ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

KATEDRA MIKROELEKTRONIKY

DIPLOMOVÁ PRÁCE

-

Příloha 1

Technické údaje a návod k použití řídicí desky pro klimatizovanou komoru

Technical data and manual for air-conditioned chamber control board

Vedoucí práce: doc. Dr. Ing. Jiří Hospodka

2018

Jan Pavlíček

Obsah

Úvod

Tento text je přílohou k diplomové práci na téma „Regulátor tepelné komory s tepelným čerpadlem“ a obsahuje celý řídicí program s vysvětlujícím textem včetně popisu řídicí desky.

Tento text by měl být nedílnou součástí zmíněné práce, jelikož obsahuje veškeré důležité informace týkající se reálného stavu vyvinutého řízení. Tímto se myslí zejména popis programu určeného k naprogramování řídicí desky pro účely budoucího vylepšení a dokončení tohoto programu.

1.Software

V této kapitole se nachází hlavní část programu, který byl částečně odzkoušen na řídicí desce. Ostatní části programu, resp. celý projekt, byl poskytnut firmě LG System spol. s r.o.

Program je určen pro procesor bývalé firmy Freescale – KV31P100M120.

Naprogramováno v MCUXpresso IDE verze 10.1.1_606.

Čísla řádků by měla odpovídat číslům uvedeným v IDE při programování.

Program obsahuje poměrně velké množství komentářů, které by měly dostatečně osvětlit jeho princip. Pro doplnění je zde tento program obohacen o další informace, pokud je to zapotřebí. Takové komentáře jsou psány fontem Times New Roman velikosti 12 (stejný font jako tento).

```
1. /*
2.  * Copyright (c) 2016, NXP Semiconductor, Inc.
3.  * All rights reserved.
4.  *
5.  * Redistribution and use in source and binary forms, with or
   without modification,
6.  * are permitted provided that the following conditions are
   met:
7.  *
8.  * o Redistributions of source code must retain the above
   copyright notice, this list
9.  *   of conditions and the following disclaimer.
10.    *
11.    * o Redistributions in binary form must reproduce the
   above copyright notice, this
12.    *   list of conditions and the following disclaimer in the
   documentation and/or
13.    *   other materials provided with the distribution.
14.    *
15.    * o Neither the name of NXP Semiconductor, Inc. nor the
   names of its
16.    *   contributors may be used to endorse or promote
   products derived from this
17.    *   software without specific prior written permission.
18.    *
19.    * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
   CONTRIBUTORS "AS IS" AND
20.    * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
   LIMITED TO, THE IMPLIED
21.    * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
   PARTICULAR PURPOSE ARE
22.    * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
   CONTRIBUTORS BE LIABLE FOR
23.    * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
   CONSEQUENTIAL DAMAGES
```

```
24.    * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
   SUBSTITUTE GOODS OR SERVICES;
25.    * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
   HOWEVER CAUSED AND ON
26.    * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
   LIABILITY, OR TORT
27.    * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
   OUT OF THE USE OF THIS
28.    * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
   DAMAGE.
29.    */
```

Začátek programu – standardní začlenění potřebných knihoven.

```
30.   /**
31.    * @file    Prvni_dipl.c
32.    * @brief   Application entry point.
33.    */
34.   #include <stdio.h>
35.   #include "board.h"
36.   #include "pin_mux.h"
37.   #include "clock_config.h"
38.   #include "MKV31F51212.h"
39.
40.   #include "stdlib.h"
41.   #include "string.h"
42.
43.   #include "fsl_gpio.h"
44.   #include "fsl_debug_console.h"
45.   #include "fsl_ftm.h"
46.   #include "fsl_adc16.h"
47.   #include "fsl_dac.h"
48.   #include "fsl_pdb.h"
49.   #include "fsl_common.h"
50.
51.   #include "pid.h"
52.
53.   /*
54.    * @brief   Application entry point.
55.    */
```

Zde jsou inicializovány konstanty pro LCD, PID regulátor a další účely. Vzhledem k tomu, že jedna z chybějících součástí programu je použití paměti FLASH pro uložení konstant mezi spuštěními, je doporučeníhodné některé konstanty v budoucnu inicializovat jako proměnné v paměti flash. Podobně je tomu i v případě některých proměnných v další sekci.

```
56.   /* Konstanty
57.    *
58.    *
59.    *
60.    * Ruzne definice pro program
61.    *
```

```
62.    *
63.    *
64.    * */
65.   /*
66.    * LCD konstanty
67.    * */
68.   //16x2 LCD line addresses
69.   #define LCD_Line1 0x00 // start of line 1
70.   #define LCD_Line2 0x40 // start of line 2
71.   // LCD Commands
72.   #define LCD_Clear 0b00000001 // replace all characters with
   ASCII 'space'
73.   #define LCD_Home 0b00000010 // return cursor to first
   position on first line
74.   #define LCD_EntryMode 0b00000110 // shift cursor from left
   to right on read/write
75.   #define LCD_DisplayOff 0b00001000 // turn display off
76.   #define LCD_DisplayOn 0b00001100 // display on, cursor off,
   don't blink character
77.   #define LCD_FunctionReset 0b00110000 // reset the LCD
78.   #define LCD_FunctionSet4bit 0x38 // 8-bit data, 2-line
   display, 5 x 7 font
79.   #define LCD_SetCursor 0b10000000 // set cursor position
80.
81.   // Kolik pruchodu programu zabezpeci stav tlacitka
82.   #define MAX_CNT_BUTTON 30u
83.   // Pocet pruchodu programu nez se nastavi vlajka stale
   stisknuteho tlacitka
84.   #define CONTINUOUS_BUTTON 25000u
85.
86.   /*! \brief P, I and D parameter values
87.    *
88.    * The K_P, K_I and K_D values (P, I and D gains)
89.    * need to be modified to adapt to the application at hand
90.    */
91.   //! \xrefitem todo "Todo" "Todo list"
92.   #define K_P     18.00
93.   //! \xrefitem todo "Todo" "Todo list"
94.   #define K_I     1.00
95.   //! \xrefitem todo "Todo" "Todo list"
96.   #define K_D     0.30
97.
98.
99.   /*
100.   * Nastaveni FTM
101.   *
102.   * Jako prvni nastaveni FTM0 pro zpozdeni pro LCD
103.   * */
104.  /* Get source clock for FTM driver */
105.  #define FTM_SOURCE_CLOCK (CLOCK_GetFreq(kCLOCK_BusClk)/4)
106.  /* The Flextimer FTM0 is used for LCD delays */
107.  #define DELAY_FTM_BASEADDR FTM0
108.  /* Interrupt number and interrupt handler for the FTM
   instance used */
109.  #define DELAY_FTM_IRQ_NUM FTM0_IRQn
110.  #define DELAY_FTM_HANDLER FTM0_IRQHandler
```

```
111.
112. /*Definice pro FTM1 - nacitani vteriny pro PID*/
113. #define PID_FTM_BASEADDR FTM1
114.
115. #define PID_FTM_IRQ_NUM FTM1_IRQn
116. #define PID_FTM_HANDLER FTM1_IRQHandler
117.
118. // Frekvence vydelena prescale 128
119. #define FTM_PID_SOURCE_CLOCK
   (CLOCK_GetFreq(kCLOCK_BusClk)/128)
120.
121. // Doplneni pro PWM - nastaveni FTM3
122. /* Interrupt number and interrupt handler for the FTM base
   address used */
123. #define FTM_INTERRUPT_NUMBER FTM3_IRQn
124. #define FTM_LED_HANDLER FTM3_IRQHandler
125.
126. /* Interrupt to enable and flag to read */
127. #define FTM_CHANNEL_INTERRUPT_ENABLE
   kFTM_Chnl7InterruptEnable
128. #define FTM_CHANNEL_FLAG kFTM_Chnl7Flag
129.
130. /* Get source clock for FTM driver */
131. #define FTM_SOURCE_CLOCK3 CLOCK_GetFreq(kCLOCK_BusClk)
132.
133. // ADC
134. #define DEMO_ADC16_CHANNEL_GROUP 0U
135. #define ADC_KONSTANTA 0.000805664
136. // Pocet pozadovanych hodnot prumerovani - slouzi spis jako
   zpozdeni pro nacitani aktualni hodnoty ADC, aby vysledky
   neprekmitavaly z jedne hodnoty na druhou
137. #define ADC_PRUMEROVANI 100
138.
139. // PDB Timer - pro citani 1 vteriny dle example
140. #define DEMO_PDB_BASE PDB0
141. #define DEMO_PDB_IRQ_ID PDB0_IRQn
142. #define DEMO_PDB_IRQ_HANDLER PDB0_IRQHandler
143.
144. // Konstanty pro klimatizaci NUTNO UPRAVIT HODNOTY DLE
   REALNE SOUSTAVY
145. // Max a Min hodnota teploty kompresoru
146. #define TKOMPRESOR_MIN 10
147. #define TKOMPRESOR_MAX 100
148. //Max a Min teplota vytlaku
149. #define TVYTLAK_MIN 10
150. #define TVYTLAK_MAX 100
151. //Max a Min teplota vyparniku
152. #define TVYPARNIK_MIN -30
153. #define TVYPARNIK_MAX 100
154.
```

Následují globální proměnné používané programem.

```
155. /*
156.  * Ruzne pomocne promenne.
```

```
157.   * */
158.
159. // PID regulator
160. //! Parameters for regulator
161. struct PID_DATA pidData;
162.
163. //Timer FTM0 pomocna promenna pricitano 1 kazdych 10us
164. volatile uint32_t oneUsCnt = 0;
165.
166. //Tlacitka - vzhledem k podstate programu nejsou
   obsluhovana pres preruseni
167. uint8_t button1 = 0;
168. uint8_t button2 = 0;
169. uint8_t button3 = 0;
170. uint8_t button4 = 0;
171. uint8_t button5 = 0;
172. uint8_t button6 = 0;
173. uint8_t button7 = 0;
174.
175. // UP opDone1
176. uint8_t buttonUPpressed = 0;
177. uint8_t buttonUPcontinuous = 0;
178. uint8_t buttonUPcontinuousHelper = 0;
179. // DOWN opDone2
180. uint8_t buttonDOWNpressed = 0;
181. uint8_t buttonDOWNcontinuous = 0;
182. uint8_t buttonDOWNcontinuousHelper = 0;
183. // RIGHT opDone3
184. uint8_t buttonRIGHTpressed = 0;
185. // LEFT opDone4
186. uint8_t buttonLEFTpressed = 0;
187. // ESC opDone5
188. uint8_t buttonESCpressed = 0;
189. // ENTER opDone6
190. uint8_t buttonENTERpressed = 0;
191. // MENU opDone7
192. uint8_t buttonMENUpressed = 0;
193.
194. uint32_t cntr1 = 0;
195. uint32_t cntr2 = 0;
196. uint32_t cntr3 = 0;
197. uint32_t cntr4 = 0;
198. uint32_t cntr5 = 0;
199. uint32_t cntr6 = 0;
200. uint32_t cntr7 = 0;
201.
202. //Pro jednu operaci pri stisknuti tlacitka, aby nedochazelo
   k opakovani
203. uint32_t opDone1 = 0;
204. uint32_t opDone2 = 0;
205. uint32_t opDone3 = 0;
206. uint32_t opDone4 = 0;
207. uint32_t opDone5 = 0;
208. uint32_t opDone6 = 0;
209. uint32_t opDone7 = 0;
210.
```

```
211. // ADC
212. // Inicializacni struktura pro prevody (je globalni kvuli
   funkci getADC)
213. adc16_channel_config_t adc16ChannelConfigStruct;
214. uint32_t aktualniKanal = 1;
215.
216. // 1 az 4 jsou diferencni kanaly
217. // Teplota kompresoru
218. uint32_t adcResultValue1 = 0;
219. // Teplota vytlaku
220. uint32_t adcResultValue2 = 0;
221. // Teplota Vypraniku
222. uint32_t adcResultValue3 = 0;
223. // Teplota v komore
224. uint32_t adcResultValue4 = 0;
225. // 5 az 8 jsou SE kanaly ADC
226. // Teplota v rozvadeci
227. uint32_t adcResultValue5 = 0;
228. // Zbytek kanalu je zatim nepouzit
229. uint32_t adcResultValue6 = 0;
230. uint32_t adcResultValue7 = 0;
231. uint32_t adcResultValue8 = 0;
232.
233. // Promenne pro prumerovani hodnot ADC
234. uint32_t adcResultValue1prumer = 0;
235. uint32_t adcResultValue2prumer = 0;
236. uint32_t adcResultValue3prumer = 0;
237. uint32_t adcResultValue4prumer = 0;
238. uint32_t adcResultValue5prumer = 0;
239. uint32_t adcResultValue6prumer = 0;
240. uint32_t adcResultValue7prumer = 0;
241. uint32_t adcResultValue8prumer = 0;
242. uint8_t pocetPrumeru = 0;
243.
244. uint32_t zpozdeni = 0;
245.
246. // MENU
247. uint8_t menu = 0;
248. uint8_t menuChange = 1;
249. uint8_t currentMenu = 1;
250. uint8_t previousMenu = 0;
251. uint8_t menuChanged = 1;
252.
253. // Teploty
254. float pozadovanaTeplota = 1.5;
255. float teplotaKompresoru = 20;
256. float teplotaVytlaku = 0;
257. float teplotaVyparniku = 0;
258. float teplotaKomory = 0;
259. float teplotaRozvadece = 0;
260.
261. // Pomocne promenne pro identifikaci chyb
262. signed int teplotaKompresoruNizka = 0;
263. signed int teplotaVytlakuVysoka = 0;
264. signed int teplotaVyparnikuNizka = 0;
265.
```

```
266. // Odecitani vterin
267. volatile uint8_t vterinaProKompresor = 0;
268. volatile uint8_t minutaProKompresor = 0;
269. volatile uint8_t vterinaProPWM = 0;
270. volatile uint8_t vterinaProTopeni = 0;
271.
272. // Pro interrupt FTM1 PID regulatoru
273. volatile uint8_t pidVterina = 0;
274.
275. // Vysledek z PID
276. float pidHodnota = 0;
277.
278. // PWM
279. volatile uint8_t updatedDutycycle = 0;
280. uint8_t oldDutycycle = 0;
281. // Potreba nastavit jestli je strida na log. 0 nebo 1
282. ftm_pwm_level_select_t pwmLevel = kFTM_LowTrue;
283.
284. // Pomocne promenne pro kompresor
285. uint8_t zapnoutKompresor = 0;
286. uint8_t vypnoutKompresor = 0;
287. uint8_t vypinaniKompresoruDokonceno = 0;
288. uint8_t prvniSpusteniKompresoru = 1;
289. uint8_t prvniSpusteniKompresoruDokonceno = 0;
290.
291. // Pomocne promenne pro topeni
292. uint8_t zapnoutTopeni = 0;
293.
294. // Promenne pro nacteni pozadovane doby v interruptu (cita
   jednotlive vteriny)
295. volatile uint8_t petMinutUplynulo = 0;
296. volatile uint8_t patnactMinutUplynulo = 0;
297. volatile uint8_t devetVterin = 0;
298. volatile uint8_t devetVterinUplynulo = 0;
299. volatile uint8_t vterinaProVypnuti = 0;
300. volatile uint8_t vterinaProDisplay = 0;
301.
302.
303. // LCD
304. // Kdyz je potreba ukazat hlasku na urcitou dobu, je
   potreba zmenit nacitatVterinaProDisplay na log 1, coz spusti
   citani vterinaProDisplay kazdou vterinu
305. // Po dosazeni potrebne doby je potreba nastavit
   menuChanged na log 1, tim se display prepise zpet (take je
   potreba zakazat tlacitka behem teto doby)
306. // po dokonceni citani vterinaProDisplay i
   nacitatVterinaProDisplay vynulovat
307. uint8_t nacitatVterinaProDisplay = 0;
308. uint8_t updateLCD = 0;
309.
310.
311.
312. // DAC
313. uint32_t pocatecniDacValue = 600;
314. uint32_t dacValue = 0;
315.
```

```
316.
317.
318. // Pouze pomocna promenna pro itoa
319. char cislo [10];
320.
321.
322. //Standardni delay
323. void delay(void) {
324.       volatile uint32_t i = 0;
325.       for (i = 0; i < 2000000; ++i) {
326.             __asm("NOP");
327.             /* delay */
328.       }
329. }
```

Několik následujících funkcí se týká provozu dvouřádkového LCD. Čekací časy jsou zajištěny pomocí časovače FTM. Přestože je odměřovaný čas zajištěn přes přerušení, čekání je zabezpečeno pomocí cyklu while(), což znamená, že kromě obsluhy ostatních přerušení nemůže hlavní program pokračovat. Dle základního testování nezpůsobuje toto řešení žádné problémy při běhu programu. Čekací časy jsou omezeny díky pomocným proměnným tak, že se informace na LCD aktualizuje pouze pokud je to nutné.

```
330. /*
331.  * Poslani bytu na LCD
332.  * */
333. void LCD_send(uint8_t theByte)
334. {
335.       GPIO_ClearPinsOutput(LCDDB7_GPIO, 1u <<
   LCDDB7_GPIO_PIN);                        // assume that data
   is '0'
336.       if (theByte & 1<<7) GPIO_SetPinsOutput(LCDDB7_GPIO,
   1u << LCDDB7_GPIO_PIN);     // make data = '1' if necessary
337.
338.       GPIO_ClearPinsOutput(LCDDB6_GPIO, 1u <<
   LCDDB6_GPIO_PIN);                        // repeat for each
   data bit
339.       if (theByte & 1<<6) GPIO_SetPinsOutput(LCDDB6_GPIO,
   1u << LCDDB6_GPIO_PIN);
340.
341.       GPIO_ClearPinsOutput(LCDDB5_GPIO, 1u <<
   LCDDB5_GPIO_PIN);                        // repeat for each
   data bit
342.       if (theByte & 1<<5) GPIO_SetPinsOutput(LCDDB5_GPIO,
   1u << LCDDB5_GPIO_PIN);
343.
344.       GPIO_ClearPinsOutput(LCDDB4_GPIO, 1u <<
   LCDDB4_GPIO_PIN);                        // repeat for each
   data bit
345.       if (theByte & 1<<4) GPIO_SetPinsOutput(LCDDB4_GPIO,
   1u << LCDDB4_GPIO_PIN);
346.
347.       GPIO_ClearPinsOutput(LCDDB3_GPIO, 1u <<
   LCDDB3_GPIO_PIN);                        // repeat for each
   data bit
```

```
348.       if (theByte & 1<<3) GPIO_SetPinsOutput(LCDDB3_GPIO,
   1u << LCDDB3_GPIO_PIN);
349.
350.       GPIO_ClearPinsOutput(LCDDB2_GPIO, 1u <<
   LCDDB2_GPIO_PIN);                          // repeat for each
   data bit
351.       if (theByte & 1<<2) GPIO_SetPinsOutput(LCDDB2_GPIO,
   1u << LCDDB2_GPIO_PIN);
352.
353.       GPIO_ClearPinsOutput(LCDDB1_GPIO, 1u <<
   LCDDB1_GPIO_PIN);                          // repeat for each
   data bit
354.       if (theByte & 1<<1) GPIO_SetPinsOutput(LCDDB1_GPIO,
   1u << LCDDB1_GPIO_PIN);
355.
356.       GPIO_ClearPinsOutput(LCDDB0_GPIO, 1u <<
   LCDDB0_GPIO_PIN);                          // repeat for each
   data bit
357.       if (theByte & 1<<0) GPIO_SetPinsOutput(LCDDB0_GPIO,
   1u << LCDDB0_GPIO_PIN);
358.
359.       FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
360.       // Write the data + small delay for the line to
   settle
361.       GPIO_SetPinsOutput(LCDE_GPIO, 1u << LCDE_GPIO_PIN);
   // Enable pin high
362.       oneUsCnt = 0;
     // Pro jistotu nulovani pomocne promenne
363.       //Cekani 1 * 10us = 10us kvuli odrazum
364.       while(!(oneUsCnt >= 1));
365.       GPIO_ClearPinsOutput(LCDE_GPIO, 1u << LCDE_GPIO_PIN);
   // Enable pin low
366.       oneUsCnt = 0;
     // Pro jistotu nulovani pomocne promenne
367.       //Cekani 1 * 10us = 10us kvuli odrazum
368.       while(!(oneUsCnt >= 1));
369.
370.       //Zastavime timer
371.       FTM_StopTimer(DELAY_FTM_BASEADDR);
372.       oneUsCnt = 0;
373.  }
374.
375.
376.  /*
377.   *    Send an instruction to LCD
378.  */
379.  void LCD_send_instruction(uint8_t theInstruction)
380.  {
381.       GPIO_ClearPinsOutput(LCDRS_GPIO, 1u <<
   LCDRS_GPIO_PIN);                // select the Instruction
   Register (RS low)
382.       GPIO_ClearPinsOutput(LCDE_GPIO, 1u << LCDE_GPIO_PIN);
   // make sure E is initially low
383.       LCD_send(theInstruction);                    // write
   the data
384.  }
```

```
385.
386.  /*
387.   *    Send a single character to LCD
388.  */
389.  void LCD_send_character(uint8_t theData)
390.  {
391.        GPIO_SetPinsOutput(LCDRS_GPIO, 1u << LCDRS_GPIO_PIN);
   // select the Data Register (RS high)
392.        GPIO_ClearPinsOutput(LCDE_GPIO, 1u << LCDE_GPIO_PIN);
   // make sure E is initially low
393.        LCD_send(theData);                              // write
   the data
394.  }
395.
396.  /*
397.   *    Send string to LCD
398.  */
399.  void LCD_write_string(char *theString){
400.
401.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
402.
403.        while (*theString)
404.        {
405.              LCD_send_character(*theString++);
406.
407.              FTM_StartTimer(DELAY_FTM_BASEADDR,
   kFTM_SystemClock);
408.              oneUsCnt = 0;
           // Pro jistotu nulovani pomocne promenne
409.              //Cekani 8 * 10us = 80us (minimalne 40us)
410.              while(!(oneUsCnt >= 8));
411.        }
412.        //Zastavime timer
413.        FTM_StopTimer(DELAY_FTM_BASEADDR);
414.        oneUsCnt = 0;
415.  }
416.
417.  /*
418.   *    Kurzor na displyi se posune na pozici
419.   *
420.   *    @param x je pozice v ose x (0 az 19 pro 20znakovy
   display)
421.   *    @param y je pozice v ose y (0 nebo 1 pro dvouradkovy
   LCD)
422.  */
423.  void LCD_gotoxy(uint8_t x, uint8_t y)
424.  {
425.        if ( y == 0 )
   LCD_send_instruction((0x80)+LCD_Line1+x);
426.        else LCD_send_instruction((0x80)+LCD_Line2+x);
427.  }
428.
429.  /* Vycisteni LCD - v podstate tento prikaz LCD_Clear zapise
   na kazde misto LCD mezeru
430.   * Zpozdeni pres 4 ms
431.   * */
```

```
432.  void LCD_clear()
433.  {
434.        // Clear Display instruction
435.        LCD_send_instruction(LCD_Clear);             // clear
   display RAM
436.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
437.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
438.        //Cekani 400 * 10us = 4ms
439.        while(!(oneUsCnt >= 400));
   // 1.64 mS delay (min)
440.  }
441.
442.  /* Inicializace LCD
443.   * Standardni procedura pouzivajici delay (zpozdeni
   zalozeno na FTM0)
444.   * */
445.  void LCDInit(void){
446.       FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
447.        //Cekani 10 000 * 10us = 100ms
448.        while(!(oneUsCnt >= 10000));
449.
450.        // Reset the LCD controller
451.        LCD_send(LCD_FunctionReset);                 // first
   part of reset sequence
452.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
453.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
454.        //Cekani 1 000 * 10us = 10ms
455.        while(!(oneUsCnt >= 1000));
456.
457.        LCD_send(LCD_FunctionReset);                 //
   second part of reset sequence
458.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
459.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
460.        //Cekani 20 * 10us = 200us
461.        while(!(oneUsCnt >= 20));
462.
463.        LCD_send(LCD_FunctionReset);                 // third
   part of reset sequence
464.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
465.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
466.        //Cekani 20 * 10us = 200us
467.        while(!(oneUsCnt >= 20));
468.
469.        // Set 8-bit mode
470.
471.        // Function Set instruction
472.        LCD_send_instruction(LCD_FunctionSet4bit);   // set
   mode, lines, and font
473.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
474.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
475.        //Cekani 8 * 10us = 80us (minimalne 40us)
```

```
476.        while(!(oneUsCnt >= 8));
477.
478.        // Datasheet initialization
479.
480.        // Display On/Off Control instruction
481.        LCD_send_instruction(LCD_DisplayOff);        // turn
   display OFF
482.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
483.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
484.        //Cekani 8 * 10us = 80us (minimalne 40us)
485.        while(!(oneUsCnt >= 8));
486.
487.        // Clear Display instruction
488.        LCD_send_instruction(LCD_Clear);             // clear
   display RAM
489.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
490.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
491.        //Cekani 400 * 10us = 4ms
492.        while(!(oneUsCnt >= 400));
   // 1.64 mS delay (min)
493.
494.        // Entry Mode Set instruction
495.        LCD_send_instruction(LCD_EntryMode);         // set
   desired shift characteristics
496.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
497.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
498.        //Cekani 8 * 10us = 80us (minimalne 40us)
499.        while(!(oneUsCnt >= 8));
500.
501.        // Turn the display back on
502.
503.        // Display On/Off Control instruction
504.        LCD_send_instruction(LCD_DisplayOn);         // turn
   the display ON
505.        FTM_StartTimer(DELAY_FTM_BASEADDR, kFTM_SystemClock);
506.        oneUsCnt = 0;
      // Pro jistotu nulovani pomocne promenne
507.        //Cekani 8 * 10us = 80us (minimalne 40us)
508.        while(!(oneUsCnt >= 8));
509.
510.        //Zastavime timer
511.        FTM_StopTimer(DELAY_FTM_BASEADDR);
512.        oneUsCnt = 0;
513.  }
```

Obsluha tlačítek není zajištěna přes přerušení. Pokud by to však v budoucnu bylo z jakéhokoliv důvodu nutné, neměl by být větší problém takové zpracování jejich signálu naprogramovat.

Doba ustalování signálu je stanovena dle reálného času průběhu programu. Konstanta `MAX_CNT_BUTTON` udává počet průchodů programu, než je logická úroveň tlačítka jednoznačně stanovena.

V současném stavu programu je naprogramováno přičítání a odečítání pouze jedné proměnné pomocí tlačítek (požadována teplota v komoře). Z tohoto důvodu není k tomuto účelu napsána vlastní funkce, nýbrž je tato funkce implementována pouze pomocí rozhodovacího bločku naprogramována přímo v menu 0. Kvůli rychlejší změně požadované hodnoty je rozeznáván také dlouhý stisk tlačítka. Počet průchodů programu „dlouhého stisku“ je stanoven pomocí konstanty `CONTINUOUS_BUTTON`.

```
514.
515.  /* Kontrola tlacitek
516.   * Nacita countery pro jednotliva tlacitka.
517.   * Pripadne tedy umoznuje stisk vice tlacitek najednou.
518.   * Jakmile dojde k nacteni daneho poctu pruchodu programu,
519.   * nastavi promennou pro dane tlacitko na log. 1.
520.   * Poukd je potreba pouze jeden stisk, je nutne nastavit
521.   * promennou opDonex na log. 1 (pri uvolneni tlacitka se
   automaticky vynuluje)
522.   * a zaroven buttonXXXpressed na log. 0.
523.   * */
524.  void tlacitka()
525.  {
526.        // Kontrola tlacitek na zacatek
527.        button1 = GPIO_ReadPinInput(BUTTONS_UP_GPIO,
   BUTTONS_UP_GPIO_PIN);
528.        button2 = GPIO_ReadPinInput(BUTTONS_DOWN_GPIO,
   BUTTONS_DOWN_GPIO_PIN);
529.        button3 = GPIO_ReadPinInput(BUTTONS_RIGHT_GPIO,
   BUTTONS_RIGHT_GPIO_PIN);
530.        button4 = GPIO_ReadPinInput(BUTTONS_LEFT_GPIO,
   BUTTONS_LEFT_GPIO_PIN);
531.        button5 = GPIO_ReadPinInput(BUTTONS_ESC_GPIO,
   BUTTONS_ESC_GPIO_PIN);
532.        button6 = GPIO_ReadPinInput(BUTTONS_ENTER_GPIO,
   BUTTONS_ENTER_GPIO_PIN);
533.        button7 = GPIO_ReadPinInput(BUTTONS_MENU_GPIO,
   BUTTONS_MENU_GPIO_PIN);
534.
535.        /* Nacitani cntr pri kontrole tlacitek.
536.         * Dobu nacitani je treba zmenit dle delky programu.
537.         * Kontrola tlacitek neni zajistena pres interrupt!!!
538.         * */
539.        if(button1 == 0)
540.        {
541.              if(cntr1 <= CONTINUOUS_BUTTON)
542.              {
543.                    cntr1++;
544.              }
545.              if(cntr1 == CONTINUOUS_BUTTON)
546.              {
547.                    buttonUPcontinuous = 1;
548.              }
549.        }
550.        else
551.        {
552.              cntr1 = 0;
```

```
553.            opDone1 = 0;
554.            buttonUPcontinuous = 0;
555.            buttonUPcontinuousHelper = 0;
556.       }
557.       if (button2 == 0)
558.       {
559.            if(cntr2 < CONTINUOUS_BUTTON)
560.            {
561.                 cntr2++;
562.            }
563.            if(cntr2 == CONTINUOUS_BUTTON)
564.            {
565.                 buttonDOWNcontinuous = 1;
566.            }
567.       } else
568.       {
569.            cntr2 = 0;
570.            opDone2 = 0;
571.            buttonDOWNcontinuous = 0;
572.            buttonDOWNcontinuousHelper = 0;
573.       }
574.       if (button3 == 0)
575.       {
576.            if(cntr3 < MAX_CNT_BUTTON)
577.            {
578.                 cntr3++;
579.            }
580.       } else
581.       {
582.            cntr3 = 0;
583.            opDone3 = 0;
584.       }
585.       if (button4 == 0)
586.       {
587.            if(cntr4 < MAX_CNT_BUTTON)
588.            {
589.                 cntr4++;
590.            }
591.       } else
592.       {
593.            cntr4 = 0;
594.            opDone4 = 0;
595.       }
596.       if (button5 == 0)
597.       {
598.            if(cntr5 < MAX_CNT_BUTTON)
599.            {
600.                 cntr5++;
601.            }
602.       } else
603.       {
604.            cntr5 = 0;
605.            opDone5 = 0;
606.       }
607.       if (button6 == 0)
608.       {
```

```
609.            if(cntr6 < MAX_CNT_BUTTON)
610.            {
611.                 cntr6++;
612.            }
613.       } else
614.       {
615.            cntr6 = 0;
616.            opDone6 = 0;
617.       }
618.       if (button7 == 0)
619.       {
620.            if(cntr7 < MAX_CNT_BUTTON)
621.            {
622.                 cntr7++;
623.            }
624.       } else
625.       {
626.            cntr7 = 0;
627.            opDone7 = 0;
628.       }
629.       //Konec nacitani tlacitek
630.       //Vyhodnoceni tlacitek
631.       if(((cntr1 == MAX_CNT_BUTTON) && (opDone1 == 0)))
632.       {
633.            buttonUPpressed = 1;
634.       }
635.       if(((cntr2 == MAX_CNT_BUTTON) && (opDone2 == 0)))
636.       {
637.            buttonDOWNpressed = 1;
638.       }
639.       if(((cntr3 == MAX_CNT_BUTTON) && (opDone3 == 0)))
640.       {
641.            buttonRIGHTpressed = 1;
642.       }
643.       if(((cntr4 == MAX_CNT_BUTTON) && (opDone4 == 0)))
644.       {
645.            buttonLEFTpressed = 1;
646.       }
647.       if(((cntr5 == MAX_CNT_BUTTON) && (opDone5 == 0)))
648.       {
649.            buttonESCpressed = 1;
650.       }
651.       if(((cntr6 == MAX_CNT_BUTTON) && (opDone6 == 0)))
652.       {
653.            buttonENTERpressed = 1 ;
654.       }
655.       if(((cntr7 == MAX_CNT_BUTTON) && (opDone7 == 0)))
656.       {
657.            buttonMENUpressed = 1;
658.       }
659.  }
660.
661.  /* Orezana verze ftoa() - z nejakeho duvodu je k dispozici
       itoa(), ale ftoa() ne
662.   * Funkce prevadi poskytnuty float na string s jednim
       desetinnym mistem (lze upravit pri nasobeni)
```

```
663.   *
664.   * @param flt float, ktery chceme prevest
665.   * @param str string, do ktereho ulozime nas prevedeny
   float
666.   * */
667.  void ftoa(float flt, char *str)
668.  {
669.        char string1 [10];
670.        char string2 [10];
671.        itoa(flt, str, 10);
672.
673.        strcat(str, ".");                        // Doplnime
   desetinnou tecku
674.        if((flt < 0) && (flt > -1))                        //
   Problem se zapisem -0.1 aĹľ -0.9 - doplneno znamenko
675.        {
676.              strcpy(string1, "-");
677.              strcat(string1, str);
678.              strcpy(str, string1);
679.        }
680.        uint32_t i = 0;
681.        if(flt >= 0)
682.        {
683.              i = ((flt - (int)flt) * 10);        // Odecteme
   celociselnou hodnotu a vynasobime dle pozadovaneho poctu
   desetinnych mist - pro jedno misto staci * 10
684.        }
685.        else i = (((int)flt - flt) * 10);
686.        itoa(i, string2, 10);                    // Desetinna
   mista prevedeme na string
687.        strcat(str, string2);                    // Sloucime
   stringy - ptredavany str je globalni, neni potreba return
688.  }
689.
690.  /*
691.   * Pomocne funkce pro kontrolu stavu klimatizace - tyto se
   kontroluji neustale v programu
692.   * a pokud nastane problem, nastavi se prislusna promenna,
   podle ktere se provedou prislusne kroky
693.   * a je vykazana chyba.
694.   * Nebyla k dispozici soustava pro vyzkouseni, takze kvuli
   debuggingu je zatim zakomentovano
695.   * */
696.  void kontrolaKompresoru()
697.  {
698.  //    if(teplotaKompresoru < TKOMPRESOR_MIN)
699.  //    {
700.  //          teplotaKompresoruNizka = 1;
701.  //          zapnoutKompresor = 0;
702.  //          vypnoutKompresor = 1;
703.  //          menu = 50;
704.  //    }
705.  //    else teplotaKompresoruNizka = 0;
706.  //
707.  //    if(teplotaVytlaku > TVYTLAK_MAX)
708.  //    {
```

```
709. //         teplotaVytlakuVysoka = 1;
710. //         zapnoutKompresor = 0;
711. //         vypnoutKompresor = 1;
712. //         menu = 50;
713. //    }
714. //    else teplotaVytlakuVysoka = 0;
715. //
716. //    if(teplotaVyparniku < TVYPARNIK_MIN)
717. //    {
718. //         teplotaVyparnikuNizka = 1;
719. //         zapnoutKompresor = 0;
720. //         vypnoutKompresor = 1;
721. //         menu = 50;
722. //    }
723. //    else teplotaVyparnikuNizka = 0;
724. }
```

Při ovládání klimatizace není prozatím ovládán čtyřcestný ventil – jeho funkci lze velmi jednoduše doplnit – viz. komentáře.

```
725. /*
726.  * Ovladani klimatizace - startovaci sekvence pro prvni
   spusteni
727.  *
728.  * */
729. void startKompresoru()
730. {
731.       // Mela by nastat kontrola teploty kompresoru - pokud
   je prilis nizka, musi se nejprve zapnout jeho ohrev
732.       // Nasleduje kontrola teploty vytlaku - pokud je
   prilis vysoka, je nutne zastavit proces - toto je nutne
   kontrolovat neustale
733.       // Kontrola teploty vyparniku - dle navodu pokud je
   prilis nizka, obratit chod atd.
734.       kontrolaKompresoru();
735.
736.       // Kontrola tlakĹž na vstupu a vĂ˝stupu? - Dle
   domluvy se nemeri digitalne, kompresor ma standardni
   mechanickou ochranu
737.
738.
739.       // Pokud vsechno projde, pokracovat
740.       if(zapnoutKompresor == 1)
741.       {
742.            // Vynulovani promenne pro odstavku
743.            vypinaniKompresoruDokonceno = 0;
744.            // Kontrola zmÄ›ny duty cycle z predchoziho
   prubehu
745.            // Nejprve zaznamename puvodni duty cycle
746.            oldDutycycle = updatedDutycycle;
747.
748.            if(prvniSpusteniKompresoru &&
   (prvniSpusteniKompresoruDokonceno == 0))
749.            {
750.             /* Update PWM duty cycle, start compressor
```

```
751.              * Pro prvni spusteni - 5 minut PWM na minimum,
   kompresor 15%
752.              * Nasleduje 15 minut, kdy se po rampe zvysuje
   vykon kompresoru az na 98% pri konstantnim otevreni (na 5%
   spise zavreni) ventilu
753.              * */
754.
755.             if(petMinutUplynulo == 0)
756.             {
757.                 // Otevrit ventil na 5% dle domluvy
758.                 updatedDutycycle = 5;
759.                 // Ovladaci napeti pro kompresor
760.                 DAC_Enable(DAC_DAC_10V_PERIPHERAL, true);
   /* Enable output. */
761.
   DAC_SetBufferReadPointer(DAC_DAC_10V_PERIPHERAL, 0U); /* Make
   sure the read pointer to the start. */
762.
   /*
763.
   * The buffer is not enabled, so the read pointer can not move
   automatically. However, the buffer's read pointer
764.
   * and items can be written manually by user.
765.
   */
766.                 // Na vystupu DAC je priblizne 15% napeti
   (ciselna hodnota 600) - tuto hodnotu nacteme do promenne
   dacValue
767.                 DAC_SetBufferValue(DAC_DAC_10V_PERIPHERAL,
   0U, pocatecniDacValue);
768.                 dacValue = pocatecniDacValue;
769.                 // Zapneme zelenou LED 2
770.                 GPIO_ClearPinsOutput(LED_GREEN_LED2_GPIO,
   1u << LED_GREEN_LED2_GPIO_PIN);
771.                 // Dokud jsme v prvnich peti minutach,
   nulujeme vlajku pro 9 vterin (neni to uplne nutne, protoze
   krok je pouze 35 tj. cca 28 mV a prijdeme pouze o jeden,
   maximalni hodnota je stejne osetrena proti presahu 4095))
772.                 devetVterin = 0;
773.             }
774.             else if(patnactMinutUplynulo == 0)   //Pokud
   uplynulo prvnich 5 minut, pokracuje se zvysovanim vykonu
   kompresoru (PWM pro skrtici ventil zustava z minula)
775.             {
776.                 if(devetVterinUplynulo)
777.                 {
778.                     devetVterinUplynulo = 0;
779.                     dacValue += 35;                 //
   Nyni se jiz dacValue zvysuje o krok 35 (28 mV)
780.                     // Pro jistotu osetreni presahu
   maximalni hodnoty DAC vystupu (teoreticky vyjde posledni
   cislo 4100, protoze 15 min * 60 s = 900
781.                     // Po vydeleni 9 s ziskame tedy 100
   hodnot (primereny skok cca 28 mV) * 35 = 3500 a pocatecni
   stav byl 600, tj. 3500 + 600 = 4100
```

```
782.                        if(dacValue >= 4095)
783.                        {
784.                             dacValue = 4095;
785.                        }
786.                    // Ovladaci napeti pro kompresor
787.                        DAC_Enable(DAC_DAC_10V_PERIPHERAL,
   true);             /* Enable output. */
788.
   DAC_SetBufferReadPointer(DAC_DAC_10V_PERIPHERAL, 0U); /* Make
   sure the read pointer to the start. */
789.
   /*
790.
   * The buffer is not enabled, so the read pointer can not move
   automatically. However, the buffer's read pointer
791.
   * and items can be written manually by user.
792.
   */
793.                        // Na vystupu DAC je priblizne 15%
   napeti (ciselna hodnota 600)
794.                    DAC_SetBufferValue(DAC_DAC_10V_PERIPHERAL,
   0U, dacValue);
795.                    // Zapneme zelenou LED 2
796.                    GPIO_ClearPinsOutput(LED_GREEN_LED2_GPIO,
   1u << LED_GREEN_LED2_GPIO_PIN);
797.                }
798.             }
799.
800.             // Pokud se vyse zmenil duty cycle oproti stavu
   pri prichodu do funkce, zmenime tuto hodnotu v registrech PWM
801.             if(updatedDutycycle != oldDutycycle)
802.             {
803.                 /* Disable channel output before updating
   the dutycycle */
804.
     FTM_UpdateChnlEdgeLevelSelect(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, 0U);
805.
806.                 /* Update PWM duty cycle */
807.
     FTM_UpdatePwmDutycycle(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, kFTM_CenterAlignedPwm,
   updatedDutycycle);
808.
809.                 /* Software trigger to update registers */
810.
     FTM_SetSoftwareTrigger(PERIF_OUT_O7_PERIPHERAL, true);
811.
812.                 /* Start channel output with updated
   dutycycle */
813.
     FTM_UpdateChnlEdgeLevelSelect(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, pwmLevel);
814.
```

```
815.                  // A zapnout timer ovladajici PWM - pri
   vypnuti timeru je vystup ve stavu vysoke impedance (dle
   mereni napeti na jeho pinu)
816.                  FTM_StartTimer(PERIF_OUT_O7_PERIPHERAL,
   kFTM_SystemClock);
817.              }
818.            }
819.       }
820. }
821.
822. /* Funkce pro regulaci PWM ventilu po prvnim nabehu
   kompresoru
823.  *
824.  * */
825. void regulaceKompresoru(float pid)
826. {
827.       // Pokud se jiz nejedna o prvni spusteni, je situace
   o neco jednodussi a je pouze potreba nastavit PWM pro ventil,
   pripadne pak dale postupne menit vykon kompresoru (pro
   zacatek by mel byt 98%)
828.       // Pomoci PWM lze nastavit teplotu vyparniku do
   urcite miry, nasledne je treba menit vykon kompresoru
829.       // Zde je tedy treba nastavit updatedDutycycle (aby
   pro prvni pruchod byla tato hodnota prenesena do registru -
   if(updatedDutycycle != oldDutycycle) nize)
830.       // Pripadne pomoci rozhodovaciho bloku pri urcite
   teplote jiz zahajit postupnou zmenu vykonu kompresoru
831.       // Prozatim neni zmena vykonu kompresoru
   naprogramovana - ridi se pouze PWM
832.
833.       // Ovladani ctyrcenstneho ventilu neni
   implementovano, nicmene vystup je na nej pripraven
834.       // Pravdepodobne bude vystup ovladat rele, proto
   doporucuji na vystup desky pripojit MOSFET s ochrannymi prvky
   a tim ovladat toto rele
835.       // Vystup lze zapnout pomoci
   GPIO_SetPinsOutput(PERIF_OUT_O3_GPIO, 1u <<
   PERIF_OUT_O3_GPIO_PIN);
836.       // Vypnout lze pak
   GPIO_ClearPinsOutput(PERIF_OUT_O3_GPIO, 1u <<
   PERIF_OUT_O3_GPIO_PIN);
837.       // Zapnuti prip. vypnuti zavisi na pozadovane funkci
   soustavy - klimatizace umi prepnutim topit nebo chladit -
   nebylo specifikovano pri jake urovni je ventil
838.       // pripraven na chlazeni resp. topeni - timto
   zpusobem je mozne provadet odmrazovani vyparniku, pokud je
   treba
839.       // Synchronne dle pozadavku je mozne ovladat
   ventilator na vyparniku (vystup O4 - zatim neni ovladan (O5
   je povazovan za ventilator v komore))
840.
841.       // Pokud vsechno projde, pokracovat
842.       if((zapnoutKompresor == 1) &&
   (prvniSpusteniKompresoruDokonceno == 1))
843.       {
```

```
844.            if(pid < 0)     // Pokud jsou hodnoty zaporne,
   je teplota v komore vyssi nez pozadovana
845.            {
846.                  // Pro jistotu vypneme topeni
847.                  GPIO_ClearPinsOutput(PERIF_OUT_O2_GPIO, 1u
   << PERIF_OUT_O2_GPIO_PIN);
848.
849.                  // Pro zacatek je uvazovana hodnota PID
   1000 (resp. -1000) a rozdelena do 6 stupnu PWM regulace
850.                  if(pid < -1000)
851.                  {
852.                        // Nejvyssi hodnota je 100, proto se
   vzdy pricita
853.                        if (updatedDutycycle < 100)
854.                        {
855.                              if(vterinaProPWM)
856.                              {
857.                                    vterinaProPWM = 0;
858.                                    updatedDutycycle++;
859.
860.                                    // Pro jistotu omezeni
861.                                    if(updatedDutycycle >=
   100)
862.                                    {
863.                                          updatedDutycycle =
   100;
864.                                    }
865.                              }
866.                        }
867.                  }
868.                  else if ((-1001 < pid) && (pid < -800))
869.                  {
870.                        if(updatedDutycycle > 80)
871.                        {
872.                              if(vterinaProPWM)
873.                              {
874.                                    vterinaProPWM = 0;
875.                                    updatedDutycycle--;
876.                              }
877.                        }
878.                        else if (updatedDutycycle < 80)
879.                        {
880.                              if(vterinaProPWM)
881.                              {
882.                                    vterinaProPWM = 0;
883.                                    updatedDutycycle++;
884.                              }
885.                        }
886.                  }
887.                  else if ((-801 < pid) && (pid < -600))
888.                  {
889.                        if(updatedDutycycle > 60)
890.                        {
891.                              if(vterinaProPWM)
892.                              {
893.                                    vterinaProPWM = 0;
```

```
                                    updatedDutycycle--;
                              }
                        }
                        else if (updatedDutycycle < 60)
                        {
                              if(vterinaProPWM)
                              {
                                    vterinaProPWM = 0;
                                    updatedDutycycle++;
                              }
                        }
                  }
                  else if ((-601 < pid) && (pid < -400))
                  {
                        if(updatedDutycycle > 40)
                        {
                              if(vterinaProPWM)
                              {
                                    vterinaProPWM = 0;
                                    updatedDutycycle--;
                              }
                        }
                        else if (updatedDutycycle < 40)
                        {
                              if(vterinaProPWM)
                              {
                                    vterinaProPWM = 0;
                                    updatedDutycycle++;
                              }
                        }
                  }
                  else if ((-401 < pid) && (pid < -200))
                  {
                        if(updatedDutycycle > 20)
                        {
                              if(vterinaProPWM)
                              {
                                    vterinaProPWM = 0;
                                    updatedDutycycle--;
                              }
                        }
                        else if (updatedDutycycle < 20)
                        {
                              if(vterinaProPWM)
                              {
                                    vterinaProPWM = 0;
                                    updatedDutycycle++;
                              }
                        }
                  }
                  else if ((-201 < pid) && (pid < 0))
                  {
                        if(updatedDutycycle > 5)
                        {
                              if(vterinaProPWM)
                              {
```

```
950.                                    vterinaProPWM = 0;
951.                                    updatedDutycycle--;
952.                               }
953.                          }
954.                          else if (updatedDutycycle < 5)
955.                          {
956.                               if(vterinaProPWM)
957.                               {
958.                                    vterinaProPWM = 0;
959.                                    updatedDutycycle++;
960.                               }
961.                          }
962.                     }
963.
964.               // Pokud se vyse zmenil duty cycle oproti stavu
   pri prichodu do funkce, zmenime tuto hodnotu v registrech PWM
965.                     // V tomto pripade se do oldDutycycle
   zapisuje ve funkci startKompresoru
966.               if(updatedDutycycle != oldDutycycle)
967.               {
968.                     /* Disable channel output before updating
   the dutycycle */
969.
      FTM_UpdateChnlEdgeLevelSelect(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, 0U);
970.
971.                     /* Update PWM duty cycle */
972.
      FTM_UpdatePwmDutycycle(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, kFTM_CenterAlignedPwm,
   updatedDutycycle);
973.
974.                     /* Software trigger to update registers */
975.
      FTM_SetSoftwareTrigger(PERIF_OUT_O7_PERIPHERAL, true);
976.
977.                     /* Start channel output with updated
   dutycycle */
978.
      FTM_UpdateChnlEdgeLevelSelect(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, pwmLevel);
979.
980.                     // A zapnout timer ovladajici PWM - pri
   vypnuti timeru je vystup ve stavu vysoke impedance (dle
   mereni napeti na jeho pinu)
981.                     FTM_StartTimer(PERIF_OUT_O7_PERIPHERAL,
   kFTM_SystemClock);
982.               }
983.             }
984.       }
985.  }
986.
987.
988.  /* Regulace pro topnou spiralu - podobne jako PWM je
   rozdeleno do nekolika stupnu dle PID hodnoty
```

```
989.   * Ve funkci neni ostereno "vypnuti" chlazeni, protoze se
   jedna o pomerne slozity proces - predpoklada se, ze pokud
   kompresor predtim chladil,
990.   * dostal se az na hodnotu 5% duty cycle a kompresor bezi
   na minimalni vykon - vykon kompresoru zatim neni v programu
   regulace menen
991.   *
992.   * */
993.  void regulaceTopeni(float pid)
994.  {
995.        if(zapnoutTopeni)
996.        {
997.              if(pid > 0)
998.              {
999.                    if(pid > 1000)
1000.                   {
1001.                         GPIO_SetPinsOutput(PERIF_OUT_O2_GPIO,
   1u << PERIF_OUT_O2_GPIO_PIN);
1002.                   }
1003.                   else if ((800 < pid) && (pid < 1001))
1004.                   {
1005.                         if(vterinaProTopeni > 16)
1006.                         {
1007.
      GPIO_ClearPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1008.                         }
1009.                         else
1010.                         {
1011.
      GPIO_SetPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1012.                         }
1013.                   }
1014.                   else if ((600 < pid) && (pid < 801))
1015.                   {
1016.                         if(vterinaProTopeni > 12)
1017.                         {
1018.
      GPIO_ClearPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1019.                         }
1020.                         else
1021.                         {
1022.
      GPIO_SetPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1023.                         }
1024.                   }
1025.                   else if ((400 < pid) && (pid < 601))
1026.                   {
1027.                         if(vterinaProTopeni > 8)
1028.                         {
1029.
      GPIO_ClearPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
```

```
1030.                       }
1031.                       else
1032.                       {
1033.
     GPIO_SetPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1034.                       }
1035.                 }
1036.                 else if ((200 < pid) && (pid < 401))
1037.                 {
1038.                       if(vterinaProTopeni > 4)
1039.                       {
1040.
     GPIO_ClearPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1041.                       }
1042.                       else
1043.                       {
1044.
     GPIO_SetPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1045.                       }
1046.                 }
1047.                 else if ((0 < pid) && (pid < 201))
1048.                 {
1049.                       if(vterinaProTopeni > 1)
1050.                       {
1051.
     GPIO_ClearPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1052.                       }
1053.                       else
1054.                       {
1055.
     GPIO_SetPinsOutput(PERIF_OUT_O2_GPIO, 1u <<
   PERIF_OUT_O2_GPIO_PIN);
1056.                       }
1057.                 }
1058.           }
1059.      }
1060. }
1061.
1062. /*
1063.  * Funkce zajistujici postupne vypnuti kompresoru vcetne
   PWM - opet zde neni prozatim uvazovan ctyrcestny ventil
1064.  * */
1065. void odstavkaKompresoru()
1066. {
1067.      if(vypnoutKompresor && (vypinaniKompresoruDokonceno
   == 0))
1068.      {
1069.           // Ventil otevrit na maximum - uvolneni tlaku v
   soustave
1070.           // Kontrola zmÄ›ny duty cycle z predchoziho
   prubehu
1071.           // Nejprve zaznamename puvodni duty cycle
```

```
1072.           oldDutycycle = updatedDutycycle;
1073.           // Dle zadani postupne otevrit ventil
   (vterinaProVypnuti je nulovana nize pri zmene hodnoty DAC)
1074.           if(vterinaProVypnuti)
1075.           {
1076.                updatedDutycycle += 5;
1077.                if(updatedDutycycle > 99)
1078.                {
1079.                     updatedDutycycle = 100;
1080.                }
1081.           }
1082.
1083.           // Pokud se zmenil duty cycle oproti stavu pri
   prichodu do funkce, zmenime tuto hodnotu v registrech PWM
1084.           if(updatedDutycycle != oldDutycycle)
1085.           {
1086.                /* Disable channel output before updating
   the dutycycle */
1087.
      FTM_UpdateChnlEdgeLevelSelect(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, 0U);
1088.
1089.                /* Update PWM duty cycle */
1090.
      FTM_UpdatePwmDutycycle(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, kFTM_CenterAlignedPwm,
   updatedDutycycle);
1091.
1092.                /* Software trigger to update registers */
1093.
      FTM_SetSoftwareTrigger(PERIF_OUT_O7_PERIPHERAL, true);
1094.
1095.                /* Start channel output with updated
   dutycycle */
1096.
      FTM_UpdateChnlEdgeLevelSelect(PERIF_OUT_O7_PERIPHERAL,
   PERIF_OUT_O7_CHANNEL, pwmLevel);
1097.
1098.                // A zapnout timer ovladajici PWM - pri
   vypnuti timeru je vystup ve stavu vysoke impedance (dle
   mereni napeti na jeho pinu)
1099.                FTM_StartTimer(PERIF_OUT_O7_PERIPHERAL,
   kFTM_SystemClock);
1100.           }
1101.           // Nyni je potreba snizovat vykon kompresoru az
   na nulu behem 15 minut. Mezi tim by samozrejme melo byt mozne
   topit.
1102.           // Pri stisknuti tlacitka nebo splneni podminky
   pro odstaveni kompresoru je nutne dbat na aktualni hodnotu
   DAC a od te postupne snizovat!
1103.           if(vterinaProVypnuti)
1104.           {
1105.                vterinaProVypnuti = 0;
1106.                // Odlisny zpusob nez pri zapinani - zde
   dojde k vynulovani po cca 820 vterinach pokud byl kompresor
```

```
   zapnut na maximum. Pripadny zaporny vysledek je osetren
   podminkou
1107.                 dacValue -= 5;
1108.                 if(dacValue < 1)
1109.                 {
1110.                       dacValue = 0;
1111.                       // Odstavka je dokoncena
1112.                       vypinaniKompresoruDokonceno = 1;
1113.                       prvniSpusteniKompresoruDokonceno = 0;
1114.                       prvniSpusteniKompresoru = 1;
1115.                 // Vypneme zelenou LED 2
1116.                 GPIO_SetPinsOutput(LED_GREEN_LED2_GPIO, 1u
   << LED_GREEN_LED2_GPIO_PIN);
1117.                 }
1118.
1119.           // Nastavime ovladaci napeti pro kompresor
1120.            DAC_Enable(DAC_DAC_10V_PERIPHERAL, true);
   /* Enable output. */
1121.
   DAC_SetBufferReadPointer(DAC_DAC_10V_PERIPHERAL, 0U); /* Make
   sure the read pointer to the start. */
1122.
   /*
1123.
   * The buffer is not enabled, so the read pointer can not move
   automatically. However, the buffer's read pointer
1124.
   * and items can be written manually by user.
1125.
   */
1126.            // Na vystupu DAC by pri prvnim zapnuti mela
   byt 0
1127.           DAC_SetBufferValue(DAC_DAC_10V_PERIPHERAL, 0U,
   dacValue);
1128.           }
1129.
1130.      }
1131. }
1132.
1133. /* Menu cislo 0
1134.  * Zakladni obrazovka, stisknutim OK zapne regulaci.
1135.  * */
1136. void menu0()
1137. {
1138.      // Kontrola, ze tu ma program byt
1139.      if (menu == 0)
1140.      {
1141.           // Nejprve kontrola na zmenu menu
1142.           if((menuChanged == 1) || (updateLCD == 1))
1143.           {
1144.                menuChanged = 0;
1145.                updateLCD = 0;
1146.
1147.                // Cislo menu
1148.                LCD_gotoxy(15, 1);
1149.                LCD_write_string("0");
```

```
1150.                // Informace
1151.                LCD_gotoxy(0, 0);
1152.                LCD_write_string("START");
1153.                LCD_gotoxy(0, 1);
1154.                LCD_write_string("Stiskni OK");
1155.
1156.                // Vypis pozadovane teploty na LCD
1157.                ftoa(pozadovanaTeplota, cislo);
1158.                LCD_gotoxy(6, 0);
1159.                LCD_write_string(cislo);
1160.                LCD_write_string("  ");    // Doplneni 2
  mezer
1161.           }
1162.           // Odstavka kompresoru
1163.           odstavkaKompresoru();
1164.
1165.           if (buttonUPpressed == 1)
1166.           {
1167.                updateLCD = 1;
1168.                pozadovanaTeplota += 0.5;
1169.                if(pozadovanaTeplota > 120)
1170.                {
1171.                     pozadovanaTeplota = 120;
1172.                }
1173.                // Registruje pouze jeden stisk
1174.                buttonUPpressed = 0;
1175.                opDone1 = 1;
1176.           }
1177.           else if(buttonUPcontinuous)
1178.           {
1179.                updateLCD = 1;
1180.                buttonUPcontinuousHelper++;
1181.                if(buttonUPcontinuousHelper > 50)
1182.                {
1183.                     buttonUPcontinuousHelper = 0;
1184.                     pozadovanaTeplota += 0.5;
1185.                }
1186.                if(pozadovanaTeplota > 120)
1187.                {
1188.                     pozadovanaTeplota = 120;
1189.                }
1190.           }
1191.           if (buttonDOWNpressed == 1)
1192.           {
1193.                updateLCD = 1;
1194.                pozadovanaTeplota -= 0.5;
1195.                if(pozadovanaTeplota < -30)
1196.                {
1197.                     pozadovanaTeplota = -30;
1198.                }
1199.                // Registruje pouze jeden stisk
1200.                buttonDOWNpressed = 0;
1201.                opDone2 = 1;
1202.           }
1203.           else if(buttonDOWNcontinuous)
1204.           {
```

```
1205.                updateLCD = 1;
1206.                buttonDOWNcontinuousHelper++;
1207.                if(buttonDOWNcontinuousHelper > 50)
1208.                {
1209.                     buttonDOWNcontinuousHelper = 0;
1210.                     pozadovanaTeplota += 0.5;
1211.                }
1212.                if(pozadovanaTeplota < -30)
1213.                {
1214.                     pozadovanaTeplota = -30;
1215.                }
1216.           }
1217.           if (buttonENTERpressed == 1)
1218.           {
1219.                buttonENTERpressed = 0;
1220.                opDone6 = 1;
1221.                menu = 100;
1222.                // Nastavime promennou, aby kompresor
   zahajil automaticky prvni spusteni
1223.                zapnoutKompresor = 1;
1224.                vypnoutKompresor = 0;
1225.                zapnoutTopeni = 1;
1226.
1227.                // Zapneme timer pro PID regulator
1228.                FTM_StartTimer(PID_FTM_BASEADDR,
   kFTM_SystemClock);
1229.
1230.                // Zapnout ventilator v komore
1231.                GPIO_SetPinsOutput(PERIF_OUT_O5_GPIO, 1u
   << PERIF_OUT_O5_GPIO_PIN);
1232.
1233.                // Pri pouziti flash by se mela pozadovana
   teplota zapsat do pameti
1234.           }
1235.
1236.      }
1237. }
1238.
1239. /* Menu probihajici regulace
1240.  *
1241.  * Zobrazuje teplotu uvnitr komory
1242.  * */
1243. void menu100()
1244. {
1245.      // Kontrola, ze tu ma program byt
1246.      if (menu == 100)
1247.      {
1248.           // Vypis na LCD
1249.           // Nejprve kontrola na zmenu menu
1250.           if(menuChanged == 1)
1251.           {
1252.                menuChanged = 0;
1253.
1254.                // Cislo menu
1255.                LCD_gotoxy(15, 1);
1256.                LCD_write_string("T");
```

```
1257.                 // Informace
1258.                 LCD_gotoxy(0, 0);
1259.                 LCD_write_string("REGULACE");
1260.                 LCD_gotoxy(0, 1);
1261.                 LCD_write_string("Konec ESC");
1262.           }
1263.           // Funkce pro kompresor - zkontroluje prvni
   spusteni a pote by mela pokracovat regulaci
1264.           startKompresoru();
1265.
1266.           // Pokud timer FTM1 nacte jednu vterinu, zavola
   se funkce PID kontroleru a upravi se hodnota PID
1267.           if(pidVterina >= 5)
1268.           {
1269.                pidVterina = 0;
1270.                // Nacist PID - pokud jsou hodnoty
   zaporne, je realna teplota vyssi nez pozadovana
1271.                pidHodnota =
   pid_Controller(pozadovanaTeplota, teplotaKomory, &pidData);
1272.                // PRINTF("\r\nPID: %d .\r\n",
   (int)pidHodnota); // Kontrola vystupu z funkce pro PID -
   nezbytne pro prizpusobeni konstant soustave
1273.           }
1274.
1275.           // Regulace
1276.
1277.           regulaceKompresoru(pidHodnota);
1278.           regulaceTopeni(pidHodnota);
1279.
1280.           // Vypis teploty v komore
1281.           ftoa(teplotaKomory, cislo);
1282.           LCD_gotoxy(9, 0);
1283.           LCD_write_string(cislo);
1284.           LCD_write_string("  ");
1285.
1286.
1287.
1288.           if (buttonESCpressed == 1) // Ukonceni regulace
1289.           {
1290.                // Osetreni stisku tlacitka
1291.                buttonESCpressed = 0;
1292.                opDone5 = 1;
1293.
1294.                // Vypne FTM1 pro PID regulator a reset
   integratoru PID
1295.                FTM_StopTimer(PID_FTM_BASEADDR);
1296.                pid_Reset_Integrator(&pidData);
1297.
1298.                // Vypnout topeni
1299.                zapnoutTopeni = 0;
1300.                GPIO_ClearPinsOutput(PERIF_OUT_O2_GPIO, 1u
   << PERIF_OUT_O2_GPIO_PIN);
1301.
1302.                // Vypnout ventilator
1303.                GPIO_ClearPinsOutput(PERIF_OUT_O5_GPIO, 1u
   << PERIF_OUT_O5_GPIO_PIN);
```

```
1304.
1305.                 // Odstavka kompresoru je zajistena v
  menu0
1306.                 if(zapnoutKompresor)
1307.                 {
1308.                       zapnoutKompresor = 0;
1309.                       vypnoutKompresor = 1;
1310.                 }
1311.                 // Navrat do menu0
1312.                 menu = 0;
1313.
1314.           }
1315.
1316.      }
1317. }
1318.
1319. /* Menu chyby kontrolovanych teplot na kompresoru apod.
1320.  * Zobrazuje chybovou hlasku
1321.  *
1322.  * */
1323. void menuChybaKompresoru()
1324. {
1325.      // Kontrola, ze tu ma program byt
1326.      if (menu == 50)
1327.      {
1328.           odstavkaKompresoru();
1329.
1330.           // Vypne FTM1 pro PID regulator a reset
  integratoru PID
1331.           FTM_StopTimer(PID_FTM_BASEADDR);
1332.           pid_Reset_Integrator(&pidData);
1333.
1334.           // Vypis na LCD
1335.           // Nejprve kontrola na zmenu menu
1336.           if(menuChanged == 1)
1337.           {
1338.                menuChanged = 0;
1339.
1340.                // Cislo menu
1341.                LCD_gotoxy(14, 1);
1342.                LCD_write_string("F1");
1343.                // Informace
1344.                LCD_gotoxy(0, 0);
1345.                LCD_write_string("CHYBA KOMP. (T)");
1346.                LCD_gotoxy(0, 1);
1347.                LCD_write_string("Zpet ESC");
1348.           }
1349.
1350.           if (buttonESCpressed == 1)
1351.           {
1352.                buttonESCpressed = 0;
1353.                opDone5 = 1;
1354.                menu = 0;
1355.           }
1356.
1357.      }
```

```
1358. }
1359.
1360. /* Funkce pro prepocet teploty SE
1361.  * prijme hodnotu z ADC a prepocita ji na teplotu, kterou
   vrati jako double
1362.  *
1363.  * @param cislo hodnota, kterou chceme prepocitat
1364.  * @param reference hodnota referencniho odporu a privodu
   cidla (2200)
1365.  * */
1366. float prepocetSE(uint32_t cislo, uint32_t reference)
1367. {
1368.       double napeti = 0;
1369.       double odpor = 0;
1370.       float teplota = 0;
1371.       // Voltmetr
1372.       napeti = (cislo/(ADC_PRUMEROVANI + 1)) *
   ADC_KONSTANTA;
1373.       // Ohmmetr - pro SE je napeti mereno na referencnim
   odporu a 3,3 V je rozdeleno na dvou odporech
1374.       // 2200 je hodnota referencniho odporu a pro
   presnejsi vysledky je nutne u kazdeho kanalu zapsat konkretni
   realnou hodnotu
1375.       odpor = ((3.3/napeti) * reference) - 2200;
1376.       // Nyni uz zname odpor cidla, takze staci prepocet na
   teplotu dle linearni funkce
1377.       teplota = (odpor - 1000)/(3.839167);
1378.       return teplota;
1379. }
1380.
1381. /* Funkce pro prepocet teploty z diferencnich vstupu
1382.  * prijme hodnotu z ADC a prepocita ji na teplotu, kterou
   vrati jako double
1383.  *
1384.  * @param cislo hodnota, kterou chceme prepocitat
1385.  * @param reference hodnota souctu obou referencnich odporu
   a privodu cidla (2000)
1386.  * */
1387. float prepocetDiff(uint32_t cislo, uint32_t reference)
1388. {
1389.       double napeti = 0;
1390.       float odpor = 0;
1391.       float teplota = 0;
1392.       // Voltmetr - stejne jako SE
1393.       napeti = (cislo/(ADC_PRUMEROVANI + 1)) *
   ADC_KONSTANTA;
1394.       // Ohmmetr - pro diferencni zapojeni je napeti mereno
   primo na cidle a 3,3 V je rozdeleno na celkem trech odporech
1395.       // reference je hodnota souctu referencnich odporu a
   pro presnejsi vysledky je nutne u kazdeho kanalu zapsat
   konkretni realnou hodnotu tohoto souctu
1396.       // hodnota ref. odporu se muze napriklad primo
   predavat do funkce
1397.       odpor = (reference/((3.3/napeti) - 1));
1398.       // Nyni uz zname odpor cidla, takze staci prepocet na
   teplotu dle linearni funkce - opet totozne s SE
```

```
1399.      teplota = (odpor - 1000)/(3.839167);
1400.      return teplota;
1401. }
1402.
1403.
1404. /* Tato funkce prijma hodnotu 1 az 8 podle teto vybere
   vstup ADC, na nemz provede prevod - hodnotu nevraci, pouze ji
   uklada do globalni promenne
1405.  *
1406.  *
1407.  * @param teplomer je pozadovany teplomer - 1 je
   teplotaKompresoru
1408.  *
      2 je teplotaVytlaku
1409.  *
      3 je teplotaVyparniku
1410.  *
      4 je teplotaKomory
1411.  *
      5 je teplotaRozvadece
1412.  *
      Ostatni nejsou zatim pouzity, lze tedy libovolne do
   programu pridat dalsi 3 teplomery
1413.  * */
1414. void getADC(uint8_t teplomer)
1415. {
1416.      switch(teplomer)
1417.      {
1418.      case 8:
1419.         /*
1420.          When in software trigger mode, each conversion
   would be launched once calling the "ADC16_ChannelConfigure()"
1421.          function, which works like writing a conversion
   command and executing it. For another channel's conversion,
1422.          just to change the "channelNumber" field in
   channel's configuration structure, and call the
1423.          "ADC16_ChannelConfigure() again.
1424.         */
1425.      adc16ChannelConfigStruct.channelNumber =
   ADC_TEP8_CHANNEL;
1426.      adc16ChannelConfigStruct.enableDifferentialConversion
   = false;
1427.         ADC16_SetChannelConfig(ADC_TEP8_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1428.
1429.         while (0U == (kADC16_ChannelConversionDoneFlag &
1430.
   ADC16_GetChannelStatusFlags(ADC_TEP8_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1431.         {
1432.         }
1433.         adcResultValue8 =
   ADC16_GetChannelConversionValue(ADC_TEP8_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1434.           break;
1435.      case 7:
```

```
1436.         /*
1437.          When in software trigger mode, each conversion
   would be launched once calling the "ADC16_ChannelConfigure()"
1438.          function, which works like writing a conversion
   command and executing it. For another channel's conversion,
1439.          just to change the "channelNumber" field in
   channel's configuration structure, and call the
1440.          "ADC16_ChannelConfigure() again.
1441.         */
1442.       adc16ChannelConfigStruct.channelNumber =
   ADC_TEP7_CHANNEL;
1443.       adc16ChannelConfigStruct.enableDifferentialConversion
   = false;
1444.         ADC16_SetChannelConfig(ADC_TEP7_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1445.
1446.         while (0U == (kADC16_ChannelConversionDoneFlag &
1447.
   ADC16_GetChannelStatusFlags(ADC_TEP7_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1448.         {
1449.         }
1450.         adcResultValue7 =
   ADC16_GetChannelConversionValue(ADC_TEP7_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1451.           break;
1452.       case 6:
1453.         /*
1454.          When in software trigger mode, each conversion
   would be launched once calling the "ADC16_ChannelConfigure()"
1455.          function, which works like writing a conversion
   command and executing it. For another channel's conversion,
1456.          just to change the "channelNumber" field in
   channel's configuration structure, and call the
1457.          "ADC16_ChannelConfigure() again.
1458.         */
1459.       adc16ChannelConfigStruct.channelNumber =
   ADC_TEP6_CHANNEL;
1460.       adc16ChannelConfigStruct.enableDifferentialConversion
   = false;
1461.         ADC16_SetChannelConfig(ADC_TEP6_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1462.
1463.         while (0U == (kADC16_ChannelConversionDoneFlag &
1464.
   ADC16_GetChannelStatusFlags(ADC_TEP6_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1465.         {
1466.         }
1467.         adcResultValue6 =
   ADC16_GetChannelConversionValue(ADC_TEP6_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1468.           break;
1469.       case 5:
1470.         /*
```

```
1471.          When in software trigger mode, each conversion
   would be launched once calling the "ADC16_ChannelConfigure()"
1472.          function, which works like writing a conversion
   command and executing it. For another channel's conversion,
1473.          just to change the "channelNumber" field in
   channel's configuration structure, and call the
1474.          "ADC16_ChannelConfigure() again.
1475.         */
1476.      adc16ChannelConfigStruct.channelNumber =
   ADC_TEP5_CHANNEL;
1477.      adc16ChannelConfigStruct.enableDifferentialConversion
   = false;
1478.         ADC16_SetChannelConfig(ADC_TEP5_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1479.
1480.         while (0U == (kADC16_ChannelConversionDoneFlag &
1481.
   ADC16_GetChannelStatusFlags(ADC_TEP5_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1482.         {
1483.         }
1484.         adcResultValue5 =
   ADC16_GetChannelConversionValue(ADC_TEP5_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1485.           break;
1486.      case 4:
1487.         /* Diferencni kanal. */
1488.      adc16ChannelConfigStruct.channelNumber =
   ADC_TEP4P_CHANNEL;
1489.      adc16ChannelConfigStruct.enableDifferentialConversion
   = true;
1490.         ADC16_SetChannelConfig(ADC_TEP4P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1491.
1492.         while (0U == (kADC16_ChannelConversionDoneFlag &
1493.
   ADC16_GetChannelStatusFlags(ADC_TEP4P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1494.         {
1495.         }
1496.         adcResultValue4 =
   ADC16_GetChannelConversionValue(ADC_TEP4P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1497.           break;
1498.      case 3:
1499.         /* Diferencni kanal. */
1500.      adc16ChannelConfigStruct.channelNumber =
   ADC_TEP3P_CHANNEL;
1501.      adc16ChannelConfigStruct.enableDifferentialConversion
   = true;
1502.         ADC16_SetChannelConfig(ADC_TEP3P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1503.
1504.         while (0U == (kADC16_ChannelConversionDoneFlag &
```

```
1505.
   ADC16_GetChannelStatusFlags(ADC_TEP3P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1506.         {
1507.         }
1508.         adcResultValue3 =
   ADC16_GetChannelConversionValue(ADC_TEP3P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1509.           break;
1510.      case 2:
1511.         /* Diferencni kanal. */
1512.      adc16ChannelConfigStruct.channelNumber =
   ADC_TEP2P_CHANNEL;
1513.      adc16ChannelConfigStruct.enableDifferentialConversion
   = true;
1514.         ADC16_SetChannelConfig(ADC_TEP2P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1515.
1516.         while (0U == (kADC16_ChannelConversionDoneFlag &
1517.
   ADC16_GetChannelStatusFlags(ADC_TEP2P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1518.         {
1519.         }
1520.         adcResultValue2 =
   ADC16_GetChannelConversionValue(ADC_TEP2P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1521.           break;
1522.      case 1:
1523.         /* Diferencni kanal. */
1524.      adc16ChannelConfigStruct.channelNumber =
   ADC_TEP1P_CHANNEL;
1525.      adc16ChannelConfigStruct.enableDifferentialConversion
   = true;
1526.         ADC16_SetChannelConfig(ADC_TEP1P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
1527.
1528.         while (0U == (kADC16_ChannelConversionDoneFlag &
1529.
   ADC16_GetChannelStatusFlags(ADC_TEP1P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP)))
1530.         {
1531.         }
1532.         adcResultValue1 =
   ADC16_GetChannelConversionValue(ADC_TEP1P_PERIPHERAL,
   DEMO_ADC16_CHANNEL_GROUP);
1533.           break;
1534.      default: PRINTF("\r\n Zadan neplatny kanal\r\n");
     // Opet pouze pro debugging vystup na konzoli
1535.      }
1536. }
1537.
1538.
1539. /*
1540.  *
1541.  *
```

```
1542.  * Interrupt handlers
1543.  *
1544.  * */
1545.
1546. /*!
1547.  * @brief ISR for PDB interrupt function
1548.  */
1549. void DEMO_PDB_IRQ_HANDLER(void)
1550. {
1551.     PDB_ClearStatusFlags(DEMO_PDB_BASE,
   kPDB_DelayEventFlag);
1552.
1553.     GPIO_TogglePinsOutput(LED_GREEN_LED3_GPIO, 1u <<
   LED_GREEN_LED3_GPIO_PIN);     // Pouze pro informaci, ze
   interrupt funguje - je pouzita zelena LED3
1554.     // Nacist informaci, ze uplynula vterina pro regulaci
   kompresoru - je celkem nezajimave, pokud prvni vterina
1555.     // bude rovnou zmarena, protoze celkovy cas pro nabeh
   kompresoru je 5 + 15 minut
1556.     // Samozrejme je nutne prislusne promenne po pouziti
   vhodne nulovat (petMinutUplynulo, patnactMinutUplynulo)
1557.     // Pravdepodobne pokud nebyl kompresor pouzit dele nez
   urcity casovy usek, nebo doslo k jeho odstavce
1558.     if(zapnoutKompresor)
1559.     {
1560.      if(prvniSpusteniKompresoru)
1561.      {
1562.           vterinaProKompresor++;
1563.           devetVterin++;
1564.           if(vterinaProKompresor > 59)
1565.           {
1566.                vterinaProKompresor = 0;
1567.                minutaProKompresor++;
1568.                if(minutaProKompresor > 4)
1569.                {
1570.                     petMinutUplynulo = 1;
1571.                     if(minutaProKompresor > 19)
1572.                     {
1573.                          patnactMinutUplynulo = 1;
1574.                          // Zaroven tim nastavime vlajku,
   ze byl prvni start dokoncen
1575.                          prvniSpusteniKompresoruDokonceno
   = 1;
1576.                          prvniSpusteniKompresoru = 0;
1577.                          minutaProKompresor = 0;
1578.                     }
1579.                }
1580.           }
1581.           if(devetVterin > 8)
1582.           {
1583.                devetVterin = 0;
1584.                devetVterinUplynulo = 1;
1585.           }
1586.      }
1587.      vterinaProPWM = 1;
1588.     }
```

```
1589.     else minutaProKompresor = 0;
1590.     if(vypnoutKompresor)
1591.     {
1592.       // Cita pouze 1 vterinu, samozrejme po pouziti nutno
   vynulovat
1593.       vterinaProVypnuti = 1;
1594.     }
1595.     if(nacitatVterinaProDisplay)
1596.     {
1597.       vterinaProDisplay++;
1598.     }
1599.     if(zapnoutTopeni)
1600.     {
1601.       vterinaProTopeni++;
1602.       if(vterinaProTopeni > 20)
1603.       {
1604.             vterinaProTopeni = 0;
1605.       }
1606.     }
1607.
1608.     // Znovu citat
1609.     PDB_DoSoftwareTrigger(DEMO_PDB_BASE);
1610. }
1611.
1612. /*
1613.  * Prozatim je tento interrupt nevyuzity
1614.  * */
1615. void FTM_LED_HANDLER(void)
1616. {
1617.     if ((FTM_GetStatusFlags(PERIF_OUT_O7_PERIPHERAL) &
   FTM_CHANNEL_FLAG) == FTM_CHANNEL_FLAG)
1618.     {
1619.         /* Clear interrupt flag.*/
1620.         FTM_ClearStatusFlags(PERIF_OUT_O7_PERIPHERAL,
   FTM_CHANNEL_FLAG);
1621.     }
1622. }
1623.
1624. /*
1625.  * Pouze nacita vteriny pro PID - i pres nastaveni dle
   example bezi cca 3x rcyhleji nez by mel (interrupt je vyvolan
   cca 3x do 1 vteriny)
1626.  * */
1627. void PID_FTM_HANDLER(void)
1628. {
1629.     /* Clear interrupt flag.*/
1630.     FTM_ClearStatusFlags(PID_FTM_BASEADDR,
   kFTM_TimeOverflowFlag);
1631.
1632.     pidVterina++;
1633. }
```

Začátek hlavního programu. Obecně jsou zde nastaveny veškeré periferie včetně GPIO apod. Nastavování periferií (stejně tak jako například změna PWM střídy výše při ovládání klimatizace) je naprogramováno dle example projektů.

Při základním testování se ukázalo, že čítače čítají rychleji, než je nastaveno. Pravděpodobně se jedná pouze o nevhodně zvolený zdroj hodinového taktu, nicméně je pro bezhybný chod nezbytné přesně určit příčinu tohoto jevu.

Zároveň jsou na začátku programu zinicializovány všechny vstupy a výstupy – zde je nutné dle reálné soustavy určit, na jakou logickou úroveň mají být nastaveny.

```
1634.
1635. int main(void) {
1636.      /* Define the init structure for the output LED pin.
1637.       * Dle example projektu.
1638.       * Bez zapisu dle example se zdaji byt vstupy i
   vystupy dle Config Tools nefunkcni.
1639.       * */
1640.      gpio_pin_config_t led_config = { kGPIO_DigitalOutput,
   0, };
1641.
1642.      /* Define the init structure for the input switch
   pin.
1643.       * Opet dle example.
1644.       * */
1645.      gpio_pin_config_t sw_config = { kGPIO_DigitalInput,
   0, };
1646.
1647.      // Nastaveni FTM0
1648.      /* Define the init structure for FTM.
1649.      * Opet dle example.
1650.      * */
1651.      ftm_config_t ftmInfo;
1652.      FTM_GetDefaultConfig(&ftmInfo);
1653.     /* Divide FTM clock by 4 */
1654.     ftmInfo.prescale = kFTM_Prescale_Divide_4;
1655.     /* Initialize FTM module */
1656.     FTM_Init(DELAY_FTM_BASEADDR, &ftmInfo);
1657.     /*
1658.      * Set timer period.
1659.      * Enable interrupt.
1660.     */
1661.     FTM_SetTimerPeriod(DELAY_FTM_BASEADDR,
   USEC_TO_COUNT(10U, FTM_SOURCE_CLOCK));
1662.     FTM_EnableInterrupts(DELAY_FTM_BASEADDR,
   kFTM_TimeOverflowInterruptEnable);
1663.     EnableIRQ(DELAY_FTM_IRQ_NUM);
1664.
1665.     /*
1666.      * Doplneni pro FTM1
1667.      * Z nejakeho duvodu nesedi pocet ms (1000) pro jednu
   vterinu - cita mnohem rychleji
1668.      * Kvuli odezve systemu na PID je tedy hodnota odhadem
   zmenena.
1669.      * */
1670.     ftmInfo.prescale = kFTM_Prescale_Divide_128;
1671.     FTM_Init(PID_FTM_BASEADDR, &ftmInfo);
1672.     FTM_SetTimerPeriod(PID_FTM_BASEADDR,
   USEC_TO_COUNT(1000000U, FTM_PID_SOURCE_CLOCK));
```

```
1673.
1674.     FTM_EnableInterrupts(PID_FTM_BASEADDR,
   kFTM_TimeOverflowInterruptEnable);
1675.
1676.     EnableIRQ(PID_FTM_IRQ_NUM);
1677.     // FTM_StartTimer(PID_FTM_BASEADDR, kFTM_SystemClock);
1678.
1679.     /*
1680.      * Nastaveni PDB zpozdeni/timeru - 1 vterina
1681.      * */
1682.     pdb_config_t pdbConfigStruct;
1683.     EnableIRQ(DEMO_PDB_IRQ_ID);
1684.     /* Configure the PDB counter. */
1685.     /*
1686.      * pdbConfigStruct.loadValueMode =
   kPDB_LoadValueImmediately;
1687.      * pdbConfigStruct.prescalerDivider =
   kPDB_PrescalerDivider1;
1688.      * pdbConfigStruct.dividerMultiplicationFactor =
   kPDB_DividerMultiplicationFactor1;
1689.      * pdbConfigStruct.triggerInputSource =
   kPDB_TriggerSoftware;
1690.      * pdbConfigStruct.enableContinuousMode = false;
1691.      */
1692.     PDB_GetDefaultConfig(&pdbConfigStruct);
1693.     // Zmenime mod na continuous
1694.     //pdbConfigStruct.enableContinuousMode = true;
1695.     pdbConfigStruct.prescalerDivider =
   kPDB_PrescalerDivider64;
1696.     pdbConfigStruct.dividerMultiplicationFactor =
   kPDB_DividerMultiplicationFactor40;
1697.     PDB_Init(DEMO_PDB_BASE, &pdbConfigStruct);
1698.
1699.     /* Configure the delay interrupt. */
1700.     PDB_SetModulusValue(DEMO_PDB_BASE, 10000U);
1701.
1702.     /* The available delay value is less than or equal to
   the modulus value. */
1703.     PDB_SetCounterDelayValue(DEMO_PDB_BASE, 10000U);
1704.     PDB_EnableInterrupts(DEMO_PDB_BASE,
   kPDB_DelayInterruptEnable);
1705.     PDB_DoLoadValues(DEMO_PDB_BASE);
1706.     // Zapneme PDB
1707.     PDB_DoSoftwareTrigger(DEMO_PDB_BASE);
1708.
1709.
1710.     /* Inicializace ADC - zatĂm pouze SE a testovĂˇnĂ 1
   vĂ´vodu
1711.      *
1712.      * */
1713.     //Struktura pro SE ADC
1714.     adc16_config_t adc16ConfigStruct;
1715.
1716.     /* Struktura pro DAC (10 V). */
1717.     dac_config_t dacConfigStruct;
1718.
```

```
1719.
1720.
1721.      /* Init board hardware. */
1722.      BOARD_InitBootPins();
1723.      BOARD_InitBootClocks();
1724.      /* Init FSL debug console. */
1725.      BOARD_InitDebugConsole();
1726.
1727.
1728.     /*
1729.      * FTM3 - nastaveni PWM dle example ftm_simple_pwm
1730.      * Interrupt neni potreba
1731.      * */
1732.     ftm_chnl_pwm_signal_param_t ftmParam;
1733.
1734.     /* Configure ftm params with frequency 24kHZ */
1735.     ftmParam.chnlNumber = PERIF_OUT_O7_CHANNEL;
1736.     ftmParam.level = pwmLevel;
1737.     ftmParam.dutyCyclePercent = updatedDutycycle;
1738.     ftmParam.firstEdgeDelayPercent = 0U;
1739.     // Upravit prescaler, pripadne hodnotu u nastaveni pwm
   - nyni dle example je pwm 24 kHz, pri prescale na 128 by melo
   byt mene nez 200 Hz
1740.     // Je potreba vyzkouset, jestli se frekvence
   neprepocitava i s prescale!
1741.     ftmInfo.prescale = kFTM_Prescale_Divide_128;
1742.     /* Initialize FTM module */
1743.     FTM_Init(PERIF_OUT_O7_PERIPHERAL, &ftmInfo);
1744.     FTM_SetupPwm(PERIF_OUT_O7_PERIPHERAL, &ftmParam, 1U,
   kFTM_CenterAlignedPwm, 24000U, FTM_SOURCE_CLOCK3);
1745.     /* Enable channel interrupt flag.*/
1746.     //FTM_EnableInterrupts(PERIF_OUT_O7_PERIPHERAL,
   FTM_CHANNEL_INTERRUPT_ENABLE);
1747.
1748.     /* Enable at the NVIC */
1749.     //EnableIRQ(FTM_INTERRUPT_NUMBER);
1750.
1751.     // Start timer
1752.     // FTM_StartTimer(PERIF_OUT_O7_PERIPHERAL,
   kFTM_SystemClock);
1753.
1754.      /* Inicializace ADC
1755.       *
1756.       * */
1757.     // Asi bude dobrÃ© pouÍĬÄt long sample atd. dle
   struktury nÄĹľe
1758.      // kADC16_ResolutionSE12Bit znamenÃ˘ 12 bit pro SE a
   13 bit pro DM (popis v reference manuÃ˘lu)
1759.     /*
1760.      * adc16ConfigStruct.referenceVoltageSource =
   kADC16_ReferenceVoltageSourceVref;
1761.      * adc16ConfigStruct.clockSource =
   kADC16_ClockSourceAsynchronousClock;
1762.      * adc16ConfigStruct.enableAsynchronousClock = true;
1763.      * adc16ConfigStruct.clockDivider =
   kADC16_ClockDivider8;
```

```
1764.      * adc16ConfigStruct.resolution =
   kADC16_ResolutionSE12Bit;
1765.      * adc16ConfigStruct.longSampleMode =
   kADC16_LongSampleDisabled;
1766.      * adc16ConfigStruct.enableHighSpeed = false;
1767.      * adc16ConfigStruct.enableLowPower = false;
1768.      * adc16ConfigStruct.enableContinuousConversion =
   false;
1769.      */
1770.     ADC16_GetDefaultConfig(&adc16ConfigStruct);
1771.     //    adc16ConfigStruct.clockDivider =
   kADC16_ClockDivider2;
1772.     ADC16_Init(ADC_TEP7_PERIPHERAL, &adc16ConfigStruct);
1773.     //? Pokud je mux na kanĂ˘lu b? ZatĂm se ukĂ˘zaly
   funkÄŤnĂ pouze kanĂ˘ly a
1774. //#if defined(DEMO_ADC16_CHANNEL_MUXB) &&
   DEMO_ADC16_CHANNEL_MUXB
1775.     //    ADC16_SetChannelMuxMode(ADC_TEP7_PERIPHERAL,
   kADC16_ChannelMuxB);
1776. //#endif
1777.     //Budeme pouĹľĂvat poze softwarovĂ˝ trigger
1778.     ADC16_EnableHardwareTrigger(ADC_TEP7_PERIPHERAL,
   false); /* Make sure the software trigger is used. */
1779.     /*Autokalibrace - urÄŤitÄ› dobrÄ© provÄ©st*/
1780. #if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) &&
   FSL_FEATURE_ADC16_HAS_CALIBRATION
1781.     if (kStatus_Success ==
   ADC16_DoAutoCalibration(ADC_TEP7_PERIPHERAL))
1782.     {
1783.         PRINTF("ADC16_DoAutoCalibration() Done.\r\n");
1784.     }
1785.     else
1786.     {
1787.         PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
1788.     }
1789. #endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */
1790.     /* Disable interrupt. */
1791.
   adc16ChannelConfigStruct.enableInterruptOnConversionCompleted
   = false;
1792. //#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) &&
   FSL_FEATURE_ADC16_HAS_DIFF_MODE
1793. //    adc16ChannelConfigStruct.enableDifferentialConversion
   = false;
1794. //#endif /* FSL_FEATURE_ADC16_HAS_DIFF_MODE */
1795.
1796.
1797.     /* Configure the DAC. */
1798.     /*
1799.      * dacConfigStruct.referenceVoltageSource =
   kDAC_ReferenceVoltageSourceVref2;
1800.      * dacConfigStruct.enableLowPowerMode = false;
1801.      */
1802.     DAC_GetDefaultConfig(&dacConfigStruct);
1803.     DAC_Init(DAC_DAC_10V_PERIPHERAL, &dacConfigStruct);
```

```
1804.     //DAC_Enable(DAC_DAC_10V_PERIPHERAL, true);
   /* Enable output. */
1805.     //DAC_SetBufferReadPointer(DAC_DAC_10V_PERIPHERAL, 0U);
   /* Make sure the read pointer to the start. */
1806.                                                         /*
1807.                                                         * The
   buffer is not enabled, so the read pointer can not move
   automatically. However, the buffer's read pointer
1808.                                                         * and
   items can be written manually by user.
1809.                                                         */
1810.
1811.
1812.       /* Inicializace vystupu pro LCD
1813.        * Init struktura je stejna jako pro LED
1814.        * Nasleduje standardni inicializacni rutina.
1815.        * */
1816.       GPIO_PinInit(LCDDB0_GPIO, LCDDB0_GPIO_PIN,
   &led_config);
1817.       GPIO_PinInit(LCDDB1_GPIO, LCDDB1_GPIO_PIN,
   &led_config);
1818.       GPIO_PinInit(LCDDB2_GPIO, LCDDB2_GPIO_PIN,
   &led_config);
1819.       GPIO_PinInit(LCDDB3_GPIO, LCDDB3_GPIO_PIN,
   &led_config);
1820.       GPIO_PinInit(LCDDB4_GPIO, LCDDB4_GPIO_PIN,
   &led_config);
1821.       GPIO_PinInit(LCDDB5_GPIO, LCDDB5_GPIO_PIN,
   &led_config);
1822.       GPIO_PinInit(LCDDB6_GPIO, LCDDB6_GPIO_PIN,
   &led_config);
1823.       GPIO_PinInit(LCDDB7_GPIO, LCDDB7_GPIO_PIN,
   &led_config);
1824.
1825.       GPIO_PinInit(LCDRS_GPIO, LCDRS_GPIO_PIN,
   &led_config);
1826.       GPIO_PinInit(LCDE_GPIO, LCDE_GPIO_PIN, &led_config);
1827.
1828.       LCDInit();
1829.       PRINTF("\r\n LCD ready.\r\n");
1830.
1831.       /* Init output LED GPIO.
1832.        * Dle example.
1833.        * */
1834.       GPIO_PinInit(LED_GREEN_LED1_GPIO,
   LED_GREEN_LED1_GPIO_PIN, &led_config);
1835.       GPIO_PinInit(LED_GREEN_LED2_GPIO,
   LED_GREEN_LED2_GPIO_PIN, &led_config);
1836.       GPIO_PinInit(LED_GREEN_LED3_GPIO,
   LED_GREEN_LED3_GPIO_PIN, &led_config);
1837.       GPIO_PinInit(LED_GREEN_LED4_GPIO,
   LED_GREEN_LED4_GPIO_PIN, &led_config);
1838.       GPIO_PinInit(LED_GREEN_LED5_GPIO,
   LED_GREEN_LED5_GPIO_PIN, &led_config);
1839.       GPIO_PinInit(LED_RED_LED1_GPIO,
   LED_RED_LED1_GPIO_PIN, &led_config);
```

```
1840.      GPIO_PinInit(LED_RED_LED2_GPIO,
   LED_RED_LED2_GPIO_PIN, &led_config);
1841.      GPIO_PinInit(LED_RED_LED3_GPIO,
   LED_RED_LED3_GPIO_PIN, &led_config);
1842.
1843.      /* Init output 12 V driver GPIO.
1844.       * Dle example.
1845.       * */
1846.      GPIO_PinInit(PERIF_OUT_O0_GPIO,
   PERIF_OUT_O0_GPIO_PIN, &led_config);
1847.      GPIO_PinInit(PERIF_OUT_O1_GPIO,
   PERIF_OUT_O1_GPIO_PIN, &led_config);
1848.      GPIO_PinInit(PERIF_OUT_O2_GPIO,
   PERIF_OUT_O2_GPIO_PIN, &led_config);
1849.      GPIO_PinInit(PERIF_OUT_O3_GPIO,
   PERIF_OUT_O3_GPIO_PIN, &led_config);
1850.      GPIO_PinInit(PERIF_OUT_O4_GPIO,
   PERIF_OUT_O4_GPIO_PIN, &led_config);
1851.      GPIO_PinInit(PERIF_OUT_O5_GPIO,
   PERIF_OUT_O5_GPIO_PIN, &led_config);
1852.      GPIO_PinInit(PERIF_OUT_O6_GPIO,
   PERIF_OUT_O6_GPIO_PIN, &led_config);
1853.      // Cislo 7 definovano jako PWM - nutno inicializovat
   trochu jinak
1854.      GPIO_PinInit(PERIF_OUT_O8_GPIO,
   PERIF_OUT_O8_GPIO_PIN, &led_config);
1855.      GPIO_PinInit(PERIF_OUT_O9_GPIO,
   PERIF_OUT_O9_GPIO_PIN, &led_config);
1856.      GPIO_PinInit(PERIF_OUT_O10_GPIO,
   PERIF_OUT_O10_GPIO_PIN, &led_config);
1857.      GPIO_PinInit(PERIF_OUT_O11_GPIO,
   PERIF_OUT_O11_GPIO_PIN, &led_config);
1858.
1859.      /* Init input BUTTON GPIO.
1860.       * Dle example.
1861.       * */
1862.      GPIO_PinInit(BUTTONS_UP_GPIO, BUTTONS_UP_GPIO_PIN,
   &sw_config); //UP
1863.      GPIO_PinInit(BUTTONS_DOWN_GPIO,
   BUTTONS_DOWN_GPIO_PIN, &sw_config);//DOWN
1864.      GPIO_PinInit(BUTTONS_LEFT_GPIO,
   BUTTONS_LEFT_GPIO_PIN, &sw_config);//LEFT
1865.      GPIO_PinInit(BUTTONS_RIGHT_GPIO,
   BUTTONS_RIGHT_GPIO_PIN, &sw_config);//RIGHT
1866.      GPIO_PinInit(BUTTONS_ESC_GPIO, BUTTONS_ESC_GPIO_PIN,
   &sw_config);//ESC
1867.      GPIO_PinInit(BUTTONS_ENTER_GPIO,
   BUTTONS_ENTER_GPIO_PIN, &sw_config);//ENTER
1868.      GPIO_PinInit(BUTTONS_MENU_GPIO,
   BUTTONS_MENU_GPIO_PIN, &sw_config);//DOWN
1869.
1870.      // Inicializace teplot bez prumerovani - pouze
   zjistime aktualni hodnotu a zapiseme ji do prislusne promenne
1871.      for(int i = 1; i < 9; i++)
1872.      {
1873.           getADC(i);
```

```
1874.      }
1875.      teplotaKompresoru = prepocetDiff(adcResultValue1,
   2000);
1876.      teplotaVytlaku = prepocetDiff(adcResultValue2, 2000);
1877.      teplotaVyparniku = prepocetDiff(adcResultValue3,
   2000);
1878.      teplotaKomory = prepocetDiff(adcResultValue4, 2000);
1879.      teplotaRozvadece = prepocetSE(adcResultValue5, 2200);
1880.
1881.      LCD_gotoxy(0, 0);
1882.      LCD_write_string("Regulator DP");
1883.
1884.      LCD_gotoxy(0, 1);
1885.      LCD_write_string("V 0.1");
1886.
1887.      // Inicializace PID
1888.      pid_Init(K_P, K_I, K_D, &pidData);
1889.
1890.
1891.      /* Hlavni smycka programu. */
1892.      while (1) {
1893.
1894.           // Postupne ziskame hodnotu z ADC
1895.           getADC(aktualniKanal);
1896.           // Nyni prumerovani a prepocet na teplotu
1897.           switch(aktualniKanal)
1898.           {
1899.           case 1:
1900.                adcResultValue1prumer += adcResultValue1;
1901.                break;
1902.           case 2:
1903.                adcResultValue2prumer += adcResultValue2;
1904.                break;
1905.           case 3:
1906.                adcResultValue3prumer += adcResultValue3;
1907.                break;
1908.           case 4:
1909.                adcResultValue4prumer += adcResultValue4;
1910.                break;
1911.           case 5:
1912.                adcResultValue5prumer += adcResultValue5;
1913.                break;
1914.           case 6:
1915.                adcResultValue6prumer += adcResultValue6;
1916.                break;
1917.           case 7:
1918.                adcResultValue7prumer += adcResultValue7;
1919.                break;
1920.           case 8:
1921.                // Zde navic nacitame pomocnou promennou
   prumerovani
1922.                pocetPrumeru++;
1923.                adcResultValue8prumer += adcResultValue8;
1924.                break;
1925.           default: PRINTF("\r\n Zadan neplatny
   kanal\r\n");
```

```
1926.           }
1927.           // Jakmile je dosazeno pozadovaneho poctu
  mereni, celkovy vysledek predame funkci pro vypocet teploty
1928.           if(pocetPrumeru >= ADC_PRUMEROVANI)
1929.           {
1930.                pocetPrumeru = 0;
1931.
1932.                // Zjistime teplotu a vynulujeme pomocnou
  promennou pro kazdy kanal (stejne jako ostatni ukony i u
  nepouzitych)
1933.                teplotaKompresoru =
  prepocetDiff(adcResultValue1prumer, 2000);
1934.                adcResultValue1prumer = 0;
1935.
1936.                teplotaVytlaku =
  prepocetDiff(adcResultValue2prumer, 2000);
1937.                adcResultValue2prumer = 0;
1938.
1939.                teplotaVyparniku =
  prepocetDiff(adcResultValue3prumer, 2000);
1940.                adcResultValue3prumer = 0;
1941.
1942.                teplotaKomory =
  prepocetDiff(adcResultValue4prumer, 2000);
1943.                adcResultValue4prumer = 0;
1944.
1945.                teplotaRozvadece =
  prepocetSE(adcResultValue5prumer, 2200);
1946.                adcResultValue5prumer = 0;
1947.
1948.                // Nepouzite
1949.                adcResultValue6prumer = 0;
1950.                adcResultValue7prumer = 0;
1951.                adcResultValue8prumer = 0;
1952.           }
1953.
1954.           // Postupne menime vybrany teplomer
1955.           aktualniKanal++;
1956.           if(aktualniKanal > 8)
1957.           {
1958.                aktualniKanal = 1;
1959.           }
1960.
1961.
1962.           // Mazani displaye pri zmene MENU
1963.           previousMenu = currentMenu;
1964.           currentMenu = menu;
1965.           if(previousMenu != currentMenu)
1966.           {
1967.                menuChanged = 1;
1968.                // Vycistit display pred zapisem noveho
  menu.
1969.                LCD_clear();
1970.           }
1971.           else menuChanged = 0;
1972.
```

```
1973.           // Tlacitka
1974.           tlacitka();
1975.
1976.           // Zapnout/vypnout LED pri topeni
1977.           if(GPIO_ReadPinInput(PERIF_OUT_O2_GPIO,
   PERIF_OUT_O2_GPIO_PIN))
1978.           {
1979.                GPIO_ClearPinsOutput(LED_RED_LED1_GPIO, 1u
   << LED_RED_LED1_GPIO_PIN);
1980.           }
1981.           else GPIO_SetPinsOutput(LED_RED_LED1_GPIO, 1u <<
   LED_RED_LED1_GPIO_PIN);
1982.
1983.           // Zapnout/vypnout LED pokud je zapnuty
   ventilator
1984.           if(GPIO_ReadPinInput(PERIF_OUT_O5_GPIO,
   PERIF_OUT_O5_GPIO_PIN))
1985.           {
1986.                GPIO_ClearPinsOutput(LED_GREEN_LED2_GPIO,
   1u << LED_GREEN_LED2_GPIO_PIN);
1987.           }
1988.           else GPIO_SetPinsOutput(LED_GREEN_LED2_GPIO, 1u
   << LED_GREEN_LED2_GPIO_PIN);
1989.
1990.           //MENU
1991.           menu0();
1992.           menu100();
1993.           menuChybaKompresoru();
1994.
1995.      }
1996.      return 0;
1997. }
1998.
1999. /*
2000.  * Interrupt pro LCD casovac zpozdeni
2001.  * */
2002. void DELAY_FTM_HANDLER(void)
2003. {
2004.     /* Clear interrupt flag.*/
2005.     FTM_ClearStatusFlags(DELAY_FTM_BASEADDR,
   kFTM_TimeOverflowFlag);
2006.     oneUsCnt++;
2007. }
2008.
```

Na závěr je třeba zdůraznit, že program v této podobě nepoužívá pro ukládání paměť flash, takže některé proměnné, které je třeba ukládat mezi spuštěními systému, ukládány nejsou, respektive značná část těchto proměnných chybí, jelikož bez paměti flash postrádá smysl takové proměnné inicializovat.

Například se jedná o referenční hodnoty odporů, které reprezentují referenční odpor k teplotním čidlům v součtu s odporem jejich přívodních vodičů. Tyto referenční hodnoty nejsou inicializovány jako proměnné a není ani implementována funkce pro změnu těchto hodnot.

Původně to nebylo vyžadováno (proto také není uvedeno v zadání), nicméně během řešení této práce se vyskytl požadavek, že má být deska ovladatelná přes MODBUS. Vlivem různých průtahů při řešení však tato funkce implementová není. Na desce je tedy dle domluvy pouze hardware potřebný k tomuto ovládání (zapojení pro komunikační rozhraní RS485).

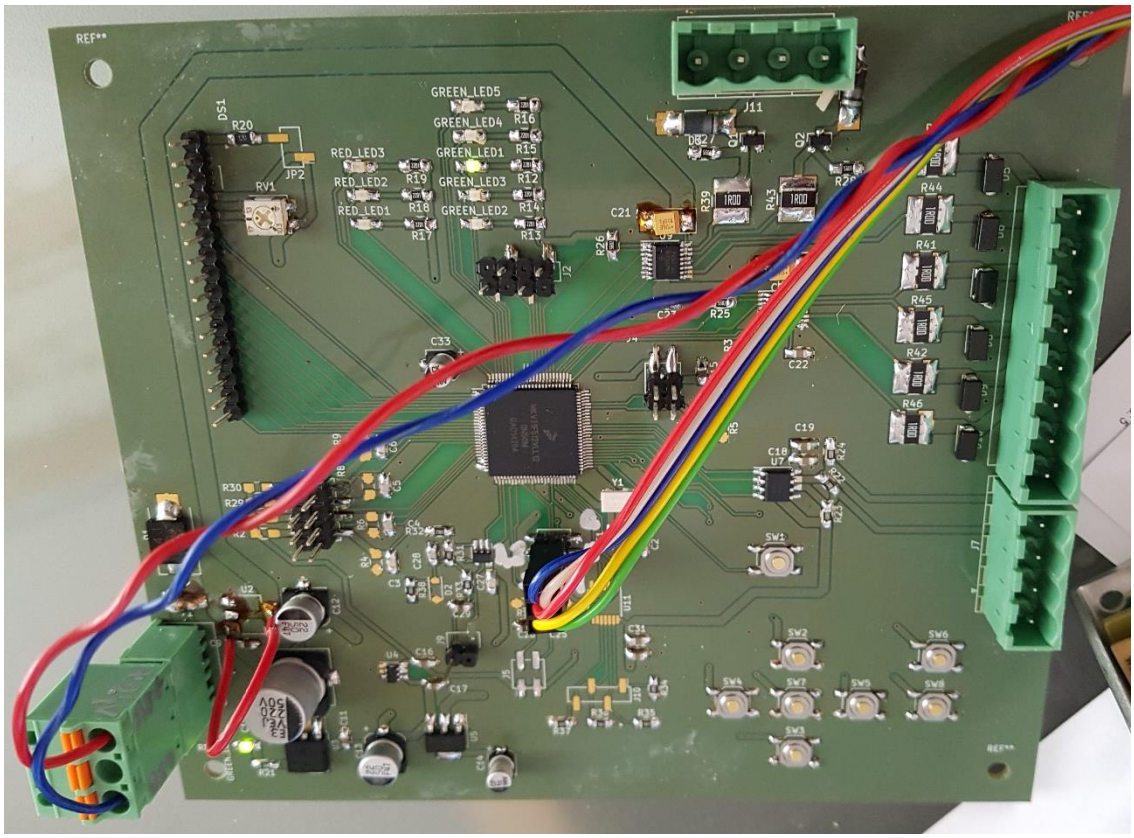
2. Hardware

Hlavní text diplomové práce obsahuje poměrně podrobný popis řídicí desky, zde jsou zcela konkrétně popsány jednotlivé konektory nacházející se na desce, případně další její důležité části.

Na obrázku níže je snímek téměř kompletně osazené řídicí desky. Ve stavu na obrázku je deska v základu funkční – nedostatky byly popsány v hlavním textu.

Zkráceně se tedy jedná o následující:

1. Na výstupu kanálu použitého pro PWM ovládání expanzního ventilu je výstup měřitelný na straně procesoru, nicméně za výkonovým driverem již ne. Prozatím nebyla zjištěna příčina tohoto jevu.
2. Při návrhu desky bylo omylem použito špatné číslování padů operačního zesilovače, takže nyní je připojen přes propojovací destičku (není na obrázku zachyceno)
3. Původní záměr byl použít co největší množství součástek pro SMD montáž. Toto se bohužel ukázalo jako nevhodné zejména s ohledem na připojovací piny s roztečí 2,54 mm. Při osazování desky se již zkracovala doba pro dokončení této práce a SMD piny nebyly k dispozici. Z tohoto důvodu jsou použity mírně upravené běžné piny, které, jak se ukázalo, nejsou pro tento účel příliš vhodné a při hrubším zacházení hrozí poškození desky.
4. Vlivem faktoru popsaného v bodu 3. byl již při pájení z desky omylem vytžen pin výstupu DAC1.

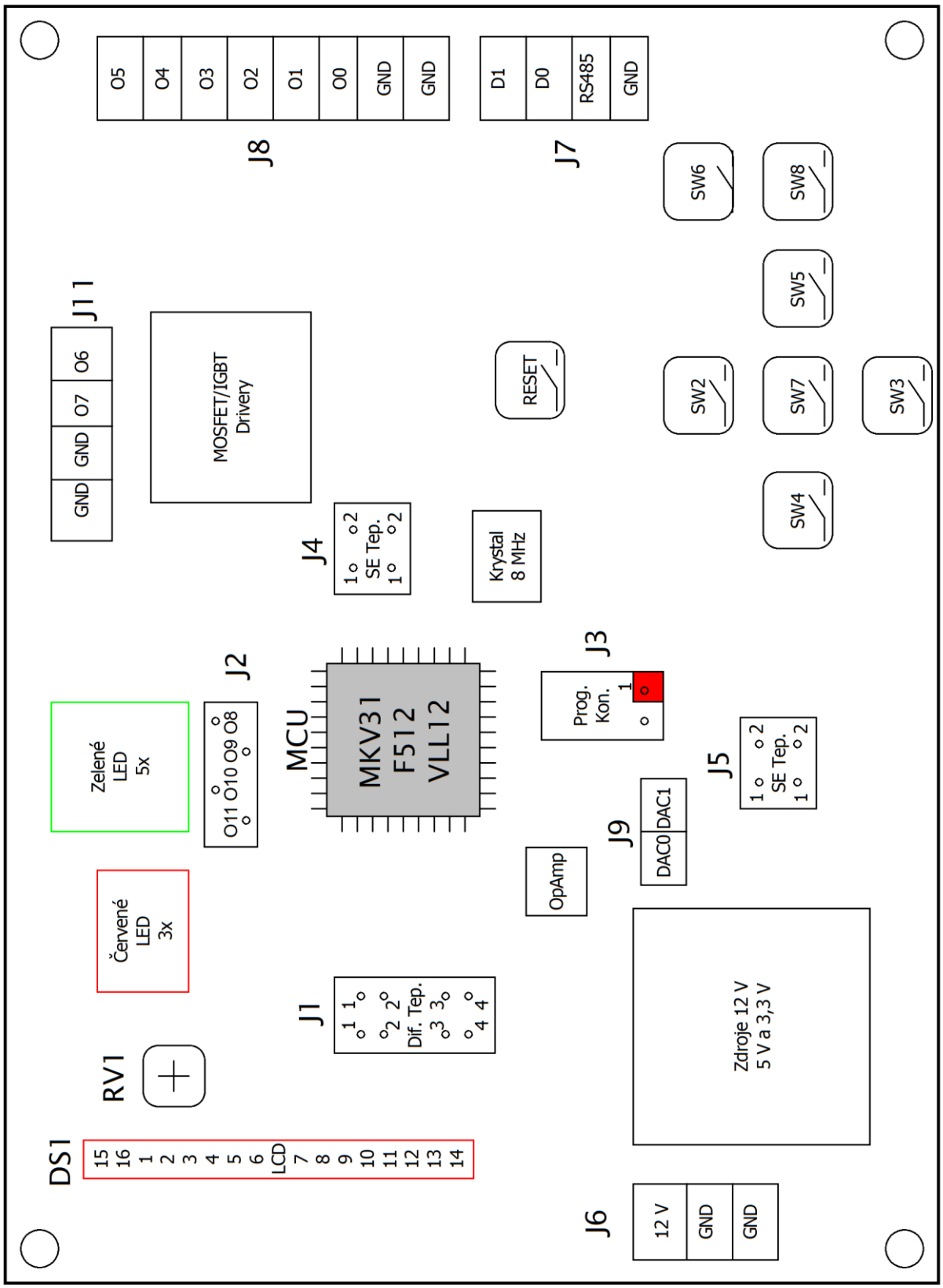


Obrázek 1 - Téměř kompletně osazená řídicí deska

Následující obrázky obsahují podrobné informace o celé řídicí desce. Obrázek níže je stylizovaným schématem pro znazší orientaci a má sloužit zejména pro usnadnění orientace na desce a připojování periferií.

UPOZORNĚNÍ – vlivem ztráty dat (HDD obsahující kompletně celý projekt a dokumentaci na konci roku 2017 byl defektní a nepodařilo se zachránit potřebná data) nemusejí následující schémata přesně reprezentovat zapojení řídicí desky. Po poměrně podrobné kontrole lze prohlásit, že souhlasí pozice a označení veškerých používaných výstupů – zejména je myšlena shoda číslování pinů procesoru a korespondujících výstupů na desce.

Dále je potřeba dodat, že část „výkonových“ výstupů je ovládána pomocí MOSFET/IGBT driverů. Pro reálné použití doporučuji vytvoření další desky, která bude obsahovat vhodně zapojené výkonové tranzistory tak, aby byla řídicí deska řádně chráněna proti destruktivním jevům a zároveň tyto drivery nebyly přetěžovány. Ovládání vnějších prvků (relé, SSR apod.) tedy bude následně zajištěno pomocí těchto tranzistorů.



Obrázek 2 - Schematické znázornění desky

DS1 je konektor pro display – číslováno dle použitého LCD (některá LCD mají číslování v řadě 1 až 16, u použitého kusu je číslování nejprve 15, 16 – napájení podsvícení – a teprve následně 1 až 14).

RV1 je nastavovací potenciometr kontrastu displaye.

J1 jsou piny s roztečí 2,54 mm pro teplotní senzory PT1000. Zapojovány jsou vždy mezi dvě stejná čísla (1-1, 2-2 atd.). V tomto případě by mělo souhlasit i číslování těchto pinů – při testování byla používána dvojice 4-4, která je v programu vedena jako teploměr 4.

J2 jsou piny zapojené přímo na nožičky procesoru – ozančení souhlasí s ozančením v programu – výstupy/vstupy O11 až O8.

J3 jsou piny určené pro programování desky dle domluvy – konektor 2x3 piny, červeně označený pin je pin číslo 1.

J4 a J5 jsou piny pro připojení dalších teplotních čidel PT1000. Opět se čidla připojují mezi piny se stejným číslem. V tomto případě již číslování nesouhlasí s programem a je třeba konzultovat schéma.

J6 je konektor s roztečí 5,08 mm a slouží k připojení napájení. Původně se mělo jednat o napájení 24 V, nicméně vlivem defektu na zdroji 12 V na desce je tento překrácen a napájecí napětí je nutné 12 V.

J7 je konektor s roztečí 5,08 mm pro rozhraní RS485. Vývody D1 a D0 jsou datové – viz schéma RS485, třetí pin je nezapojen (označení RS485) a čtvrtý pin je připojen na GND.

J8 a J11 jsou konektory s roztečí 5,08 mm zapojené na výstupy 12 V driverů (v případě výstupů O7 a O6 jsou 12 V drivery zapojeny na gate tranzistorů. Drain těchto tranzisotrů je zapojen na konektor. Označení O7 až O0 souhlasí s označením v programu.

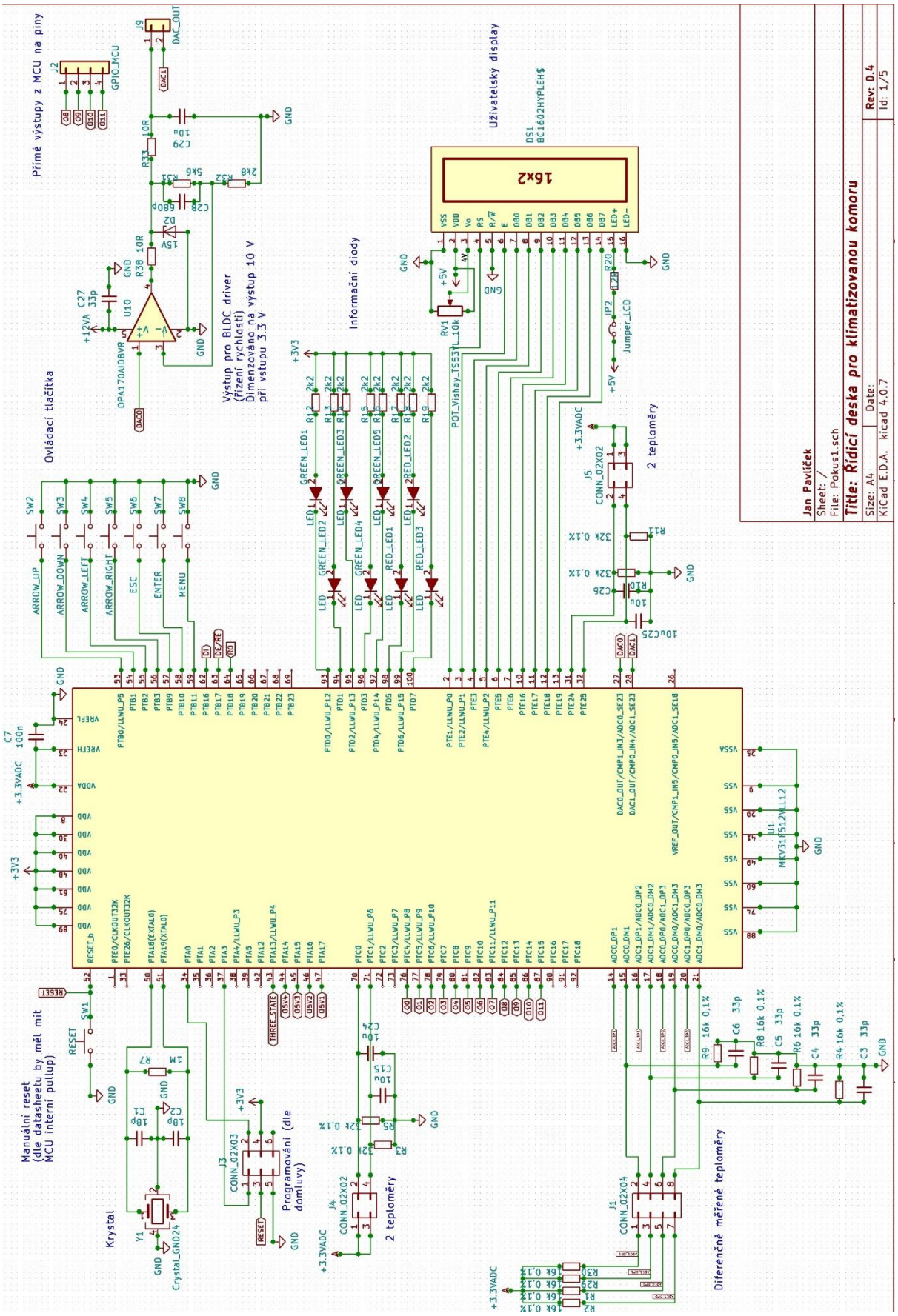
J9 jsou piny s roztečí 2,54 mm. Jedná se o výstupy ADC0 a ADC1. ADC1 je zapojeno přímo na pin procesoru. V případě ADC0 je výstup procesoru zapojen na operační zesilovač zapojený jako neinvertující zesilovač se zesílením 3.

SW2 až SW8 jsou uživatelská tlačítka. Označení souhlasí s označením v programu. Pro snadnější identifikaci jsou proměnně asociované s těmito tlačítky v programu označeny např. v případě SW2 jako buttonUP apod.

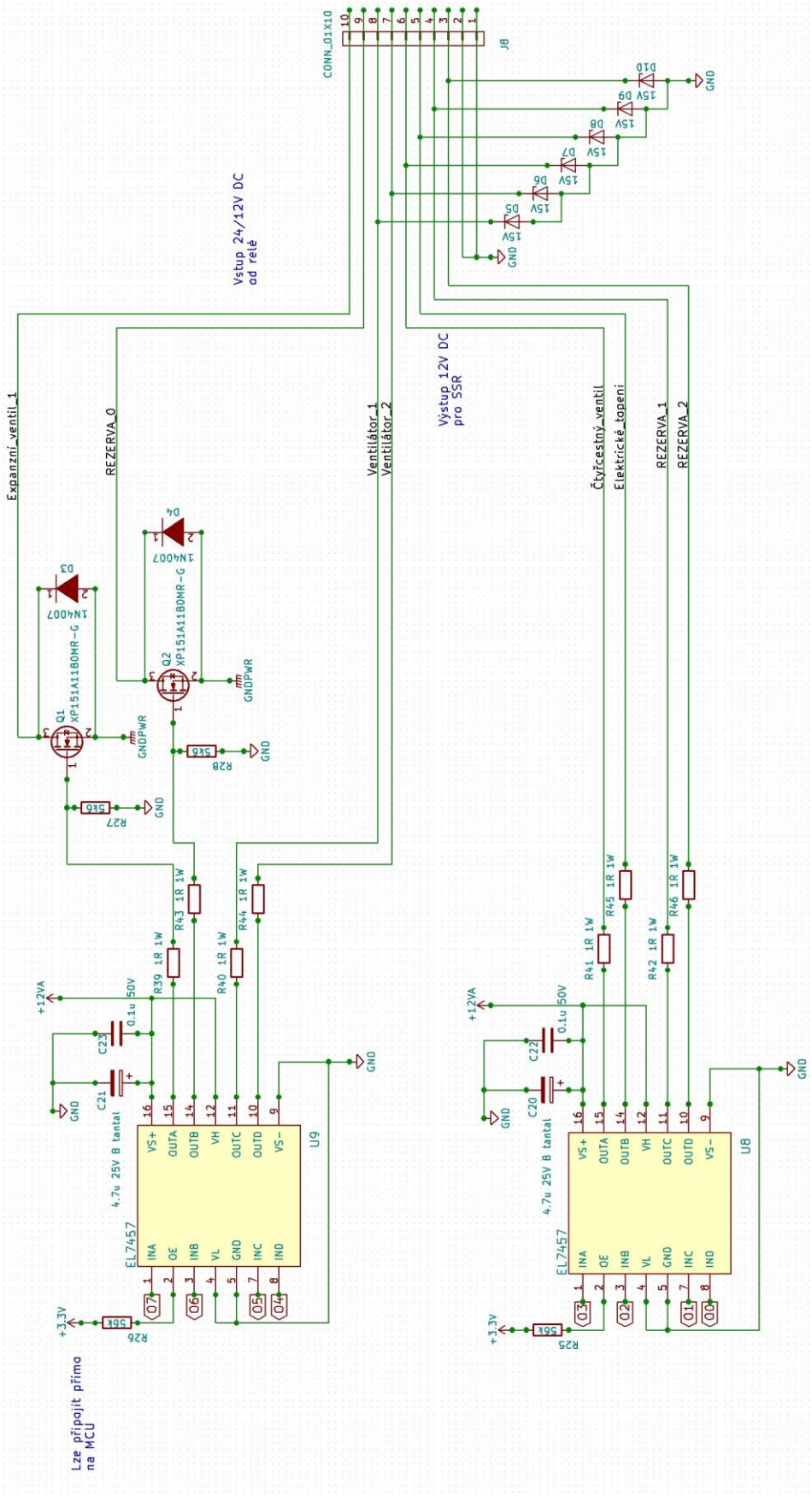
2.1. Schémata

Tato sekce obsahuje schémata popisující celou řídicí desku. Jak již bylo napsáno výše, nejnovější verze těchto schémat byly nenávratně ztraceny, proto je třeba některé detaily brát s rezevou a nemusí nutně reprezentovat aktuální stav desky.

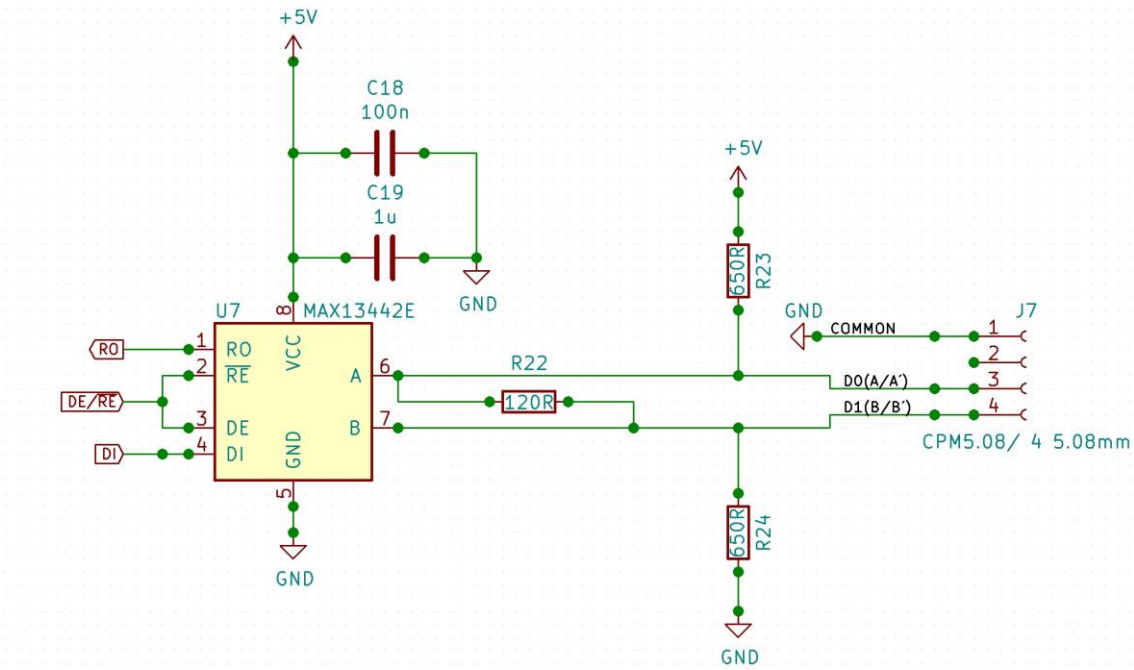
Hlavní informací celkového schématu je připojení jednotlivých periferií na procesor – tyto spoje byly zkontrolovány a odpovídají aktuálnímu stavu desky. Názvy tlačítek, LED diod a dalších komponent souhlasí s označením v programu – soubor pin_mux.h.



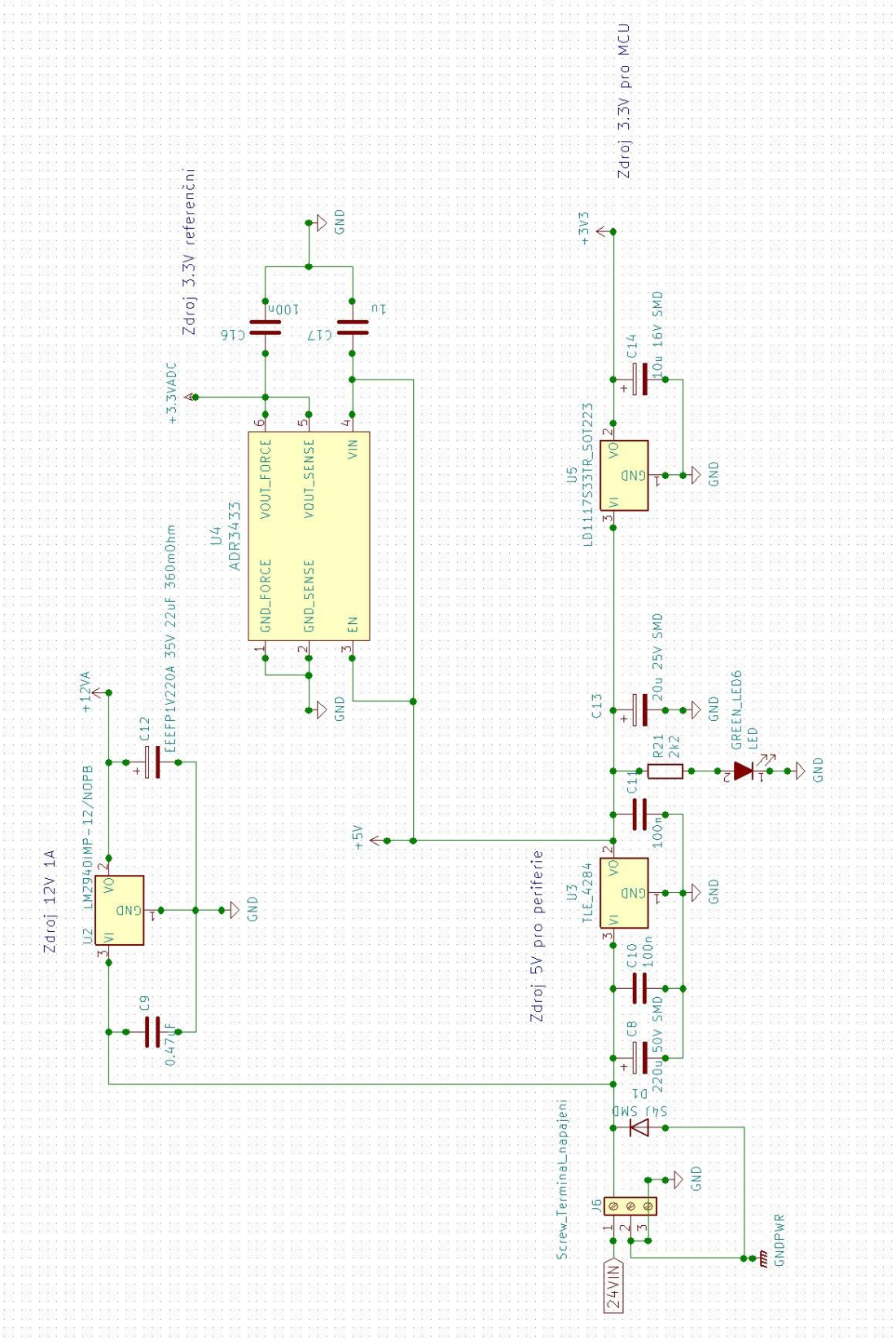
Obrázek 3 -Hlavní schéma řídicí desky



Obrázek 4 - Schéma výkonových výstupů



Obrázek 5 - Schéma zapojení rozhraní RS485



Obrázek 6 - Schéma zapojení zdrojů 12 V, 5 V a 3,3 V s 3,3 V referencí

3. Shrnutí

Tento návod obsahuje hlavní část programu řídicí desky pro klimatizovanou komoru. Kvůli časovému presu a absenci řízené soustavy již nebyl program reálně otestován, takže jeho funkčnost lze považovat pouze za teoretickou.

Uvedený program je tedy spíše potřeba považovat za jakýsi základ pro další úpravu a doladění, pokud by měla řídicí deska řádně plnit svoji funkci.

Po hradwarové stránce lze projekt taktéž považovat za funkční, nicméně se návrh a osazení neobešly bez některých chyb a je doporučeníhodné vytvořit novou, vylepšenou verzi této desky.

SEZNAM OBRÁZKŮ

Seznam použité literatury

Pokud se v tomto dokumentu nacházejí informace, které nejsou všeobecně známé nebo nejsou výsledkem této práce, zdroje těchto informací jsou uvedeny v hlavním textu této diplomové práce.