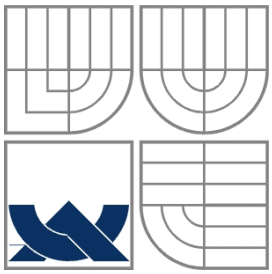
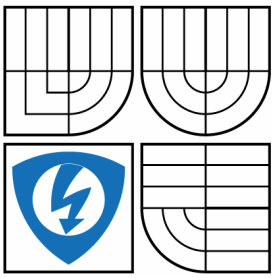


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ZÁZNAMOVÁ JEDNOTKA DO UL LETADLA

DATALOGGER FOR UL AIRCRAFT

PŘÍLOHA DIPLOMOVÉ PRÁCE

AUTOR PRÁCE — Bc. MICHAL MAREK
AUTHOR

VEDOUCÍ PRÁCE — Ing. STANISLAV KLUSÁČEK, Ph.D.
SUPERVISOR

BRNO 2013

Zdrojové kódy programu

Deska plošných spojů

Schéma

```
 1: /********     ZLD    *******/
 2:  /** Zapisovac letovych udaju **/
 3:  //     ***Michal Marek***
 4:  //
 5:  //
 6:  //
 7:  /* MCU PIC24FJ64GA004  */
 8:  //
 9:  //   pin               oznaceni        funkce          typ cidla
10:  //   2                 RC6             teplota OW       DS18B20
11:  //   3                 RC7  RP23       SD CS
12:  //   4                 RC8  RP24       SD MOSI
13:  //   5                 RC9  RP25       SD CLK
14:  //   8                 RB10 RP10       SD MISO
15:  //   10      AN12      RB12 AN12       AD rychlost      MPX5010DP
16:  //   11      AN11      RB13 AN11       AD vyska         MPX5100AP
17:  //   15      AN9       RB15 AN9        AD napeti
18:  //   25      AN6       RC0  AN6        AD gmetr X      MMA7260
19:  //   26      AN7       RC1  AN7        AD gmetr Y
20:  //   27      AN8       RC2  AN8        AD gmetr Z
21:  //   33                RB6  RP6        vstup RPM        citac impulsu
22:  //   36                RC3  RP19       Tx2 PC
23:  //   37                RC4  RP20       Rx2 PC
24:  //   38                RC5  RP21       Rx1 GPS           gps-04
25:  //
26:
27:  #include "gmetr.h"
28:  #include "gps_parse.h"
29:  #include "eeprom.h"
30:  #include "DS1820.h"
31:
32:
33: unsigned int rtc_ms=0;
34: unsigned int rtc_wh=0;
35: unsigned short rtc_s=0;
36: unsigned short rtc_m=0;
37: unsigned short rtc_h=0;
38:  char batteryFlag=0;
39:  char batteryCounter=0;
40: char ready=0;
41: char output[200];
42: char change = 0;
43: char vnc2OK=1;
44: struct gps dataGPS = {"0000000","00","0","000000","00000","00000","0",
    "0000000000","0","000000000","0","0000000000","000000",
    N00°00.0000","0000000000"};
45:
46: char kk=0;
47: float fx_g=0;
48: float fy_g=0;
49: float fz_g=0;
50: float temp_air;
51: unsigned error2;
52: unsigned short int offset[5];
53: char command = 0;
54: unsigned int rpm;
55: unsigned int rpm_tmp;
56: unsigned adc_value;
57: unsigned adc_vyska;
58: unsigned adc_rychlost;
59: unsigned adc_napeti_sit;
60: unsigned adc_napeti_bat;
61:  unsigned adc_teplota;
62:  unsigned pom_adc;
```

```
 63:   unsigned int time_old;
 64: unsigned int start_year;
 65: unsigned adc;
 66: float g;
 67: float temp_calc;
 68: float napeti_sit=0;
 69: float napeti_bat=0;
 70: unsigned rychlost;
 71: float vyska;
 72: unsigned head_temperature;
 73: float temp_pressure;
 74: char output2[50];
 75:  char  uart_rd;
 76:  char iic_read=2;
 77:  char iic_read2=0;
 78:   char  uart2_rd;
 79:  int vinbuf;
 80:  int l=0;
 81:  int lenStr=5;
 82:
 83: char txttmp[6];
 84: bit LED_Y ;
 85: bit LED_G ;
 86:  char txt[200];
 87:  char txt2[50];
 88: signed int latitude, longitude;
 89: char *string="aaaste";
 90: char *g_string="aaa";
 91: char *pom="00000";
 92: char string2[6];
 93:   int ii=0;
 94:   int ii2=0;
 95: char filename[]="123456AC.csv";
 96: char fileset = 0;
 97: char filewait = 0;
 98:    // MMC module connections
 99: sbit Mmc_Chip_Select           at LATC7_bit;  // for writing to output pin
    always use latch
100: sbit Mmc_Chip_Select_Direction at TRISC7_bit;
101: // eof MMC module connections
102: char err_txt[20]       = "FAT16 not found";
103: unsigned short loop, loop2;
104: unsigned long  i, size;
105: char           Buffer[512];
106: char counter=0;
107:
108:  void uartinterrupt() iv IVT_ADDR_U1RXINTERRUPT ics ICS_AUTO {
109:    if (U1RXIF_bit == 1) {
110:      uart_rd = UART1_Read();
111:      ii++;
112:      if ( uart_rd == '$') ii=0;
113:      txt[ii] = uart_rd;
114:     if (uart_rd == 13)
115:     {
116:       pom = memcpy(output,txt,ii+1);
117:       ready = 1;                    // Ready for parsing GPS data
118:     }
119:     }
120:     U1RXIF_bit = 0;                // Set RCIF to 0
121:   }
122:
123: void M_Open_File_Append() {
124:  Mmc_Fat_Write(Buffer, lenStr);
125: }
```

```
126:
127: void IntDetection() org 0x003C{ //Interrupt on INT1
128:   rpm_tmp++;
129:   IFS1.F4 = 0;               //interrupt flag cleared
130: }
131:
132: void RTCInterrupt() org 0x000090{
133:   RTCIF_bit= 0;
134:   ALRMEN_bit=1;
135:   change=1;
136:
137:   rpm=rpm_tmp*60;
138:   rpm_tmp=0;
139:
140: }
141:
142: void rtc_get(){
143:      RTCWREN_bit=1;                          // povolení úprav
144:      if (!RTCSYNC_bit)                       // probíhající synchronizace?
145:         {
146:         RTCPTR_0_bit=1;                      // nastavení ukazatele na registr
147:         RTCPTR_1_bit=0;                      // nastavení ukazatele na registr
148:         rtc_wh=RTCVAL;                       // čtení registru WKDYHR,
149:                                              // ukazatel automaticky -1
150:         rtc_ms=RTCVAL;                       // čtení registru MINSEC
151:         }
152:      RTCWREN_bit=0;                          // zakázání úprav
153:      rtc_s= bcd2dec(rtc_ms & 0xFF);          // zpracování hodnot
154:      rtc_m= bcd2dec((rtc_ms>>8)& 0xFF) ;     // zpracování hodnot
155:      rtc_h= bcd2dec(rtc_wh & 0xFF);          // zpracování hodnot
156:  }
157:
158: void vinculumSend(char* datasend, int len)
159: {
160: int sendcounter=0;
161:  for (sendcounter=0;sendcounter<len; sendcounter++)
162:  {
163:   while(LATC2_bit==1);
164:    asm CLRWDT;
165:   if (datasend[sendcounter]){
166:    UART2_Write(datasend[sendcounter]);
167:     }
168:  }
169: }
170: void onerror ()
171: {
172:   while(1)
173:   {
174:    LATA1_bit=1;
175:    Delay_ms(150);
176:     LATA1_bit=0;
177:    Delay_ms(150);
178:   }
179:  }
180:  void init()
181: {
182:       asm mov #0x742, w1
183:     asm mov #0x46, w2
184:     asm mov #0x57, w3
185:     asm mov.b #0x02, w0
186:     asm mov.b w2, [w1]
187:     asm mov.b w3, [w1]
188:     asm mov.b w0, [w1]
189:
```

```
190:    asm disi #5
191:     asm mov #0x55, w7
192:     asm mov w7, NVMKEY
193:     asm mov #0xAA, w8
194:     asm mov w8, NVMKEY
195:     asm bset RCFGCAL, #13
196:     asm nop
197:     asm nop
198:
199: RTCWREN_bit=1;
200: RTCPTR_0_bit=1;
201:  RTCPTR_1_bit=0;
202:  RTCVAL=0;
203:  RTCVAL=0;
204:  AMASK_0_bit=1;
205:  AMASK_1_bit=0;
206:  AMASK_2_bit=0;
207:  AMASK_3_bit=0;
208:  ALRMEN_bit=1;
209: RTCEN_bit=1;
210: RTCWREN_bit=0;
211:
212:    ADPCFG = 0x003F;          // Initialize AN pins as digital
213:
214:   TRISA  = 0x0600;              // Initialize PORTA as output
215:   LATA   = 0x0000;          // Initialize PORTA value
216:   TRISB  = 0xE010;              // Initialize PORTB as output
217:   LATB   = 0x0000;          // Initialize PORTB value
218:   TRISC  = 0x0135;              // Initialize PORTC as output
219:   LATC   = 0x0000;          // Initialize PORTC value
220:   ODC1_bit=1;
221:   IPC5   = IPC5 | 0x0010;
222:   IPC2   = IPC2 | 0x7000;
223:
224:  LATB12_bit = 1;      // batery rele on
225:  LATC1_bit = 0;       // reset vinculum
226:  LATA8_bit = 0;       // reset gps
227:  LATA0_bit = 0;       // vnc cts
228:   Unlock_IOLOCK();
229:   PPS_Mapping_NoLock(24, _OUTPUT,_SDO1);  // MOSI
230:   PPS_Mapping_NoLock(25, _OUTPUT,_SCK1OUT);  // CLK
231:   PPS_Mapping_NoLock(10, _INPUT,_SDI1);  // MISO
232:   PPS_Mapping_NoLock(6, _INPUT,_INT1);  // RPM
233:   PPS_Mapping_NoLock(21, _INPUT,_U1RX);  // GPS RX
234:   PPS_Mapping_NoLock(7, _OUTPUT,_U1TX);  // GPS TX
235:   PPS_Mapping_NoLock(19, _INPUT,_U2RX);  // PC RX
236:   PPS_Mapping_NoLock(20, _OUTPUT,_U2TX);  // PC TX
237:   Lock_IOLOCK();
238:
239:    ADC1_Init();
240:
241:   asm CLRWDT;
242:   ready = 0;                      // Initialize variables
243:   i = 0;
244:   counter=0;
245:   fileset = 0;
246:  filewait = 0;
247: SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1,
     _SPI_PRESCALE_PRI_64,
248:                   _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH,
     _SPI_ACTIVE_2_IDLE);
249: // LATA1_bit =~ LATA1_bit;
250:    Delay_ms(300);
251:   UART1_Init(9600);               // Initialize UART module at 9600
```

```
252:    Delay_ms(300);
253:    UART2_Init(115200);               // Initialize UART module at 57600
254:    Delay_ms(300);
255: //LATA1_bit =~ LATA1_bit;
256:    DS18B20_init(&PORTC,6);
257:  //  Delay_ms(100);
258:  //  UART2_Write_Text("RESET");
259:  // UART2_Write(13);
260:  URXISEL_1_U1STA_bit = 0;
261:  URXISEL_1_U2STA_bit = 0;
262:   NSTDIS_bit = 1;                  // no nesting of interrupts
263:   U1RXIF_bit = 0;                    // ensure interrupt not pending
264:   U1RXIE_bit = 1;
265:   U2RXIF_bit = 0;                    // ensure interrupt not pending
266:   U2RXIE_bit = 1;
267:
268:  INT1IE_bit=1;
269:   INT1EP_bit=1;
270:  //--- init the FAT library
271: if (!Mmc_Fat_Init()) {
272:      // reinitialize spi at higher speed
273:     SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1,
     _SPI_PRESCALE_PRI_4,
274:                        _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE,
     _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE);
275:   }
276:   else {
277:      onerror ();
278:    }
279:
280:  asm CLRWDT;
281:  I2C1_Init(100000);
282:    Delay_ms(100);
283:   TRISB.B2 = 1;
284:   TRISB.B3 = 1;
285: I2C2_Init(100000);
286:    Delay_ms(100);
287:    asm CLRWDT;
288:
289: }
290: void napeti()
291: {
292:    adc_napeti_sit = ADC1_Get_Sample(9);
293:   adc_napeti_bat = ADC1_Get_Sample(10);
294:      napeti_sit = adc_napeti_sit*0.0322266*1.15;
295:      if (napeti_sit>10)
296:      {
297:       batteryFlag='M';
298:        LATA1_bit =0;
299:       }
300:       else
301:       {
302:          napeti_bat = adc_napeti_bat*0.0322266*1.17;
303:       batteryFlag='B';
304:       LATA1_bit =~ LATA1_bit;
305:       if ((rpm<100)&&(rychlost==0))
306:       {
307:        batteryCounter++;
308:        asm CLRWDT;
309:        if(batteryCounter>2)
310:        {
311:        LATA1_bit=0;
312:        Delay_ms(100);
313:         LATA1_bit=1;
```

```
314:         Delay_ms(100);
315:         LATA1_bit=0;
316:         Delay_ms(100);
317:         LATA1_bit=1;
318:         Delay_ms(100);
319:         Mmc_Fat_Close();
320:         LATB12_bit=0;
321:         }
322:       }
323:       else
324:           {
325:           batteryCounter=0;
326:           }
327:       }
328:  }
329: void main() {
330:
331:    init();
332:    Delay_ms(1000);
333:     asm CLRWDT;
334:    if (Button(&PORTA, 9, 10, 0))
335:      {
336:       LATA7_bit=1;
337:       LATC1_bit = 1;  // start vinculum
338:
339:       while(1)
340:       {
341:       asm CLRWDT;
342:       LATA1_bit =~ LATA1_bit;
343:       adc_napeti_sit = ADC1_Get_Sample(9);
344:      napeti_sit = adc_napeti_sit*0.0322266;
345:      if (napeti_sit<10)
346:      LATB12_bit=0;
347:       Delay_ms(1000);
348:       }
349:      }
350:     LATA8_bit=1;   // start GPS
351:     LATA1_bit=0;
352:     LATC1_bit = 1;  // start vinculum
353:       asm CLRWDT;
354:     Delay_ms(1000);
355:
356:   get_offset(&offset);
357:    asm CLRWDT;
358:  RTCIE_bit=1;
359:   while(1)
360:   {
361:  OERR_bit = 0;                  // Set OERR to 0
362:  FERR_bit = 0;                  // Set FERR to 0
363:
364:   if (ready==1){
365:   ready = gps_parse_f(&dataGPS,&output);
366:    }
367:    if ((fileset==0)&&(filewait==0))
368:    {
369:    if (strcmp(datagps.Date, "000000"))
370:    {
371:     filename[0]=dataGPS.Date[4];
372:     filename[1]=dataGPS.Date[5];
373:     filename[2]=dataGPS.Date[2];
374:     filename[3]=dataGPS.Date[3];
375:     filename[4]=dataGPS.Date[0];
376:     filename[5]=dataGPS.Date[1];
377:     Mmc_Fat_Assign(filename, 0x80);
```

```
378:     Mmc_Fat_Append();                                    // Prepare file for
    append
379:     vinculumSend(filename,8);
380:     filewait=1;
381:     LATA1_bit=1;
382:    }
383:     loop++;
384:    if (loop>50)
385:    {
386:    LATA1_bit =~ LATA1_bit;
387:    loop=0;
388:    }
389:    Delay_ms(10);
390:    }
391:    if (filewait==1) {
392:       if(vnc2OK==1) fileset=1;
393:                    }
394:    if ((change==1)&&(fileset==1)) {
395:       asm CLRWDT;
396:      napeti();
397:      gmetr_f(&fx_g, &fy_g, &fz_g, &offset);
398:      asm CLRWDT;
399:     adc_teplota = ADC1_Get_Sample(6);
400:     temp_air=DS18B20_teplota();
401:
402: I2C2_Start();              // issue I2C start signal
403:   I2C2_Write(0x9B);          // send byte via I2C  (device address + R)
404:   adc_rychlost = I2C2_Read(0);      // Read the data (NO acknowledge)
405:   adc_rychlost = (adc_rychlost<<8) + I2C2_Read(1);      // Read the data (NO
    acknowledge)I2C1_Read(0);      // Read the data (NO acknowledge)
406:   I2C2_Stop();
407:
408:   I2C1_Start();              // issue I2C start signal
409:   I2C1_Write(0x9B);          // send byte via I2C  (device address + R)
410:   adc_vyska = I2C1_Read(0);      // Read the data (NO acknowledge)
411:   adc_vyska = (adc_vyska<<8) + I2C1_Read(1);      // Read the data (NO
    acknowledge)I2C1_Read(0);      // Read the data (NO acknowledge)
412:   I2C1_Stop();
413:   asm CLRWDT;
414:
415:     counter++;
416:     if (counter>20){
417:         if ((!strcmp(datagps.Altitude, "0000000"))|(!strcmp(datagps.Longitude,
    "0000000000")))
418:          { asm RESET; }
419:     else {counter=0;}
420:          }
421:     temp_calc=3.6*sqrt(4.4288 * adc_rychlost - 725.624) + *offset - 128 ;
422:     if (temp_calc <40 ) temp_calc = 0;
423:     rychlost = temp_calc;
424:     asm CLRWDT;
425:     temp_pressure = 27.1267*adc_vyska + 10555.56;
426:
427:   temp_calc = -(log(temp_pressure)-11.5261)*27671.23  + *(offset+1) - 128  ;
428:       vyska = temp_calc;
429:
430:       temp_calc=-35*log(adc_teplota*0.00322266) + 41;
431:       head_temperature=temp_calc;
432:      rtc_get();
433:
434:     asm CLRWDT;
435:    sprintf(Buffer,"%.2s-%.2s-20%.2s,%.2s:%.2s:%.2s,%s%.3s°%.2s.%.4s
    %s%.2s°%.2s.%.4s,%s,%s,%s,%.2s:%.2s:%.2s,%s,%s,%02u:%02u:%02u,%u,%.1f,%.1f,%1.2f,
    %1.2f,%1.2f,%c,%.2f,%.2f,%u,%u,%.2f\n",
```

```
436:       datagps.Date,datagps.Date+2,datagps.Date+4,
437:       datagps.UTC_Time,datagps.UTC_Time+2,datagps.UTC_Time+4,
438:       dataGPS.Indicator_E_or_W,datagps.Longitude,datagps.Longitude+3,
     datagps.Longitude+6,datagps.Indicator_N_or_S,datagps.Latitude,datagps.Latitude+2,
     datagps.Latitude+5,
439:       datagps.RMC_Status,
440:       datagps.Course_Over_Ground,
441:       datagps.Speed_Over_Ground,
442:       datagps.UTC_Time_Alt,datagps.UTC_Time_Alt+2,datagps.UTC_Time_Alt+4,
443:       datagps.Altitude,
444:       datagps.Satelit,
445:       rtc_h,rtc_m,rtc_s,
446:      rychlost,//rychlost
447:      vyska,temp_pressure,//vyska
448:      fx_g,
449:      fy_g,
450:      fz_g,
451:       batteryFlag,napeti_sit,//napeti sit
452:        napeti_bat,//napeti baterie
453:      rpm,
454:      head_temperature,//adc_teplota,//teplota hlav
455:      temp_air
456:        );
457:
458:     asm CLRWDT;
459: lenStr= strlen(Buffer);
460:  M_Open_File_Append();
461:   asm CLRWDT;
462:   vinculumSend(&Buffer, lenStr);
463:     change=0;
464:     }
465:     }
466: }
467:
468:
469:
470: ********************************************************************
471: gps_parse.h
472: ********************************************************************
473: #ifndef GPS_PARSE_H
474: #define GPS_PARSE_H
475:
476:  struct gps{
477:      char Altitude[8];
478:      char Satelit[3];
479:      char Mode[2];
480:      char Date[7];
481:      char Course_Over_Ground[6];
482:      char Speed_Over_Ground[6];
483:      char Indicator_E_or_W[2];
484:      char Longitude[11];
485:      char Indicator_N_or_S[2];
486:      char Latitude[10];
487:      char RMC_Status[2];
488:      char UTC_Time[11];
489:      char Message_ID[7];
490:      char Position[25];
491:      char UTC_Time_Alt[11];
492: } ;
493:
494:
495: char gps_parse_f(struct gps * dataGPS, char*output);
496:
497: #endif
```

```
498:
499: ********************************************************************
500: gps_parse.c
501: ********************************************************************
502: #include "gps_parse.h"
503:
504:
505:  char j=0;
506:  char k=0;
507:  char nn=0;
508:
509:
510: char gps_parse_f(struct gps * dataGPS, char* output)
511:  {
512:      asm CLRWDT;
513:         if (output[0]=='$')
514:        {if (output[1]=='G')
515:        {if (output[2]=='P')
516:        {if (output[3]=='R')
517:        {if (output[4]=='M')
518:        {if (output[5]=='C')
519:        { j=1; }}}}}}
520:
521:        if (output[0]=='$')
522:        {if (output[1]=='G')
523:        {if (output[2]=='P')
524:        {if (output[3]=='G')
525:        {if (output[4]=='G')
526:        {if (output[5]=='A')
527:        { j=2; }}}}}}
528:
529:      if(j == 1) {
530:       k=0;
531:       nn=0;
532:        while(output[k]!=',')
533:        {
534:         if (nn>6) break;
535:         dataGPS->Message_ID[nn]=output[k];
536:         nn++;
537:        k++;
538:        }
539:        k++;
540:       dataGPS->Message_ID[nn]='\0';
541:       nn=0;
542:        while(output[k]!=',')
543:        {
544:         if (nn>10) break;
545:         dataGPS->UTC_Time[nn]=output[k];
546:         nn++;
547:        k++;
548:        }
549:        k++;
550:         dataGPS->UTC_Time[nn]='\0';
551:       nn=0;
552:        while(output[k]!=',')
553:        {
554:         if (nn>1) break;
555:         dataGPS->RMC_Status[nn]=output[k];
556:         nn++;
557:        k++;
558:        }
559:        k++;
560:        dataGPS->RMC_Status[nn]='\0';
561:        nn=0;
```

```
562:        while(output[k]!=',')
563:        {
564:         if (nn>9) break;
565:         dataGPS->Latitude[nn]=output[k];
566:         nn++;
567:        k++;
568:        }
569:        k++;
570:        dataGPS->Latitude[nn]='\0';
571:        nn=0;
572:        while(output[k]!=',')
573:        {
574:         if (nn>1) break;
575:         dataGPS->Indicator_N_or_S[nn]=output[k];
576:         nn++;
577:        k++;
578:        }
579:        k++;
580:        dataGPS->Indicator_N_or_S[nn]='\0';
581:        nn=0;
582:        while(output[k]!=',')
583:        {
584:         if (nn>10) break;
585:         dataGPS->Longitude[nn]=output[k];
586:         nn++;
587:        k++;
588:        }
589:        k++;
590:        dataGPS->Longitude[nn]='\0';
591:        nn=0;
592:        while(output[k]!=',')
593:        {
594:         if (nn>1) break;
595:         dataGPS->Indicator_E_or_W[nn]=output[k];
596:         nn++;
597:        k++;
598:        }
599:        k++;
600:        dataGPS->Indicator_E_or_W[nn]='\0';
601:        nn=0;
602:        while(output[k]!=',')
603:        {
604:         if (nn>5) break;
605:         dataGPS->Speed_Over_Ground[nn]=output[k];
606:         nn++;
607:        k++;
608:        }
609:        k++;
610:        dataGPS->Speed_Over_Ground[nn]='\0';
611:        nn=0;
612:        while(output[k]!=',')
613:        {
614:         if (nn>5) break;
615:         dataGPS->Course_Over_Ground[nn]=output[k];
616:         nn++;
617:        k++;
618:        }
619:        k++;
620:        dataGPS->Course_Over_Ground[nn]='\0';
621:        nn=0;
622:        while(output[k]!=',')
623:        {
624:         if (nn>6) break;
625:         dataGPS->Date[nn]=output[k];
```

```
626:         nn++;
627:        k++;
628:        }
629:         asm CLRWDT;
630:        dataGPS->Date[nn]='\0';
631:         dataGPS->Mode[0]=output[k+3];
632:        dataGPS->Mode[1]='\0';
633:
634:           j=0;
635:           }
636:       asm CLRWDT;
637:      if(j == 2) {
638:        // LATA1_bit = ~LATA1_bit;
639:       k=0;
640:       nn=0;
641:        while(output[k]!=',')
642:        {
643:         if (nn>6) break;
644:         dataGPS->Message_ID[nn]=output[k];
645:         nn++;
646:        k++;
647:        }
648:        k++;
649:        dataGPS->Message_ID[nn]='\0';
650:       nn=0;
651:        while(output[k]!=',')
652:        {
653:         if (nn>10) break;
654:         dataGPS->UTC_Time_Alt[nn]=output[k];
655:         nn++;
656:        k++;
657:        }
658:        k++;
659:         dataGPS->UTC_Time[nn]='\0';
660:       nn=0;
661:        while(output[k]!=',')
662:        {
663:        k++;
664:        }
665:        k++;
666:        while(output[k]!=',')
667:        {
668:        k++;
669:        }
670:        k++;
671:        while(output[k]!=',')
672:        {
673:        k++;
674:        }
675:        k++;
676:        while(output[k]!=',')
677:        {
678:        k++;
679:        }
680:        k++;
681:        while(output[k]!=',')
682:        {
683:        k++;
684:        }
685:        k++;
686:        nn=0;
687:        while(output[k]!=',')
688:        {
689:         if (nn>2) break;
```

```
690:         dataGPS->Satelit[nn]=output[k];
691:         nn++;
692:        k++;
693:        }
694:        k++;
695:        dataGPS->Satelit[nn]='\0';
696:        while(output[k]!=',')
697:        {
698:        k++;
699:        }
700:        k++;
701:        nn=0;
702:        while(output[k]!=',')
703:        {
704:         if (nn>7) break;
705:         dataGPS->Altitude[nn]=output[k];
706:         nn++;
707:        k++;
708:        }
709:        k++;
710:        dataGPS->Altitude[nn]='\0';
711:          j=0;
712:           }
713:          asm CLRWDT;
714:     return 0;
715:     }
716:
717:
718: ********************************************************************
719: ds1820.h
720: ********************************************************************
721: #ifndef _DS1820_H
722: #define _DS1820_H
723:
724:  void DS18B20_init(char * iport, unsigned int ipin);
725:  float DS18B20_teplota();
726:
727: #endif
728:
729:
730: ********************************************************************
731: ds1820.c
732: ********************************************************************
733:
734:
735:  unsigned int xpin;
736:  char * xport;
737:  unsigned int temp;
738:  unsigned int temp_fraction;
739:  float temp_out=0.0;
740:  char text[8];
741:  signed char negative;
742:
743:
744: void DS18B20_init()
745: {
746:     Ow_Reset(&PORTC, 6);                        // Reset signál
747:     Ow_Write(&PORTC, 6, 0xCC);                  // Příkaz SKIP_ROM
748:     Ow_Write(&PORTC, 6, 0x4E);                  // Příkaz nastavení Th+Tl+config
749:     Ow_Write(&PORTC, 6, 0x00);                  // Th
750:     Ow_Write(&PORTC, 6, 0x00);                  // Tl
751:     Ow_Write(&PORTC, 6, 0x3F);                  // config - 10bit rozlišení
752: }
753:
```

```
754: float DS18B20_teplota()
755: {
756:     Ow_Reset(&PORTC, 6);                          // Reset signál
757:     Ow_Write(&PORTC, 6, 0xCC);                    // Příkaz SKIP_ROM
758:     Ow_Write(&PORTC, 6, 0x44);                    // Příkaz CONVERT_T
759:     Delay_us(120);                                // Doba nutná pro převod T
760:     Ow_Reset(&PORTC, 6);                          // Reset signál
761:     Ow_Write(&PORTC, 6, 0xCC);                    // Příkaz SKIP_ROM
762:     Ow_Write(&PORTC, 6, 0xBE);                    // Příkaz READ_SCRATCHPAD
763:     temp =  Ow_Read(&PORTC, 6);                   // čtení prvního byte
764:     temp = (Ow_Read(&PORTC, 6) << 8) + temp;      // čtení druhého byte
765:
766:     if (temp & 0x8000)                            // kontrola záporné teploty
767:       {
768:       temp = ~temp + 1;                           // převod na kladné číslo
769:       negative = -1;                              // nastavení koeficeintu
770:       }
771:     else
772:       {
773:       negative = 1;                               // nastavení koeficeintu
774:       }
775:
776:     temp_out =  temp >> 4;                        // získání celé části
777:     temp_fraction  = temp >> 2 ;                  // získání desetinné části
778:     temp_fraction &= 0x03;                        // získání desetinné části
779:     temp_out = temp_out + temp_fraction * 0.25;   // výpočet teploty
780:     temp_out *= negative;                         // výpočet teploty
781:
782:     return temp_out;                              // zjištěná teplota
783: }
784:
785:
786:
787:
788: ********************************************************************
789: gmetr.h
790: ********************************************************************
791:  #ifndef GMETR_H
792: #define GMETR_H
793:
794: void gmetr_f(float * x, float * y, float * z,unsigned short int * offset);
795:
796: #endif
797:
798: ********************************************************************
799: gmetr.c
800: ********************************************************************
801: #include "gmetr.h"
802:
803:  signed char x_g;
804:  signed char y_g;
805:  signed char z_g;
806:
807:  void gmetr_f(float * x, float * y, float * z, unsigned short int * offset){
808:    I2C1_Start();              // issue I2C start signal
809:   I2C1_Write(0x3A);          // send byte via I2C  (device address + W)
810:   I2C1_Write(0x16);             // send byte (address of EEPROM location)
811:   I2C1_Write(0x01);          // send data (data to be written)
812:   I2C1_Stop();               // issue I2C stop signal
813:   I2C1_Start();              // issue I2C start signal
814:   I2C1_Write(0x3A);          // send byte via I2C  (device address + W)
815:   I2C1_Write(0x06);             // send byte (data address)
816:   I2C1_Restart();            // issue I2C signal repeated start
817:   I2C1_Write(0x3B);          // send byte (device address + R)
```

```
818:   x_g = I2C1_Read(1);      // Read the data (NO acknowledge)
819:   I2C1_Stop();              // issue I2C stop signal
820:   I2C1_Start();             // issue I2C start signal
821:   I2C1_Write(0x3A);         // send byte via I2C  (device address + W)
822:   I2C1_Write(0x07);            // send byte (data address)
823:   I2C1_Restart();           // issue I2C signal repeated start
824:   I2C1_Write(0x3B);         // send byte (device address + R)
825:   y_g = I2C1_Read(1);      // Read the data (NO acknowledge)
826:   I2C1_Stop();              // issue I2C stop signal
827:   I2C1_Start();             // issue I2C start signal
828:   I2C1_Write(0x3A);         // send byte via I2C  (device address + W)
829:   I2C1_Write(0x08);            // send byte (data address)
830:   I2C1_Restart();           // issue I2C signal repeated start
831:   I2C1_Write(0x3B);         // send byte (device address + R)
832:   z_g = I2C1_Read(1);      // Read the data (NO acknowledge)
833:   I2C1_Stop();              // issue I2C stop signal
834:
835:   *x = 0.063*(x_g+*(offset+2)-128);
836:   *y = 0.063*(y_g+*(offset+3)-128);
837:   *z = 0.063*(z_g+*(offset+4)-128);
838:  }
839:
840:
841:
842: ********************************************************************
843: eeprom.h
844: ********************************************************************
845:
846: #ifndef EEPROM_H
847: #define EEPROM_H
848:
849: void set_offset(signed int * offset);
850: void get_offset(signed int * offset);
851:
852: #endif
853:
854: ********************************************************************
855: eeprom.c
856: ********************************************************************
857: #include "eeprom.h"
858:
859: char ei=0;
860: char ea=0;
861:
862:  void set_offset(unsigned short int * offset){
863:
864:   for ( ei = 0; ei<5; ei++ )
865:   {
866:    ea = *(offset+ei);
867:   I2C2_Start();             // issue I2C start signal
868:   I2C2_Write(0xA0);         // send byte via I2C  (device address + W)
869:   I2C2_Write(ei+10);            // send byte (address of EEPROM location)
870:   I2C2_Write(ea);          // send data (data to be written)
871:   I2C2_Stop();              // issue I2C stop signal
872:   Delay_5ms();
873:
874:   }
875:
876:  }
877:
878: void get_offset(unsigned short int * offset){
879:
880:   for ( ei = 0; ei<5; ei++ )
881:   {
```

```
882:   I2C2_Start();              // issue I2C start signal
883:   I2C2_Write(0xA0);          // send byte via I2C  (device address + W)
884:   I2C2_Write(ei+10);             // send byte (address of EEPROM location)
885:   I2C2_Restart();            // issue I2C signal repeated start
886:   I2C2_Write(0xA1);          // send byte (device address + R)
887:   *(offset+ei) =  I2C2_Read(1);      // Read the data (NO acknowledge)
888:   I2C2_Stop();               // issue I2C stop signal
889:   Delay_5ms();
890:     }
891: }
892:
893:
894:
895: ********************************************************************
896: ********************************************************************
897: Vinculum-II
898: ********************************************************************
899: ********************************************************************
900:
901: /*
902: ** Filename: ZLD_iomux.c
903: **
904: ** Automatically created by Application Wizard 2.0.0
905: **
906: ** Part of solution ZLD in project ZLD
907: **
908: ** Comments:
909: **
910: ** Important: Sections between markers "FTDI:S*" and "FTDI:E*" will be
    overwritten by
911: ** the Application Wizard
912: */
913: #include "vos.h"
914:
915: void iomux_setup()
916: {
917:   /* FTDI:SIO IOMux Functions */
918:   unsigned char packageType;
919:
920:   packageType = vos_get_package_type();
921:   if (packageType == VINCULUM_II_32_PIN)
922:   {
923:     // Debugger to pin 11 as Bi-Directional.
924:     vos_iomux_define_bidi(199, IOMUX_IN_DEBUGGER, IOMUX_OUT_DEBUGGER);
925:     // GPIO_Port_A_1 to pin 12 as OUTPUT.
926:     //vos_iomux_define_output(12, IOMUX_OUT_GPIO_PORT_A_1);
927:     // GPIO_Port_A_2 to pin 14 as OUTPUT.
928:     vos_iomux_define_output(14, IOMUX_OUT_GPIO_PORT_A_2);
929:     // GPIO_Port_A_3 to pin 15 as OUTPUT.
930:     vos_iomux_define_output(15, IOMUX_OUT_GPIO_PORT_A_3);
931:     // UART_TXD to pin 23 as Output.
932:     vos_iomux_define_output(23, IOMUX_OUT_UART_TXD);
933:     // UART_RXD to pin 24 as Input.
934:     vos_iomux_define_input(24, IOMUX_IN_UART_RXD);
935:     // UART_RTS_N to pin 25 as Output.
936:     vos_iomux_define_output(25, IOMUX_OUT_UART_RTS_N);
937:     // UART_CTS_N to pin 26 as Input.
938:     vos_iomux_define_input(26, IOMUX_IN_UART_CTS_N);
939:     // GPIO_Port_A_4 to pin 29 as Input.
940:     //vos_iomux_define_output(29, IOMUX_OUT_GPIO_PORT_A_4);
941:     // GPIO_Port_B_5 to pin 30 as Input.
942:     //vos_iomux_define_input(30, IOMUX_IN_GPIO_PORT_B_5);
943:     // GPIO_Port_B_6 to pin 31 as Input.
944:     vos_iomux_define_input(31, IOMUX_IN_GPIO_PORT_B_6);
```

```
945:     // GPIO_Port_B_7 to pin 32 as Input.
946:     //vos_iomux_define_input(32, IOMUX_IN_GPIO_PORT_B_7);
947:
948:   }
949:
950:   /* FTDI:EIO */
951: }
952:
953: ************************************************************
954:
955: /*
956: ** Filename: ZLD.c
957: **
958: ** Automatically created by Application Wizard 2.0.0
959: **
960: ** Part of solution ZLD in project ZLD
961: **
962: ** Comments:
963: **
964: ** Important: Sections between markers "FTDI:S*" and "FTDI:E*" will be
     overwritten by
965: ** the Application Wizard
966: */
967:
968: #include "ZLD.h"
969:
970: /* Default settings for UART interface */
971: #define DEF_UART_BAUD   UART_BAUD_115200
972: #define DEF_UART_DATA_BITS  UART_DATA_BITS_8
973: #define DEF_UART_FLOW   UART_FLOW_RTS_CTS
974: #define DEF_UART_STOP_BITS  UART_STOP_BITS_1
975: #define DEF_UART_PARITY   UART_PARITY_NONE
976:
977: /* FTDI:STP Thread Prototypes */
978: vos_tcb_t *tcbSETUP;
979: vos_tcb_t *tcbUART;
980: vos_tcb_t *tcbBOMS;
981: //vos_tcb_t *tcbFIRMWARE;
982: vos_tcb_t *tcbCOPYD;
983:
984: void setup();
985: void UART();
986: void BOMS();
987: void COPYD();
988: //void firmware();
989: /* FTDI:ETP */
990:
991: /* FTDI:SDH Driver Handles */
992: VOS_HANDLE hUSBHOST_1; // USB Host Port 1
993: VOS_HANDLE hUSBHOST_2; // USB Host Port 2
994: VOS_HANDLE hUART; // UART Interface Driver
995: VOS_HANDLE hGPIO_PORT_A; // GPIO Port A Driver
996: VOS_HANDLE hFAT_FILE_SYSTEM_1; // FAT File System for FAT32 and FAT16
997: VOS_HANDLE hFAT_FILE_SYSTEM_2; // FAT File System for FAT32 and FAT16
998: VOS_HANDLE hBOMS_1; // Bulk Only Mass Storage for USB disks
999: VOS_HANDLE hBOMS_2; // Bulk Only Mass Storage for USB disks
1000: VOS_HANDLE hGPIO_PORT_B;
1001: /* FTDI:EDH */
1002:
1003: vos_semaphore_t setupSem;
1004: vos_semaphore_t command;
1005: vos_semaphore_t cmdCpy;
1006: //vos_semaphore_t write;
1007: vos_semaphore_t dataSem;
```

```
1008: vos_mutex_t mBufAccess;
1009:
1010: fat_context fatContext1; fat_context fatContext2;
1011: FILE *file;
1012: unsigned char buf[300];
1013: unsigned short pBuf = 0;
1014: unsigned char filenameu[] = "TESTZLD .CSV";
1015: unsigned char fileuart = 0;
1016: unsigned char usbout = 0;
1017: /* Declaration for IOMUx setup function */
1018: void iomux_setup();
1019: void monError();
1020: void LED1();
1021: void NLED1();
1022:
1023: unsigned char copyFile(fat_context fatContextSrc, fat_context fatContextDest,
      char *filename);
1024: unsigned char changeDirUp(fat_context fatContext);
1025: unsigned char copyDir(fat_context fatContextSrc, fat_context fatContextDest);
1026: unsigned char copyDisk(fat_context fatContextSrc, fat_context fatContextDest);
1027: //fat_context *opendisk(VOS_HANDLE hUsbHost, VOS_HANDLE hBoms);
1028:
1029: /* Main code - entry point to firmware */
1030: void main()
1031: {
1032:   /* FTDI:SDD Driver Declarations */
1033:   // UART Driver configuration context
1034:   uart_context_t uartContext;
1035:   // GPIO Port A configuration context
1036:   gpio_context_t gpioContextA;
1037:   gpio_context_t gpioContextB;
1038:   // USB Host configuration context
1039:   usbhost_context_t usbhostContext;
1040:   /* FTDI:EDD */
1041:
1042:   /* FTDI:SKI Kernel Initialisation */
1043:   vos_init(50, VOS_TICK_INTERVAL, VOS_NUMBER_DEVICES);
1044:   vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
1045:   vos_set_idle_thread_tcb_size(400);
1046:   /* FTDI:EKI */
1047:
1048:   iomux_setup();
1049:
1050:   /* FTDI:SDI Driver Initialisation */
1051:   // Initialise UART
1052:   uartContext.buffer_size = VOS_BUFFER_SIZE_256_BYTES;
1053:   uart_init(VOS_DEV_UART,&uartContext);
1054:
1055:   // Initialise GPIO A
1056:   gpioContextA.port_identifier = GPIO_PORT_A;
1057:   gpio_init(VOS_DEV_GPIO_PORT_A,&gpioContextA);
1058:   // Initialise GPIO B
1059:   gpioContextA.port_identifier = GPIO_PORT_B;
1060:   gpio_init(VOS_DEV_GPIO_PORT_B,&gpioContextB);
1061:
1062:   fat_init();
1063:
1064:   // Initialise FAT File System Driver
1065:   fatdrv_init(VOS_DEV_FAT_FILE_SYSTEM_1);
1066:
1067:   // Initialise FAT File System Driver
1068:   fatdrv_init(VOS_DEV_FAT_FILE_SYSTEM_2);
1069:
1070:   // Initialise BOMS Device Driver
```

```
1071:   boms_init(VOS_DEV_BOMS_1);
1072:
1073:   // Initialise BOMS Device Driver
1074:   boms_init(VOS_DEV_BOMS_2);
1075:
1076:   // Initialise USB Host
1077:   usbhostContext.if_count = 8;
1078:   usbhostContext.ep_count = 16;
1079:   usbhostContext.xfer_count = 2;
1080:   usbhostContext.iso_xfer_count = 2;
1081:   usbhost_init(VOS_DEV_USBHOST_1, VOS_DEV_USBHOST_2, &usbhostContext);
1082:   /* FTDI:EDI */
1083:
1084:   /* FTDI:SCT Thread Creation */
1085:   tcbCOPYD = vos_create_thread_ex(24,2000, COPYD, "Copydisc", 0);
1086:   tcbSETUP = vos_create_thread_ex(25,800, setup, "Setup", 0);
1087:   tcbUART = vos_create_thread_ex(24,1200, UART, "Read", 0);
1088:     tcbBOMS = vos_create_thread_ex(24,1000, BOMS, "Write", 0);
1089:
1090:
1091:   /* FTDI:ECT */
1092:
1093:   vos_init_semaphore(&setupSem, 0);
1094:   vos_init_semaphore(&command, 0);
1095:   vos_init_semaphore(&dataSem, 0);
1096:   vos_init_semaphore(&cmdCpy, 0);
1097:   vos_init_mutex(&mBufAccess, VOS_MUTEX_UNLOCKED);
1098:
1099:   vos_start_scheduler();
1100:
1101: main_loop:
1102:   goto main_loop;
1103: }
1104:
1105: unsigned char usbhost_connect_state(VOS_HANDLE hUSB)
1106: {
1107:   unsigned char connectstate = PORT_STATE_DISCONNECTED;
1108:   usbhost_ioctl_cb_t hc_iocb;
1109:
1110:   if (hUSB)
1111:   {
1112:     hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_GET_CONNECT_STATE;
1113:     hc_iocb.get        = &connectstate;
1114:     vos_dev_ioctl(hUSB, &hc_iocb);
1115:
1116:     // repeat if connected to see if we move to enumerated
1117:     if (connectstate == PORT_STATE_CONNECTED)
1118:     {
1119:       vos_dev_ioctl(hUSB, &hc_iocb);
1120:     }
1121:   }
1122:   return connectstate;
1123: }
1124:
1125:
1126: VOS_HANDLE fat_attach(VOS_HANDLE hMSI, unsigned char devFAT)
1127: {
1128:   fat_ioctl_cb_t           fat_ioctl;
1129:   fatdrv_ioctl_cb_attach_t fat_att;
1130:   VOS_HANDLE hFAT;
1131:
1132:   // currently the MSI (BOMS or other) must be open
1133:         // open the FAT driver
1134:   hFAT = vos_dev_open(devFAT);
```

```
1135:
1136:         // attach the FAT driver to the MSI driver
1137:   fat_ioctl.ioctl_code = FAT_IOCTL_FS_ATTACH;
1138:   fat_ioctl.set = &fat_att;
1139:   fat_att.msi_handle = hMSI;
1140:   fat_att.partition = 0;
1141:
1142:   if (vos_dev_ioctl(hFAT, &fat_ioctl) != FAT_OK)
1143:   {
1144:                 // unable to open the FAT driver
1145:     vos_dev_close(hFAT);
1146:     monError(); return NULL;
1147:   }
1148:
1149:   return hFAT;
1150: }
1151:
1152: void fat_detach(VOS_HANDLE hFAT)
1153: {
1154:   fat_ioctl_cb_t           fat_ioctl;
1155:
1156:   if (hFAT)
1157:   {
1158:     fat_ioctl.ioctl_code = FAT_IOCTL_FS_DETACH;
1159:     fat_ioctl.set = NULL;
1160:     fat_ioctl.get = NULL;
1161:
1162:     vos_dev_ioctl(hFAT, &fat_ioctl);
1163:     vos_dev_close(hFAT);
1164:   }
1165: }
1166:
1167:
1168: VOS_HANDLE boms_attach(VOS_HANDLE hUSB, unsigned char devBOMS)
1169: {
1170:   usbhost_device_handle_ex ifDisk = 0;
1171:   usbhost_ioctl_cb_t hc_iocb;
1172:   usbhost_ioctl_cb_class_t hc_iocb_class;
1173:   msi_ioctl_cb_t boms_iocb;
1174:   boms_ioctl_cb_attach_t boms_att;
1175:   VOS_HANDLE hBOMS;
1176:     char mystate=0;
1177:
1178:   // find BOMS class device
1179:   hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
1180:   hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE_SCSI;
1181:   hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;
1182:
1183:   // user ioctl to find first hub device
1184:   hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
1185:   hc_iocb.handle.dif = NULL;
1186:   hc_iocb.set = &hc_iocb_class;
1187:   hc_iocb.get = &ifDisk;
1188:
1189:   if (vos_dev_ioctl(hUSB, &hc_iocb) != USBHOST_OK)
1190:   {
1191:     return NULL;
1192:   }
1193:
1194:   // now we have a device, intialise a BOMS driver with it
1195:   hBOMS = vos_dev_open(devBOMS);
1196:
1197:   // perform attach
1198:   boms_att.hc_handle = hUSB;
```

```
1199:   boms_att.ifDev = ifDisk;
1200:
1201:   boms_iocb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
1202:   boms_iocb.set = &boms_att;
1203:   boms_iocb.get = NULL;
1204:
1205:   mystate = vos_dev_ioctl(hBOMS, &boms_iocb);
1206:   if (mystate != MSI_OK)
1207:   {
1208:     vos_dev_close(hBOMS);
1209:     hBOMS = NULL;
1210:     monError(); return NULL;
1211:   }
1212:
1213:   return hBOMS;
1214: }
1215:
1216: void boms_detach(VOS_HANDLE hBOMS)
1217: {
1218:   msi_ioctl_cb_t boms_iocb;
1219:
1220:   if (hBOMS)
1221:   {
1222:     boms_iocb.ioctl_code = MSI_IOCTL_BOMS_DETACH;
1223:     boms_iocb.set = NULL;
1224:     boms_iocb.get = NULL;
1225:
1226:     vos_dev_ioctl(hBOMS, &boms_iocb);
1227:     vos_dev_close(hBOMS);
1228:   }
1229: }
1230:
1231: /* FTDI:ESP */
1232:
1233: /* Application Threads */
1234:
1235:
1236: void monError()
1237: {
1238:   unsigned char data;
1239:   vos_gpio_read_port(GPIO_PORT_A, &data);
1240:   data |= 0x04;
1241:   vos_gpio_write_port(GPIO_PORT_A, data);
1242:   while(1);
1243: }
1244:
1245: void setup()
1246: {
1247:   // Open a handle to all our devices...
1248:   unsigned char vstup=0;
1249:
1250:   hGPIO_PORT_A = vos_dev_open(VOS_DEV_GPIO_PORT_A);
1251:   hGPIO_PORT_B = vos_dev_open(VOS_DEV_GPIO_PORT_B);
1252:
1253:   vos_gpio_set_port_mode(GPIO_PORT_A, 0xff);
1254:   vos_gpio_set_port_mode(GPIO_PORT_B, 0x00);
1255:
1256:   vos_gpio_write_port(GPIO_PORT_A, 0);
1257:   vos_gpio_read_port(GPIO_PORT_B, &vstup);
1258:   vstup=vstup & 0x40;
1259:   //vstup=vstup >>4;
1260:   switch (vstup) {
1261:                 case 0x40: LED1();vos_signal_semaphore(&cmdCpy);break;
1262:                 default:   fileuart=1;vos_signal_semaphore(&command);break;
```

```
1263:
1264:                     }
1265: }
1266:
1267:
1268: void UART()
1269: {
1270:   unsigned short numRead;
1271:   common_ioctl_cb_t uart_iocb;
1272:   unsigned short dataAvail = 0;
1273:     unsigned char counter=0;
1274:   unsigned int handle;
1275:   usbhost_ioctl_cb_t hc_iocb;
1276: //  common_ioctl_cb_t uart_iocb;
1277:   unsigned char connect1, connect2;
1278:   unsigned char state1, state2;
1279:
1280:   vos_wait_semaphore(&command);
1281:
1282:
1283:   hUSBHOST_1 = vos_dev_open(VOS_DEV_USBHOST_1);
1284:   hUSBHOST_2 = vos_dev_open(VOS_DEV_USBHOST_2);
1285:
1286:   state1 = state2 = PORT_STATE_DISCONNECTED;
1287:
1288:   fat_init();
1289:
1290:   counter=0;
1291:   while (usbhost_connect_state(hUSBHOST_1) != PORT_STATE_ENUMERATED)
1292:   {
1293:
1294:     vos_delay_msecs(250);
1295:     LED1();
1296:     vos_delay_msecs(250);
1297:     NLED1();
1298:   }
1299:   state1=usbhost_connect_state(hUSBHOST_1);
1300:
1301:   counter=0;
1302:   while (usbhost_connect_state(hUSBHOST_2) != PORT_STATE_ENUMERATED)
1303:   {
1304:     // wait for enumeration to complete
1305:     vos_delay_msecs(100);
1306:     if (counter>30)
1307:     {
1308:       usbout=1;
1309:       break;
1310:     }
1311:     counter++;
1312:   }
1313:     state2=usbhost_connect_state(hUSBHOST_2);
1314:
1315:   hUART = vos_dev_open(VOS_DEV_UART);
1316:
1317:   uart_iocb.ioctl_code = VOS_IOCTL_COMMON_ENABLE_DMA;
1318:   uart_iocb.set.param = DMA_ACQUIRE_AS_REQUIRED;
1319:   vos_dev_ioctl(hUART, &uart_iocb);
1320:
1321:   // UART set baud rate
1322:   uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_BAUD_RATE;
1323:   uart_iocb.set.uart_baud_rate = DEF_UART_BAUD;
1324:   vos_dev_ioctl(hUART, &uart_iocb);
1325:
1326:   // UART set flow control
```

```
1327:   uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_FLOW_CONTROL;
1328:   uart_iocb.set.param = DEF_UART_FLOW;
1329:   vos_dev_ioctl(hUART, &uart_iocb);
1330:
1331:   // UART set data bits
1332:   uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_DATA_BITS;
1333:   uart_iocb.set.param = DEF_UART_DATA_BITS;
1334:   vos_dev_ioctl(hUART, &uart_iocb);
1335:
1336:   // UART set stop bits
1337:   uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_STOP_BITS;
1338:   uart_iocb.set.param = DEF_UART_STOP_BITS;
1339:   vos_dev_ioctl(hUART, &uart_iocb);
1340:
1341:   // UART set parity
1342:   uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_PARITY;
1343:   uart_iocb.set.param = DEF_UART_PARITY;
1344:   vos_dev_ioctl(hUART, &uart_iocb);
1345:
1346:   fatContext1 = fatContext2 = NULL;
1347:
1348:   if ((state1 == PORT_STATE_ENUMERATED))
1349:   {
1350:     hBOMS_1 = boms_attach(hUSBHOST_1, VOS_DEV_BOMS_1);
1351:
1352:     if (hBOMS_1)
1353:     {
1354:       fatContext1 = fat_open(hBOMS_1, 0, NULL);
1355:
1356:       if (!fatContext1)
1357:       {
1358:         monError();
1359:       }
1360:     }
1361:   }
1362:
1363:   if ((state2 == PORT_STATE_ENUMERATED))
1364:   {
1365:     hBOMS_2 = boms_attach(hUSBHOST_2, VOS_DEV_BOMS_2);
1366:
1367:     if (hBOMS_2)
1368:     {
1369:       fatContext2 = fat_open(hBOMS_2, 0, NULL);
1370:
1371:       if (!fatContext2)
1372:       {
1373:         monError();
1374:       }
1375:     }
1376:   }
1377:
1378:
1379:   hFAT_FILE_SYSTEM_1 = fat_attach(hBOMS_1, VOS_DEV_FAT_FILE_SYSTEM_1);
1380:
1381:   if (hFAT_FILE_SYSTEM_1 == NULL)
1382:   {
1383:     monError();
1384:   }
1385:
1386:   if (fatContext2)
1387:   {
1388:     hFAT_FILE_SYSTEM_2 = fat_attach(hBOMS_2, VOS_DEV_FAT_FILE_SYSTEM_2);
1389:     if (hFAT_FILE_SYSTEM_2 == NULL)
1390:     {
```

```
1391:       monError();return;
1392:     }
1393:   }
1394:
1395:
1396:   while(1)
1397:   {
1398:     // get bytes available...
1399:     uart_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
1400:     vos_dev_ioctl(hUART, &uart_iocb);
1401:     dataAvail = uart_iocb.get.queue_stat; // How much data to read?
1402:
1403:     if (dataAvail)
1404:     {
1405:       if (dataAvail > (sizeof(buf) - pBuf))
1406:         dataAvail = (sizeof(buf) - pBuf);
1407:
1408:       vos_lock_mutex(&mBufAccess);
1409:       vos_dev_read(hUART, &buf[pBuf], dataAvail, &numRead);
1410:       pBuf += numRead;
1411:       vos_unlock_mutex(&mBufAccess);
1412:       vos_signal_semaphore(&dataSem);
1413:     }
1414:   }
1415: }
1416:
1417: void BOMS()
1418: {
1419: file_context_t FILEC;
1420: unsigned char status;
1421: unsigned char OKstatus={'#','V', 13};
1422:   while(1)
1423:   {
1424:     vos_wait_semaphore(&dataSem);
1425:
1426:
1427:     if (pBuf)
1428:     {
1429:       if ((fileuart==1)&&(pBuf>7))     //zmena z 8 na 7
1430:       {
1431:         vos_lock_mutex(&mBufAccess);
1432:         pBuf=0;
1433:         for (pBuf=0;pBuf<8; pBuf++)
1434:         {
1435:           filenameu[pBuf]=buf[pBuf];
1436:         }
1437:         pBuf=0;
1438:         fileuart=0;
1439:         vos_dev_write(hUART, &OKstatus, 3, NULL);
1440:         vos_unlock_mutex(&mBufAccess);
1441:       }
1442:       if  (fileuart!=1)
1443:       {
1444:         vos_lock_mutex(&mBufAccess);
1445:         fsAttach(hFAT_FILE_SYSTEM_1);
1446:         file = fopen(filenameu, "a+");
1447:         if (file)
1448:         {
1449:           fwrite(buf, (size_t)pBuf, sizeof(char), file);
1450:           fclose(file);
1451:           file = NULL;
1452:         }
1453:         else
1454:         {
```

```
1455:           monError();
1456:         }
1457:         if (fatContext2)
1458:         {
1459:           fsAttach(hFAT_FILE_SYSTEM_2);
1460:           file = fopen(filenameu, "a+");
1461:           if (file)
1462:           {
1463:             fwrite(buf, (size_t)pBuf, sizeof(char), file);
1464:             fclose(file);
1465:             file = NULL;
1466:           }
1467:           else
1468:           {
1469:             monError();
1470:           }
1471:
1472:         }
1473:         pBuf = 0;
1474:         vos_unlock_mutex(&mBufAccess);
1475:       }
1476:     }
1477:   }
1478: }
1479:
1480: /*
1481: ** copyFile
1482: **
1483: ** Copy a file from the source to the destination disk.
1484: ** Source file is opened for read only, destination is opened for write.
1485: ** Destination file will be overwritten if it exists.
1486: **
1487: ** Parameters: handle to source file system
1488: **             handle to destination file system
1489: **             filename
1490: ** Returns: status of file system operation
1491: ** Comments:
1492: */
1493: unsigned char copyFile(fat_context fatContextSrc, fat_context fatContextDest,
     char *filename)
1494: {
1495:   unsigned char status;
1496:   // file handles for source and destination files
1497:   static file_context_t fileSource;
1498:   static file_context_t fileDest;
1499:   unsigned long length;
1500:
1501:   status = fat_fileOpen(fatContextSrc, &fileSource, filename, FILE_MODE_READ);
1502:   status = fat_fileOpen(fatContextDest, &fileDest, filename, FILE_MODE_WRITE);
1503:   length = fat_dirEntrySize(&fileSource);
1504:
1505:   status = fat_fileCopy(&fileSource, &fileDest);
1506:
1507:   // close the file handles
1508:   fat_fileClose(&fileSource);
1509:   fat_fileClose(&fileDest);
1510:
1511:   return status;
1512: }
1513:
1514: /*
1515: ** changeDirUp
1516: **
1517: ** Change directory up one level on selected file system.
```

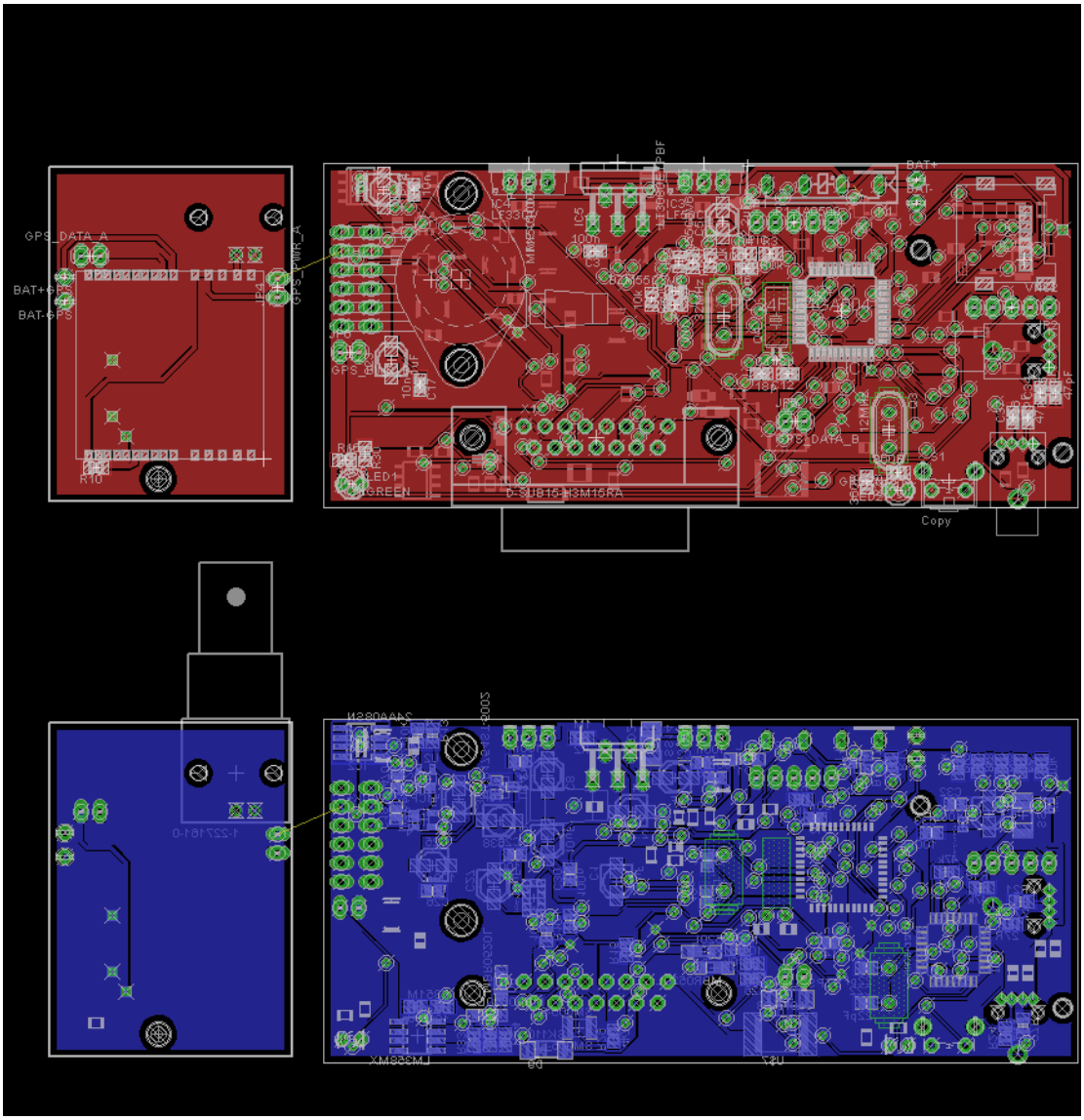
```
1518: **
1519: ** Parameters: handle to  file system
1520: ** Returns: status of file system operation
1521: ** Comments:
1522: */
1523: unsigned char changeDirUp(fat_context fatContext)
1524: {
1525:   unsigned char status;
1526:
1527:   status = fat_dirChangeDir(fatContext, "..         ");
1528:   return status;
1529: }
1530:
1531: /*
1532: ** copyDir
1533: **
1534: ** Recursively copy a directory from the source to the destination disk.
1535: **
1536: ** Parameters: handle to source file system
1537: **             handle to destination file system
1538: ** Returns: status of file system operation
1539: ** Comments:
1540: */
1541:
1542: unsigned char copyDir(fat_context fatContextSrc, fat_context fatContextDest)
1543: {
1544:   file_context_t fileFind;
1545:   char filename[12];
1546:   unsigned char status;
1547:
1548:   memset(filename, 0, sizeof(filename));
1549:
1550:   if (fat_dirTableFindFirst(fatContextSrc, &fileFind) == FAT_OK)
1551:   {
1552:     do
1553:     {
1554:       if (fat_dirEntryIsValid(&fileFind))
1555:       {
1556:         fat_dirEntryName(&fileFind, filename);
1557:
1558:         if (fat_dirEntryIsDirectory(&fileFind))
1559:         {
1560:           status = fat_dirChangeDir(fatContextSrc, (char *) filename);
1561:
1562:           if (status == FAT_OK)
1563:           {
1564:             status = fat_dirCreateDir(fatContextDest, (char *) filename);
1565:
1566:             if ((status == FAT_OK) || (status == FAT_EXISTS))
1567:             {
1568:               status = fat_dirChangeDir(fatContextDest, (char *) filename);
1569:
1570:               if (status == FAT_OK)
1571:               {
1572:                 copyDir(fatContextSrc, fatContextDest);
1573:                 status = changeDirUp(fatContextDest);
1574:
1575:                 if (status == FAT_OK)
1576:                 {
1577:                 }
1578:               }
1579:             }
1580:
1581:             status = changeDirUp(fatContextSrc);
```

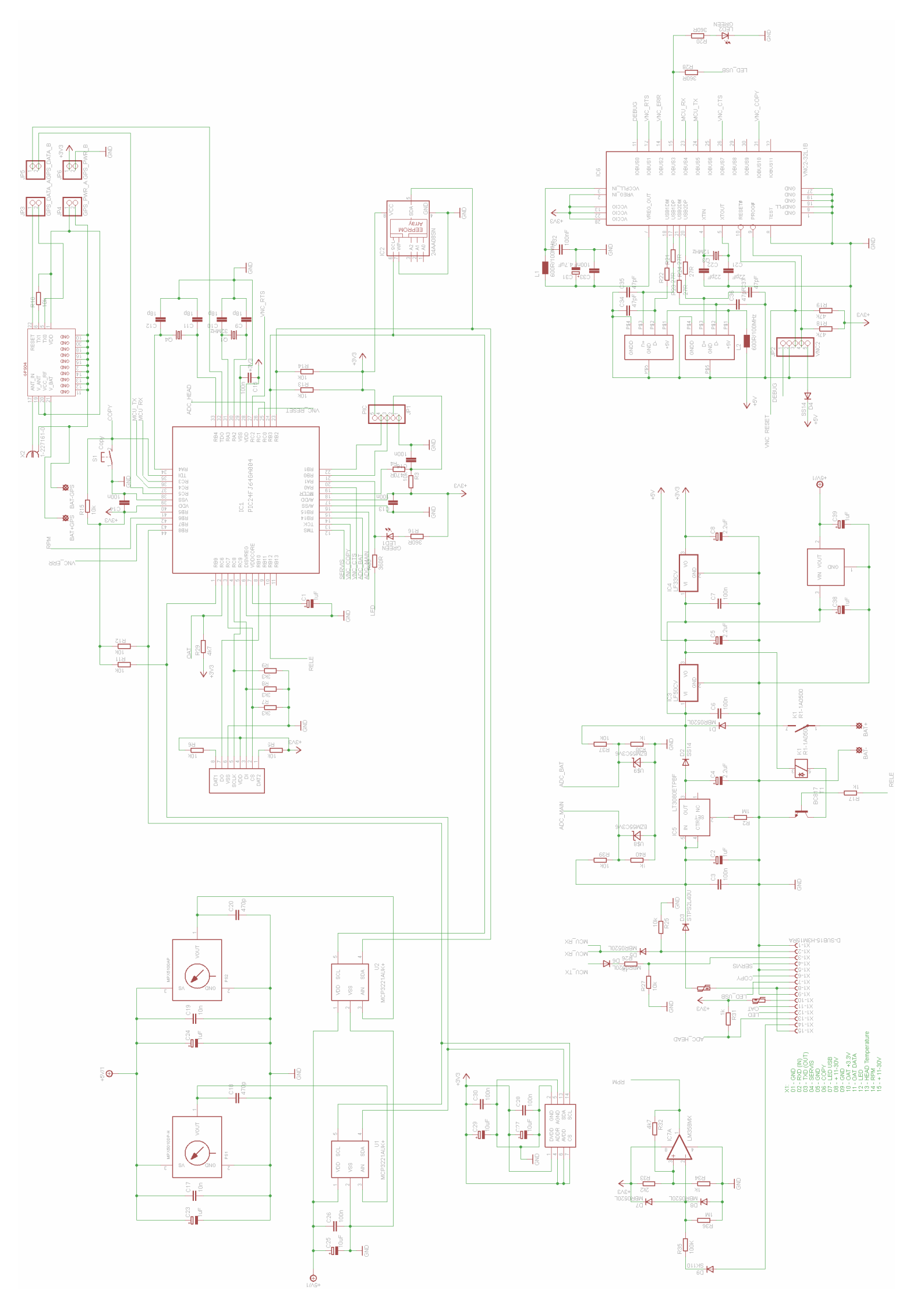
```
1582:
1583:             if (status == FAT_OK)
1584:             {
1585:             }
1586:           }
1587:
1588:         }
1589:         else
1590:         {
1591:           copyFile(fatContextSrc, fatContextDest, filename);
1592:         }
1593:       }
1594:     }
1595:     while (fat_dirTableFindNext(fatContextSrc, &fileFind) == FAT_OK);
1596:   }
1597:
1598:   return status;
1599: }
1600:
1601: /*
1602: ** copyDisk
1603: **
1604: ** Copy the source disk from the current directory to the destination disk
1605: ** current directory.
1606: **
1607: ** Parameters: handle to source file system
1608: **             handle to destination file system
1609: ** Returns: status of file system operation
1610: ** Comments:
1611: */
1612: unsigned char copyDisk(fat_context fatContextSrc, fat_context fatContextDest)
1613: {
1614:   return copyDir(fatContextSrc, fatContextDest);
1615: }
1616:
1617: unsigned char destMakeUniqueDir(fat_context fatContext)
1618: {
1619:   file_context_t fileFind;
1620:
1621:
1622:   char dirName[12] = "ZLD_0000   \0";
1623:   unsigned short count = 1;
1624:   unsigned short dirnum;
1625:   unsigned char i;
1626:   unsigned char status;
1627:
1628:   do
1629:   {
1630:     if (count > 9999)
1631:       break;
1632:
1633:     dirName[4] = ((count / 1000) % 10) + '0';
1634:     dirName[5] = ((count / 100) % 10) + '0';
1635:     dirName[6] = ((count / 10) % 10) + '0';
1636:     dirName[7] = (count % 10) + '0';
1637:
1638:     status = fat_dirTableFind(fatContext, &fileFind, (char *) dirName);
1639:
1640:     if (status != FAT_OK)
1641:     {
1642:       status = fat_dirCreateDir(fatContext, (char *) dirName);
1643:
1644:       if (status == FAT_OK)
1645:       {
```

```
1646:         status = fat_dirChangeDir(fatContext, (char *) dirName);
1647:
1648:         if (status == FAT_OK)
1649:         {
1650:           return 0;          // passed
1651:         }
1652:       }
1653:
1654:       break;
1655:     }
1656:
1657:     count++;
1658:   }
1659:   while (1);
1660:
1661:   return -1;                        // failed
1662: }
1663:
1664: void COPYD()
1665: {
1666:   unsigned char status;
1667:   unsigned int handle;
1668:   usbhost_ioctl_cb_t hc_iocb;
1669:   unsigned char connect1, connect2;
1670:   unsigned char state1, state2;
1671:
1672:   vos_wait_semaphore(&cmdCpy);
1673:
1674:   LED1();
1675:
1676:   hUSBHOST_1 = vos_dev_open(VOS_DEV_USBHOST_1);
1677:   hUSBHOST_2 = vos_dev_open(VOS_DEV_USBHOST_2);
1678:
1679:   state1 = state2 = PORT_STATE_DISCONNECTED;
1680:
1681:   fat_init();
1682:
1683:   while (usbhost_connect_state(hUSBHOST_1) != PORT_STATE_ENUMERATED)
1684:   {
1685:     // wait for enumeration to complete
1686:     vos_delay_msecs(100);
1687:
1688:   }
1689:   state1=usbhost_connect_state(hUSBHOST_1);
1690:
1691:   while (usbhost_connect_state(hUSBHOST_2) != PORT_STATE_ENUMERATED)
1692:   {
1693:     // wait for enumeration to complete
1694:     vos_delay_msecs(250);
1695:     LED1();
1696:     vos_delay_msecs(250);
1697:     NLED1();
1698:   }
1699:   state2=usbhost_connect_state(hUSBHOST_2);
1700:   LED1();
1701:
1702:
1703:     if ((state1 == PORT_STATE_ENUMERATED) && (state2 == PORT_STATE_ENUMERATED))
1704:     {
1705:       fatContext1 = fatContext2 = NULL;
1706:
1707:       hBOMS_1 = boms_attach(hUSBHOST_1, VOS_DEV_BOMS_1);
1708:
1709:       if (hBOMS_1)
```

```
1710:       {
1711:         fatContext1 = fat_open(hBOMS_1, 0, NULL);
1712:
1713:         if (!fatContext1)
1714:         {
1715:           monError();
1716:         }
1717:       }
1718:
1719:       hBOMS_2 = boms_attach(hUSBHOST_2, VOS_DEV_BOMS_2);
1720:
1721:       if (hBOMS_2)
1722:       {
1723:         fatContext2 = fat_open(hBOMS_2, 0, NULL);
1724:
1725:         if (!fatContext2)
1726:         {
1727:           if (fatContext1)
1728:             fat_close(fatContext1);
1729:
1730:           if (fatContext2)
1731:             fat_close(fatContext2);
1732:
1733:           boms_detach(hBOMS_1);
1734:           boms_detach(hBOMS_2);
1735:           while(1)
1736:           {
1737:             vos_delay_msecs(200);
1738:             LED1();
1739:             vos_delay_msecs(200);
1740:             NLED1();
1741:
1742:           }
1743:         }
1744:       }
1745:
1746:       if ((fatContext1) && (fatContext2))
1747:       {
1748:
1749:         // create (if required) and change into target direcory on destination
      disk
1750:         if (destMakeUniqueDir(fatContext2) == 0)
1751:         {
1752:           copyDisk(fatContext1, fatContext2);
1753:
1754:           // restore destination disk to top level directory
1755:           changeDirUp(fatContext2);
1756:         }
1757:
1758:         // restore source disk to top level directory
1759:         changeDirUp(fatContext1);
1760:
1761:       }
1762:
1763:       if (fatContext1)
1764:         fat_close(fatContext1);
1765:
1766:       if (fatContext2)
1767:         fat_close(fatContext2);
1768:
1769:       boms_detach(hBOMS_1);
1770:       boms_detach(hBOMS_2);
1771:
1772:       // copy once only
```

```
1773:
1774:     while (1)
1775:     {
1776:       vos_delay_msecs(1000);
1777:       LED1();
1778:       vos_delay_msecs(1000);
1779:       NLED1();
1780:     }
1781:   }
1782: }
1783:
1784: void LED1()
1785: {
1786:   unsigned char data;
1787:
1788:   vos_gpio_read_port(GPIO_PORT_A, &data);
1789:   data |= 0x08;
1790:   vos_gpio_write_port(GPIO_PORT_A, data);
1791: }
1792: void NLED1()
1793: {
1794:   unsigned char data;
1795:
1796:   vos_gpio_read_port(GPIO_PORT_A, &data);
1797:   data &= (~0x08);
1798:   vos_gpio_write_port(GPIO_PORT_A, data);
1799: }
1800:
1801:
```


GPS_DATA_A
BAT+GPS
BAT-GPS
GPS_PWR_A
R10
GPS_B
LED1
GREEN
D-SUB15-HSM15RA
BAT+
GPS_DATA_B
Copy


X1 - GND
01 - GND (IN)
02 - TXD (OUT)
03 - RXD (IN)
04 - SERVS
05 - GND
06 - +5V
07 - LED USB
08 - +11-30V
09 - GND
10 - BAT +9.3V
11 - BAT DATA
12 - LED
13 - HEAD Temperature
14 - RPM
15 - +11-30V