: * later version.                                                            *
 9: *                                                                           *
10: * This program is distributed in the hope that it will be useful,           *
11: * but WITHOUT ANY WARRANTY; without even the implied warranty of            *
12: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the              *
13: * GNU Lesser General Public License for more details.                       *
14: *                                                                           *
15: * You should have received a copy of the GNU Lesser General Public License  *
16: * along with this program. If not, see <http://www.gnu.org/licenses/>.      *
17: *****************************************************************************
18:
19: Remember that the TWI bus must have pull-up resistors on SDA and SCL.
20: As LCD03/LCD05 are 5V, if the microcontroller is 3.3V a logic level
21: converter must be used. I recommend 1.8K or 4.7K resistors.
22:
23: The TWI must be initialized as TWIC, Peripheral Clock frequency: 32000000 Hz,
24: SCL Rate: 100000 bps and Master interrupt: Low Level
25:
26: *****************************************************************************/
27:
28: #include <twix.h>
29: #include <delay.h>
30:
31: #define ad_aeternum while(1)
32: #define delta_time(t) (unsigned long int)(*LCD_inst - (unsigned long int)t)
33: #define lcd_cursor_home() lcd_byte_write(1)
34: #define lcd_set_cursor_1_80(pos) lcd_2byte_write(2,pos)
35: #define lcd_set_cursor_l_c(l,c) lcd_3byte_write(3,l,c)
36: #define lcd_hide_cursor() lcd_byte_write(4)
37: #define lcd_cursor_type(tipo) if (tipo) lcd_byte_write(6);else lcd_byte_write(5)
38: #define lcd_backspace() lcd_byte_write(8)
39: #define lcd_h_tab() lcd_byte_write(9)
40: #define lcd_v_tab() lcd_byte_write(11)
41: #define lcd_smart_line_feed() lcd_byte_write(10)
42: #define lcd_carriage_return() lcd_byte_write(13)
```

```
43: #define lcd_clear_column() lcd_byte_write(17)
44: #define lcd_tab_set(tab) if ((tab >= 1) & (tab <= 10)) lcd_2byte_write(18,tab)
45: #define lcd_backlight_on() lcd_byte_write(19)
46: #define lcd_backlight_off() lcd_byte_write(20)
47: #define lcd_char_write(ascii) lcd_byte_write(ascii)
48: #define lcd_clear_screen() lcd_byte_write(12)
49:
50: static TWI_MASTER_INFO_t *ptwiLCD;
51: unsigned char LCD_ender;
52: unsigned long int *LCD_inst;
53: unsigned char TIME_OUT_TECLADO;
54: unsigned char display_buffer[81];
55:
56: //_____________________________________________________________________________________________________________
57:
58: void lcd03_twi_init(unsigned char addr,unsigned long int *rel,unsigned char tot)
59: {
60:     //Initializes LCD03
61:     //addr - address
62:     //rel - pointer to the relogio variable (in milliseconds)
63:     //tot - keyboard time (in milliseconds)
64:
65:     extern TWI_MASTER_INFO_t twic_master;
66:
67:     ptwiLCD = &twic_master;
68:     LCD_ender = addr;
69:     LCD_inst = rel;
70:     TIME_OUT_TECLADO = tot;
71: }
72:
73: //_____________________________________________________________________________________________________________
74:
75: void lcd_byte_write(unsigned char data)
76: {
77:     //writes 1 byte to LCD03 (adress LCD_ender) register 0
78:     //data - Byte to write
79:
80:     unsigned char w[2];
81:
82:     w[0] = 0;
83:     w[1] = data;
84:     twi_master_trans(ptwiLCD,LCD_ender>>1,w,2,NULL,0);
```

```
 85: }
 86:
 87: //_____________________________________________________________________________________________________________________
 88:
 89: void lcd_2byte_write(unsigned char data1, unsigned char data2)
 90: {
 91:     //writes 2 bytes to LCD03 (adress LCD_ender) register 0
 92:     //data1 and data2 - bytes to write
 93:
 94:     unsigned char w[3];
 95:
 96:     w[0] = 0;
 97:     w[1] = data1;
 98:     w[2] = data2;
 99:     twi_master_trans(ptwiLCD,LCD_ender>>1,w,3,NULL,0);
100: }
101:
102: //_____________________________________________________________________________________________________________________
103:
104: void lcd_3byte_write(unsigned char data1, unsigned char data2, unsigned char data3)
105: {
106:     //writes 3 bytes to LCD03 (adress LCD_ender) register 0
107:     //data1, data2 e data3 - bytes to write
108:
109:     unsigned char w[4];
110:
111:     w[0] = 0;
112:     w[1] = data1;
113:     w[2] = data2;
114:     w[3] = data3;
115:     twi_master_trans(ptwiLCD,LCD_ender>>1,w,4,NULL,0);
116: }
117:
118: //_____________________________________________________________________________________________________________________
119:
120: unsigned char lcd_register_read(unsigned char reg)
121: {
122:     //Reads the LCD03 reg register
123:
124:     unsigned char data,LCD_reg;
125:
126:     LCD_reg = reg;
```

```
127:     twi_master_trans(ptwiLCD,LCD_ender>>1,&LCD_reg,1,&data,1);
128:     return data;
129: }
130:
131: //_____________________________________________________________________________________________________________
132:
133: void lcd_char_string_write(char *str)
134: {
135:     //Sends the str string to the LCD03
136:
137:     unsigned char n;
138:     char * ptr;
139:
140:     n = 0;
141:     ptr = str + 1;
142:     while(*ptr != '\0') {n++;ptr++;}
143:     n++;
144:     twi_master_trans(ptwiLCD,LCD_ender>>1,str,n,NULL,0);
145: }
146:
147: //_____________________________________________________________________________________________________________
148:
149: void lcd_clear_line(unsigned char lin)
150: {
151:     //Cleans the lin line and places the cursor at the beginning of the line
152:
153:     unsigned char i;
154:
155:     lcd_set_cursor_l_c(lin,1);
156:     for(i=1;i<=20;i++) lcd_char_write(32);
157:     lcd_set_cursor_l_c(lin,1);
158: }
159:
160: //_____________________________________________________________________________________________________________
161:
162: char convert_keypad(unsigned char tipo, unsigned char car)
163: {
164:     //Converts the keypad bits to the corresponding characters
165:
166:     char baite;
167:
168:     baite = 0;      // Inválido
```

```
169:     switch (tipo)
170:     {
171:         case 1:     // É LSB
172:         {
173:             switch (car)
174:             {
175:                 case 1:     {baite = 49; break;}    // ASCII 1
176:                 case 2:     {baite = 50; break;}    // ASCII 2
177:                 case 4:     {baite = 51; break;}    // ASCII 3
178:                 case 8:     {baite = 52; break;}    // ASCII 4
179:                 case 16:    {baite = 53; break;}    // ASCII 5
180:                 case 32:    {baite = 54; break;}    // ASCII 6
181:                 case 64:    {baite = 55; break;}    // ASCII 7
182:                 case 128:   {baite = 56; break;}    // ASCII 8
183:                 default:    break;
184:             }
185:             break;
186:         }
187:
188:         case 2  :   // É MSB
189:         {
190:             switch (car)
191:             {
192:                 case 1:     {baite = 57; break;}    // ASCII 9
193:                 case 2:     {baite = 42; break;}    // ASCII *
194:                 case 4:     {baite = 48; break;}    // ASCII 0
195:                 case 8:     {baite = 35; break;}    // ASCII #
196:                 default:    break;
197:             }
198:             break;
199:         }
200:         default:    break;
201:     }
202:     return baite;
203: }
204:
205: //_____________________________________________________________________________________________________________________
206:
207: char read_display_keypad(unsigned char mostra)
208: {
209:     //Reads the keypad character, sends it to the LCD03 and returns it
210:     //mostra = 1 - send to the LCD03 the own characters
```

```
211:     //mostra = 0 - send to the LCD03 the characters '_'
212:
213:     unsigned char msb,lsb;
214:     char baite;
215:
216:     lsb = lcd_register_read(1);
217:     msb = lcd_register_read(2);
218:     if (lsb != 0)
219:     {
220:         baite = convert_keypad(1,lsb);
221:         if (mostra) lcd_char_write(baite);
222:         else lcd_char_write('_');
223:     }
224:     else if (msb != 0)
225:     {
226:         baite = convert_keypad(2,msb);
227:         if (mostra) lcd_char_write(baite);
228:         else lcd_char_write('_');
229:     }
230:     else baite = 0;
231:     return baite;
232: }
233:
234: //_____________________________________________________________________________________________________________________
235:
236: unsigned char calcula(char *cmp)
237: {
238:     //Calculates the command code value. If OK, returns the value. On the contrary returns 0
239:
240:     unsigned char i,j;
241:     unsigned char *ptr;
242:     unsigned int val,k;
243:
244:     ptr = cmp;
245:     k = 1;
246:     val = 0;
247:     for (i=0;i<4;i++)
248:     {
249:         if (*ptr == '*')
250:         {
251:             ptr--;
252:             for (j=1;j<=i;j++)
```

```
253:             {
254:                 val = val + (k * (*ptr - 48));
255:                 k = k * 10;
256:                 ptr--;
257:             }
258:             if (val <= 255) return (unsigned char)val;
259:             else return 0;
260:         }
261:         else ptr++;
262:     }
263:     return 0;
264: }
265:
266: //_____________________________________________________________________________________________________________________________
267:
268: char receive_keypad_cod(unsigned char t_o)
269: {
270:     //Receives the command code from keypad
271:
272:     unsigned char cod;
273:     unsigned char nc,baite;
274:     unsigned char campo[4];
275:     unsigned char *ptca;
276:     unsigned long int inst1;
277:
278:     if (t_o > 0) inst1 = *LCD_inst;
279:     campo[3] = '\0';
280:     ptca = campo;
281:     while(*ptca != '\0') *ptca++ = ' ';
282:     nc = 0;
283:     ptca = campo;
284:     ad_aeternum
285:     {
286:
287:         if (t_o > 0) if (delta_time(inst1) > t_o) return 255;
288:         baite = read_display_keypad(1);
289:         if (nc > 3)
290:         {
291:             delay_ms(200);
292:             return 0;
293:         }
294:         switch (baite)
```

```
295:         {
296:             case '#':
297:             {
298:                 if (nc > 0)
299:                 {
300:                     lcd_backspace();
301:                     lcd_backspace();
302:                     ptca--;
303:                     nc--;
304:                 }
305:                 else lcd_backspace();
306:                 break;
307:             }
308:             case 0:
309:                 break;
310:             case '*':
311:             {
312:                 *ptca = baite;
313:                 cod = calcula(campo);
314:                 delay_ms(200);
315:                 return cod;
316:             }
317:             default:
318:             {
319:                 *ptca = baite;
320:                 nc++;
321:                 ptca++;
322:                 break;
323:             }
324:         }
325:         delay_ms(200);
326:     }
327: }
328:
329: //_____________________________________________________________________________________________________________
330:
331: unsigned char receive_keypad_par(unsigned char t_o, unsigned char mostra, char *ptro, unsigned char nmax)
332: {
333:     //Receives a command parameter and returns its size
334:     //t_o - Maximum wait time in seconds
335:     //mostra = 1 - displays the real characters
336:     //mostra = 0 - displays '_' characters
```

```
337:     //ptro - pointer to the command area
338:     //nmax - maximum bytes number to receive
339:
340:     unsigned char nc,baite;
341:     unsigned long int inst1;
342:
343:     if (t_o > 0) inst1 = *LCD_inst;
344:     nc = 0;
345:     ad_aeternum
346:     {
347:         if (t_o > 0) if (delta_time(inst1) > t_o) return 255;
348:         baite = read_display_keypad(mostra);
349:         if (nc > nmax)
350:         {
351:             delay_ms(200);
352:             return 254;
353:         }
354:         switch (baite)
355:         {
356:             case '#':
357:             {
358:                 if (nc > 0)
359:                 {
360:                     lcd_backspace();
361:                     lcd_backspace();
362:                     ptro--;
363:                     nc--;
364:                 }
365:                 else lcd_backspace();
366:                 break;
367:             }
368:             case 0:
369:                 break;
370:             case '*':
371:             {
372:                 delay_ms(200);
373:                 return nc;
374:             }
375:             default:
376:             {
377:                 *ptro = baite;
378:                 nc++;
```

```
379:                 ptro++;
380:                 break;
381:             }
382:         }
383:         delay_ms(200);
384:     }
385: }
386:
387: //_____________________________________________________________________________________________________________________
388:
389: unsigned char receive_command(char *cmdo)
390: {
391:     //Prepares the LCD03 to receive the keyed command
392:
393:     unsigned char cc[] = " Command code:";
394:     unsigned char pc[] = " Parameter:";
395:     unsigned char inv[] = " Invalid command try again";
396:     unsigned char npar;
397:     unsigned char *ptr;
398:
399:     cc[0] = 0;
400:     pc[0] = 0;
401:     inv[0] = 0;
402:     lcd_clear_screen();
403:     lcd_cursor_home();
404:     lcd_char_string_write(cc);
405:     lcd_carriage_return();
406:     ptr = cmdo;
407:     ptr++;
408:     *ptr = receive_keypad_cod(TIME_OUT_TECLADO);
409:     if (*ptr == 0) return 0;
410:     if (*ptr == 255) return 255;
411:     lcd_carriage_return();
412:     lcd_char_string_write(pc);
413:     lcd_carriage_return();
414: volta:
415:     ptr = cmdo;
416:     ptr++;
417:     ptr++;
418:     while(*ptr != '\0') *ptr++ = ' ';
419:     ptr = cmdo;
420:     ptr++;
```

```
421:     ptr++;
422:     npar = receive_keypad_par(TIME_OUT_TECLADO,1,ptr,18);
423:     if (npar == 255) return 255;
424:     if (npar == 254)
425:     {
426:         lcd_clear_screen();
427:         lcd_cursor_home();
428:         lcd_hide_cursor();
429:         lcd_char_string_write(inv);
430:         delay_ms(2000);
431:         goto volta;
432:     }
433:     *cmdo = npar + 50;
434:     return 0;
435: }
436:
437: //_____________________________________________________________________________________________________________________
438:
439: char read_keypad(void)
440: {
441:     //Reads the keypad character
442:
443:     unsigned char msb,lsb;
444:     char baite;
445:
446:     lsb = lcd_register_read(1);
447:     msb = lcd_register_read(2);
448:     if (lsb != 0) baite = convert_keypad(1,lsb);
449:     else if (msb != 0) baite = convert_keypad(2,msb);
450:     else baite = 0;
451:     return baite;
452: }
453:
454: //_____________________________________________________________________________________________________________________
455:
456: void lcd_talk(char *fr1,char *fr2,char *fr3,unsigned char aguarda)
457: {
458:     //Displays a message
459:     //If aguarda>=1 => waits the '*' keying
460:     //If aguarda=1 => do not waits
461:
462:     lcd_hide_cursor();
```

```
463:     lcd_clear_screen();
464:     lcd_cursor_home();
465:     lcd_char_string_write(fr1);
466:     lcd_set_cursor_l_c(2,1);
467:     lcd_char_string_write(fr2);
468:     lcd_set_cursor_l_c(3,1);
469:     lcd_char_string_write(fr3);
470:     if (aguarda)
471:     {
472:         lcd_set_cursor_l_c(4,1);
473:         while(read_keypad() != '*') delay_ms(200);
474:         lcd_cursor_type(0);
475:     }
476:     delay_ms(200);
477: }
478:
479: //_____________________________________________________________________________________________________________
480:
481: void lcd_display123(char *fr1,char *fr2,char *fr3, unsigned char seg)
482: {
483:     //Displays lines 1,2,3
484:     //If seg > 0 => waits seg seconds and clears the display
485:
486:     lcd_clear_line(1);
487:     lcd_set_cursor_l_c(1,1);
488:     lcd_char_string_write(fr1);
489:     lcd_clear_line(2);
490:     lcd_set_cursor_l_c(2,1);
491:     lcd_char_string_write(fr2);
492:     lcd_clear_line(3);
493:     lcd_set_cursor_l_c(3,1);
494:     lcd_char_string_write(fr3);
495:     lcd_hide_cursor();
496:     if (seg)
497:     {
498:         delay_ms(1000*seg);
499:         lcd_clear_screen();
500:     }
501:     else delay_ms(200);
502: }
503:
504: //_____________________________________________________________________________________________________________
```

```
505:
506: void lcd_display1234(char *fr1,char *fr2,char *fr3,char *fr4, unsigned char seg)
507: {
508:     //Displays lines 1,2,3,4
509:     //If seg > 0 => waits seg seconds and clears the display
510:
511:     lcd_clear_line(1);
512:     lcd_set_cursor_l_c(1,1);
513:     lcd_char_string_write(fr1);
514:     lcd_clear_line(2);
515:     lcd_set_cursor_l_c(2,1);
516:     lcd_char_string_write(fr2);
517:     lcd_clear_line(3);
518:     lcd_set_cursor_l_c(3,1);
519:     lcd_char_string_write(fr3);
520:     lcd_clear_line(4);
521:     lcd_set_cursor_l_c(4,1);
522:     lcd_char_string_write(fr4);
523:     lcd_hide_cursor();
524:     if (seg)
525:     {
526:         delay_ms(1000*seg);
527:         lcd_clear_screen();
528:     }
529:     else delay_ms(200);
530: }
531:
532: //_____________________________________________________________________________________________________________________________
533:
534: void limpa_display_buffer(void)
535: {
536:     //Clears the display buffer
537:
538:     unsigned char i,n;
539:
540:     n = sizeof display_buffer;
541:     for (i=0;i<n;i++) display_buffer[i] = ' ';
542:     display_buffer[n-1] = '\0';
543: }
544:
545: //_____________________________________________________________________________________________________________________________
546:
```

```
547: void display_flash(unsigned char limpa, unsigned char linha, signed char center, flash unsigned char *str,
548:                    signed char seg)
549: {
550:     //Displays the str string from flash in the linha line
551:     //limpa=1 => clears the display before
552:     //limpa=0 => do not clears the display before
553:     //seg>=0 => displays the line for seg seconds
554:     //seg<0 => do not returns
555:     //center=0 => line aligned to the left
556:     //center=1 => line aligned to the center
557:     //center<0 => line aligned at the present cursor position
558:     //If remainder(linha/4) = 0 => line 4
559:     //If remainder(linha/4) > 0 => this is the line
560:
561:     unsigned char i,resto,n,k;
562:     const unsigned char quatro = 4;
563:     const unsigned char dois = 2;
564:
565:     if (limpa) lcd_clear_screen();
566:     n = strlenf(str) + 1;
567:     limpa_display_buffer();
568:     display_buffer[0] = 0;
569:     for (i=1;i<=n;i++)
570:     {
571:         display_buffer[i] = *str;
572:         str++;
573:     }
574:     if (center > 0)
575:     {
576:         k = (22 - n) / dois;
577:         resto = linha - (linha / quatro * quatro);
578:         if (resto == 0) linha = quatro;
579:         else linha = resto;
580:         lcd_clear_line(linha);
581:         lcd_set_cursor_l_c(linha,k);
582:     }
583:     else if (center < 0);
584:     else
585:     {
586:         k = 1;
587:         resto = linha - (linha / quatro * quatro);
588:         if (resto == 0) linha = quatro;
```

```
589:         else linha = resto;
590:         lcd_clear_line(linha);
591:         lcd_set_cursor_l_c(linha,k);
592:     }
593:     lcd_hide_cursor();
594:     lcd_char_string_write(display_buffer);
595:     if (seg == 0) return;
596:     if (seg > 0)
597:     {
598:         delay_ms(1000*seg);
599:         lcd_clear_screen();
600:     }
601:     else ad_aeternum;
602: }
603:
604: //_____________________________________________________________________________________________________________________
605:
606: void display(unsigned char limpa, unsigned char linha, signed char center, unsigned char *str, signed char seg)
607: {
608:     //Displays the str string in the linha line
609:     //limpa=1 => clears the display before
610:     //limpa=0 => do not clears the display before
611:     //seg>=0 => displays the line for seg seconds
612:     //seg<0 => do not returns
613:     //center=0 => line aligned to the left
614:     //center=1 => line aligned to the center
615:     //center<0 => line aligned at the present cursor position
616:     //If remainder(linha/4) = 0 => line 4
617:     //If remainder(linha/4) > 0 => this is the line
618:
619:     unsigned char i,resto,n,k;
620:     const unsigned char quatro = 4;
621:     const unsigned char dois = 2;
622:
623:     if (limpa) lcd_clear_screen();
624:     n = strlen(str) + 1;
625:     limpa_display_buffer();
626:     display_buffer[0] = 0;
627:     for (i=1;i<=n;i++)
628:     {
629:         display_buffer[i] = *str;
630:         str++;
```

```
631:     }
632:     if (center > 0)
633:     {
634:         k = (22 - n) / dois;
635:         resto = linha - (linha / quatro * quatro);
636:         if (resto == 0) linha = quatro;
637:         else linha = resto;
638:         lcd_clear_line(linha);
639:         lcd_set_cursor_l_c(linha,k);
640:     }
641:     else if (center < 0);
642:     else
643:     {
644:         k = 1;
645:         resto = linha - (linha / quatro * quatro);
646:         if (resto == 0) linha = quatro;
647:         else linha = resto;
648:         lcd_clear_line(linha);
649:         lcd_set_cursor_l_c(linha,k);
650:     }
651:     lcd_hide_cursor();
652:     lcd_char_string_write(display_buffer);
653:     if (seg == 0) return;
654:     if (seg > 0)
655:     {
656:         delay_ms(1000*seg);
657:         lcd_clear_screen();
658:     }
659:     else ad_aeternum;
660: }
661:
662: //*****************************************************************************************************************************
```