Středoškolská technika 2014

Setkání a prezentace prací středoškolských studentů na ČVUT

NÁVRH A REALIZACE STUDENTSKÉHO MINISATELITU CANSAT PRO EVROPSKOU SOUTĚŽ ESA 2014

Vladimír Váňa, Ondřej Vrba, Jan Ružička, Karel Urban, Ondřej Piskáček, Jakub Janovský, Vít Kalhus

Střední průmyslová škola elektrotechnická
Ječná 30, Praha 2

1. Úvod

Cílem práce je navrhnout a realizovat minisatelit Cansat pro finále evropské soutěže ESA 2014 European CanSat competition 1 - 5.června 2014 na základně Andøya Rocket Range v Norsku .

2. Co je to Cansat

Před několika lety se objevila zajímavá technická novinka - stavba „minisatelitů“ soukromými osobami využívajícími k jejich konstrukci plechovku od limonády. Odtud pochází i jejich název - CanSat. Poprvé s myšlenkou CanSatu přišel profesor Robert Twiggs [1] na konci 90. let. Tvůrci CanSatů do plechovky od limonády obvykle umisťují nějaká čidla jako např. čidlo tlaku a teploty, GPS moduly, kamery apod., jednočipové mikropočítače a vysílač pro VKV či UKV pásma ISP či pro některé z radioamaterských pásem 2m, 70cm, 23cm či 12cm. K vypuštění CanSatu vybaveného vlastním padáčkem obvykle slouží balon či signální raketa. Poté, co CanSat opustí raketu či balon, padá s pomocí padáčku k zemi a přitom vysílá naměřené údaje. Stavbou a provozem CanSatů se zabývají často zejména vysokoškolští studenti. Někdy je to i součást jejich studia. Příkladem může být studium SpaceMaster na ČVUT, kdy student tohoto magisterského studia je současně studentem ČVUT FEL i Luleå University of Technology, Kiruna Space Campus, Sweden a po úspěšném studiu získá tituly obou univerzit. 1. semestr absolvují studenti tohoto studia na Julius-Maximilians Universität

Würzburg, Germany, kde absolvují 6 předmětů včetně XE35CSP CanSat - Projekt (3 ECTS).

Obr.1 CanSat Nederland

V několika posledních letech jsou dokonce pořádány národní i mezinárodní soutěže založené na použití CanSatů

2.1. Podmínky soutěže

Úkolem je navrhnout a realizovat minisatelit rozměrů plechovky na nápoje 350ml , který bude vypuštěn raketou do výše cca 1km a poté bude na padáku klesat k Zemi a přitom bude plnit tři mise. Dvě jsou pro každý tým povinné – jde o měření atmosférického tlaku a teploty. Třetí mise již závisí na volbě soutěžního týmu. Náš tým PragSAT si zvolil 3D měření zrychlení a 3D měření magnetického pole Země.
Během sestupu Cansatu k Zemi jsou hodnoty naměřených údajů vysílány k zemi pomocí trancieveru pro pásmu 70 cm a jsou zachycovány pozemní stanicí soutěžního týmu a jsou jím dále zpracovávány a zobrazovány na PC.

2.2. Výběr třetí mise

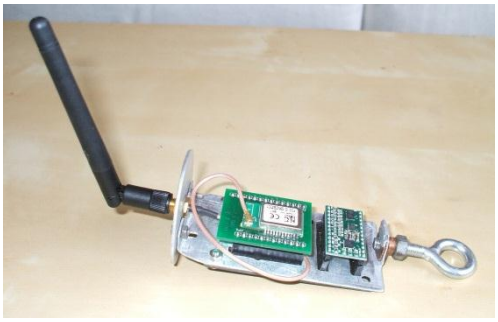
Při výběru třetí mise – měření zemského magnetického pole jsme vycházeli z tradice měření zemského magnetického pole pomocí satelitů Magion v České republice či předtím i v Československé republice - MAGION1 – MAGION5 (MAGnetics , IONosphere):MAGION1 (1979 – 1991), MAGION2 (1989), MAGION3 (1991-1992), MAGION4 (1995-1997) a MAGION5 (1996-2002).
Dalším zdrojem inspirace nám byl projekt ESA SWARM.
Na přípravě těchto přístrojů pro misi Swarm se podílelo 15 českých firem. Jde o dosud největší projekt vývoje letového hardwaru, na kterém Česká republika s agenturou ESA spolupracovala.
Proto jsme i my do své třetí mise kromě měření magnetického pole Země zahrnuli i akcelerometr.

3. Hardware

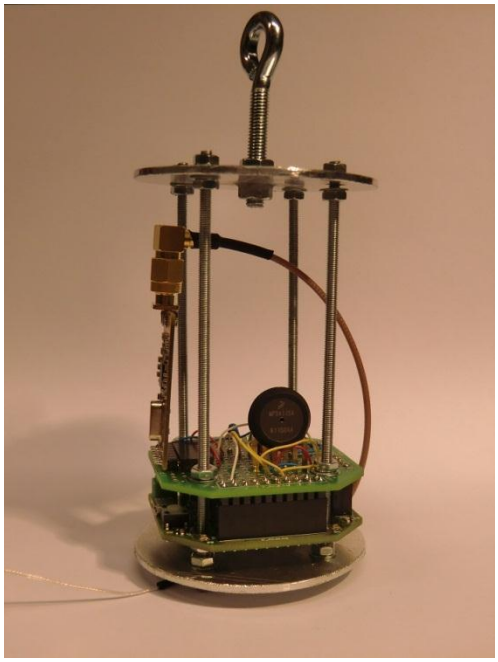
Elektronické vybavení Cansatu je dáno jeho požadovanou funkcí. Bude obsahovat blok čidel teploty, atmosférického tlaku, 3D akcelerometr a 3D magnetometr. Dále bude obsahovat palubní počítač pro komunikaci s čidly (nastavení jejich vlastností, přečtení dat) a z vysílače. Dalším blokem tedy bude ještě vysílač. Protože v případě poruchy radiového spojení bychom přišli o naměřená data, doplnili jsme elektroniku ještě o záznam dat na SD kartu. Nedílnou částí elektroniky je i napájecí zdroj.

3.1. Konstrukční provedení Cansatu

V praxi se ustálily dvě konstrukční provedení Cansatů. Ukazují je dva následující obrázky.

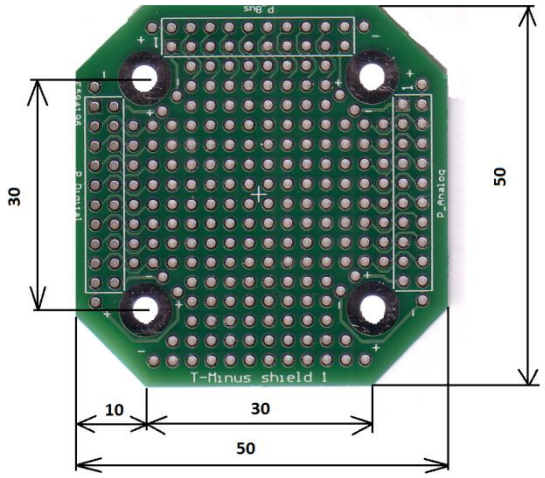


Obr.2



Obr.3

Zvolili jsme variantu z obr.3, protože máme dvojici trancieverů 433MHz vyráběné holandskou firmou T-minus pro ESA. Tím jsou ale určeny rozměry desek plošných spojů námi vyvíjených bloků.

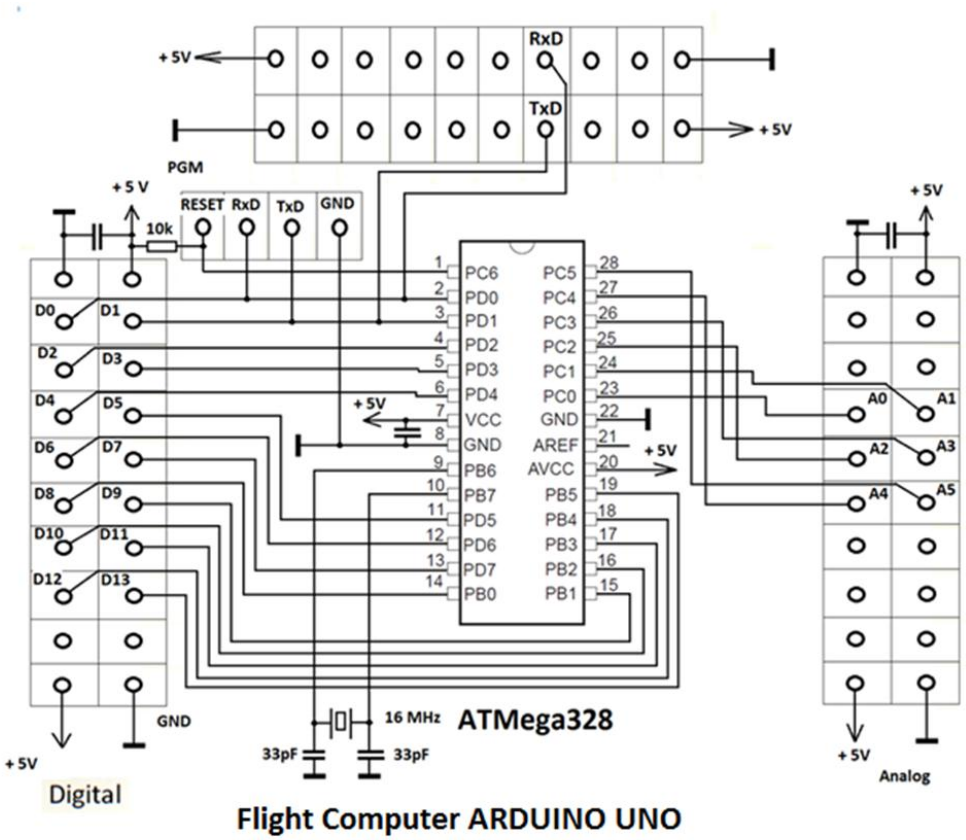


Obr.4

Čtyři otvory o Ø 3,2 mm slouží k navlečení desek na čtveřici svorníků se závitem M3 – viz obr.3

3.2. Palubní počítač

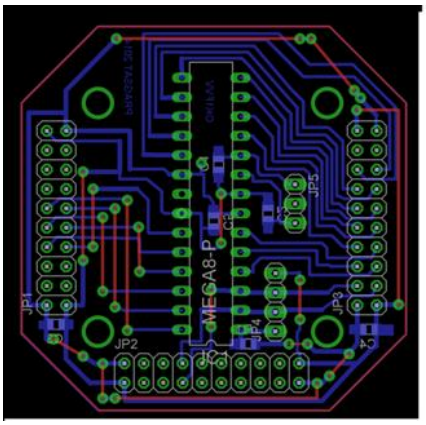
Z důvodů kompatibility se startkitem ESA vyráběným firmou T-minus jsme za základ použili MCU ATMEL ATMega. Použili jsme ATMega328, neboť v GMElectronic lze tento MCU zakoupit s naprogramovaným bootloaderem Arduino UNO. Z důvodů kompatibility jsme na této i dalších deskách použili trojici konektorů 2x10 pinů (viz např. univerzální deska na obr.4). Rovněž signály na těchto konektorech jsme volili tak, aby byly kompatibilní se startkitem ESA a mohli jsme tak použít desku tcvr z tohoto startkitu.Tím je dáno zapojení našeho palubního počítače.

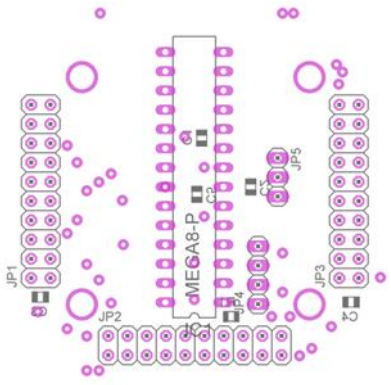


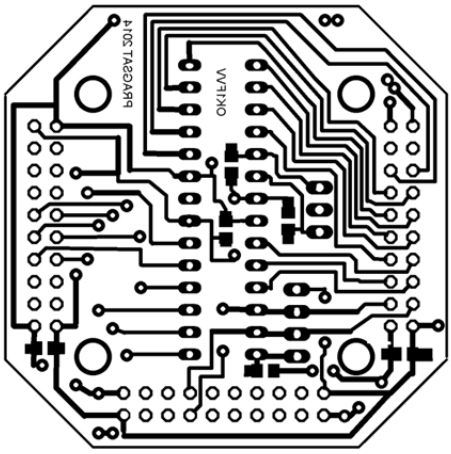
Flight Computer ARDUINO UNO

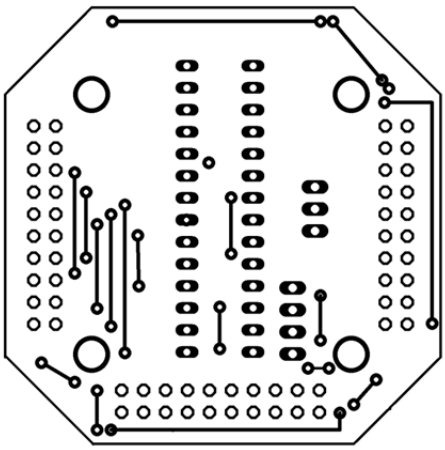
Zapojení počítače je velice jednoduché – obsahuje krystal 16MHz stejně jako ARDUINO UNO a dále obsahuje již jen propojení svých pinů s odpovídajícími dutinkami konektoru tak, aby tato deska byla v rozsahu D0 –D13 a A0 až A5 kompatibilní s počítačem od T-minus. Potom obsahuje již jen 4 pinový konektor pro připojení programátoru Arduino, např. s FT232RL ze stavebnice Cansatu z 2012.

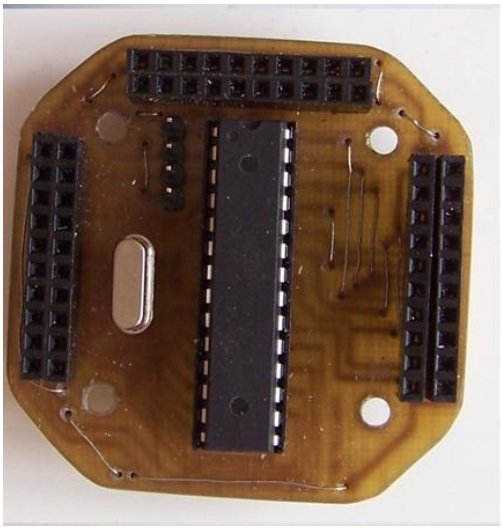
Následující obrázky ukazují návrh plošných spojů a fotografie hotového vzorku. Pro návrh PCB jsme použili systém EAGLE education [9] který je pro nekomerční použití a malé velikosti desek zdarma.

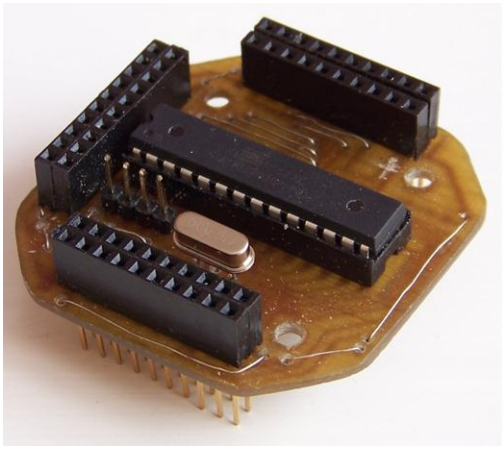












3.3. Blok čidel

Předpokládáme použití čidel tlaku, teploty, 3D akcelerometru, 3D magnetometru a 3D gyroskopu od firmy STMicroelectronics. Využijeme přitom modul STEVAL MKI124V1 module, který obsahuje LPS331AP (čidlo tlaku), LSM303DLHC (3D akcelerometr, 3D magnetometr) a L3GD20 (3D gyroscop) . S modulem se komunikuje prostřednictvím I2C.

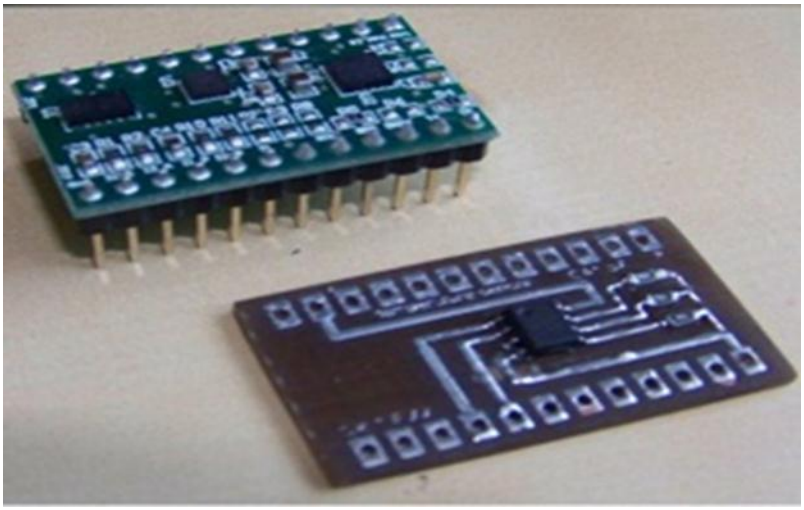
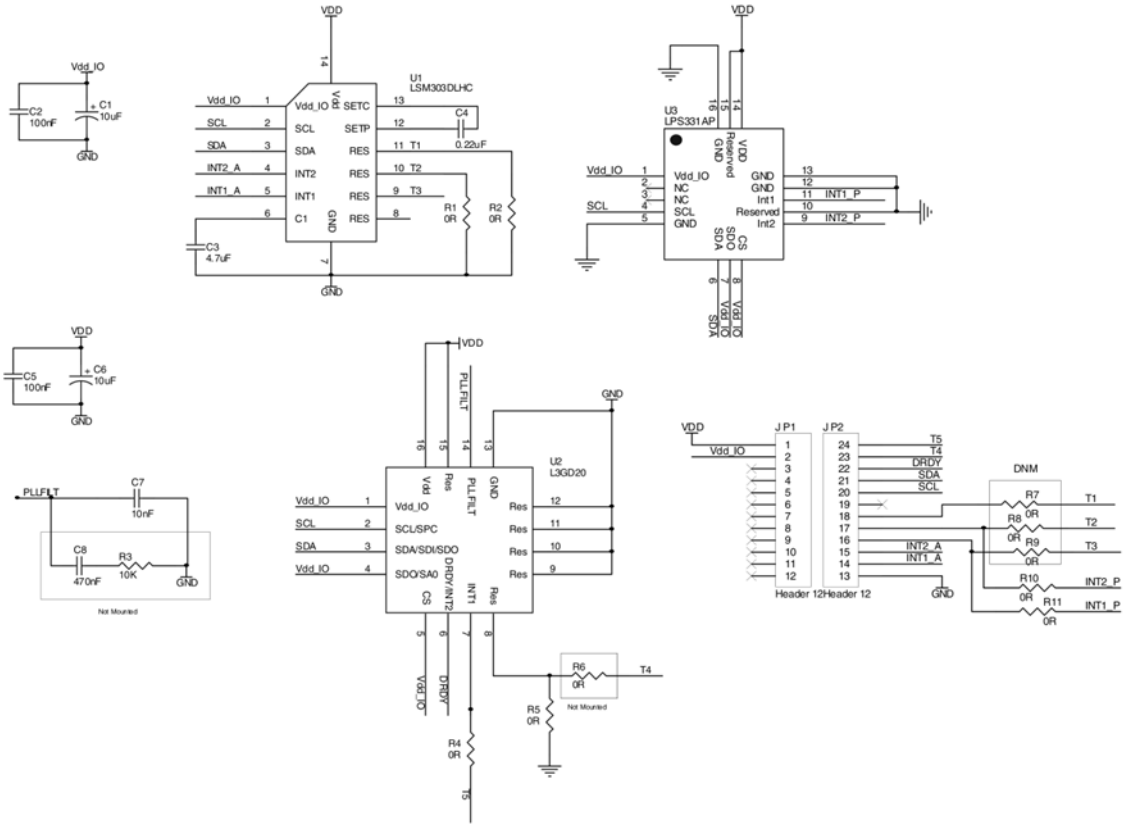
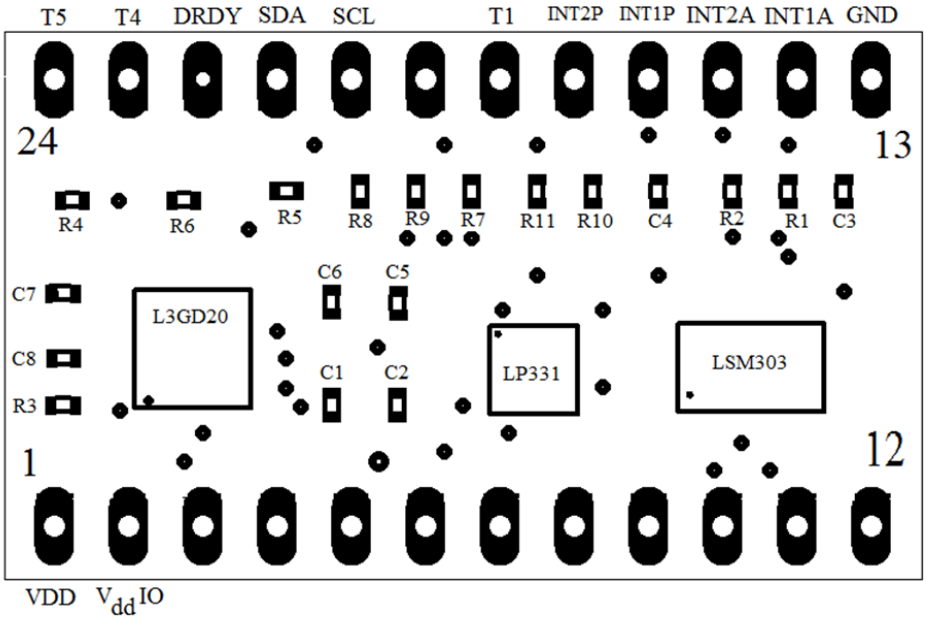


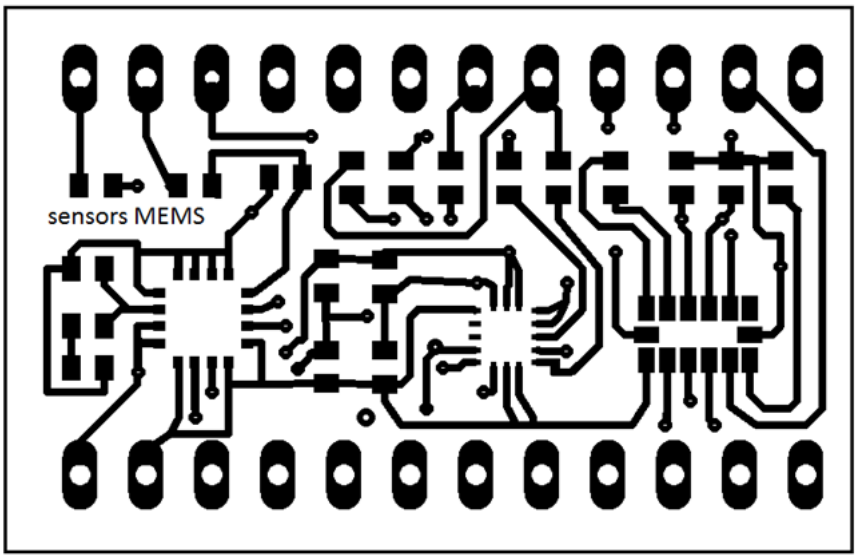
Schéma modulu MKI124V1:

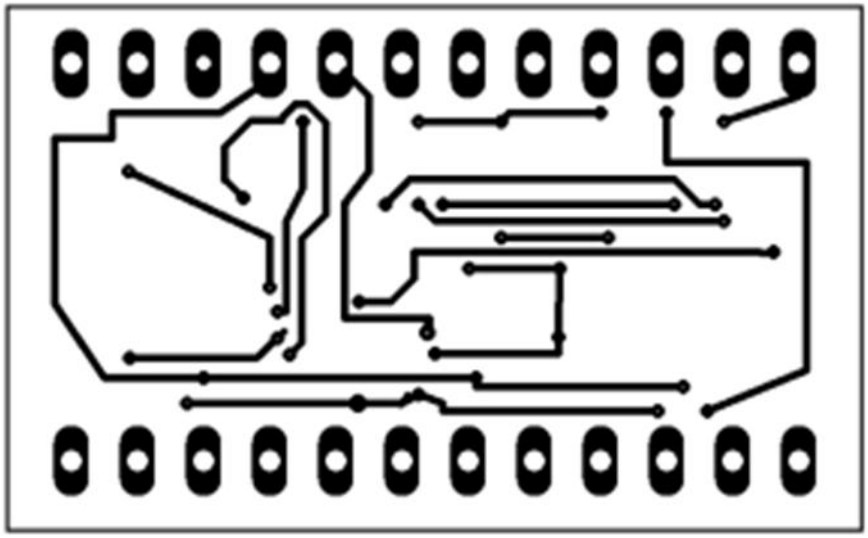


Rozložení součástí tohoto modulu:

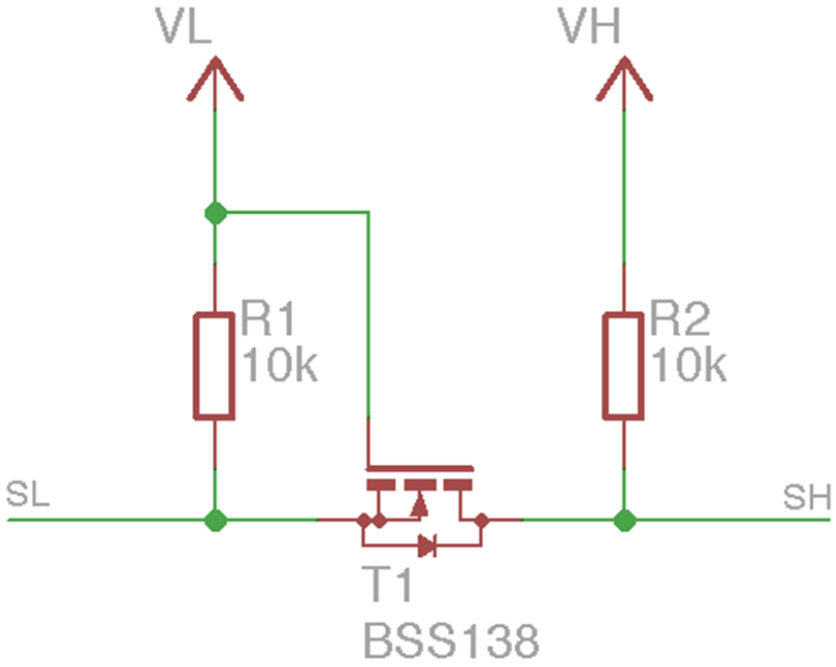

T5
T4
DRDY
SDA
SCL
T1
INT2P
INT1P
INT2A
INT1A
GND
24
13
R4
R6
R5
R8
R9
R7
R11
R10
C4
R2
R1
C3
C6
C5
C7
L3GD20
C8
C1
C2
LP331
LSM303
R3
1
12
VDD
V_{dd} IO

A ještě PCB:


sensors MEMS

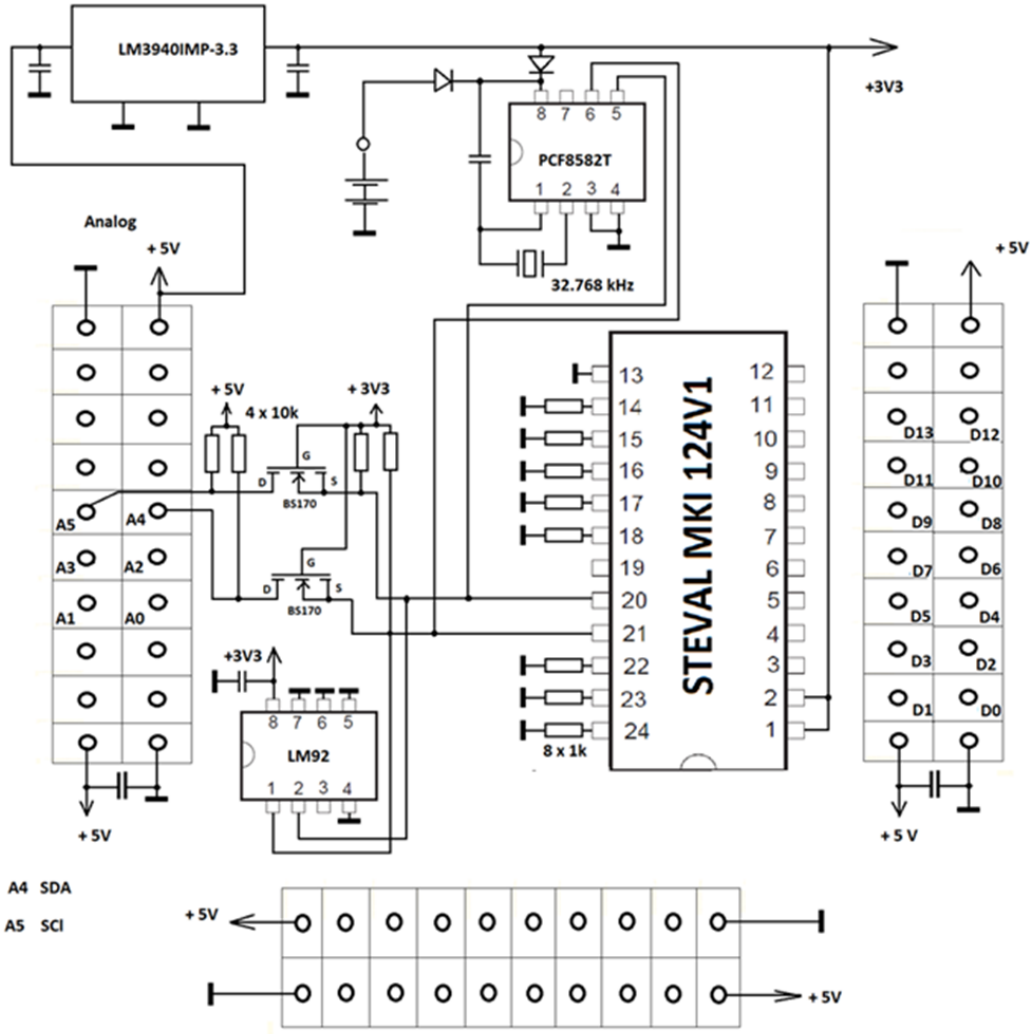


Dále se podíváme na základní parametry MKI124V1 týkající se napájení:
Napájení 2,4 až 3,6 V. I/O napěťové úrovně max. 3,6V takže se nesmí připojit na TTL přímo, tj ani k Arduino UNO, které je napájeno 5V !
Protože i2c signály SDA a SCL jsou obousměrné, musí být převodník mezi 3V3 a 5V také obousměrný. Je jím integrovaný obvod SN74LVC2T45. Lze však místo něj použít i zapojení s tranzistorem BSS138

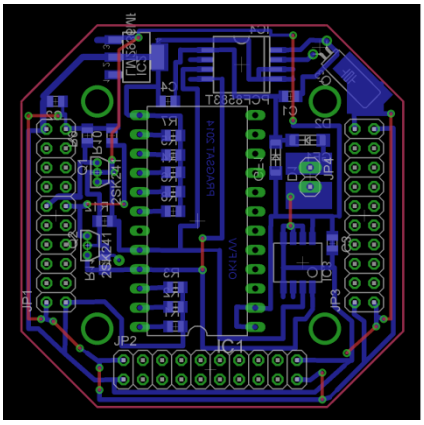


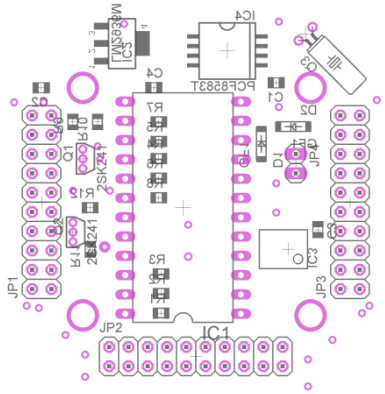
Destičku čidel MKI124V1 nakonec umístíme na destičku s trojicí konektorů 2 x 10 pinů, zapojených stejně, jako na desce palubního počítače a trancieveru. Deska bude ještě obsahovat zdroj 3.3V pro čidla, dvojici tranzistorů řízených

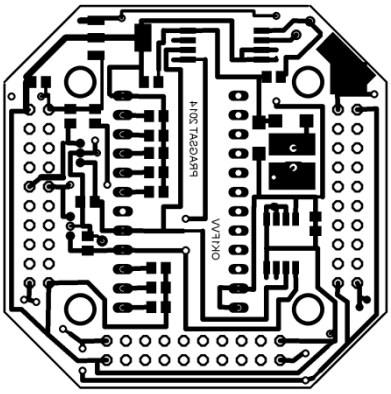
polem pro oboustranný převod úrovní signálů SDA a SCL (I2C) 3.3V a 5V. Dále ke sběrnici I2C máme na této desce připojeno čidlo teploty a obvod hodin. Současně se záznamem či vysílám naměřených dat potřebujeme totiž vědět i čas, kdy byla data pořízena.

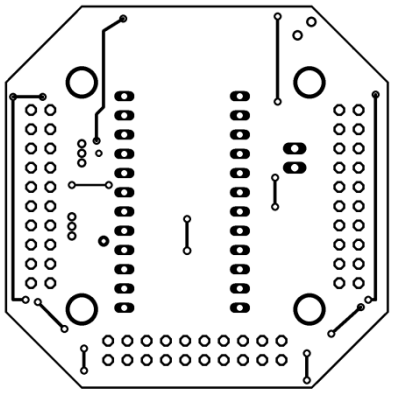


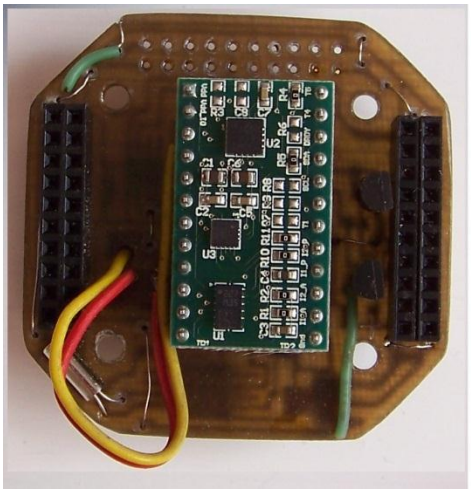
Návrh plošných spojů (v EAGLE), rozložení součásti i fotografie vzorku opět ukazují následující obrázky:

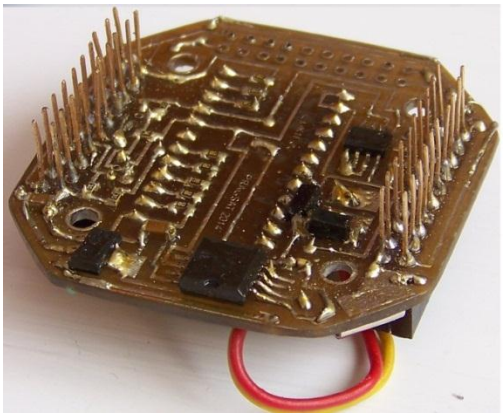








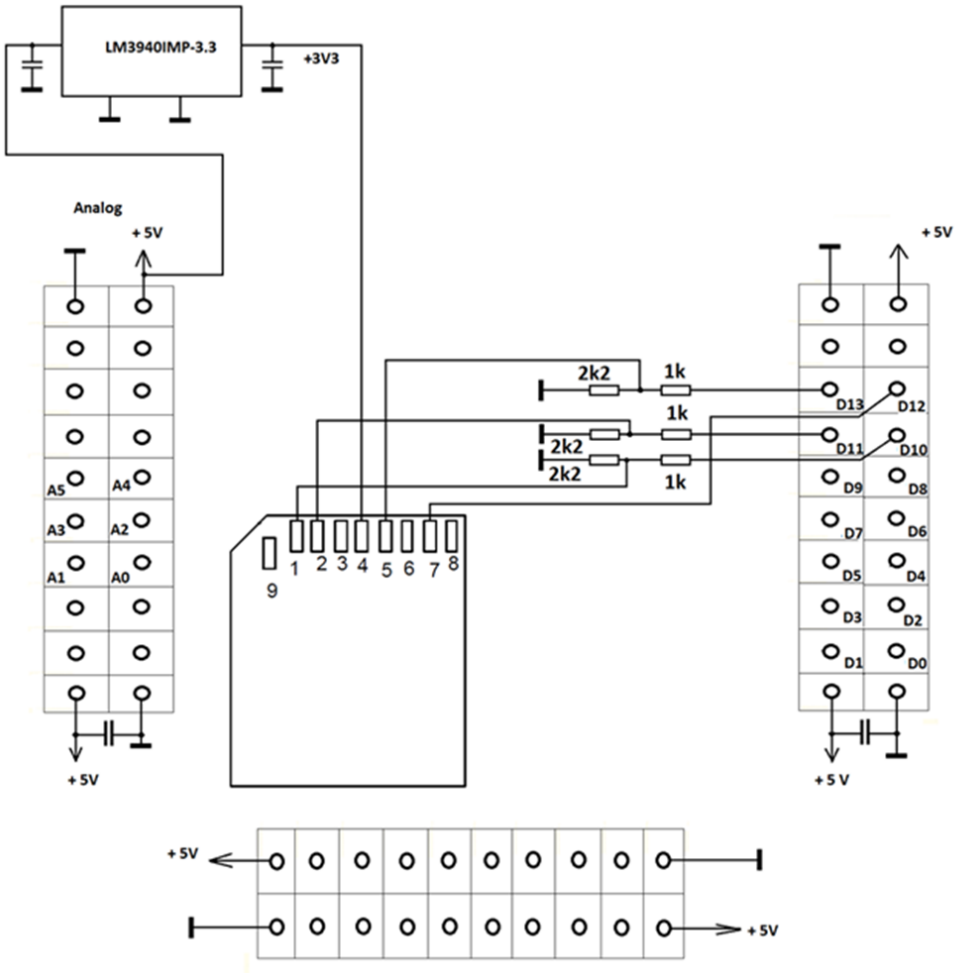


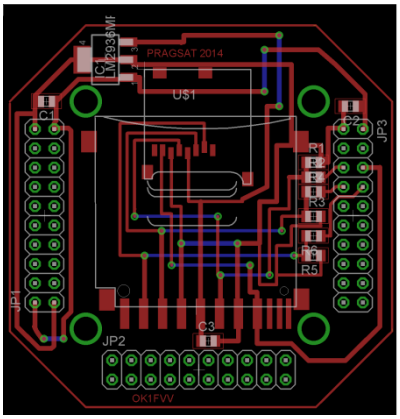


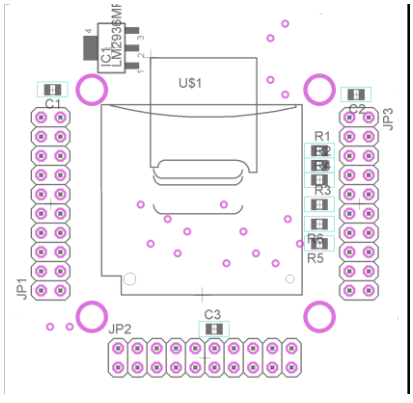
3.4. SD karta

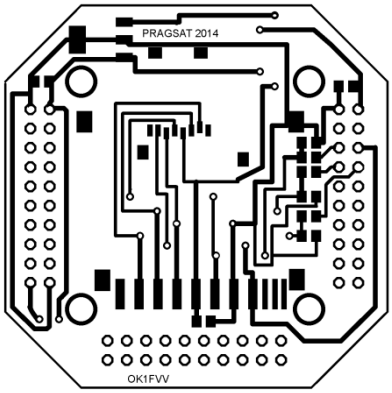
Kromě přenosu naměřených dat do pozemní stanice jsme se rozhodli cenná data zálohovat na SD kartu. Dá se s ní snadno komunikovat pomocí SPI čtveřice signálů MOSI, MISO, SCL a CS. Jedinou skutečností, na kterou nesmíme zapomenout je to, že karta předpokládá napájení 3,3V a stejné i úrovně log 1 signálů s ní komunikujících. Pro převod z 5V na 3,3V jsme použili odporové

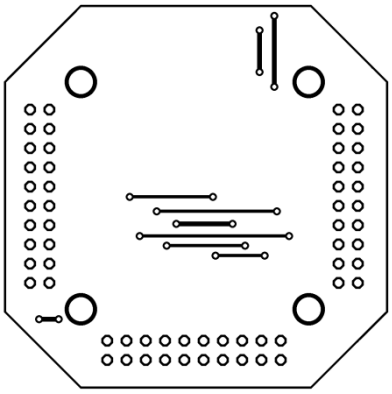
děliče. Naopak signál 3.3V z karty do ATMega jsme nepřeváděli vůbec, neboť tato úroveň je dostačující. Ostatně tak jsou zapojeny i SD karty připojované k normálnímu Arduinu.

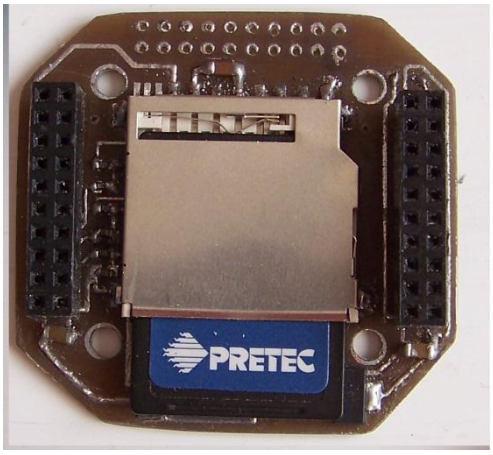


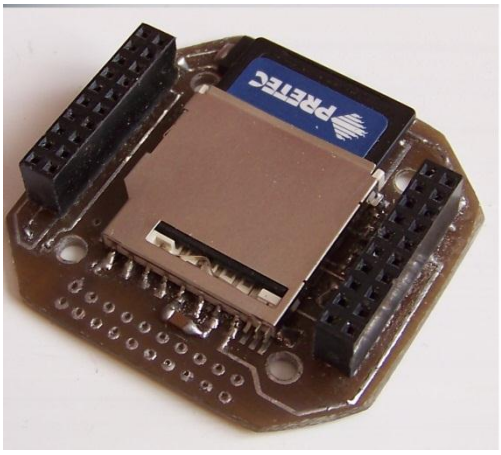












3.5. Vysílač

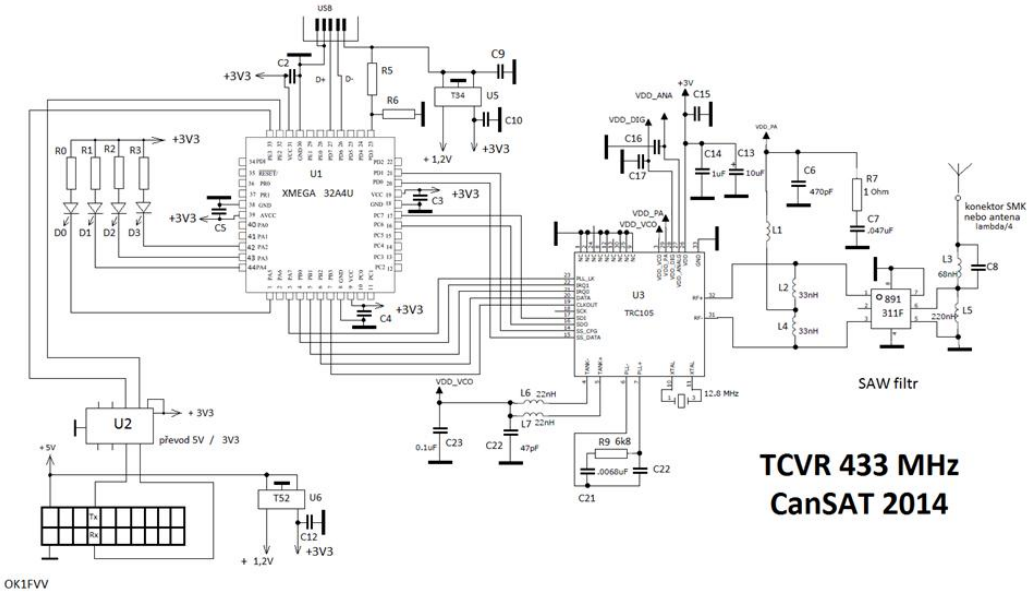
Tento blok jsme použili již hotový. To, že jde o zcela nový výrobek je zřejmě důvodem, že k němu ještě neexistuje dokumentace. Tak jsme provedli "reverze engineering". Zde jsou výsledky:

Základem trancieveru je obvod TRC105 firmy RFM. Vysílač může pracovat se dvěma druhy modulace buď OOK (on-off keyed) nebo **FSK** (frequency-shift keyed). Je-li nastaveno OOK pak log 1 znamená plný výkon a log 0 je bez výkonu. Je to tedy obdoba provozu A1. Napájecí napětí je 2.1 až 3.6V. Citlivost přijímače -112 dBm. Maximální výkon vysilače je +13 dBm. Řídícím mikrokontrolerem je u tohoto tcvr XMEGA32A4U. Tranciever pracuje na jednom z 24 kanálů v pásmu 70 cm s roztečí 90kHz mezi kanály. Jejich kmitočty jsou:

432.99 MHz	433.53 MHz	434.07 MHz	434.61 MHz
432.08 MHz	433.62 MHz	434.16 MHz	434.70 MHz
433.17 MHz	433.71 MHz	434.25 MHz	434.79 MHz
433.26 MHz	433.80 MHz	434.34 MHz	434.89 MHz

433.35 MHz	433.89 MHz	434.43 MHz	434.97 MHz
433.44 MHz	433.98 MHz	434.52 MHz	435.06 MHz

Schéma:



Oba dva obvody U1 a U3 jsou napájeny 3V3. Toto napětí získáme při napájení přes usb konektor (+5V) ze stabilizátoru U5, při napájení 5V z konektoru od bloku s MCU ATMega opět ze stabilizátoru, tentokrát U6. Dalšími signály z konektoru jsou Tx a Rx (přes konektor spojené s RxD a Txd ATmega) na úrovni TTL (5V) z Arduina. Obvod U2 slouží pro jejich převod na úroveň 3V3 XMega 32A4U.

4. Software

Tvorba software je poměrně jednoduchá vzhledem k použití vývojového prostředí i jazyka Arduina a knihoven systému Arduino pro sériovou komunikaci UART (s vysílačem), I2C (s čidly) a SPI (s SD kartou).

4.1. Programová obsluha čidel

Programovou obsluhu čidel si ukážeme na příkladě čidla atmosférického tlaku . Na začátku sketche uvedeme použité knihovny a vytvoříme instanci ps třídy LP331

```
#include <Wire.h>
#include <LPS331.h>

LPS331 ps;
```

Poté, jak je v Arduino sketchi běžné, uvedeme definice metod setup()

a loop() . To je podstaný rozdíl od jazyka C++. Jsou použity místo metody main() v C++.

```
void setup()
{
  Serial.begin(9600);
  Wire.begin();

  if (!ps.init())
  {
    Serial.println("Failed to autodetect pressure sensor!");
    while (1);
  }

  ps.enableDefault();
}
```

Ve výše uvedené metodě setup nejprve nastavíme parametry sériové komunikace. Poté se zavolá metoda init instance ps `ps.init()`
a provede se kontrola její úspěšnosti. Pokud se inicializace nepodaří, vyšle se chybová hláška. V případě úspěchu se zavolá metoda enableDefault(), čímž se zapne čidlo a nastaví se jeho nepřetržitá funkce.
Dále již probíhá veškerá činnost ve smyčce loop()

```
void loop()
{
  float pressure = ps.readPressureMillibars();
  float altitude = ps.pressureToAltitudeMeters(pressure);
  float temperature = ps.readTemperatureC();

  Serial.print("p: ");
  Serial.print(pressure);
  Serial.print(" mbar\ta: ");
  Serial.print(altitude);
  Serial.print(" m\tt: ");
  Serial.print(temperature);
  Serial.println(" deg C");

  delay(1000);
}
```

Nejprve se v ní zavolá metoda readPressureMillibars, která do proměnné pressure načte hodnotu změřeného atmosférického tlaku v milibarech. Poté se zavolá metoda pro přepočet tohoto tlaku na nadmořskou výšku a dále metoda pro přečtení teplota.
Následujících sedm řádků kódu slouží k odeslání právě naměřených hodnot sériovou linkou a nakonec je volána metoda delay, která pozdrží průběh smyčkou, neboť se měření má provádět 1 x za sekundu.

Celkový kód pro obsluhu čidla tlaku i dalších čidel uvádíme v příloze.

5. Závěr

Cílem projektu bylo navrhnout a zrealizovat zařízení Cansat, které by náš tým použil ve finále evropské soutěže ESA v červnu 2014 v Norsku. Již tato první verze splňuje požadavky zadání, nicméně návrh v sobě skrývá mnoho možných vylepšení. Mezi prvními bude implementován výkonnější mikrokontroler ARM Cortex STM32F103 a systém Arduino pak nahražen Maple, což je implementace Arduina právě pro STM32F103 provedená na MIT.

6. Přílohy

A. Sketch pro obsluhu čidla teploty

Pro programovou obsluhu jsme odzkoušeli následující sketch:

```
#include <Wire.h>
#include <LM75.h>

LM75 sensor;  // initialize an LM75 object
// You can also initiate with another address as follows:
//LM75 sensor(LM75_ADDRESS | 0b001);  // if A0->GND, A1->GND and A2->Vcc

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}

void loop()
{
  // get temperature from sensor
  Serial.print("Current temp: ");
  Serial.print(sensor.temp());
  Serial.println(" C");

  // Tos Set-point
  //sensor.tos(47.5); // set at 47.5'C
  //Serial.print("Tos set at ");
  //Serial.print(sensor.tos());
  //Serial.println(" C");
```

```
  // Thyst Set-point
  //sensor.thyst(42); // set at 42'C
  //Serial.print("Thyst set at ");
  //Serial.print(sensor.thyst());
  //Serial.println(" C");

  // shutdown the sensor and wait a while
  sensor.shutdown(true);
  delay(1000);
  // wake up sensor for next time around
  sensor.shutdown(false);

  Serial.println();
}
```

Potřebujeme k němu ještě knihovnu LM75:

LM75.h

```
#ifndef LM75_h

#define LM75_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#define LM75_ADDRESS 0x48

#define LM75_TEMP_REGISTER 0
#define LM75_CONF_REGISTER 1
#define LM75_THYST_REGISTER 2
#define LM75_TOS_REGISTER 3

#define LM75_CONF_SHUTDOWN  0
#define LM75_CONF_OS_COMP_INT 1
#define LM75_CONF_OS_POL 2
#define LM75_CONF_OS_F_QUE 3

class LM75 {
    int address;
    word float2regdata (float);
    float regdata2float (word);
    word _register16 (byte);
    void _register16 (byte, word);
    word _register8 (byte);
    void _register8 (byte, byte);
  public:
    LM75 ();
    LM75 (byte);
    float temp (void);
    byte conf (void);
    void conf (byte);
    float tos (void);
    void tos (float);
    float thyst (void);
    void thyst (float);
```

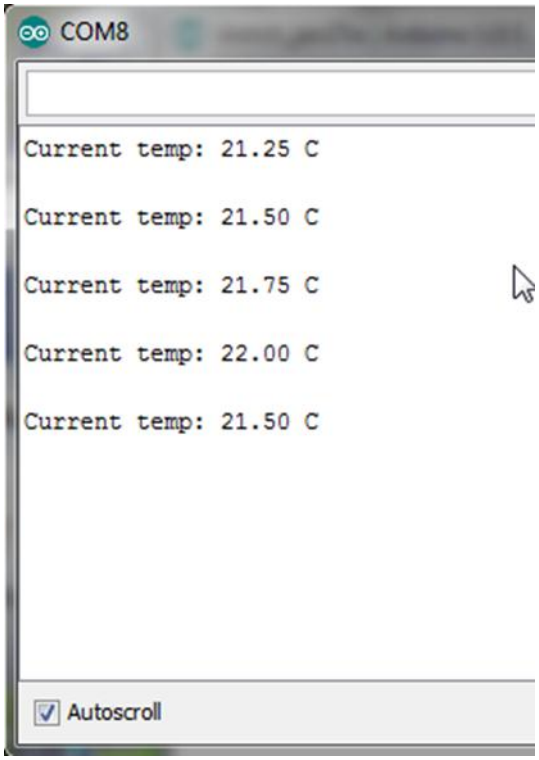
```
    void shutdown (boolean);
    boolean shutdown (void);
};

#endif
```

LM75.cpp

```
#include <Wire.h>
#include "LM75.h"

LM75::LM75 () {
  address = LM75_ADDRESS;
}

LM75::LM75 (byte addr) {
  address = addr;
}

word LM75::float2regdata (float temp)
{
  // First multiply by 8 and coerce to integer to get +/- whole numbers
  // Then coerce to word and bitshift 5 to fill out MSB
  return (word)((int)(temp * 8) << 5);
}
float LM75::regdata2float (word regdata)
{
  return ((float)(int)regdata / 32) / 8;
}word LM75::_register16 (byte reg) {
  Wire.beginTransmission(address);
  Wire.write(reg);
  Wire.endTransmission();

  Wire.requestFrom(address, 2);
  word regdata = (Wire.read() << 8) | Wire.read();
  return regdata;
}

void LM75::_register16 (byte reg, word regdata) {
  byte msb = (byte)(regdata >> 8);
  byte lsb = (byte)(regdata);

  Wire.beginTransmission(address);
  Wire.write(reg);
  Wire.write(msb);
  Wire.write(lsb);
  Wire.endTransmission();
}

word LM75::_register8 (byte reg) {
  Wire.beginTransmission(address);
  Wire.write(reg);
  Wire.endTransmission();

  Wire.requestFrom(address, 1);
  return Wire.read();
}

void LM75::_register8 (byte reg, byte regdata) {
  Wire.beginTransmission(address);
  Wire.write(reg);
```

```
  Wire.write(regdata);
  Wire.endTransmission();
}

float LM75::temp (void) {
  return regdata2float(_register16(LM75_TEMP_REGISTER));
}

byte LM75::conf () {
  return _register8(LM75_CONF_REGISTER);
}

void LM75::conf (byte data) {
  _register8(LM75_CONF_REGISTER, data);
}

float LM75::tos () {
  return regdata2float(_register16(LM75_TOS_REGISTER));
}

void LM75::tos (float temp) {
  _register16(LM75_TOS_REGISTER, float2regdata(temp));
}

float LM75::thyst () {
  return regdata2float(_register16(LM75_THYST_REGISTER));
}

void LM75::thyst (float temp) {
  _register16(LM75_THYST_REGISTER, float2regdata(temp));
}

boolean LM75::shutdown () {
  return conf() & 0x01;
}

void LM75::shutdown (boolean val) {
  conf(val << LM75_CONF_SHUTDOWN);
}
```

Výstup tohoto programu přes sériový port na monitoru je např.



A. Sketch pro obsluha čidla tlaku

Pro programovou obsluhu jsme odzkoušeil následující sketch:

```
#include <Wire.h>
#include <LPS331.h>

LPS331 ps;

void setup()
{
  Serial.begin(9600);
  Wire.begin();

  if (!ps.init())
  {
    Serial.println("Failed to autodetect pressure sensor!");
    while (1);
  }

  ps.enableDefault();
}

void loop()
{
  float pressure = ps.readPressureMillibars();
  float altitude = ps.pressureToAltitudeMeters(pressure);
  float temperature = ps.readTemperatureC();

  Serial.print("p: ");
  Serial.print(pressure);
  Serial.print(" mbar\ta: ");
  Serial.print(altitude);
  Serial.print(" m\tt: ");
```

```
  Serial.print(temperature);
  Serial.println(" deg C");

  delay(1000);
}
```

Potřebujeme k němu ještě knihovnu LPS331:

LPS331.h

```
#ifndef LPS331_h
#define LPS331_h

#include <Arduino.h> // for byte data type

// SA0 states

#define LPS331_SA0_LOW  0
#define LPS331_SA0_HIGH 1
#define LPS331_SA0_AUTO 2

// register addresses
// Note: Some of the register names in the datasheet are inconsistent
// between Table 14 in section 6 and the register descriptions in
// section 7. Where they differ, the names from section 7 have been
// used here.

#define LPS331_REF_P_XL       0x08
#define LPS331_REF_P_L        0x09
#define LPS331_REF_P_H        0x0A

#define LPS331_WHO_AM_I       0x0F

#define LPS331_RES_CONF       0x10

#define LPS331_CTRL_REG1      0x20
#define LPS331_CTRL_REG2      0x21
#define LPS331_CTRL_REG3      0x22
#define LPS331_INTERRUPT_CFG  0x23
#define LPS331_INT_SOURCE     0x24
#define LPS331_THS_P_L        0x25
#define LPS331_THS_P_H        0x26
#define LPS331_STATUS_REG     0x27

#define LPS331_PRESS_OUT_XL   0x28
#define LPS331_PRESS_OUT_L    0x29
#define LPS331_PRESS_OUT_H    0x2A

#define LPS331_TEMP_OUT_L     0x2B
#define LPS331_TEMP_OUT_H     0x2C

#define LPS331_AMP_CTRL       0x30

#define LPS331_DELTA_PRESS_XL 0x3C
#define LPS331_DELTA_PRESS_L  0x3D
#define LPS331_DELTA_PRESS_H  0x3E

class LPS331
{
```

```
  public:
    LPS331(void);

    bool init(byte sa0 = LPS331_SA0_AUTO);

    void enableDefault(void);

    void writeReg(byte reg, byte value);
    byte readReg(byte reg);

    float readPressureMillibars(void);
    float readPressureInchesHg(void);
    int32_t readPressureRaw(void);
    float readTemperatureC(void);
    float readTemperatureF(void);
    int16_t readTemperatureRaw(void);

    static float pressureToAltitudeMeters(float pressure_mbar, float
altimeter_setting_mbar = 1013.25);
    static float pressureToAltitudeFeet(float pressure_inHg, float
altimeter_setting_inHg = 29.9213);

  private:
    byte address;

    bool autoDetectAddress(void);
    bool testWhoAmI(void);
};

#endif
```

LPS331.cpp

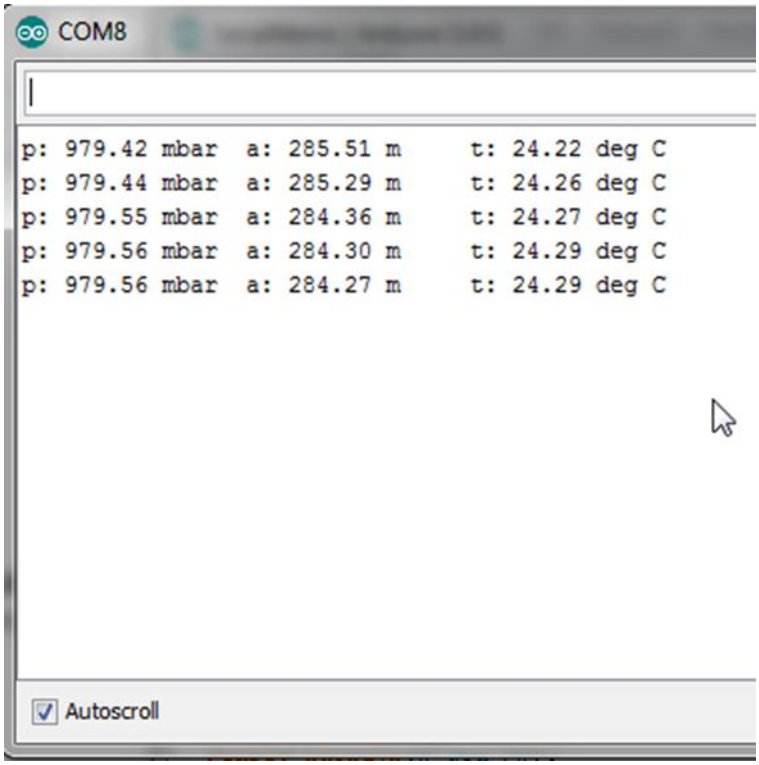
```
#include <LPS331.h>
#include <Wire.h>

// Defines /////////////////////////////////////////////////////////////

// The Arduino two-wire interface uses a 7-bit number for the address,
// and sets the last bit correctly based on reads and writes
#define LPS331AP_ADDRESS_SA0_LOW  0b1011100
#define LPS331AP_ADDRESS_SA0_HIGH 0b1011101

// Constructors ////////////////////////////////////////////////////////

LPS331::LPS331(void)
{
  // Pololu board pulls SA0 high, so default assumption is that it is
  // high
  address = LPS331AP_ADDRESS_SA0_HIGH;
}

// Public Methods //////////////////////////////////////////////////////

// sets or detects slave address; returns bool indicating success
bool LPS331::init(byte sa0)
{
```

```
  switch(sa0)
  {
    case LPS331_SA0_LOW:
      address = LPS331AP_ADDRESS_SA0_LOW;
      return testWhoAmI();

    case LPS331_SA0_HIGH:
      address = LPS331AP_ADDRESS_SA0_HIGH;
      return testWhoAmI();

    default:
      return autoDetectAddress();
  }
}

// turns on sensor and enables continuous output
void LPS331::enableDefault(void)
{
  // active mode, 12.5 Hz output data rate
  writeReg(LPS331_CTRL_REG1, 0b11100000);
}

// writes register
void LPS331::writeReg(byte reg, byte value)
{
  Wire.beginTransmission(address);
  Wire.write(reg);
  Wire.write(value);
  Wire.endTransmission();
}

// reads register
byte LPS331::readReg(byte reg)
{
  byte value;

  Wire.beginTransmission(address);
  Wire.write(reg);
  Wire.endTransmission(false); // restart
  Wire.requestFrom(address, (byte)1);
  value = Wire.read();
  Wire.endTransmission();

  return value;
}

// reads pressure in millibars (mbar)/hectopascals (hPa)
float LPS331::readPressureMillibars(void)
{
  return (float)readPressureRaw() / 4096;
}

// reads pressure in inches of mercury (inHg)
float LPS331::readPressureInchesHg(void)
{
  return (float)readPressureRaw() / 138706.5;
}

// reads pressure and returns raw 24-bit sensor output
int32_t LPS331::readPressureRaw(void)
{
  Wire.beginTransmission(address);
```

```
  // assert MSB to enable register address auto-increment
  Wire.write(LPS331_PRESS_OUT_XL | (1 << 7));
  Wire.endTransmission();
  Wire.requestFrom(address, (byte)3);

  while (Wire.available() < 3);

  uint8_t pxl = Wire.read();
  uint8_t pl = Wire.read();
  uint8_t ph = Wire.read();

  // combine bytes
  return (int32_t)(int8_t)ph << 16 | (uint16_t)pl << 8 | pxl;
}

// reads temperature in degrees C
float LPS331::readTemperatureC(void)
{
  return 42.5 + (float)readTemperatureRaw() / 480;
}

// reads temperature in degrees F
float LPS331::readTemperatureF(void)
{
  return 108.5 + (float)readTemperatureRaw() / 480 * 1.8;
}

// reads temperature and returns raw 16-bit sensor output
int16_t LPS331::readTemperatureRaw(void)
{
  Wire.beginTransmission(address);
  // assert MSB to enable register address auto-increment
  Wire.write(LPS331_TEMP_OUT_L | (1 << 7));
  Wire.endTransmission();
  Wire.requestFrom(address, (byte)2);

  while (Wire.available() < 2);

  uint8_t tl = Wire.read();
  uint8_t th = Wire.read();

  // combine bytes
  return (int16_t)(th << 8 | tl);
}

// converts pressure in mbar to altitude in meters, using 1976 US
// Standard Atmosphere model (note that this formula only applies to a
// height of 11 km, or about 36000 ft)
//  If altimeter setting (QNH, barometric pressure adjusted to sea
//  level) is given, this function returns an indicated altitude
//  compensated for actual regional pressure; otherwise, it returns
//  the pressure altitude above the standard pressure level of 1013.25
//  mbar or 29.9213 inHg
float LPS331::pressureToAltitudeMeters(float pressure_mbar, float
altimeter_setting_mbar)
{
  return (1 - pow(pressure_mbar / altimeter_setting_mbar, 0.190263)) *
44330.8;
}

// converts pressure in inHg to altitude in feet; see notes above
float LPS331::pressureToAltitudeFeet(float pressure_inHg, float
```

```
altimeter_setting_inHg)
{
  return (1 - pow(pressure_inHg / altimeter_setting_inHg, 0.190263)) *
145442;
}

// Private Methods /////////////////////////////////////////////////

bool LPS331::autoDetectAddress(void)
{
  // try each possible address and stop if reading WHO_AM_I returns the
expected response
  address = LPS331AP_ADDRESS_SA0_LOW;
  if (testWhoAmI()) return true;
  address = LPS331AP_ADDRESS_SA0_HIGH;
  if (testWhoAmI()) return true;

  return false;
}

bool LPS331::testWhoAmI(void)
{
  return (readReg(LPS331_WHO_AM_I) == 0xBB);
}
```

Výstup tohoto programu přes sériový port na monitoru je např.



B. Sketch pro obsluhu 3D akcelerometru

Pro programovou obsluhu jsme odzkoušeli následující sketch:

```
#include <Wire.h>
#include <Sensor.h>
#include <LSM303_U.h>

/* Assign a unique ID to this sensor at the same time */
LSM303_Accel_Unified accel = LSM303_Accel_Unified(54321);

void displaySensorDetails(void)
{
  sensor_t sensor;
  accel.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value);
Serial.println(" m/s^2");
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value);
Serial.println(" m/s^2");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution);
Serial.println(" m/s^2");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Accelerometer Test"); Serial.println("");

  /* Initialise the sensor */
  if(!accel.begin())
  {
    /* There was a problem detecting the ADXL345 ... check your connections
*/
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }

  /* Display some basic information on this sensor */
  displaySensorDetails();
}

void loop(void)
{
  /* Get a new sensor event */
  sensors_event_t event;
  accel.getEvent(&event);

  /* Display the results (acceleration is measured in m/s^2) */
  Serial.print("X: "); Serial.print(event.acceleration.x); Serial.print("
");
  Serial.print("Y: "); Serial.print(event.acceleration.y); Serial.print("
");
  Serial.print("Z: "); Serial.print(event.acceleration.z); Serial.print("
");Serial.println("m/s^2 ");
  delay(500);
}
```

Potřebujeme k němu ještě knihovny Sensor a LSM303_U:

Sensor.h

```
#ifndef SENSOR_H
#define SENSOR_H

#if ARDUINO >= 100
 #include "Arduino.h"
 #include "Print.h"
#else
 #include "WProgram.h"
#endif

/* Intentionally modeled after sensors.h in the Android API:
 *
https://github.com/android/platform_hardware_libhardware/blob/master/includ
e/hardware/sensors.h */

/* Constants */
#define SENSORS_GRAVITY_EARTH             (9.80665F)              /**<
Earth's gravity in m/s^2 */
#define SENSORS_GRAVITY_MOON              (1.6F)                  /**< The
moon's gravity in m/s^2 */
#define SENSORS_GRAVITY_SUN               (275.0F)                /**< The
sun's gravity in m/s^2 */
#define SENSORS_GRAVITY_STANDARD          (SENSORS_GRAVITY_EARTH)
#define SENSORS_MAGFIELD_EARTH_MAX        (60.0F)                 /**<
Maximum magnetic field on Earth's surface */
#define SENSORS_MAGFIELD_EARTH_MIN        (30.0F)                 /**<
Minimum magnetic field on Earth's surface */
#define SENSORS_PRESSURE_SEALEVELHPA      (1013.25F)              /**<
Average sea level pressure is 1013.25 hPa */
#define SENSORS_DPS_TO_RADS               (0.017453293F)          /**<
Degrees/s to rad/s multiplier */
#define SENSORS_GAUSS_TO_MICROTESLA       (100)                   /**<
Gauss to micro-Tesla multiplier */

/** Sensor types */
typedef enum
{
  SENSOR_TYPE_ACCELEROMETER         = (1),   /**< Gravity + linear
acceleration */
  SENSOR_TYPE_MAGNETIC_FIELD        = (2),
  SENSOR_TYPE_ORIENTATION           = (3),
  SENSOR_TYPE_GYROSCOPE             = (4),
  SENSOR_TYPE_LIGHT                 = (5),
  SENSOR_TYPE_PRESSURE              = (6),
  SENSOR_TYPE_PROXIMITY             = (8),
  SENSOR_TYPE_GRAVITY               = (9),
  SENSOR_TYPE_LINEAR_ACCELERATION   = (10),  /**< Acceleration not
including gravity */
  SENSOR_TYPE_ROTATION_VECTOR       = (11),
  SENSOR_TYPE_RELATIVE_HUMIDITY     = (12),
  SENSOR_TYPE_AMBIENT_TEMPERATURE   = (13),
  SENSOR_TYPE_VOLTAGE               = (15),
  SENSOR_TYPE_CURRENT               = (16),
  SENSOR_TYPE_COLOR                 = (17)
} sensors_type_t;

/** struct sensors_vec_s is used to return a vector in a common format. */
typedef struct {
```

```
    union {
        float v[3];
        struct {
            float x;
            float y;
            float z;
        };
        /* Orientation sensors */
        struct {
            float roll;    /**< Rotation around the longitudinal axis (the
plane body, 'X axis'). Roll is positive and increasing when moving
downward. -90°<=roll<=90° */
            float pitch;   /**< Rotation around the lateral axis (the wing
span, 'Y axis'). Pitch is positive and increasing when moving upwards. -
180°<=pitch<=180°) */
            float heading; /**< Angle between the longitudinal axis (the
plane body) and magnetic north, measured clockwise when viewing from the
top of the device. 0-359° */
        };
    };
    int8_t status;
    uint8_t reserved[3];
} sensors_vec_t;

/** struct sensors_color_s is used to return color data in a common format.
*/
typedef struct {
    union {
        float c[3];
        /* RGB color space */
        struct {
            float r;       /**< Red component */
            float g;       /**< Green component */
            float b;       /**< Blue component */
        };
    };
    uint32_t rgba;         /**< 24-bit RGBA value */
} sensors_color_t;

/* Sensor event (36 bytes) */
/** struct sensor_event_s is used to provide a single sensor event in a
common format. */
typedef struct
{
    int32_t version;                            /**< must be sizeof(struct
sensors_event_t) */
    int32_t sensor_id;                          /**< unique sensor identifier
*/
    int32_t type;                               /**< sensor type */
    int32_t reserved0;                          /**< reserved */
    int32_t timestamp;                          /**< time is in milliseconds
*/
    union
    {
        float           data[4];
        sensors_vec_t   acceleration;           /**< acceleration values are
in meter per second per second (m/s^2) */
        sensors_vec_t   magnetic;               /**< magnetic vector values
are in micro-Tesla (uT) */
        sensors_vec_t   orientation;            /**< orientation values are
in degrees */
        sensors_vec_t   gyro;                   /**< gyroscope values are in
```

```
rad/s */
        float           temperature;          /**< temperature is in
degrees centigrade (Celsius) */
        float           distance;             /**< distance in centimeters
*/
        float           light;                /**< light in SI lux units */
        float           pressure;             /**< pressure in hectopascal
(hPa) */
        float           relative_humidity;    /**< relative humidity in
percent */
        float           current;              /**< current in milliamps
(mA) */
        float           voltage;              /**< voltage in volts (V) */
        sensors_color_t color;                /**< color in RGB component
values */
    };
} sensors_event_t;

/* Sensor details (40 bytes) */
/** struct sensor_s is used to describe basic information about a specific
sensor. */
typedef struct
{
    char     name[12];                        /**< sensor name */
    int32_t  version;                         /**< version of the hardware
+ driver */
    int32_t  sensor_id;                       /**< unique sensor identifier
*/
    int32_t  type;                            /**< this sensor's type (ex.
SENSOR_TYPE_LIGHT) */
    float    max_value;                       /**< maximum value of this
sensor's value in SI units */
    float    min_value;                       /**< minimum value of this
sensor's value in SI units */
    float    resolution;                      /**< smallest difference
between two values reported by this sensor */
    int32_t  min_delay;                       /**< min delay in
microseconds between events. zero = not a constant rate */
} sensor_t;

class Sensor {
 public:
  // Constructor(s)
  // Sensor();
  void constructor();

  // These must be defined by the subclass
  virtual void getEvent(sensors_event_t*);
  virtual void getSensor(sensor_t*);
};

#endif
```

Sensor.cpp

```
#include "Sensor.h"
#include <avr/pgmspace.h>

void Sensor::constructor() {
}
```

LSM303_U.h

```
#ifndef __LSM303_H__
#define __LSM303_H__

#if (ARDUINO >= 100)
 #include "Arduino.h"
#else
 #include "WProgram.h"
#endif

#include <Sensor.h>
#include <Wire.h>

/*=========================================================================
    I2C ADDRESS/BITS
    -----------------------------------------------------------------------
*/
    #define LSM303_ADDRESS_ACCEL          (0x32 >> 1)         // 0011001x
    #define LSM303_ADDRESS_MAG            (0x3C >> 1)         // 0011110x
/*=========================================================================
*/

/*=========================================================================
    REGISTERS
    -----------------------------------------------------------------------
*/
    typedef enum
    {                                                   // DEFAULT
TYPE
      LSM303_REGISTER_ACCEL_CTRL_REG1_A         = 0x20,   // 00000111   rw
      LSM303_REGISTER_ACCEL_CTRL_REG2_A         = 0x21,   // 00000000   rw
      LSM303_REGISTER_ACCEL_CTRL_REG3_A         = 0x22,   // 00000000   rw
      LSM303_REGISTER_ACCEL_CTRL_REG4_A         = 0x23,   // 00000000   rw
      LSM303_REGISTER_ACCEL_CTRL_REG5_A         = 0x24,   // 00000000   rw
      LSM303_REGISTER_ACCEL_CTRL_REG6_A         = 0x25,   // 00000000   rw
      LSM303_REGISTER_ACCEL_REFERENCE_A         = 0x26,   // 00000000   r
      LSM303_REGISTER_ACCEL_STATUS_REG_A        = 0x27,   // 00000000   r
      LSM303_REGISTER_ACCEL_OUT_X_L_A           = 0x28,
      LSM303_REGISTER_ACCEL_OUT_X_H_A           = 0x29,
      LSM303_REGISTER_ACCEL_OUT_Y_L_A           = 0x2A,
      LSM303_REGISTER_ACCEL_OUT_Y_H_A           = 0x2B,
      LSM303_REGISTER_ACCEL_OUT_Z_L_A           = 0x2C,
      LSM303_REGISTER_ACCEL_OUT_Z_H_A           = 0x2D,
      LSM303_REGISTER_ACCEL_FIFO_CTRL_REG_A     = 0x2E,
      LSM303_REGISTER_ACCEL_FIFO_SRC_REG_A      = 0x2F,
      LSM303_REGISTER_ACCEL_INT1_CFG_A          = 0x30,
      LSM303_REGISTER_ACCEL_INT1_SOURCE_A       = 0x31,
      LSM303_REGISTER_ACCEL_INT1_THS_A          = 0x32,
      LSM303_REGISTER_ACCEL_INT1_DURATION_A     = 0x33,
      LSM303_REGISTER_ACCEL_INT2_CFG_A          = 0x34,
      LSM303_REGISTER_ACCEL_INT2_SOURCE_A       = 0x35,
      LSM303_REGISTER_ACCEL_INT2_THS_A          = 0x36,
      LSM303_REGISTER_ACCEL_INT2_DURATION_A     = 0x37,
      LSM303_REGISTER_ACCEL_CLICK_CFG_A         = 0x38,
      LSM303_REGISTER_ACCEL_CLICK_SRC_A         = 0x39,
      LSM303_REGISTER_ACCEL_CLICK_THS_A         = 0x3A,
      LSM303_REGISTER_ACCEL_TIME_LIMIT_A        = 0x3B,
      LSM303_REGISTER_ACCEL_TIME_LATENCY_A      = 0x3C,
      LSM303_REGISTER_ACCEL_TIME_WINDOW_A       = 0x3D
    } lsm303AccelRegisters_t;
```

```
    typedef enum
    {
      LSM303_REGISTER_MAG_CRA_REG_M             = 0x00,
      LSM303_REGISTER_MAG_CRB_REG_M             = 0x01,
      LSM303_REGISTER_MAG_MR_REG_M              = 0x02,
      LSM303_REGISTER_MAG_OUT_X_H_M             = 0x03,
      LSM303_REGISTER_MAG_OUT_X_L_M             = 0x04,
      LSM303_REGISTER_MAG_OUT_Z_H_M             = 0x05,
      LSM303_REGISTER_MAG_OUT_Z_L_M             = 0x06,
      LSM303_REGISTER_MAG_OUT_Y_H_M             = 0x07,
      LSM303_REGISTER_MAG_OUT_Y_L_M             = 0x08,
      LSM303_REGISTER_MAG_SR_REG_Mg             = 0x09,
      LSM303_REGISTER_MAG_IRA_REG_M             = 0x0A,
      LSM303_REGISTER_MAG_IRB_REG_M             = 0x0B,
      LSM303_REGISTER_MAG_IRC_REG_M             = 0x0C,
      LSM303_REGISTER_MAG_TEMP_OUT_H_M          = 0x31,
      LSM303_REGISTER_MAG_TEMP_OUT_L_M          = 0x32
    } lsm303MagRegisters_t;
/*=========================================================================
*/

/*=========================================================================
    MAGNETOMETER GAIN SETTINGS
    -----------------------------------------------------------------------
*/
    typedef enum
    {
      LSM303_MAGGAIN_1_3                        = 0x20,  // +/- 1.3
      LSM303_MAGGAIN_1_9                        = 0x40,  // +/- 1.9
      LSM303_MAGGAIN_2_5                        = 0x60,  // +/- 2.5
      LSM303_MAGGAIN_4_0                        = 0x80,  // +/- 4.0
      LSM303_MAGGAIN_4_7                        = 0xA0,  // +/- 4.7
      LSM303_MAGGAIN_5_6                        = 0xC0,  // +/- 5.6
      LSM303_MAGGAIN_8_1                        = 0xE0   // +/- 8.1
    } lsm303MagGain;
/*=========================================================================
*/

/*=========================================================================
    INTERNAL MAGNETOMETER DATA TYPE
    -----------------------------------------------------------------------
*/
    typedef struct lsm303MagData_s
    {
        float x;
        float y;
        float z;
      float orientation;
    } lsm303MagData;
/*=========================================================================
*/

/*=========================================================================
    INTERNAL ACCELERATION DATA TYPE
    -----------------------------------------------------------------------
*/
    typedef struct lsm303AccelData_s
    {
      float x;
      float y;
      float z;
```

```
    } lsm303AccelData;
/*=========================================================================
*/

/*=========================================================================
    CHIP ID
    -----------------------------------------------------------------------
*/
    #define LSM303_ID                     (0b11010100)
/*=========================================================================
*/

/* Unified sensor driver for the accelerometer */
class LSM303_Accel_Unified : public Sensor
{
  public:
    LSM303_Accel_Unified(int32_t sensorID = -1);

    bool begin(void);
    void getEvent(sensors_event_t*);
    void getSensor(sensor_t*);

  private:
    lsm303AccelData _accelData;   // Last read accelerometer data will be
available here
    int32_t         _sensorID;

    void write8(byte address, byte reg, byte value);
    byte read8(byte address, byte reg);
    void read(void);
};

/* Unified sensor driver for the magnetometer */
class LSM303_Mag_Unified : public Sensor
{
  public:
    LSM303_Mag_Unified(int32_t sensorID = -1);

    bool begin(void);
    void setMagGain(lsm303MagGain gain);
    void getEvent(sensors_event_t*);
    void getSensor(sensor_t*);

  private:
    lsm303MagGain   _magGain;
    lsm303MagData   _magData;     // Last read magnetometer data will be
available here
    int32_t         _sensorID;

    void write8(byte address, byte reg, byte value);
    byte read8(byte address, byte reg);
    void read(void);
};
```

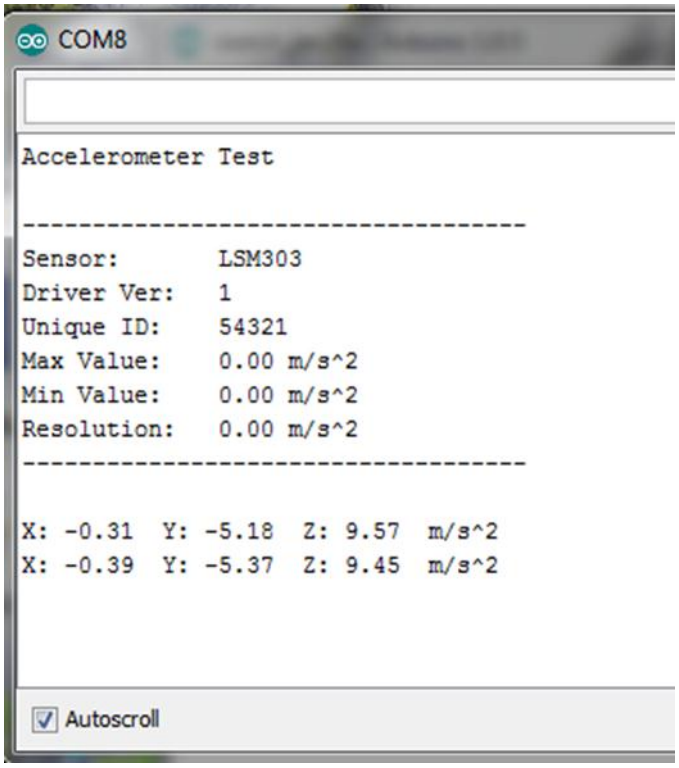
LSM303_U.cpp

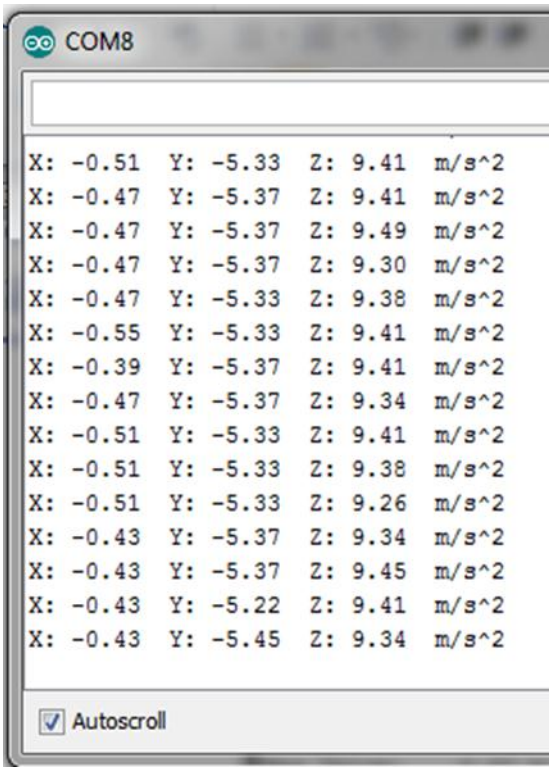
```
/**************************************************************************
/
/*!
    @brief  Sets the magnetometer's gain
```

```
*/
/**************************************************************************
/
void LSM303_Mag_Unified::setMagGain(lsm303MagGain gain)
{
  write8(LSM303_ADDRESS_MAG, LSM303_REGISTER_MAG_CRB_REG_M, (byte)gain);

  _magGain = gain;

  switch(gain)
  {
    case LSM303_MAGGAIN_1_3:
      _lsm303Mag_Gauss_LSB_XY = 1100;
      _lsm303Mag_Gauss_LSB_Z  = 980;
      break;
    case LSM303_MAGGAIN_1_9:
      _lsm303Mag_Gauss_LSB_XY = 855;
      _lsm303Mag_Gauss_LSB_Z  = 760;
      break;
    case LSM303_MAGGAIN_2_5:
      _lsm303Mag_Gauss_LSB_XY = 670;
      _lsm303Mag_Gauss_LSB_Z  = 600;
      break;
    case LSM303_MAGGAIN_4_0:
      _lsm303Mag_Gauss_LSB_XY = 450;
      _lsm303Mag_Gauss_LSB_Z  = 400;
      break;
    case LSM303_MAGGAIN_4_7:
      _lsm303Mag_Gauss_LSB_XY = 400;
      _lsm303Mag_Gauss_LSB_Z  = 255;
      break;
    case LSM303_MAGGAIN_5_6:
      _lsm303Mag_Gauss_LSB_XY = 330;
      _lsm303Mag_Gauss_LSB_Z  = 295;
      break;
    case LSM303_MAGGAIN_8_1:
      _lsm303Mag_Gauss_LSB_XY = 230;
      _lsm303Mag_Gauss_LSB_Z  = 205;
      break;
  }
}

/**************************************************************************
/
/*!
    @brief  Gets the most recent sensor event
*/
/**************************************************************************
/
void LSM303_Mag_Unified::getEvent(sensors_event_t *event) {
  /* Clear the event */
  memset(event, 0, sizeof(sensors_event_t));

  /* Read new data */
  read();

  event->version   = sizeof(sensors_event_t);
  event->sensor_id = _sensorID;
  event->type      = SENSOR_TYPE_MAGNETIC_FIELD;
  event->timestamp = 0;
  event->magnetic.x = _magData.x / _lsm303Mag_Gauss_LSB_XY *
SENSORS_GAUSS_TO_MICROTESLA;
```

```
  event->magnetic.y = _magData.y / _lsm303Mag_Gauss_LSB_XY *
SENSORS_GAUSS_TO_MICROTESLA;
  event->magnetic.z = _magData.z / _lsm303Mag_Gauss_LSB_Z *
SENSORS_GAUSS_TO_MICROTESLA;
}

/***************************************************************************
/
/*!
    @brief  Gets the sensor_t data
*/
/***************************************************************************
/
void LSM303_Mag_Unified::getSensor(sensor_t *sensor) {
  /* Clear the sensor_t object */
  memset(sensor, 0, sizeof(sensor_t));

  /* Insert the sensor name in the fixed length char array */
  strncpy (sensor->name, "LSM303", sizeof(sensor->name) - 1);
  sensor->name[sizeof(sensor->name)- 1] = 0;
  sensor->version     = 1;
  sensor->sensor_id   = _sensorID;
  sensor->type        = SENSOR_TYPE_MAGNETIC_FIELD;
  sensor->min_delay   = 0;
  sensor->max_value   = 0.0F; // TBD
  sensor->min_value   = 0.0F; // TBD
  sensor->resolution  = 0.0F; // TBD
}
```

Výstup tohoto programu přes sériový port na monitoru je např.





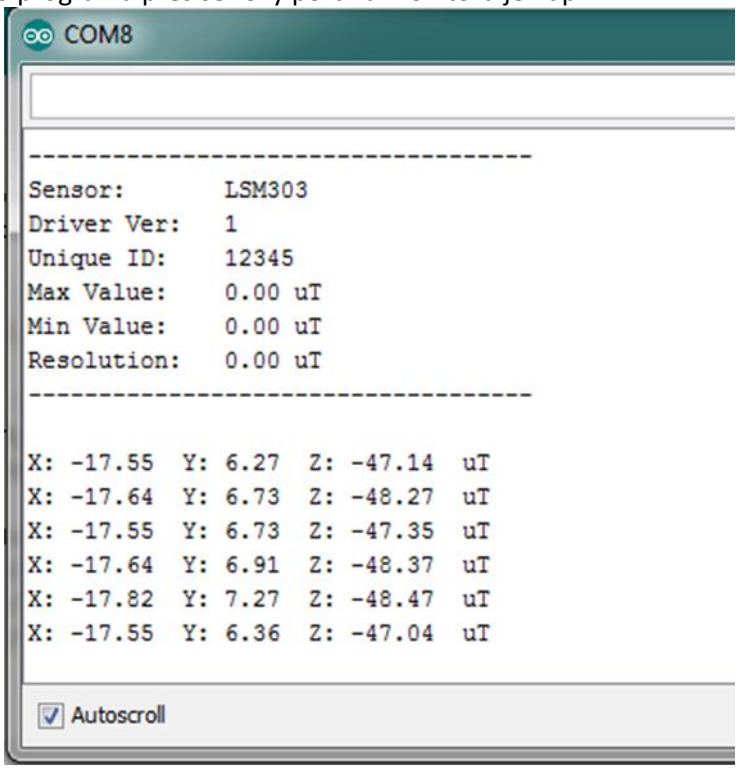
C. Sketch pro obsluhu 3D magnetometru

Pro programovou obsluhu jsme odzkoušeli následující sketch:

```
#include <Wire.h>
#include <Sensor.h>
#include <LSM303_U.h>

/* Assign a unique ID to this sensor at the same time */
LSM303_Mag_Unified mag = LSM303_Mag_Unified(12345);

void displaySensorDetails(void)
{
  sensor_t sensor;
  mag.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value);
Serial.println(" uT");
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value);
Serial.println(" uT");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution);
Serial.println(" uT");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}

void setup(void)
{
```

```
  Serial.begin(9600);
  Serial.println("Magnetometer Test"); Serial.println("");

  /* Initialise the sensor */
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections
*/
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }

  /* Display some basic information on this sensor */
  displaySensorDetails();
}

void loop(void)
{
  /* Get a new sensor event */
  sensors_event_t event;
  mag.getEvent(&event);

  /* Display the results (magnetic vector values are in micro-Tesla (uT))
*/
  Serial.print("X: "); Serial.print(event.magnetic.x); Serial.print("  ");
  Serial.print("Y: "); Serial.print(event.magnetic.y); Serial.print("  ");
  Serial.print("Z: "); Serial.print(event.magnetic.z); Serial.print("
");Serial.println("uT");
  delay(500);
}
```

Potřebujeme k němu ještě knihovny Sensor a LSM303_U, stejné jako v předchozím odstavci (akcelerometr).
Výstup tohoto programu přes sériový port na monitoru je např.



D. Sketch pro obsluhu obvodu kalendáře a času

Pro programovou obsluhu jsme odzkoušeli následující sketch:

```
#include <Wire.h> // necessary, or the application won't build properly
#include <stdio.h>
#include <PCF8583.h>
/*************************************************************************
***
 *  read/write serial interface to PCF8583 RTC via I2C interface
 *
 *  Arduino analog input 5 - I2C SCL (PCF8583 pin 6)
 *  Arduino analog input 4 - I2C SDA (PCF8583 pin 5)
 *
 *  You can set the type by sending it YYMMddhhmmss;
 *  the semicolon on the end tells it you're done...
 *

*************************************************************************
***/

int correct_address = 0;
PCF8583 p (0xA0);
void setup(void){
  Serial.begin(9600);
  Serial.print("booting...");
  Serial.println(" done");
     /* p.hour = 13;
      p.minute = 15;
      p.second = 0;
      p.year = 2014;
      p.month = 1;
      p.day = 25;
      p.set_time();
    */

}



void loop(void){
 if(Serial.available() > 0){
       p.year= (byte) ((Serial.read() - 48) *10 +  (Serial.read() - 48)) +
2000;
       p.month = (byte) ((Serial.read() - 48) *10 +  (Serial.read() - 48));
       p.day = (byte) ((Serial.read() - 48) *10 +  (Serial.read() - 48));
       p.hour  = (byte) ((Serial.read() - 48) *10 +  (Serial.read() - 48));
       p.minute = (byte) ((Serial.read() - 48) *10 +  (Serial.read() -
48));
       p.second = (byte) ((Serial.read() - 48) * 10 + (Serial.read() -
48)); // Use of (byte) type casting and ascii math to achieve result.

       if(Serial.read() == ';'){
         Serial.println("setting date");
        p.set_time();
```

```
      }
  }


  p.get_time();
  char time[50];
  sprintf(time, "%02d/%02d/%02d %02d:%02d:%02d",
         p.year, p.month, p.day, p.hour, p.minute, p.second);
  Serial.println(time);

  delay(1000);
}
```

Potřebujeme k němu ještě knihovnu PCF8583:

PCF8583.h

```
/*

   Usage:
      PCF8583 p(0xA0);
      p.get_time();

      Serial.print("year: ");
      Serial.println(p.year);


      p.hour = 14;
      p.minute = 30
      p.second = 0
      p.year = 2014
      p.month = 1
      p.day = 19
      p.set_time();


*/

#ifndef PCF8583_H
#define PCF8583_H

#include <Arduino.h>
#include <../Wire/Wire.h>

class PCF8583 {
    int address;
    int dow;
  public:
    int second;
    int minute;
    int hour;
    int day;
    int month;
    int year;
    int year_base;

    int alarm_milisec;
    int alarm_second;
    int alarm_minute;
    int alarm_hour;
```

```
    int alarm_day;

    PCF8583(int device_address);
    void init ();

    void get_time();
    void set_time();
    void get_alarm();
    int  get_day_of_week() const {
      return dow;
    }

    void set_daily_alarm();
    int bcd_to_byte(byte bcd);
    byte int_to_bcd(int in);
};


#endif  //PCF8583_H
```

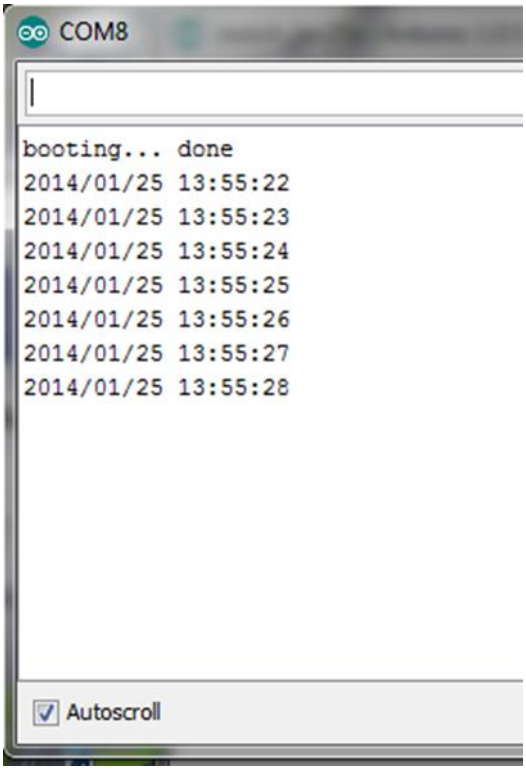
PCF8583.cpp

```
#include <Arduino.h>
#include <Wire.h>
#include "PCF8583.h"

namespace {
  bool IsLeapYear(int year) {
    return !(year % 400) || ((year % 100) && !(year % 4));
  }

  byte DayOfWeek(const PCF8583 &now) {
    static char PROGMEM MonthTable[24] = {0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3,
5, -1, 2, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5};
    byte y = now.year % 100, c = 6 - 2 * ((now.year / 100) % 4);
    return (now.day + pgm_read_byte_near(MonthTable + IsLeapYear(now.year)
* 12 + now.month - 1) + y + (y / 4) + c) % 7;
  }
}

// provide device address as a full 8 bit address (like the datasheet)
PCF8583::PCF8583(int device_address) {
  address = device_address >> 1;  // convert to 7 bit so Wire doesn't choke
  Wire.begin();
}

// initialization
void PCF8583::init()
{

Wire.beginTransmission(address);
Wire.write(0x00);
Wire.write(0x04);   // Set alarm on int\ will turn to vcc
Wire.endTransmission();

}

void PCF8583::get_time(){

  Wire.beginTransmission(address);
  Wire.write(0xC0);   // stop counting, don't mask
```

```
  Wire.endTransmission();

  Wire.beginTransmission(address);
  Wire.write(0x02);
  Wire.endTransmission();
  Wire.requestFrom(address, 5);

  second = bcd_to_byte(Wire.read());
  minute = bcd_to_byte(Wire.read());
  hour   = bcd_to_byte(Wire.read());
  byte incoming = Wire.read(); // year/date counter
  day    = bcd_to_byte(incoming & 0x3f);
  year   = (int)((incoming >> 6) & 0x03);      // it will only hold 4
years...
  incoming = Wire.read();
  month  = bcd_to_byte(incoming & 0x1f);
  dow = incoming >> 5;

  //  but that's not all - we need to find out what the base year is
  //  so we can add the 2 bits we got above and find the real year
  Wire.beginTransmission(address);
  Wire.write(0x10);
  Wire.endTransmission();
  Wire.requestFrom(address, 2);
  year_base = 0;
  year_base = Wire.read();
  year_base = year_base << 8;
  year_base = year_base | Wire.read();
  year = year + year_base;
}


void PCF8583::set_time()
 {

  if (!IsLeapYear(year) && 2 == month && 29 == day) {
    month = 3;
    day = 1;
  }

  // Attempt to find the previous leap year
  year_base = year - year % 4;
  if (!IsLeapYear(year_base)) {
    // Not a leap year (new century), make sure the calendar won't use a 29
days February.
    year_base = year - 1;
  }

  dow = DayOfWeek(*this);

  Wire.beginTransmission(address);
  Wire.write(0xC0);   // stop counting, don't mask
  Wire.endTransmission();

  Wire.beginTransmission(address);
  Wire.write(0x02);
  Wire.write(int_to_bcd(second));
  Wire.write(int_to_bcd(minute));
  Wire.write(int_to_bcd(hour));
  Wire.write(((byte)(year - year_base) << 6) | int_to_bcd(day));
  Wire.write((dow << 5) | (int_to_bcd(month) & 0x1f));
  Wire.endTransmission();
```

```
  Wire.beginTransmission(address);
  Wire.write(0x10);
  Wire.write(year_base >> 8);
  Wire.write(year_base & 0x00ff);
  Wire.endTransmission();

  init(); // re set the control/status register to 0x04

  }

//Get the alarm at 0x09 adress
void PCF8583::get_alarm()
{
Wire.beginTransmission(address);
Wire.write(0x0A); // Set the register pointer to (0x0A)
Wire.endTransmission();

Wire.requestFrom(address, 4); // Read 4 values

alarm_second = bcd_to_byte(Wire.read());
alarm_minute = bcd_to_byte(Wire.read());
alarm_hour   = bcd_to_byte(Wire.read());

Wire.beginTransmission(address);
Wire.write(0x0E);
Wire.endTransmission();

Wire.requestFrom(address, 1); // Read weekday value

alarm_day = bcd_to_byte(Wire.read());
}

//Set a daily alarm
void PCF8583::set_daily_alarm()
{
Wire.beginTransmission(address);
Wire.write(0x08);
Wire.write(0x90);  // daily alarm set
Wire.endTransmission();

Wire.beginTransmission(address);
Wire.write(0x09); // Set the register pointer to (0x09)
Wire.write(0x00); // Set 00 at milisec
Wire.write(int_to_bcd(alarm_second));
Wire.write(int_to_bcd(alarm_minute));
Wire.write(int_to_bcd(alarm_hour));
Wire.write(0x00); // Set 00 at day
Wire.endTransmission();
}

int PCF8583::bcd_to_byte(byte bcd){
  return ((bcd >> 4) * 10) + (bcd & 0x0f);
}

byte PCF8583::int_to_bcd(int in){
  return ((in / 10) << 4) + (in % 10);
}
```

Výstup tohoto programu přes sériový port na monitoru je např.



7. Použité zdroje

[1] Wang Torstein: CanSat Competition, <http://www.rocketrange.no>

[2] WWW SPŠE Ječná k CanSatu: <http://www.spsejecna.net/cansat/>

[3] WWW stránky České kosmické kanceláře:
<http://www.czechspace.cz/cs/vzdelavani/cansat>

[5] Váňa V.: Co je to Cansat. Praktická elektronika č.11/2013, str.15-17

[6] Váňa V.: Mikrokontroléry STM32F prakticky. Praktická elektronika
č.12/2013 str.15-17 a č.1/2014 str.14-15

[7] Váňa V.: Mikrokontroléry STM32 bezdrátově. Praktická elektronika č.4/2014
str.21-24

[8] <http://www.st.com>

[9] <http://arduino.cc/>

[10] <http://www.cadsoftusa.com/>

[11] < http://www.esa.int/ESA>