
Bodová zváračka riadená mikrokontrolérom ATmega88PA.

Nedávno mi padla do oka nabíjačka akumulátorov, ktorá podľa hrúbky prachu na skrinke bola už nejaký ten čas nepoužívaná. Nedalo mi to a nazrel som do jej útrob. Moje oči uvideli transformátor, chladiče, na nich usmerňovací mostík a tyristor. Hned som vedel, že by sa s tým dalo niečo urobiť a tak som ju jednoducho prerobil na bodovú zváračku.



Na napájanie ovládacieho obvodu som použil klasiku, transformátor, mostíkový usmerňovač, vyhľadzovací kondenzátor a LDO regulátor (stabilizátor). Kedže ovládací obvod má nízky odber, napätie na vstupe stabilizátora bolo vďaka kondenzátoru bez akéhokoľvek zvlnenia. Chybou pri oživovaní obvodu sa mi však podarilo upiecť mikrokontrolér, LCD displej a aj ďalšie komponenty..... nuž aj to sa stáva...



Regulačný (riadiaci) obvod v podstate riadi tyristor pripojený na výstup mostíkového usmerňovača za hlavným transformátorom (na obrázku hore veľké trafo vľavo). Regulátor umožňuje nastaviť dobu zvárania a veľkosť usmerneného prúdu, ktorý je dodávaný do miesta zvaru. Pre dodanie požadovaného množstva energie do zvaru, regulátor ovláda otváranie a zatváranie tyristora počas vopred stanoveného času v každej polperiode sieťového napäťa. Na časovanie sa využíva prechod sieťového napäťa nulou, ktorý detektuje analógový komparátor, pripojený na výstup druhého usmerňovacieho mostíka cez rezistor R1 (1MOhm). Zenerova dióda obmedzuje napätie na 4,8V, tak aby neprišlo k poškodeniu vstupu mikrokontroléra. Pri prechode zostupnej hrany napäťa nulou spustí komparátor čítač/časovač1. Prahová hodnota napäťa je nastaviteľná potenciometrom (zero set) nachádzajúcim sa na doske. K regulátoru som tak tiež pridal prúdový senzor, ktorý sa mi povaľoval na stole ešte od [projektu snímania prúdu z alternátora](#). Je to niečo naviac, ale umožňuje merať špičkovú hodnotu pretekajúceho prúdu a zobrazovať ju na LCD displeji.



Firmware mikrokontroléru ATmega88PA (pracujúceho na 8MHz) skompliovaný s pomocou AVR-GCC pod AVRStudio IDE nájdete vo výpise na konci článku, alebo na [homepage autora](#).

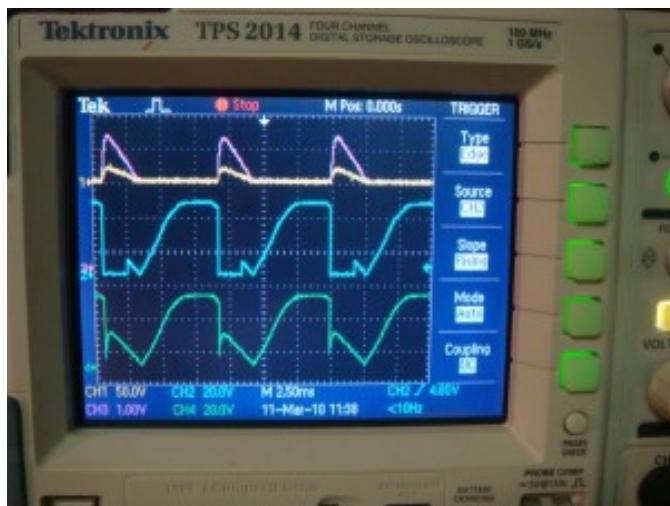
Mikrokontrolér zobrazuje na displeji informácie o veľkosti energie priviedenej do bodu zvaru, dĺžku trvania zvaru spolu s maximálnym prúdom tečúcim počas posledného uskutočneného zvaru a teplotou vo vnútri prístroja z interného senzora nachádzajúceho sa v mikrokontroléri. Mikrokontrolér taktiež zaobstaráva časovanie, detekciu prechodu napäťia nulou a riadenie tyristora. Riadiaca elektróda tyristora je ovládaná cez PMOS tranzistor Q1, ktorý je riadený jedným z pinov mikrokontroléra, nie však priamo, ale cez NPN tranzistor T2 (vid. schéma).

Zváranie sa spúšta pomocou nožného spínača pripojeného na ďalší vstup mikrokontroléra. Oba vstupné signály, signál pochádzajúci z nožného spínača, aj detekcia prechodu sieťového napäťia nulou sú vzorkované a synchronizované rutinou vo firmvéri tak, aby nedochádzalo k chybám pri spúštaní. Systém taktiež kontroluje vstup nožného spínača pri zapnutí napájania prístroja ako aj po skončení zvárania (doby zvaru). Pokial je nožný spínač stlačený, tak systém čaká na jeho uvoľnenie a užívateľ je o tejto skutočnosti informovaný výpisom na displeji. Týmto sa zvyšuje bezpečnosť a odstraňuje sa náhodné spustenie zvárania (zákmity príp. iné možné rušenie). Doba zvaru je nastaviteľná približne od jednej po 60 period (1s) sieťového napäťia. Ovládanie privádzaného výkonu umožňuje, aby sa do miesta zvaru dostalo od 5% do 95% prúdu každej polperiody. Veľkosť napäťia na katóde tyristora je po predelení rezistorovým deličom 10:1 k dispozícii na pine PC2 a pin je chránený zenerovou diódou D2, aby neprišlo k jeho poškodeniu. Neskôr som zistil, že informáciu o veľkosti tohto napäťia nebudem potrebovať a tak sa vo firmware s pinom nepracuje.

Na výstup zváračky som zapojil ochranný výkonový rezistor s odpornom niekoľko desatín Ohmu (použil som uhlíkovú tyčinku, pretože pracujem v továrni na výrobu uhlíka a tým pádom som to mal zadarmo), ktorý obmedzuje výstupný prúd na úroveň, pri ktorej nedôjde k poškodeniu diód a tyristora. Zdroj je krátkodobo preťažovaný odberom prúdu 130A. Vyzerá to tak, že zdroj tieto krátke prúdové nárazy veľmi dobre zvláda.

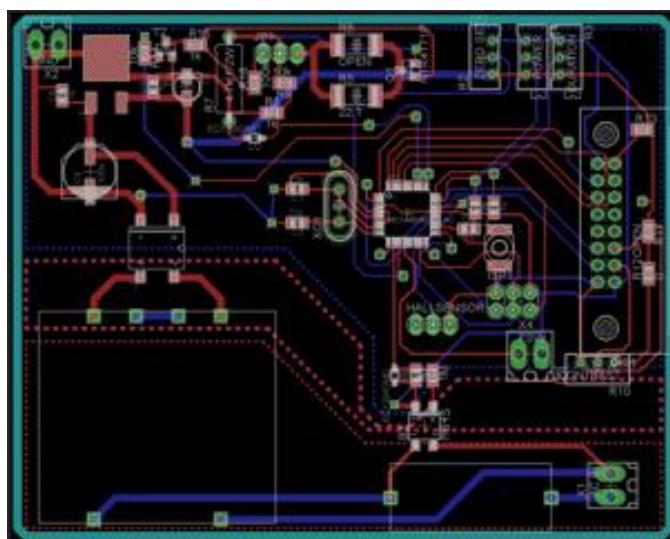
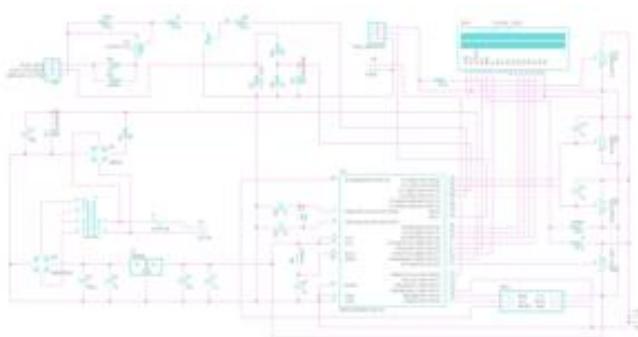
Poznámka: Ale 130A predsa len nie je dosť. Vymenil som teda vodiče na vstupe vedúce ku diódam a tyristoru za iné zo silnejším priezerom a potom som mohol zmenou hodnoty odporu rezistora (mojej uhlíkovej tyčky), alebo jej odstránením zväčšiť prúd, ktorý sa privádzza do bodu zvaru. Ukázalo sa, že aj prítlak elektródy na spoj je dôležitý, ale ešte stále zistújem ako ovplyvňuje proces zvárania.

Na nasledujúcich obrázkoch vidíme niekoľko časových priebehov, meraných na rôznych miestach zapojenia. Žltou farbou je zobrazené výstupné napätie, ktoré sa zdá byť malé, ale v skutočnosti tomu tak nie je (na danom kanále osciloskopu je nastavené vertikálne rozlíšenie 50V/dielik). Fialovou farbou je zobrazený priebeh výstupného prúdu meraného Hallovým snímačom, modrou je zobrazený priebeh napäťia na gate tranzistora, ktorým sa riadi tyristor a zelená je napätie na mostíku.



Tento projekt mi vnukol celkom zaujímavú myšlienku ako upraviť moju starú zváračku. Mám niekoľko veľkokapacitných kondenzátorov a IGBT tranzistorov zo starých budiacich jednotiek určených pre motory, z ktorých by sa dala urobiť celkom pekná [TIG zváračka](#). Myslím, že som čítal o možnosti zvárať prúdom obdĺžnikového priebehu pri frekvenciach 1 - 2 kHz. Ak si dobre spomínam, takýto priebeh prúdu z kladnou jednosmernou zložkou sa používa v niektorých typoch TIG zváračiek. Ktokolvek s informáciami o danej problematike mi kludne môže poslať mail, ja sa len potešíم.

Na posledných dvoch obrázkoch sa nachádzajú schéma zapojenia a návrh dosky plošných spojov.



A kód? S kódom je všetko v poriadku a je plne funkčný, akurát som nakoniec musel poprepájať niektoré veci, takže predtým ako sa pustíte do stavby si ešte raz skontrolujte, či sedia I/O piny.

```
1. #include // basic io header
2. #include // time header
3. #include // Interrupt header
4. //#include // can't get the watchdog to work yet
5.
6. #define F_CPU 8000000; //8MHz
7.
8. //function declarations
9. void lcd_write_byte(unsigned char CONTROL, unsigned char DATA);
10. void initLCD(void);
11. void updateLCD(void);
12. void switch_up(void);
13. void weld_message(void);
14.
15. //global variables
16. volatile unsigned int power, duration, temperature;
17. volatile unsigned char current, max_current;
18.
19. void main(void)
```

```
20.  
 {  
  
21.     unsigned int old_power, old_duration= 0; //variables for           comparison  
  
22.  
  
23.     PRR &= ~(1<<PRADC); //disable ADC           power reduction  
  
24.     ADMUX |= (1<<REFS0); //setup VCC as           reference  
  
25.     ADMUX |= (1<<ADLAR); //left adjust result       for 8 bit ADC  
  
26.     ADCSRA |= (1<<ADPS0)|(1<<ADPS1)|(1<<ADPS2); //prescaler clk/64 125kHz @ 8MHz clock  
  
27.     ADCSRA |= (1<<ADEN); //enable ADC  
  
28.  
  
29.     TCCR1B |= (1<<CS10); //enable timer1, no           prescale (clk/1)  
  
30.  
  
31.     ACSR |= (1<<ACIS1); //enable analog           comparator falling edge interrupt  
  
32.     ACSR |= (1<<ACIE); //enable analog           comparator interrupt  
  
33.  
  
34.     DDRC |= (1<<PORTC3); //SCR gate           drive  
  
35.     DDRC &= ~((1<<PORTC0)|(1<<PORTC1)|(1<<PORTC2)); //set as input  
  
36.     PORTC &= ~((1<<PORTC0)|(1<<PORTC1)|(1<<PORTC2)); //PUD  
  
37.  
  
38.     DDRD |= (1<<PORTD0)|(1<<PORTD1)|(1<<PORTD2)|(1<<PORTD3)|(1<<PORTD4)|(1<<PORTD5);  
  
39.     DDRD &= ~((1<<PORTD6)|(1<<PORTD7)); //set as input  
  
40.
```

```
PORTD &= ~((1<<PORTD6)|(1<<PORTD7)); //PUD  
41.  
  
42. DDRB &= ~(1<<PORTB0); //footswitch      input  
43. PORTB &= ~(1<<PORTB0); //PUD  
44.  
  
45. initLCD(); //set      up LCD  
46.  
  
47. switch_up(); //check footswitch  
48.  
  
49. max_current= 0x7F;  
50. updateLCD(); //display settings  
51.  
  
52. //wdt_reset(); //reset the watchdog timer  
53. // WDTCSR |= (1<<  
54. // WDTCSR |= (1<<<  
55.  
  
56.  
  
57. while(1)  
58. {  
59.   old_power= power; //set up the comparison value      before getting new value  
60.   ADMUX &= ~((1<<MUX0)|(1<<MUX1)|(1<<MUX2)|(1<<MUX3)); //sample ADC0 (power      setting)
```

```
61.    ADCSRA |= (1<<ADSC); //start           conversion
62.    while(ADCSRA &amp;      (1<<ADSC)){ } //wait for conversion
63.    power= ADCH;
64.    power*= 234; //scale power for use later: ((2^8)*234)= 59904 max value
65.    //59904 clk cycles= (1/8000000)*59904= 7.5ms
66.    //7.5ms= (.0075/(1/120))*100= 90% of one half wave (60Hz)
67.    //with these values power can be up to 90% of each half wave,
68.    //which with the 5% coded into the ISR, yields a range of 5%-95%.
69.
70.    old_duration= duration; //set up the comparison           value before getting new value
71.    ADMUX |= (1<<MUX0); //sample ADC1           (duration setting)
72.    ADCSRA |= (1<<ADSC); //start           conversion
73.    while(ADCSRA &amp;      (1<<ADSC)){ } //wait for conversion
74.    duration= ADCH;
75.    duration= (duration>>1); //convert to 7-bit number,           limits duration to ~1 second
76.    if(duration<=0x0007) duration= 0x0000;
77.    else duration-= 0x0007; //subtract 7 so duration can't exceed 3 digits (999ms)
78.
79.    ADMUX &= ~((1<<MUX0)|(1<<MUX1)|(1<<MUX2)); //set up for ADC8 (temp)
80.    ADMUX |= (1<<MUX3);
```

```
81. ADMUX |= (1<<REFS0)|(1<<REFS1); //1.1V ADC reference
82. ADMUX &= ~(1<<ADLAR); //undo left adjust           result for 8 bit ADC
83. ADCSRA |= (1<<ADSC); //start                 conversion
84. while(ADCSRA & (1<<ADSC)){} //wait for conversion
85. temperature= ADC;
86. temperature/= 12; //scale temp value to degrees C
87. ADMUX |= (1<<ADLAR); //restore left           adjust result
88. ADMUX &= ~(1<<REFS1); //restore VCC          ADC reference
89.
90. if(power!=old_power)      updateLCD(); //update      LCD if values changed
91. else if(duration!=old_duration) updateLCD();
92. if((!(PINB & (1<<PINB0))) & (duration>0)) //check for footswitch input
93. {
94.     char j= 0;
95.
96.     for(j= 0;j< 3;) //three sample noise filter
97.     {
98.         if(!(PINB & (1<<PINB0))) j++; //increment loop value if PORTC2 (fsw) is low
99.         else j= 4;                  //break out of the loop if high
100.
101.
```

```
102.    if(j==3) //should only get here if we got three low samples
103. {
104.     weld_message();
105.     ADMUX |= (1<<MUX0)|(1<<MUX1)|(1<<MUX2); //sample ADC7 (current sensor)
106.     ADMUX &= ~(1<<MUX3);
107.     ADCSRA |= (1<<ADATE); //ADC free running      mode
108.     ADCSRA |= (1<<ADSC); //start      conversion
109.     _delay_us(5); //wait for ADC's first reading
110.     max_current= 0x0000; //reset max current from last cycle
111.     sei();
112.     while((!(PINB & (1<<PINB0))) & (duration>0)){} //wait for zero cross
113.     cli();
114.     ADCSRA &= ~(1<<ADATE); //turn off      free running
115.     switch_up(); //wait for footswitch release
116.     updateLCD();
117. }
118. }
119. }
120. }
121.
```

```
122.  
  
123. ISR (ANALOG_COMP_vect)  
  
124. {  
  
125. char i= 0;  
  
126.  
  
127. for(i= 0;i< 3;) //three sample noise filter  
  
128. {  
  
129. if(!(ACSR && (1<<ACO))) i++; //increment loop value if ACO is low  
  
130. else i= 4; //break out of the loop if high  
  
131. }  
  
132.  
  
133. if(i==3) //should only get here if we got three low samples,  
  
134. { //which indicates a zero crossing.  
  
135. TCNT1= 0;  
  
136. while(TCNT1< power){} //wait for phase rotation  
  
137. PORTC |= (1<<PORTC3); //fire SCR  
  
138. while(TCNT1< 63333) //wait for 7.9ms, 95% of one half cycle (60Hz)  
  
139. {  
  
140. current= ADCH;  
  
141. if(current>max_current) max_current= current; //record the highest value
```

```
142.  
}  
  
143.    PORTC &= ~(1<<PORTC3); //turn off           SCR gate  
  
144.    duration-;  
  
145.}  
  
146.}  
  
147.  
  
148.  
  
149.  
  
150. void initLCD(void)  
  
151. {  
  
152.    _delay_ms(250);           // Wait for HD44780  
  
153.    PORTD &= ~(1<<PORTD4);  
  
154.    PORTD |= (1<<PORTD1);  
  
155.    PORTD |= (1<<PORTD0);  
  
156.    PORTD |= (1<<PORTD5); // function          set  
  
157.    _delay_ms(2);  
  
158.    PORTD &= ~(1<<PORTD5);  
  
159.    _delay_ms(20);  
  
160.    PORTD |= (1<<PORTD5); // function          set  
  
161.    _delay_ms(2);  
  
162.
```

```
PORTD &= ~(1<<PORTD5);  
  
163. _delay_ms(10);  
  
164. PORTD |= (1<<PORTD5); // function           set  
  
165. _delay_ms(2);  
  
166. PORTD &= ~(1<<PORTD5);  
  
167. _delay_ms(10);  
  
168. PORTD &= ~(1<<PORTD0);  
  
169. PORTD |= (1<<PORTD5); // initialize to 4      bit  
  
170. _delay_ms(2);  
  
171. PORTD &= ~(1<<PORTD5);  
  
172. _delay_ms(10);  
  
173.  
  
174. lcd_write_byte(0,0x28); //set interface width, #          of lines, and font size  
  
175. lcd_write_byte(0,0x0C); //display on  
  
176. lcd_write_byte(0,0x01); //clear display  
  
177. lcd_write_byte(0,0x06); //increment address by           one, shift cursor at write  
  
178. }  
  
179.  
  
180.  
  
181.  
  
182. void lcd_write_byte(unsigned char           CONTROL, unsigned char DATA)
```

```
183.  
 {  
  
184.    if(CONTROL == 1) PORTD |= (1<<PORTD4); else PORTD &= ~(1<<PORTD4);  
  
185.    if((DATA & 0x80) ==      0x80) PORTD |= (1<<PORTD3); else PORTD &= ~(1<<PORTD3);  
  
186.    if((DATA & 0x40) ==      0x40) PORTD |= (1<<PORTD2); else PORTD &= ~(1<<PORTD2);  
  
187.    if((DATA & 0x20) ==      0x20) PORTD |= (1<<PORTD1); else PORTD &= ~(1<<PORTD1);  
  
188.    if((DATA & 0x10) ==      0x10) PORTD |= (1<<PORTD0); else PORTB &= ~(1<<PORTD0);  
  
189.    PORTD |= (1<<PORTD5);  
  
190.    _delay_ms(1);  
  
191.    PORTD &= ~(1<<PORTD5);  
  
192.  
  
193.    if((DATA & 0x08) ==      0x08) PORTD |= (1<<PORTD3); else PORTD &= ~(1<<PORTD3);  
  
194.    if((DATA & 0x04) ==      0x04) PORTD |= (1<<PORTD2); else PORTD &= ~(1<<PORTD2);  
  
195.    if((DATA & 0x02) ==      0x02) PORTD |= (1<<PORTD1); else PORTD &= ~(1<<PORTD1);  
  
196.    if((DATA & 0x01) ==      0x01) PORTD |= (1<<PORTD0); else PORTD &= ~(1<<PORTD0);  
  
197.    PORTD |= (1<<PORTD5);  
  
198.    _delay_ms(2);  
  
199.    PORTD &= ~(1<<PORTD5);  
  
200.    _delay_ms(10);  
  
201. }  
  
202.
```

203.

204.

205.

```
void updateLCD(void)
```

206.

{

207.

```
unsigned int temp_duration, temp_power, temp_max_current;
```

208.

209.

```
temp_duration= (duration*8); //scale duration for BCD           conversion to milliseconds      //
```

210.

```
unsigned int duration_BCD= (((temp_duration/10)+((temp_duration/100)*6))*16)+(temp_duration%10));
```

211.

212.

```
unsigned char ONES= 0x00;
```

213.

```
unsigned char TENS= 0x00;
```

214.

```
unsigned char HUND= 0x00;
```

215.

216.

```
if((duration_BCD & 0x0800) == 0x0800)           HUND |= 0x08; else HUND &= ~0x08;
```

217.

```
if((duration_BCD & 0x0400) == 0x0400)           HUND |= 0x04; else HUND &= ~0x04;
```

218.

```
if((duration_BCD & 0x0200) == 0x0200)           HUND |= 0x02; else HUND &= ~0x02;
```

219.

```
if((duration_BCD & 0x0100) == 0x0100)           HUND |= 0x01; else HUND &= ~0x01;
```

220.

221.

```
if((duration_BCD & 0x0080) == 0x0080)           TENS |= 0x08; else TENS &= ~0x08;
```

222.

```
if((duration_BCD & 0x0040) == 0x0040)           TENS |= 0x04; else TENS &= ~0x04;
```

223.

```
if((duration_BCD & 0x0020) == 0x0020)           TENS |= 0x02; else TENS &= ~0x02;  
224.  
if((duration_BCD & 0x0010) == 0x0010)           TENS |= 0x01; else TENS &= ~0x01;  
225.  
  
226. if((duration_BCD & 0x0008) == 0x0008)           ONES |= 0x08; else ONES &= ~0x08;  
227. if((duration_BCD & 0x0004) == 0x0004)           ONES |= 0x04; else ONES &= ~0x04;  
228. if((duration_BCD & 0x0002) == 0x0002)           ONES |= 0x02; else ONES &= ~0x02;  
229. if((duration_BCD & 0x0001) == 0x0001)           ONES |= 0x01; else ONES &= ~0x01;  
230.  
  
231. ONES |= 0x30;  
232. TENS |= 0x30;  
233. HUND |= 0x30;  
234.  
  
235. lcd_write_byte(0x00,          0x01); //clear screen  
236. lcd_write_byte(0x01,          0x20); //space  
237. lcd_write_byte(0x01,          0x20); //space  
238. lcd_write_byte(0x01,          0x20); //space  
239. lcd_write_byte(0x01,          HUND);  
240. lcd_write_byte(0x01,          TENS);  
241. lcd_write_byte(0x01,          ONES);  
242. lcd_write_byte(0x01,          0x20); //space  
243. lcd_write_byte(0x01,          0x6D); //'m'
```

```
244. lcd_write_byte(0x01,          0x69); //'i'  
245. lcd_write_byte(0x01,          0x6C); //'l'  
246. lcd_write_byte(0x01,          0x6C); //'l'  
247. lcd_write_byte(0x01,          0x69); //'i'  
248. lcd_write_byte(0x01,          0x73); //'s'  
249. lcd_write_byte(0x01,          0x65); //'e'  
250. lcd_write_byte(0x01,          0x63); //'c'  
251. lcd_write_byte(0x01,          0x6F); //'o'  
252. lcd_write_byte(0x01,          0x6E); //'n'  
253. lcd_write_byte(0x01,          0x64); //'d'  
254. lcd_write_byte(0x01,          0x73); //'s'  
255. temp_power= (power/665); //scale power for BCD           conversion to percent  
256. temp_power=(95-temp_power);  
257. unsigned int power_BCD= ((temp_power/10)*16)+(temp_power%10);  
258.  
  
259. ONES= 0x00;  
260. TENS= 0x00;  
261.  
  
262. if((power_BCD & 0x0080) == 0x0080)      TENS |= 0x08; else TENS &= ~0x08;  
263. if((power_BCD & 0x0040) == 0x0040)      TENS |= 0x04; else TENS &= ~0x04;
```

```
264. if((power_BCD & 0x0020) == 0x0020) TENS |= 0x02; else TENS &= ~0x02;  
265. if((power_BCD & 0x0010) == 0x0010) TENS |= 0x01; else TENS &= ~0x01;  
266.  
267. if((power_BCD & 0x0008) == 0x0008) ONES |= 0x08; else ONES &= ~0x08;  
268. if((power_BCD & 0x0004) == 0x0004) ONES |= 0x04; else ONES &= ~0x04;  
269. if((power_BCD & 0x0002) == 0x0002) ONES |= 0x02; else ONES &= ~0x02;  
270. if((power_BCD & 0x0001) == 0x0001) ONES |= 0x01; else ONES &= ~0x01;  
271.  
272. ONES |= 0x30;  
273. TENS |= 0x30;  
274.  
275. lcd_write_byte(0x00, 0xC0); //go to second line  
276. lcd_write_byte(0x01, 0x20); //space  
277. lcd_write_byte(0x01, 0x20); //space  
278. lcd_write_byte(0x01, TENS);  
279. lcd_write_byte(0x01, ONES);  
280. lcd_write_byte(0x01, 0x25); //'%'  
281. lcd_write_byte(0x01, 0x20); //space  
282. lcd_write_byte(0x01, 0x6F); //'o'  
283. lcd_write_byte(0x01, 0x66); //'f'  
284.
```

```
1 lcd_write_byte(0x01,           0x20); //space
285.
3 lcd_write_byte(0x01,           0x70); //'p'
286.
4 lcd_write_byte(0x01,           0x68); //'h'
287.
5 lcd_write_byte(0x01,           0x61); //'a'
288.
6 lcd_write_byte(0x01,           0x73); //'s'
289.
7 lcd_write_byte(0x01,           0x65); //'e'
290.
8 temp_max_current= max_current;
291.
9 temp_max_current-= 0x7F;          //remove 2.5V sensor offset
292.
10 temp_max_current*= 3;           //scaling
293.
11 temp_max_current/= 2;           //scaling
294.
295.
12 unsigned int max_current_BCD= (((temp_max_current/10)+((temp_max_current/100)*6))*16)+(temp_max_current%10);
296.
297.
13 ONES= 0x00;
298.
14 TENS= 0x00;
299.
15 HUND= 0x00;
300.
301.
16 if((max_current_BCD & 0x0800) == 0x0800)           HUND |= 0x08; else HUND &= ~0x08;
302.
17 if((max_current_BCD & 0x0400) == 0x0400)           HUND |= 0x04; else HUND &= ~0x04;
303.
18 if((max_current_BCD & 0x0200) == 0x0200)           HUND |= 0x02; else HUND &= ~0x02;
304.
```

```
if((max_current_BCD & 0x0100) == 0x0100)           HUND |= 0x01; else HUND &= ~0x01;
305.

306. if((max_current_BCD & 0x0080) == 0x0080)           TENS |= 0x08; else TENS &= ~0x08;
307. if((max_current_BCD & 0x0040) == 0x0040)           TENS |= 0x04; else TENS &= ~0x04;
308. if((max_current_BCD & 0x0020) == 0x0020)           TENS |= 0x02; else TENS &= ~0x02;
309. if((max_current_BCD & 0x0010) == 0x0010)           TENS |= 0x01; else TENS &= ~0x01;
310.

311. if((max_current_BCD & 0x0008) == 0x0008)           ONES |= 0x08; else ONES &= ~0x08;
312. if((max_current_BCD & 0x0004) == 0x0004)           ONES |= 0x04; else ONES &= ~0x04;
313. if((max_current_BCD & 0x0002) == 0x0002)           ONES |= 0x02; else ONES &= ~0x02;
314. if((max_current_BCD & 0x0001) == 0x0001)           ONES |= 0x01; else ONES &= ~0x01;
315.

316. ONES |= 0x30;
317. TENS |= 0x30;
318. HUND |= 0x30;
319.

320. lcd_write_byte(0x00,          0x95); //go to third line (DDRAM address 0x15)
321. lcd_write_byte(0x01,          0x20); //space
322. lcd_write_byte(0x01,          HUND);
323. lcd_write_byte(0x01,          TENS);
324. lcd_write_byte(0x01,          ONES);
```

```
325.    lcd_write_byte(0x01,          0x20); //space
326.    lcd_write_byte(0x01,          0x61); //'a'
327.    lcd_write_byte(0x01,          0x6D); //'m'
328.    lcd_write_byte(0x01,          0x70); //'p'
329.    lcd_write_byte(0x01,          0x73); //'s'
330.    lcd_write_byte(0x01,          0x20); //space
331.    lcd_write_byte(0x01,          0x28); //('
332.    lcd_write_byte(0x01,          0x6D); //'m'
333.    lcd_write_byte(0x01,          0x61); //'a'
334.    lcd_write_byte(0x01,          0x78); //'x'
335.    lcd_write_byte(0x01,          0x29); //)'
336.
337.    unsigned int temperature_BCD= ((temperature/10)*16)+(temperature%10);
338.
339.    ONES= 0x00;
340.    TENS= 0x00;
341.
342.    if((temperature_BCD & 0x0080) == 0x0080)           TENS |= 0x08; else TENS &= ~0x08;
343.    if((temperature_BCD & 0x0040) == 0x0040)           TENS |= 0x04; else TENS &= ~0x04;
344.    if((temperature_BCD & 0x0020) == 0x0020)           TENS |= 0x02; else TENS &= ~0x02;
```

345.
if((temperature_BCD & 0x0010) == 0x0010) TENS |= 0x01; else TENS &= ~0x01;

346.

347.
if((temperature_BCD & 0x0008) == 0x0008) ONES |= 0x08; else ONES &= ~0x08;

348.
if((temperature_BCD & 0x0004) == 0x0004) ONES |= 0x04; else ONES &= ~0x04;

349.
if((temperature_BCD & 0x0002) == 0x0002) ONES |= 0x02; else ONES &= ~0x02;

350.
if((temperature_BCD & 0x0001) == 0x0001) ONES |= 0x01; else ONES &= ~0x01;

351.

352.
ONES |= 0x30;

353.
TENS |= 0x30;

354.

355.
lcd_write_byte(0x00, 0xD5); //go to fourth line (DDRAM address 0x55)

356.
lcd_write_byte(0x01, 0x20); //space

357.
lcd_write_byte(0x01, TENS);

358.
lcd_write_byte(0x01, ONES);

359.
lcd_write_byte(0x01, 0xDF); //degree symbol

360.
lcd_write_byte(0x01, 0x43); //'C'

361.
lcd_write_byte(0x01, 0x20); //space

362.
lcd_write_byte(0x01, 0x63); //'c'

363.
lcd_write_byte(0x01, 0x61); //'a'

364.
lcd_write_byte(0x01, 0x73); //'s'

365.

```
    lcd_write_byte(0x01,      0x65); //'e'  
366.  
    lcd_write_byte(0x01,      0x20); //space  
367.  
    lcd_write_byte(0x01,      0x74); //'t'  
368.  
    lcd_write_byte(0x01,      0x65); //'e'  
369.  
    lcd_write_byte(0x01,      0x6D); //'m'  
370.  
    lcd_write_byte(0x01,      0x70); //'p'  
371.  
}  
372.  
373.  
374.  
375. void switch_up(void)  
376. {  
377. if(!(PINB & (1<<PINB0)))  
378. {  
379.     lcd_write_byte(0x00,      0x01); //clear screen  
380.     lcd_write_byte(0x01,      0x20); //space  
381.     lcd_write_byte(0x01,      0x20); //space  
382.     lcd_write_byte(0x01,      0x72); //'r'  
383.     lcd_write_byte(0x01,      0x65); //'e'  
384.     lcd_write_byte(0x01,      0x6C); //'l'  
385.     lcd_write_byte(0x01,      0x65); //'e'
```

```
386.     lcd_write_byte(0x01,          0x61); //'a'  
387.     lcd_write_byte(0x01,          0x73); //'s'  
388.     lcd_write_byte(0x01,          0x65); //'e'  
389.     lcd_write_byte(0x01,          0x20); //space  
390.     lcd_write_byte(0x01,          0x66); //'f'  
391.     lcd_write_byte(0x01,          0x6F); //'o'  
392.     lcd_write_byte(0x01,          0x6F); //'o'  
393.     lcd_write_byte(0x01,          0x74); //'t'  
394.     lcd_write_byte(0x01,          0x73); //'s'  
395.     lcd_write_byte(0x01,          0x77); //'w'  
396.     lcd_write_byte(0x01,          0x69); //'i'  
397.     lcd_write_byte(0x01,          0x74); //'t'  
398.     lcd_write_byte(0x01,          0x63); //'c'  
399.     lcd_write_byte(0x01,          0x68); //'h'  
400.     while(!(PINB & (1<<PINB0))){};  
401. }  
402. }  
403.  
404.  
405.
```

```
406. void weld_message(void)
407. {
408.     lcd_write_byte(0x00,      0x01); //clear screen
409.     lcd_write_byte(0x01,      0x20); //space
410.     lcd_write_byte(0x01,      0x20); //space
411.     lcd_write_byte(0x01,      0x77); //'w'
412.     lcd_write_byte(0x01,      0x65); //'e'
413.     lcd_write_byte(0x01,      0x6C); //'l'
414.     lcd_write_byte(0x01,      0x64); //'d'
415.     lcd_write_byte(0x01,      0x69); //'i'
416.     lcd_write_byte(0x01,      0x6E); //'n'
417.     lcd_write_byte(0x01,      0x67); //'g'
418.     lcd_write_byte(0x01,      0x2E); //'.'
419.     lcd_write_byte(0x01,      0x2E); //'.'
420.     lcd_write_byte(0x01,      0x2E); //'.'
421. }
```

Zverejnené zo súhlasmom autora.

Homepage projektu: <http://www.imsolidstate.com/archives/590>

Preklad: [Kiwewicek](#)