CanSat

poznámky z 2012

Praha 2012 **Vladimír Váňa**

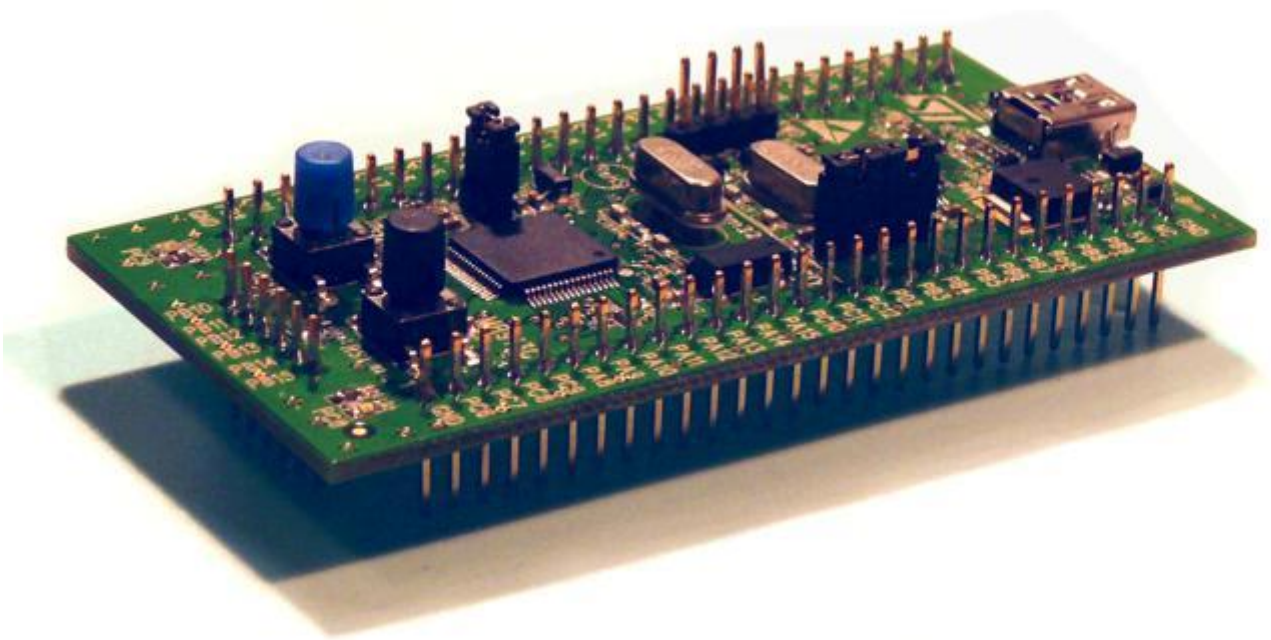
1. Úvod

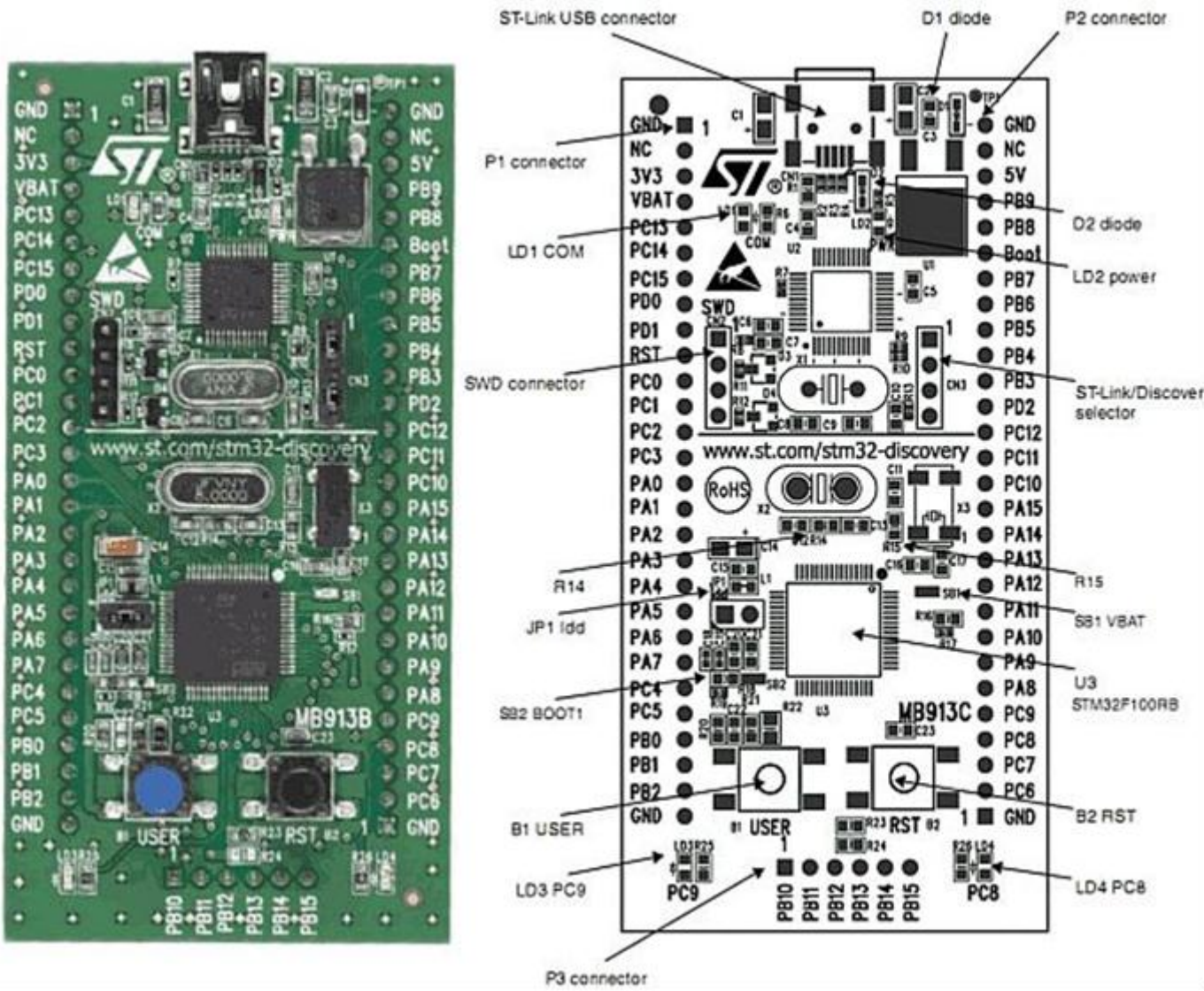
Úkolem těchto poznámek je podpora středoškolských studentů zapojených do soutěží CANSAT a je jejich 3 částí. Navazuje na poznámky z roku 2011. Neobsahuje tedy již základní informace obsažené v předchozích dílech. Zabývá se pokročilými technologiemi použitelnými při konstrukci CANSATů.
Jde především o konstrukci s programování palubního počítače s ARM Cortex STM32 a dále o čidla s digitálními rozhraními i2c a/nebo SPI. Proto je možné využít tento materiál i pro podporu výuky odborných předmětů zabývajících se touto problematikou.
Tento materiál se dále zabývá i komunikačním systémem CANSATů tj. fligh vysílačem a pozemním přijímačem jak klasickým, tak softwarově definovaným.

2. Palubní počítač

2.1 palubní počítač s ARM Cortex počítači řady STM32F

Při volbě konkrétních typů a jejich provedení jsem vycházel z toho, že studenti při vývoji software pro Cansat zabezpečující sběr dat z čidel a následné odeslání do vysílače komunikujícího s pozemní stanicí použijí startkit a bude-li palubní počítač obsahovat stejný MCU jako startkit, nebude sebemenší problém s přenesením (implementací) hotového a funkčního sw na palubní počítač.
Vzhledem k dostupnosti laciného startkitu *STM32VL-Discovery* (viz též druhý díl skript [1.2])

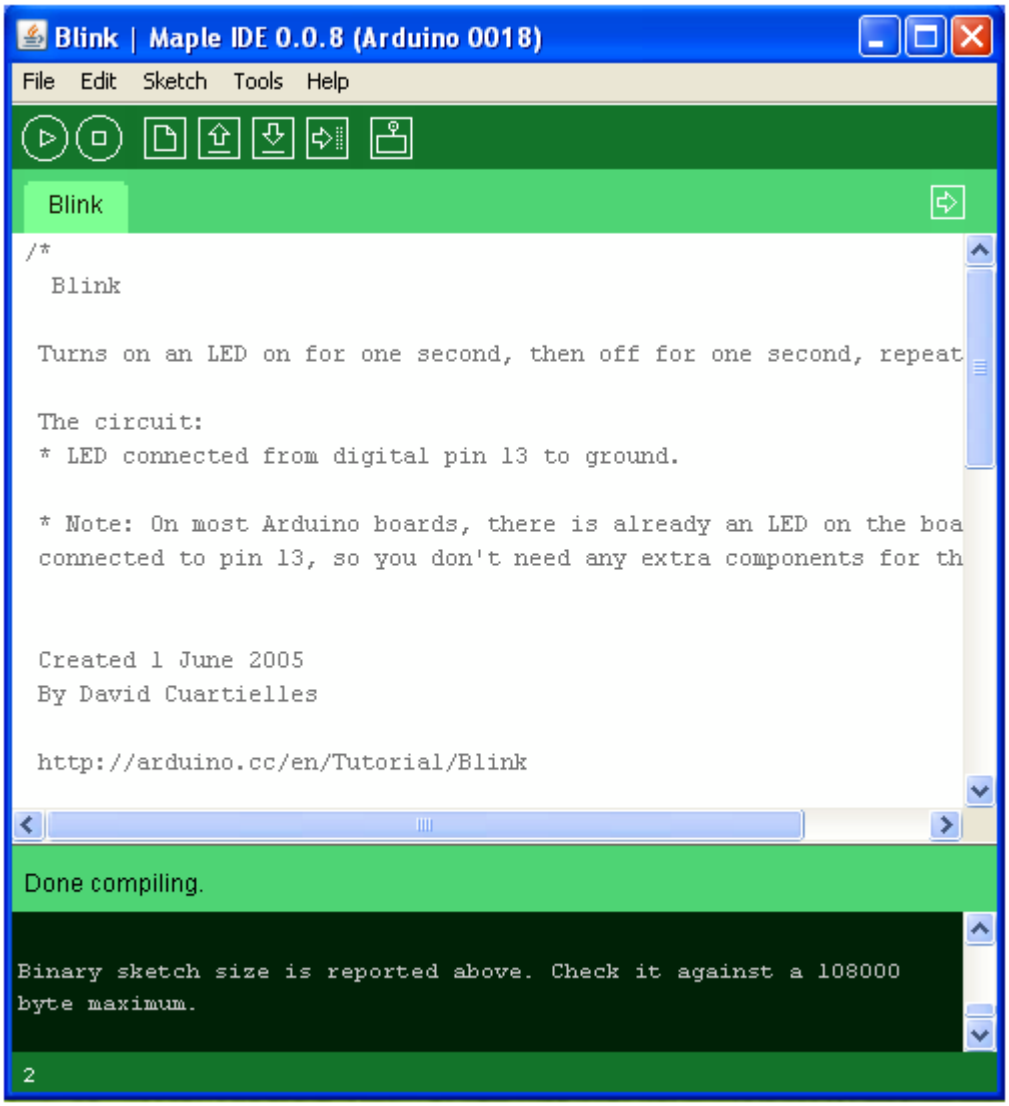




1. padla volba na *STM32F100RBT6* obsahující 128 kB Flash, 8 kB SRAM a je zapouzdřen v LQFP 64, tedy poměrně velkém pouzdře 10 x 10 mm. Startkit obsahuje i *STM32F103C8T6* sloužící k propojení MCU startkitu *STM32F100RBT6* s PC prostřednictvím USB a *ST linku* a tedy k jeho programování a k debugingu. Není tedy používán ve startkitu jako ten MCU, pro který vytváříme uživatelské programy.

Nicméně vytvořil jsem i verzi palubního počítače s *STM32F103C8T6*. Obsahuje 128 kB Flash, a 20 kB SRAM a je zapouzdřen v LQFP 48 tj. v pouzdře o velikosti 7 x 7 mm. Oproti *STM32F100RBT6* mezi vestavěnými periferiemi má i USB.

Hlavním důvodem, proč jsem vytvořil další palubní počítač právě s *STM32F103C8T6* je projekt *Maple* (resp. *Maple Mini*) z http://leaflabs.com/ a http://wiki.leaflabs.com/index.php?title=Main_Page využívající právě tento MCU. Výsledkem projektu *Maple* je systém *Arduino* kompatibilní a to včetně vývojového prostředí i programovacího jazyka *Wiring*. Výkon 32bitového ARM Cortex procesoru je ovšem větší než 8bitové ATMega původního systému *Arduino*. Systém *Arduino* se v poslední době stává populárním zejména pro různé amaterské hobby konstrukce a byl ESA použit v stavebnicích *PrattHobbies* v 1. a 2. Ročníku evropské soutěže středoškoláku CANSAT.



Poznámka k programování paměti Flash obvodů STM32

V dalších podkapitolách si uvedeme konstrukční provedení palubních počítačů s *STM32F100RBT6*, *STM32F103C8T6* a *STM32F405RGT6*. Jsou tvořeny malou destiškou PCB obsahující vlastní obvod MCU, blokovací kondenzátory, signalizační LED, krystal spolu s dvěma kondenzátory a dále konektory, které jsou propojeny se všemi piny obvodu.
Tento obvod je vždy nový, dodaný výrobcem a tak při prvním programování jeho programové paměti flash nemůžeme použít SL-link a tedy např. využít programátor z nějakého startkitu jako je *STM32VL-Discovery*, *STM32L-Discovery* či *STM32F4-Discovery*. Čím je to způsobeno a jak proto budeme postupovat si nyní vysvětlíme.

Pro využití protokolu *SWD* a tedy i výše uvedených startkitů či protokolu *JTAG*, musí být toto v MCU povoleno. Bohužel u nových MCU toto není defaultně provedeno. Povolení se provádí **programově** pomocí volání funkce (pro MCU řady *STM32F10x*)

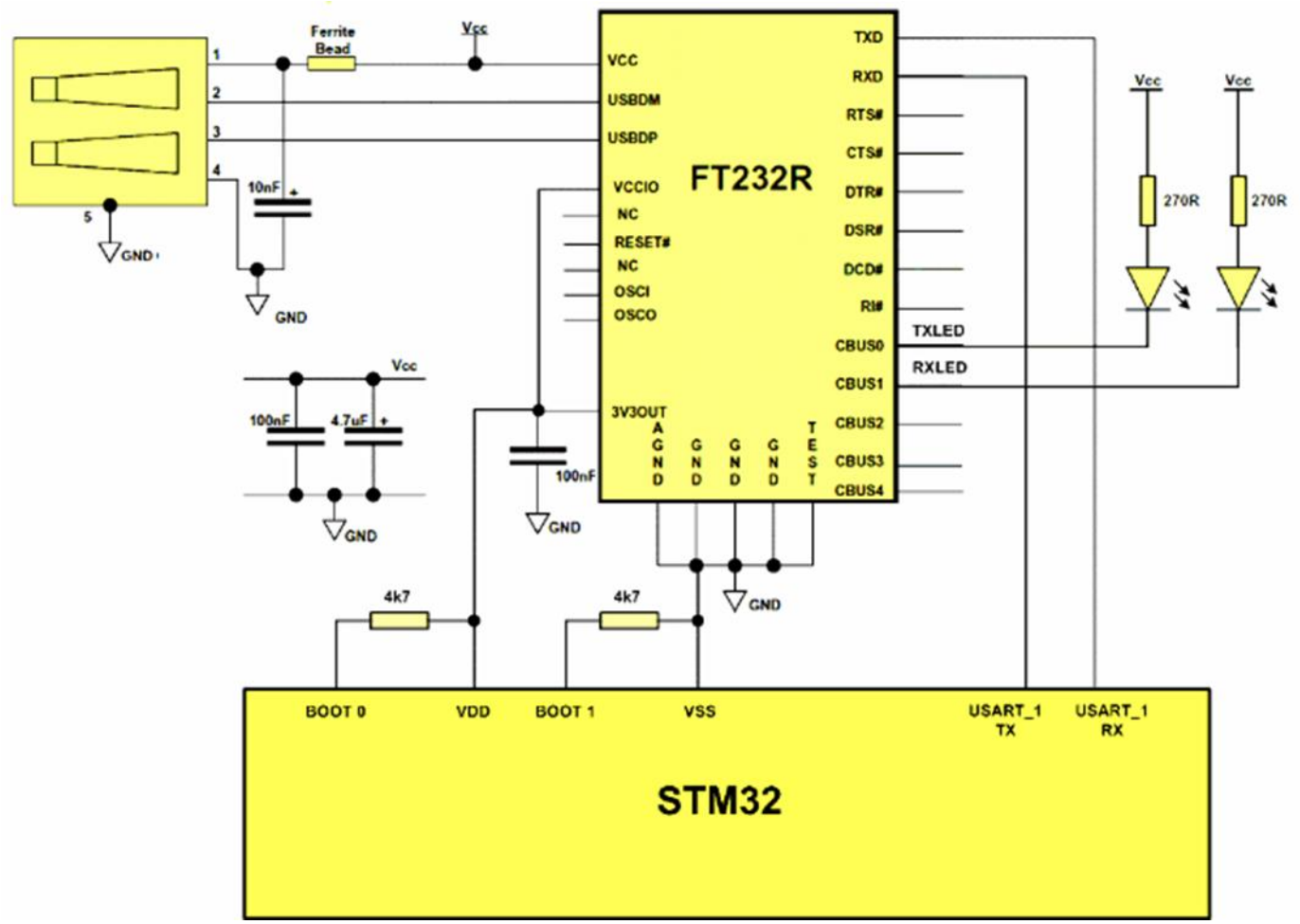
```
GPIO_PinRemapConfig(GPIO_Remap_SWJ_NoJTRST, ENABLE);    // Full Enable SWJ
```

Toto volání funkce umístíme ve zdrojovém kódu nějakého našeho programu v jeho inicializační části, program přeložíme a získaný .hex nebo .bin soubor použijeme jako zdroj pro naprogramování paměti flesh. Po spuštění programu v MCU je povoleno *SWJ* a poté již k programování budeme moci používat *SWD*. Takže zbývá ještě vyřešit, jak tento program dostaneme do MCU v době, kdy ještě *SWJ* není povoleno.
Využijeme k tomu výrobcem vestavěný bootloader. Bootloader přes RS232C je v mikrokontroléru *STM32F10x* automaticky aktivován v případě, že je na Boot pinech mcu nastavena konfigurace „System memory“. Jedná se o stav kdy je vývod **Boot0** = High a **Boot1** = Low. Hodnota pro Boot piny je mikrokontrolérem stanovena při náběžné hraně SYSCLK a současně je-li aktivní Reset pin mikrokontroléru.

Konfigurace Boot pinů

BOOT1	BOOT0	Boot mode	Popis
X	0	User Flash memory	User Flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space

Bootloader v mikrokontroléru *STM32F10x* je podporován přes jeden interface, a to USART1, vývody **PA.9/USART1_Tx**, **PA.10/USART1_Rx**. U 64 pinového *STM32F100* je **PA9** na pinu 42, **PA10** na pinu43, BOOT0 na pinu 60 a BOOT1 (PB2) na pinu 28. Zároveň z tabulky vidíme důvod, proč na našich palubních počítačích máme pomocí odporů zajištěno, že na pinech **BOOT1** a **BOOT0** máme nastavenu úroveň logická 0 – po resetování počítače bude počítač vykonávat náš uživatelský program.
Pokud v době resetování bude na **BOOT0** úroveň 1, bude MCU řízen programem Bootloader a bude se pokoušet komunikovat přes UART1. Interfejs mezi MCU a USB portem počítače PC může vypadat např. takto:



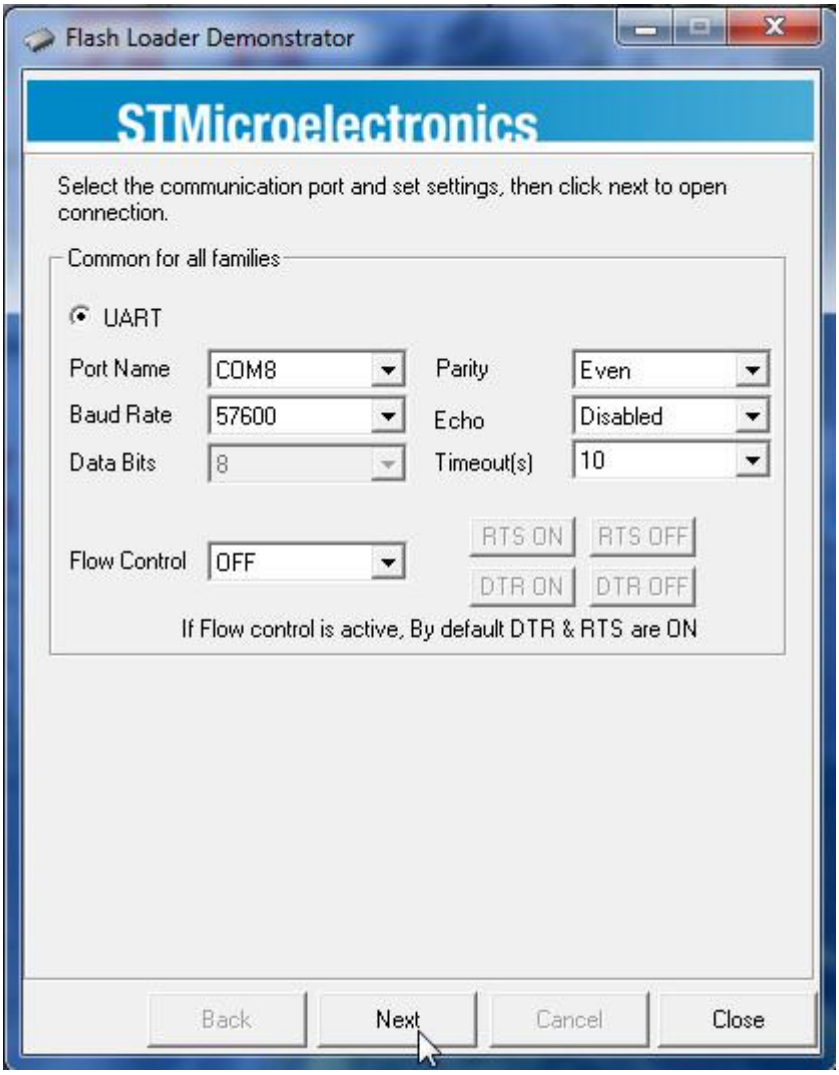
Pokud se mu to podaří a na druhé straně komunikace (tj. v PC) bude spuštěn vhodný program, můžeme pomocí tohoto programu poslat náš nový uživatelský program do MCU a poté, po odstranění 1 z jeho pinu **BOOT0** a resetování, již náš program bude v MCU řídit jeho chod.
Doporučeným programem na PC pro naprogramování flash paměti obvodů STM32 je *Flash Loader Demonstrator*. Jeho poslední verze je 2.5 a získáme ho zdarma stažením ze stránek firmy ST jako instalační soubor *Flash_Loader_Demonstrator_v2.5.0_Setup.exe* (je to program pro Windows). Po ninstalování Flash Load Demonstratoru nahrajeme náš program do programové paměti flash. Dále je uveden postup tohoto programování:

1. Propojíme palubní počítač s PC pomocí sériové linky RS232 (nemáme-li na PC RS232, lze použít i převodník USB -> RS232)
2. Připojíme napájení + 3,3 V palubního počítače
3. Stiskneme a držíme tlačítko **BOOT0** připojené k palubnímu počítači a stiskneme a uvolníme tlačítko **RESET**, rovněž vně připojené k palubnímu počítači. Poté uvolníme i tlačítko **BOOT0**. Takto je mikroprocesor připraven k nahrání programu.

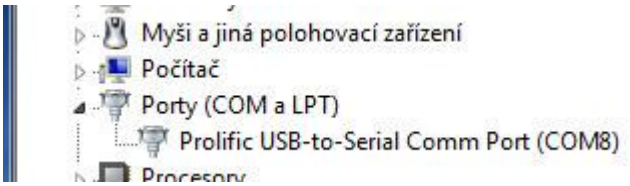
(V některých návodech je ale uvedeno, že tlačítko **BOOT0** musíme mít stlačeno po celou dobu práce se SW Flash Loader Demonstrator!!! - to ale nebyl můj případ, ostatně jednou rukou držet tlačítko

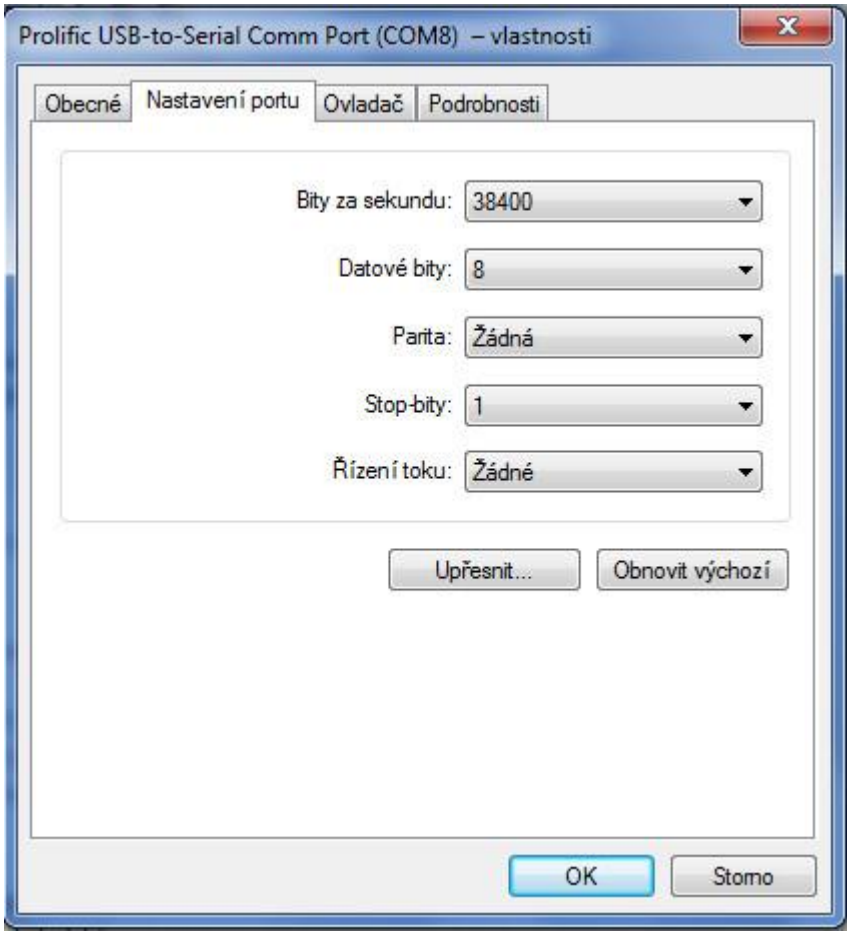
BOOT0 a jen druhou snímat obrazovky bych nezvládl …)

4. Na PC spustíme aplikaci *STMicroelectronics Flash Loader Demonstrator V2.4.0* a na úvodní obrazovce vybereme komunikační port (COM0, COM1, ..),

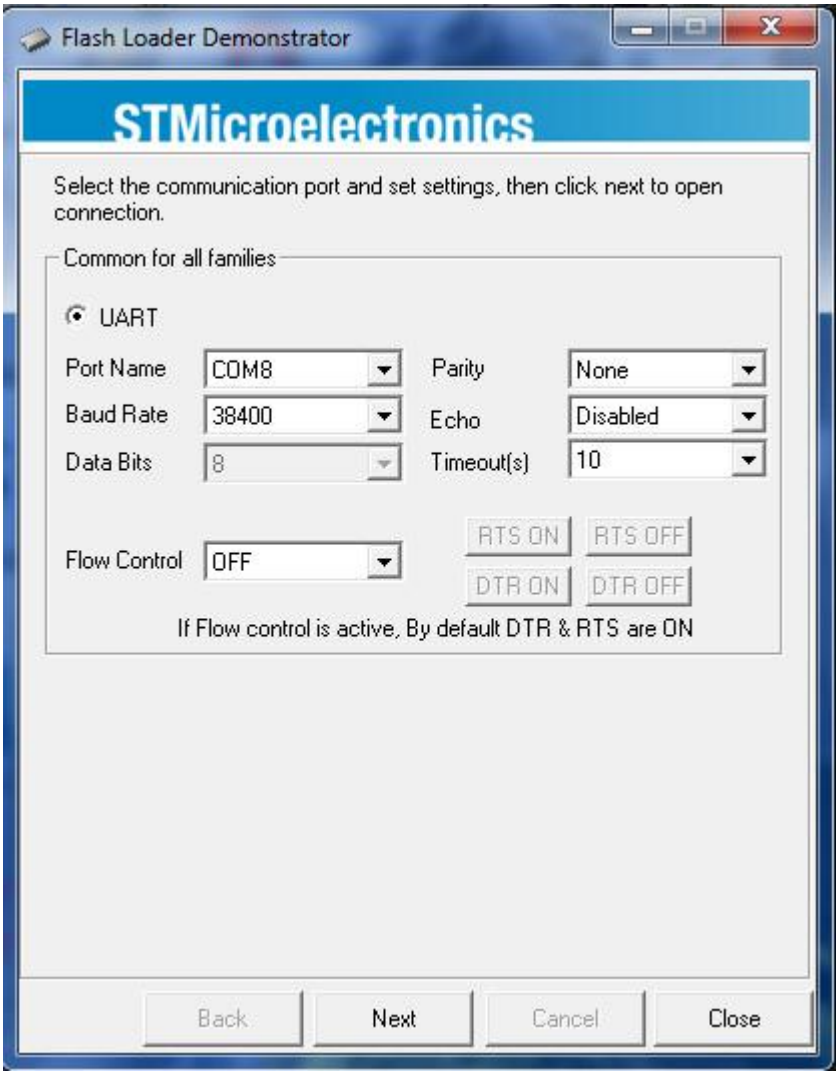


Nastavíme parametry sériového přenosu (pracoval jsem s notebookem s USB a s USB/com převodníkem PL2303) a virtuálním portem COM8

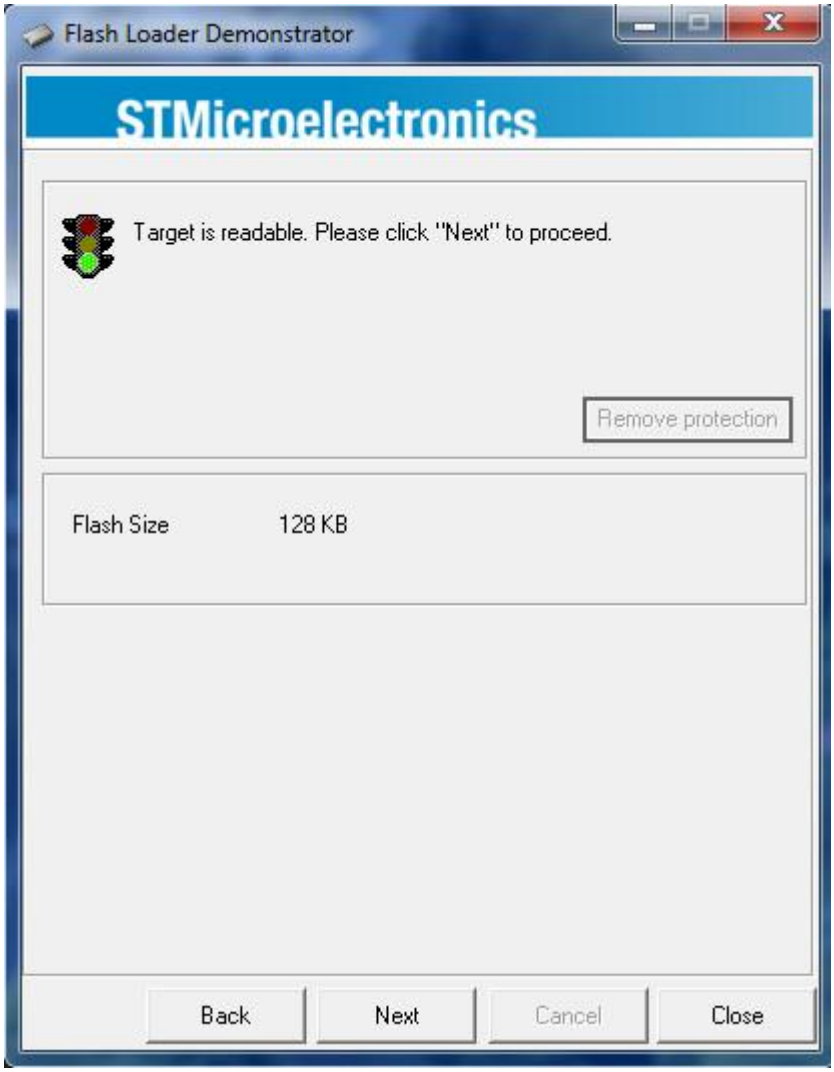




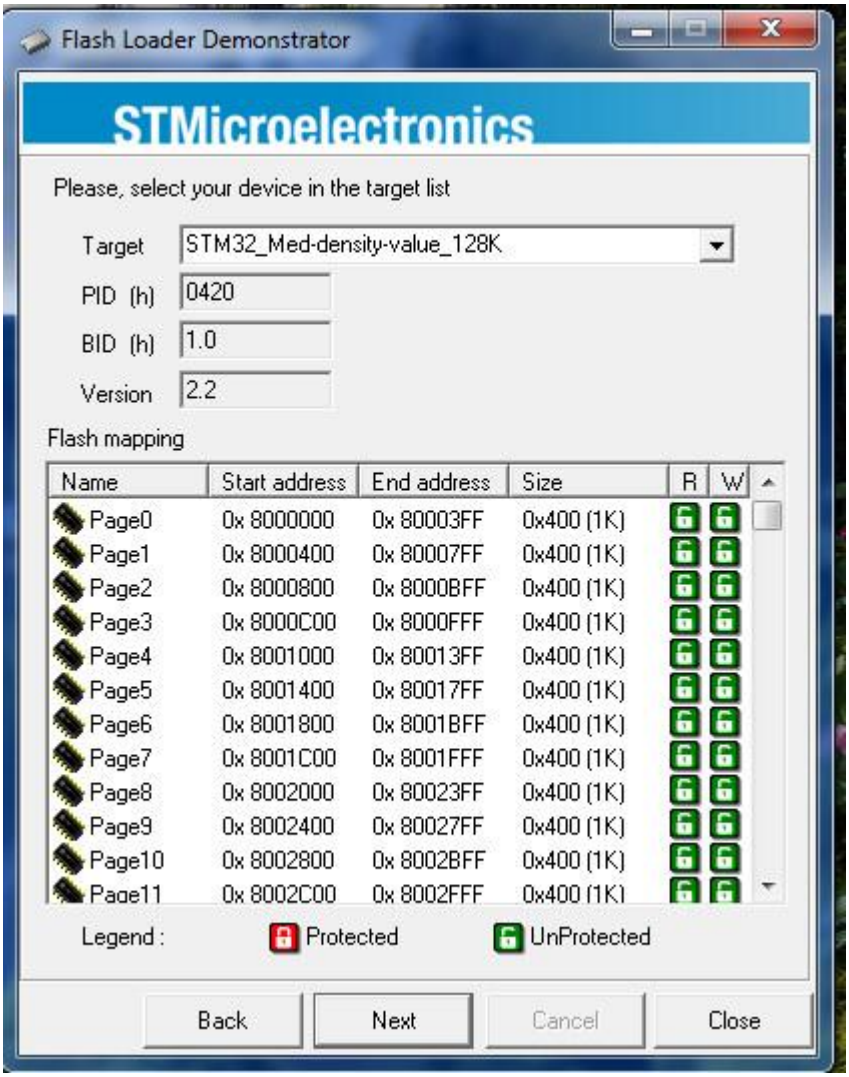
Pozn.
38400 Bd jsem nastavil z toho důvodu, že stejnou rychlostí se komunikuje s vysílačem a protože com port PC využívám i při testování palubního počítače kdy terminálovým programem kontroluji, co se vysílá a mám proto tento port nastavený na rychlost 38400. Kdybych měl jinou rychlost nastavenou ve *Flash Loader Demonstrator*, musel bych měnit *properties* com portu.



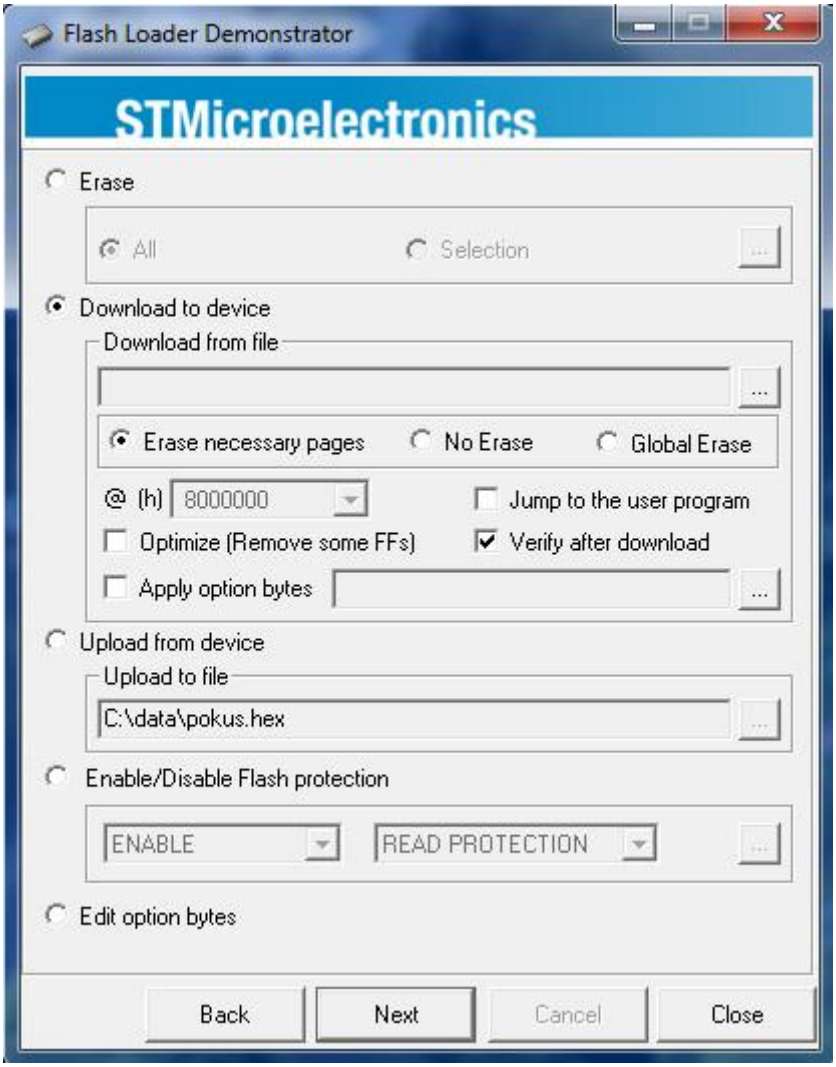
ostatní nastavení ponecháme a pokračujeme dále (tlačítko **NEXT**).



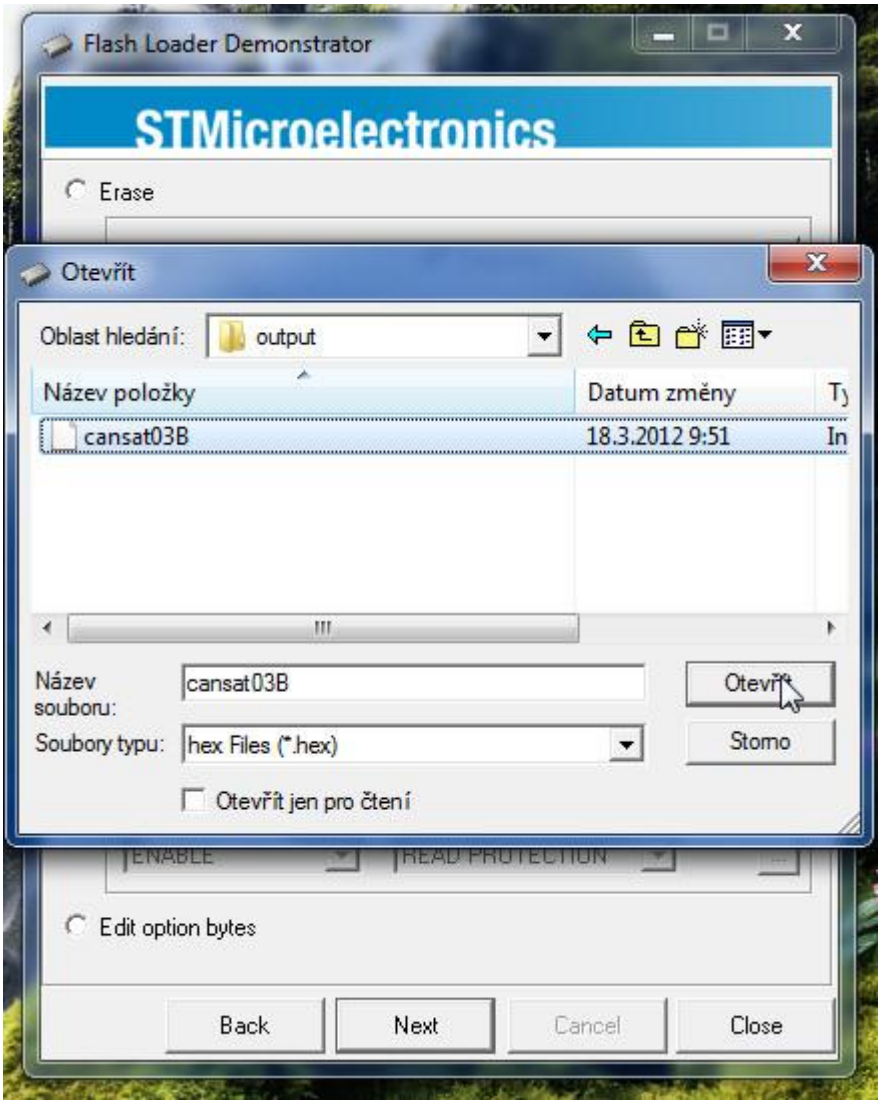
To již máme vyhráno, s navázáním spojení bývají totiž občas problémy. Dáme **Next**, dostaneme



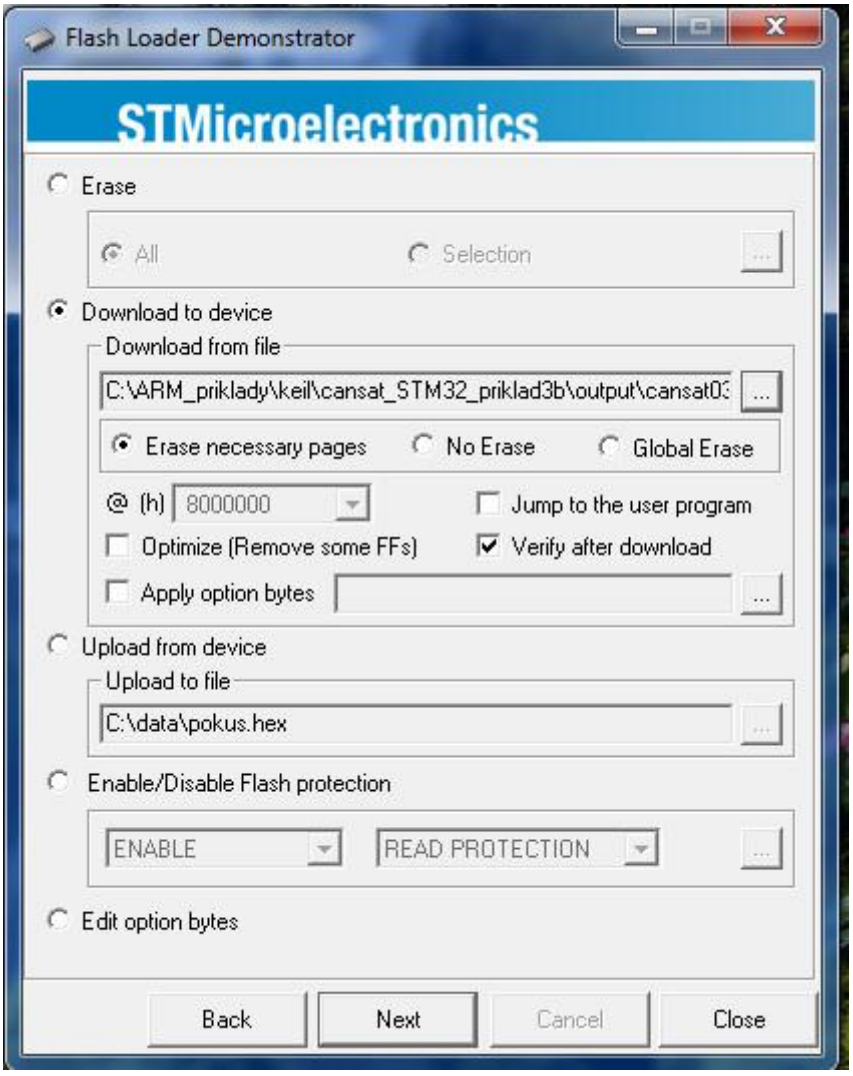
Opět dáme **Next** (předtím se pro jistotu podíváme, že to opravdu poznalo STM32 – viz textbox **Target**). Dostaneme



Klikneme na ikonku s třemi tečkami napravo od textboxu **Download to device** a najdeme náš soubor hex

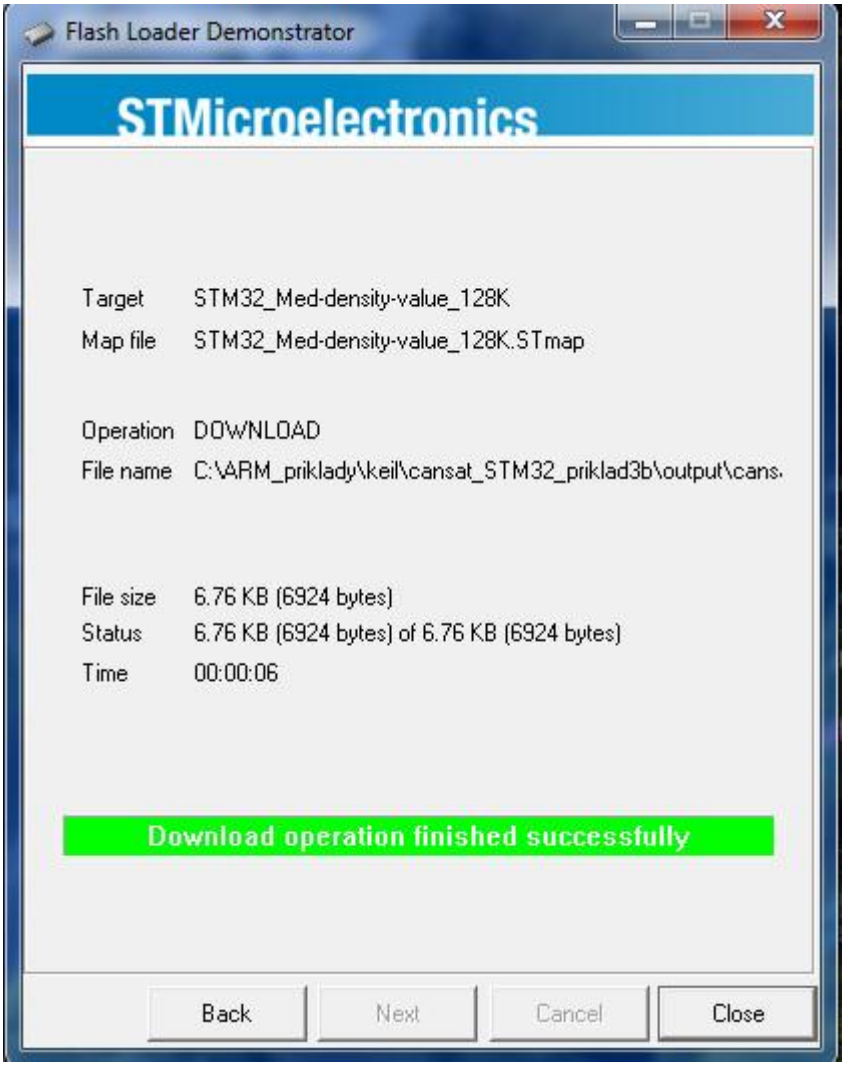


Klikneme na tlačítko **Otevřít**, máme



Klikneme na **Next** a začne se nahrávat náš program do flash paměti procesoru

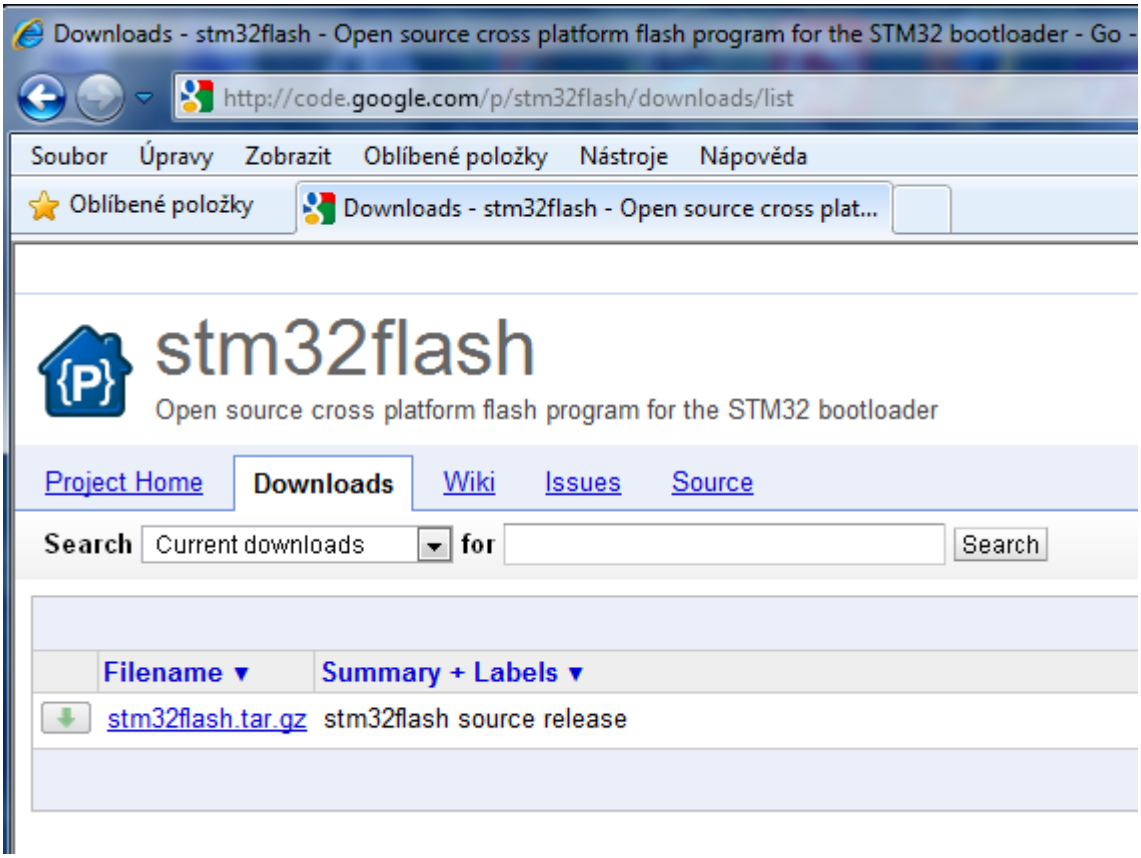
Poté bychom měli dostat



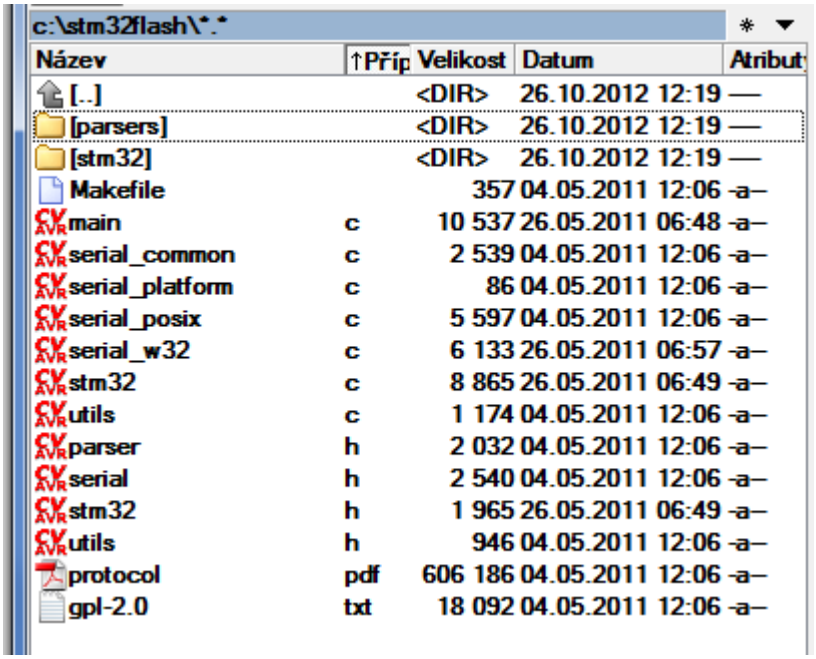
Stačí již jen kliknout na **Close**. Po resetování by už měl pracovat procesor pod nahraným uživatelským programem.

S nahráváním programu do paměti flash pomocí bootloaderu a programu *FlashLoader Demonstrator* bývají občas problémy s navázáním spojení s programovaným MCU! Někdy stačí jen snížit rychlost COM např. na 9600 Bd , popř. použít jinou verzi *Flash Loader Demonstrator*. Mě se však nejvíce osvědčilo pracovat pod OS Linux a používat program *STM32Flash*.

Proto si teď práci s STM32 pod Linuxem také ukážeme. Nejprve si z
http://code.google.com/p/stm32flash/downloads/list



Stáhneme soubor *stm32flash.tar.gz* a poté z něj vyextrahujeme (ve windows nebo v Linuxu) adresář *STM32flash* jehož obsah je

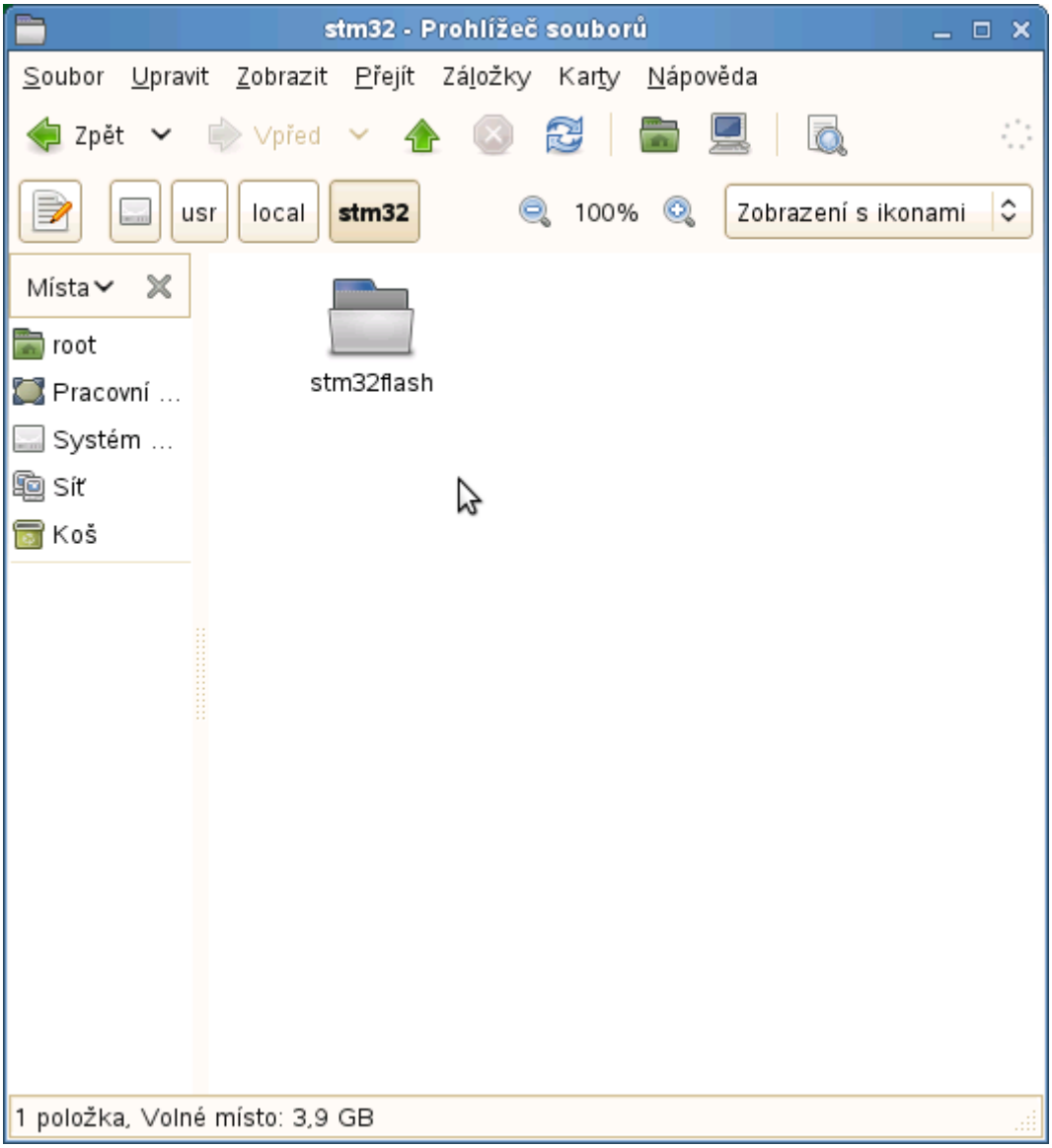


To může být překvapením pro uživatele windows zvyklé na instalační soubory msi, exe apod. U linuxu totiž často dostaneme jen zdrojový kód programu a jeho binární verzi, potřebnou ke spouštění programu si musíme vytvořit sami. Obvykle jde o překlad jednotlivých souborů zdrojového kódu free překladačem jazyka C **gcc**, slinkováním takto vzniklých object souborů, knihoven apod. Naštěstí na

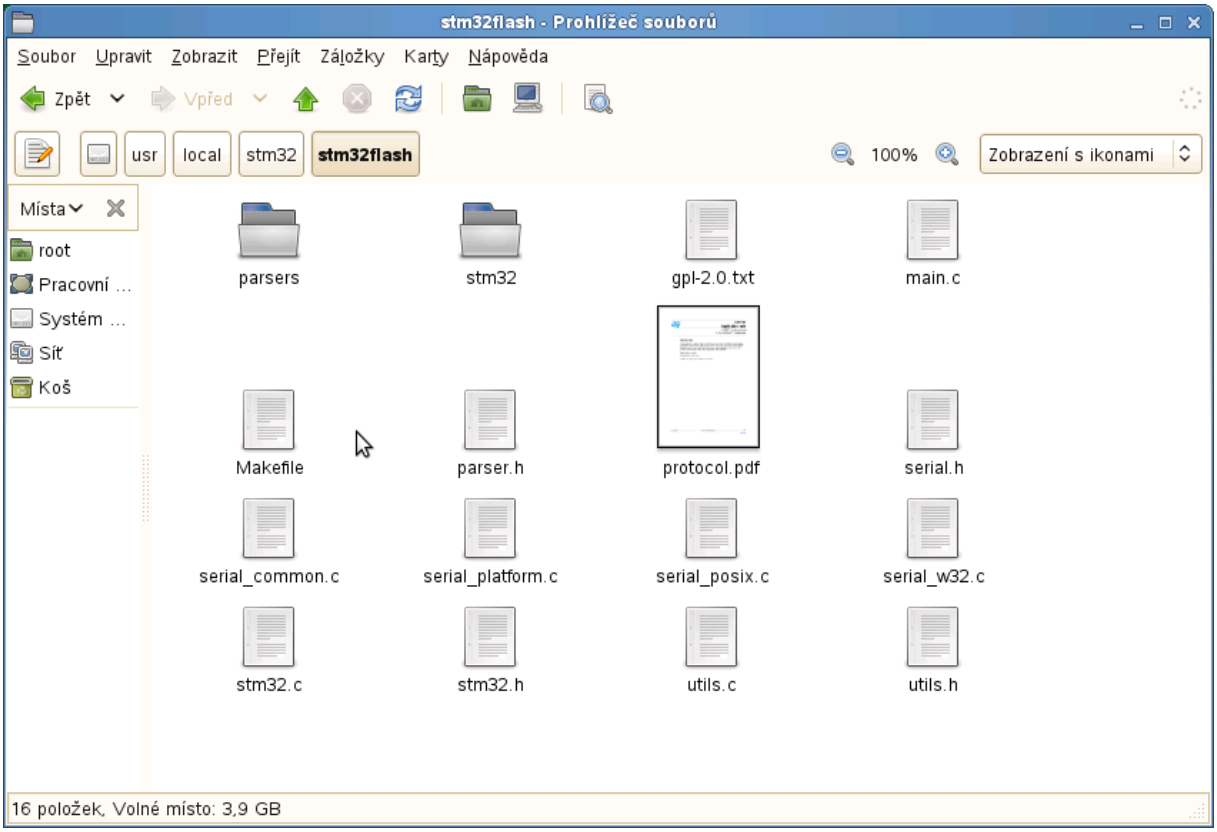
našem linuxu již obvykle máme vše potřebné nainstalované a hlavně, jak je vidět na obrázku výše, kromě zdrojového kódu jsme k instalaci dostali i soubor *Makefile*, obsahující potřebné informace k překladu a sestavení výsledného binárního kódu. Soubor *Makefile* totiž slouží jako podklad pro program *make*, který tuto činnost provede za nás. Na nás zbývá v linuxu, v režimu příkazového řádku (terminálu) v adresáři, ve kterém máme umístěn soubor *Makefile* i soubory se zdrojovými kódy spustit příkaz

Make install

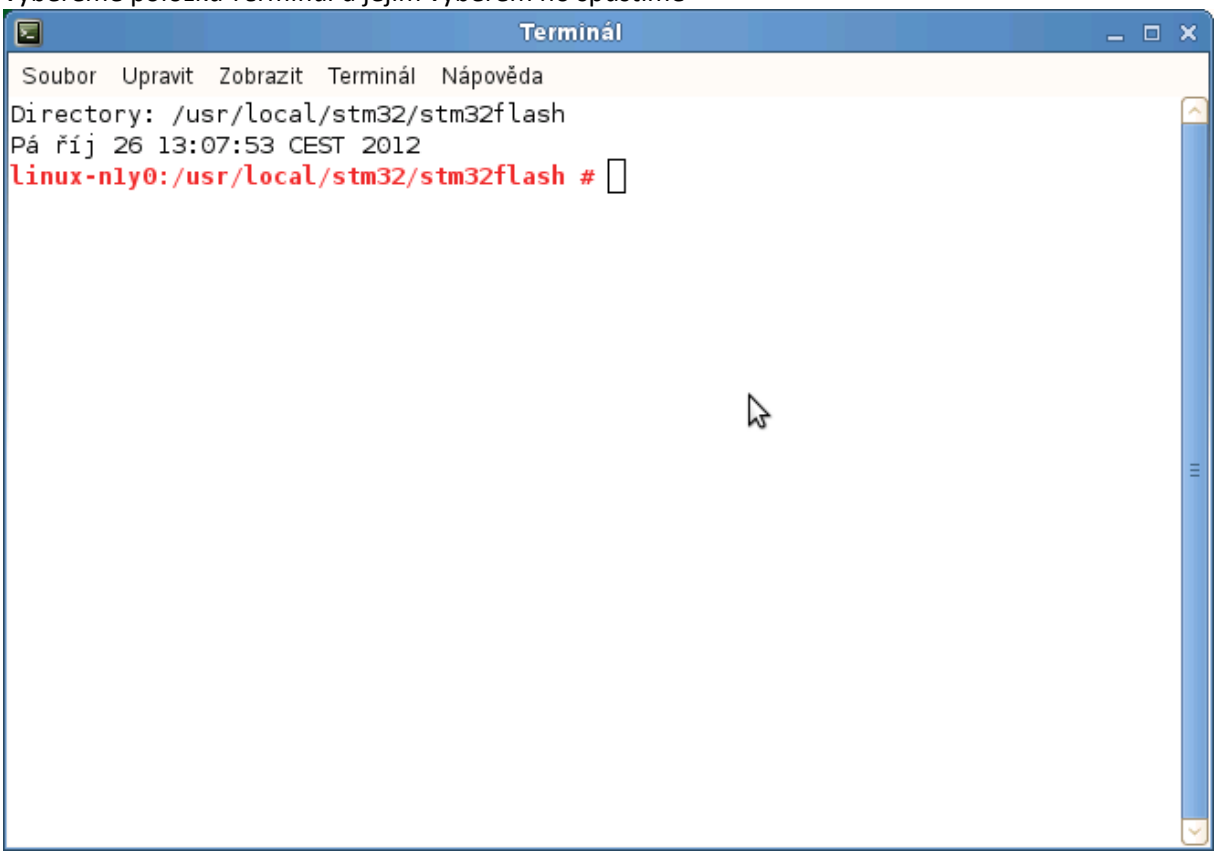
Ukážeme si to nyní na konkrétním případě. Pod OS linux vytvoříme v *usr/local* adresář *stm32* a do něj zkopírujeme adresář ***stm32flash*** i s jeho obsahem:



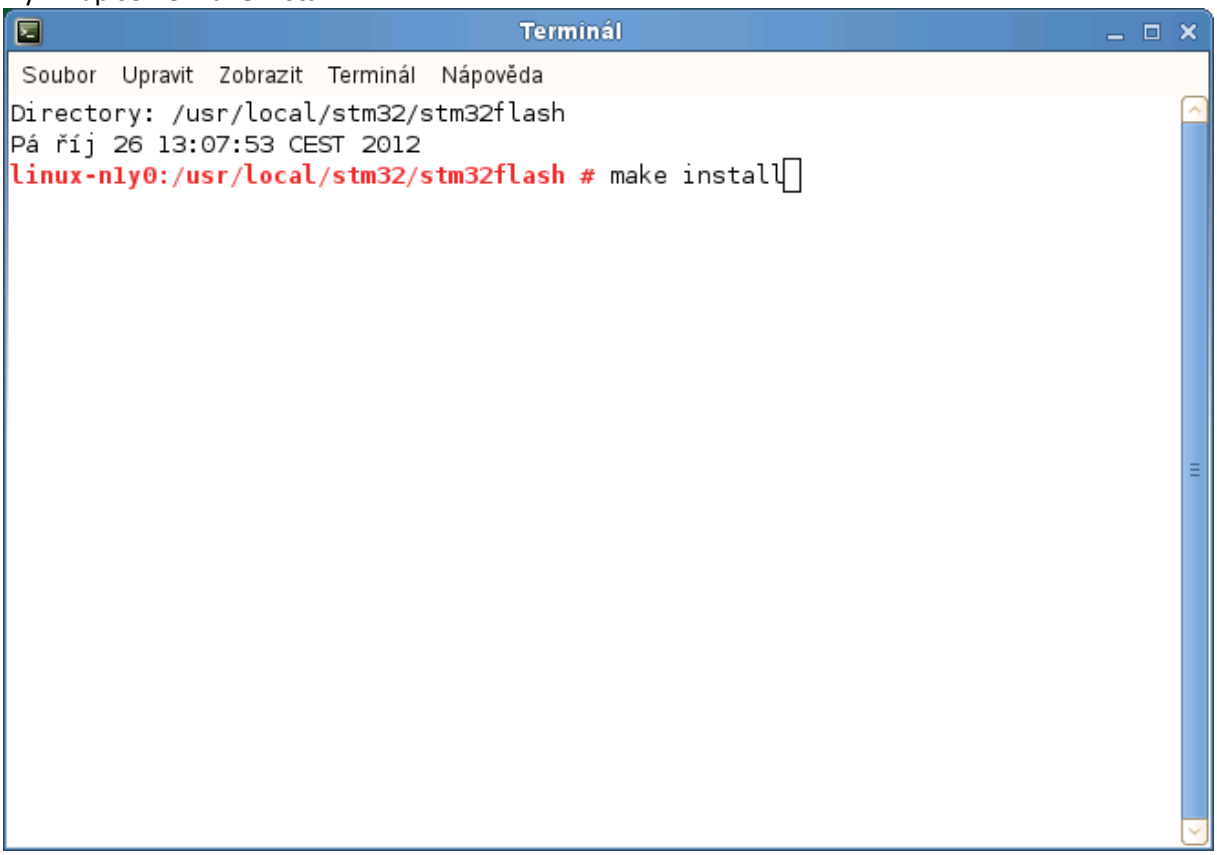
Obsah odresáře *stm32flash* je



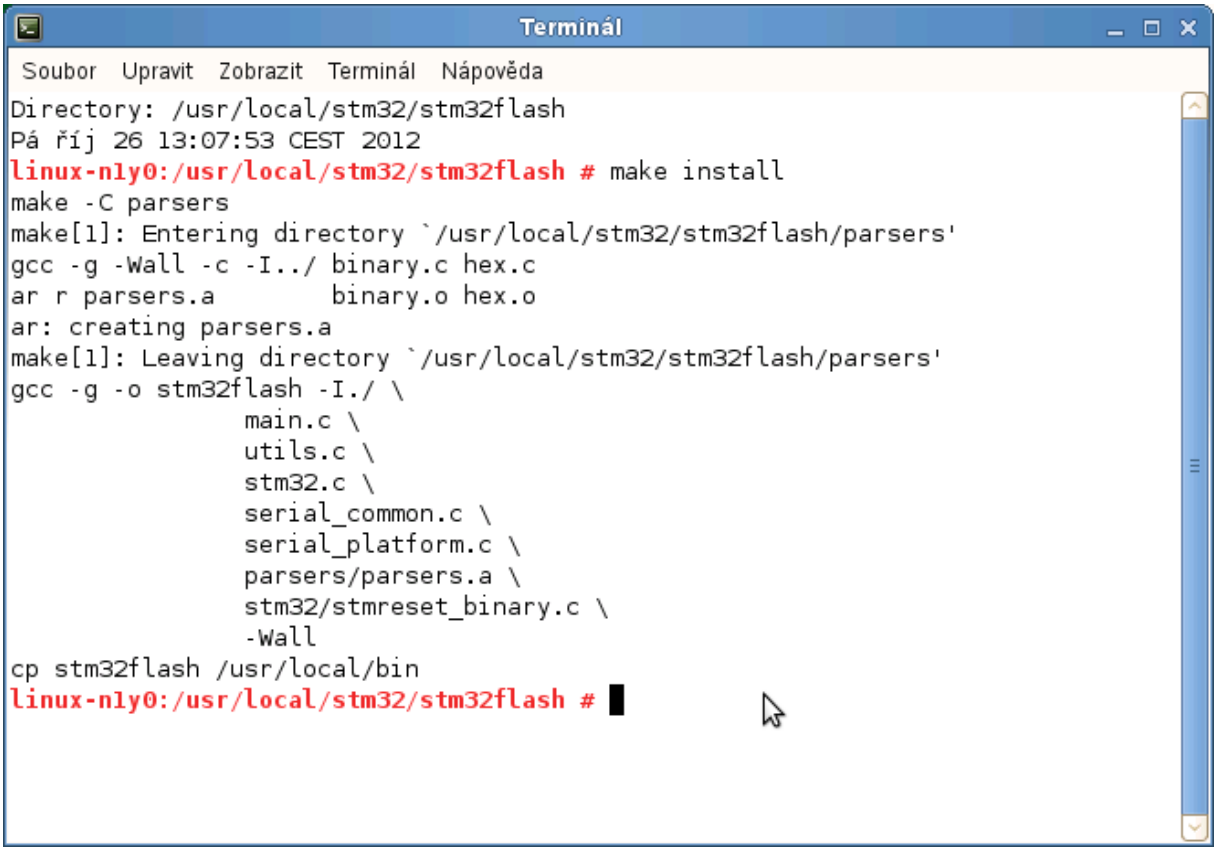
Kliknutím pravým tlačítkem myši na ploše *Prohlížeče souborů* spustíme lokální menu. V něm vybereme položku *Terminál* a jejím výběrem ho spustíme



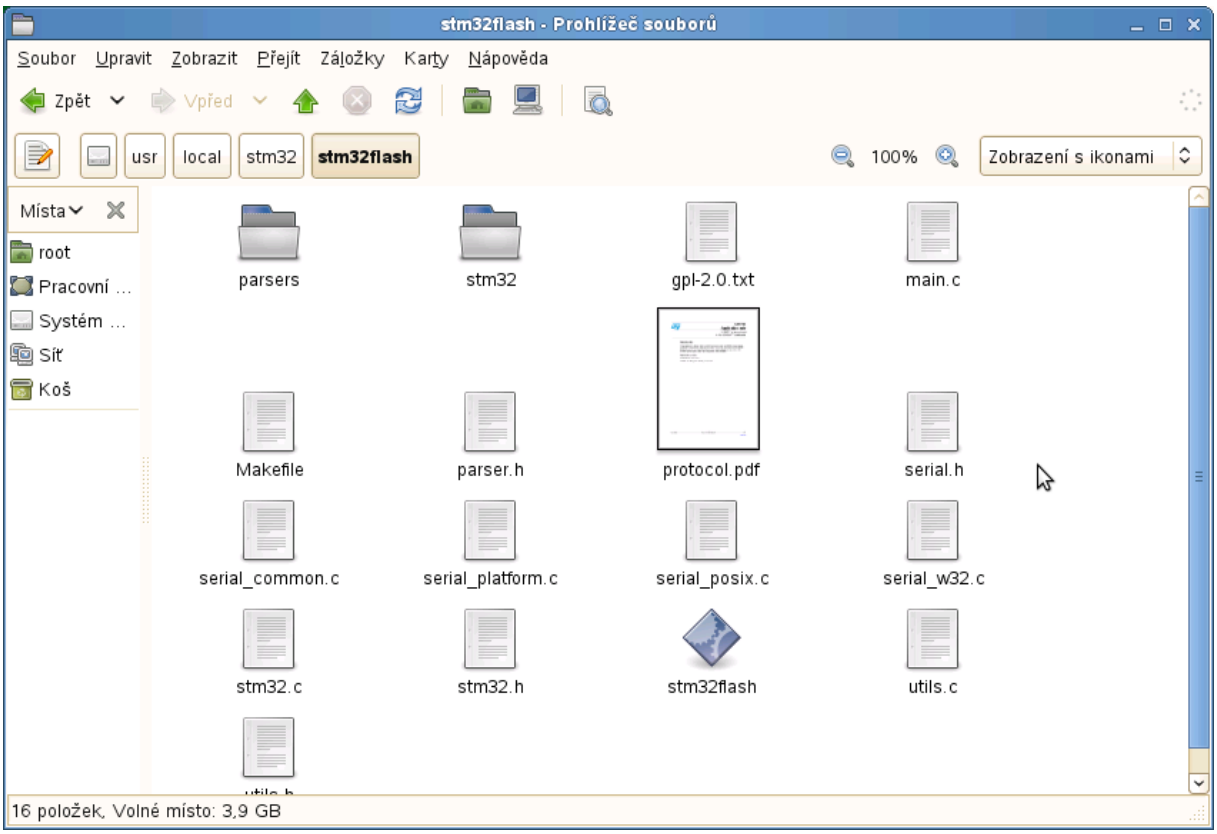
Nyní napíšeme **make install**



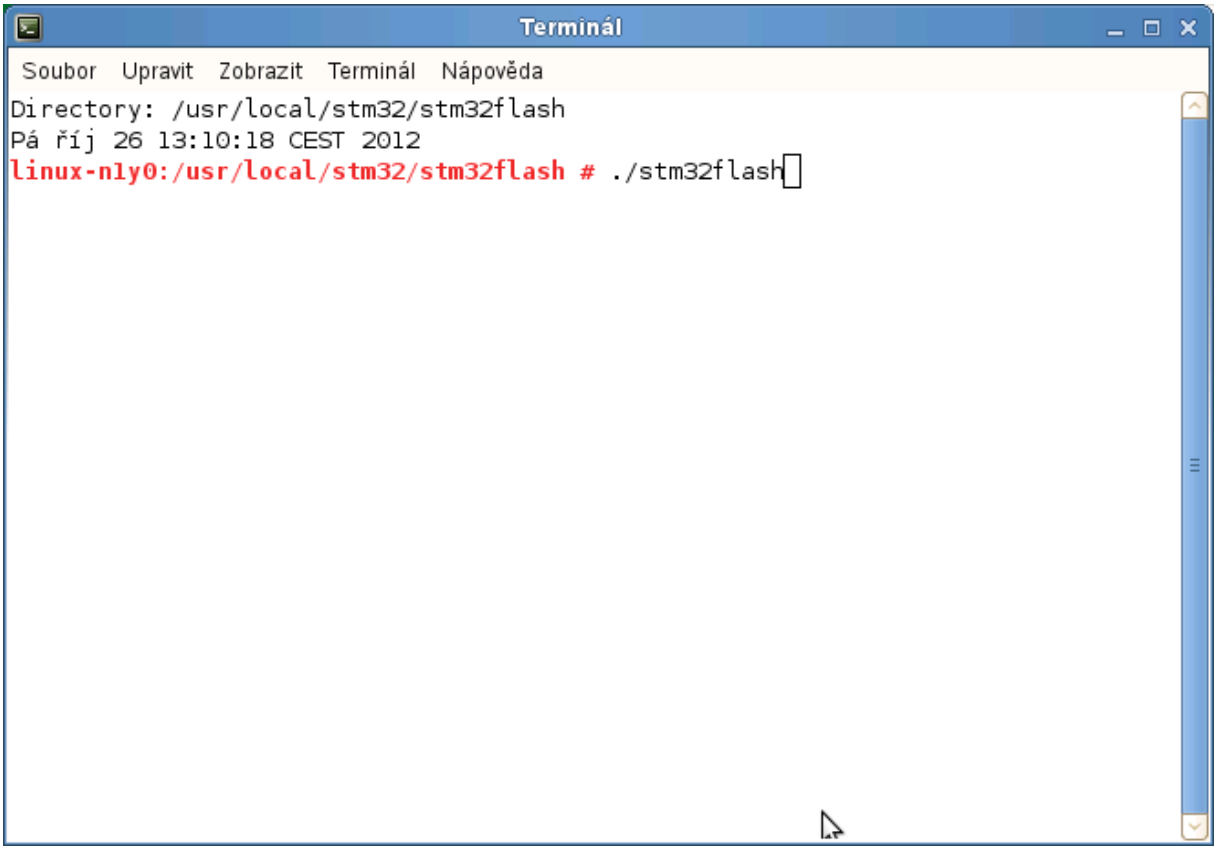
A tento příkaz spustíme, po chvíli dostaneme



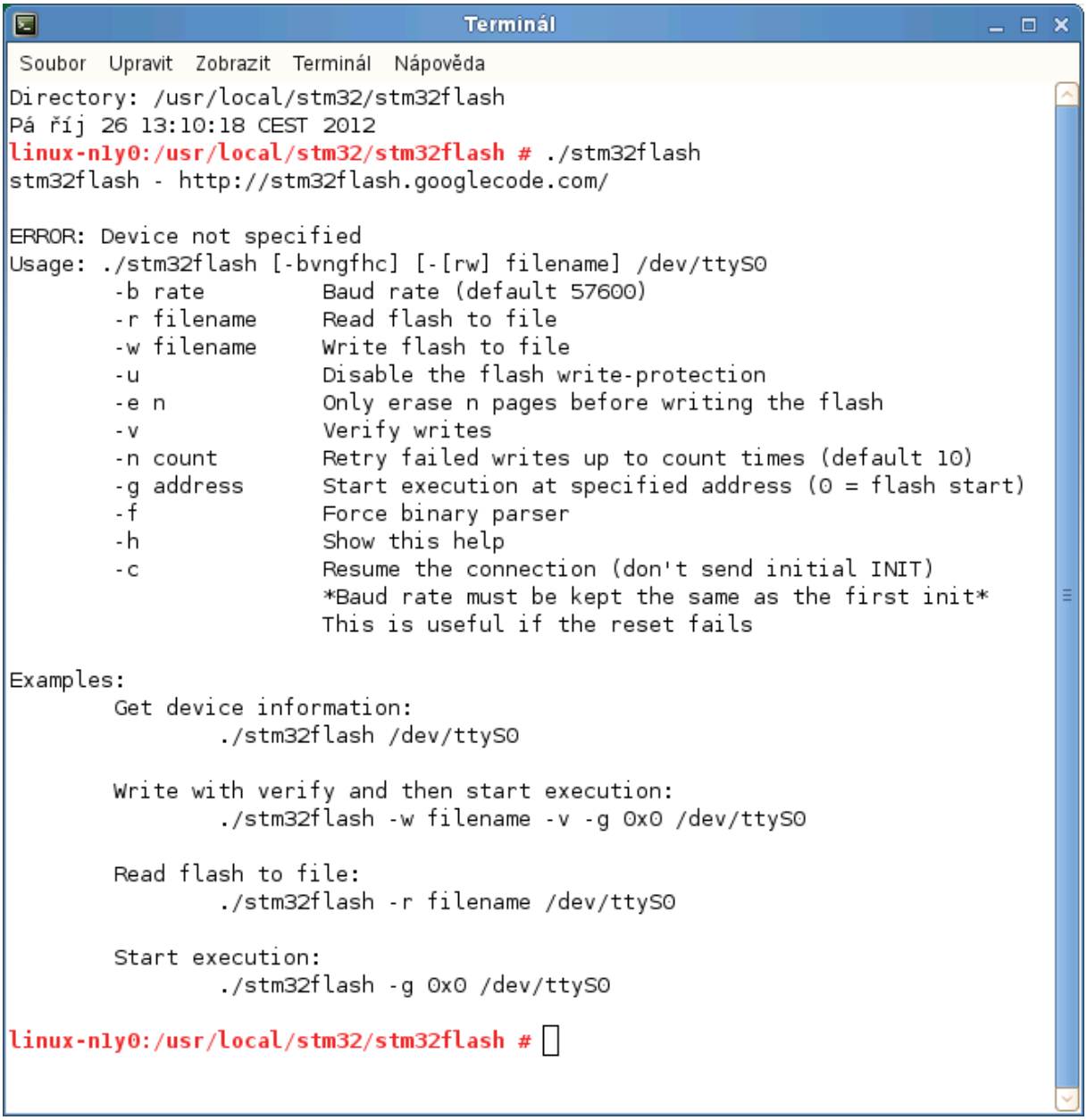
Tím je spustitelný program *stm32flash* vytvořen. Všimněme si jeho ikonky



Tento program můžeme spustit. Opět spustíme **Terminál** a napíšeme **./stm32flash**

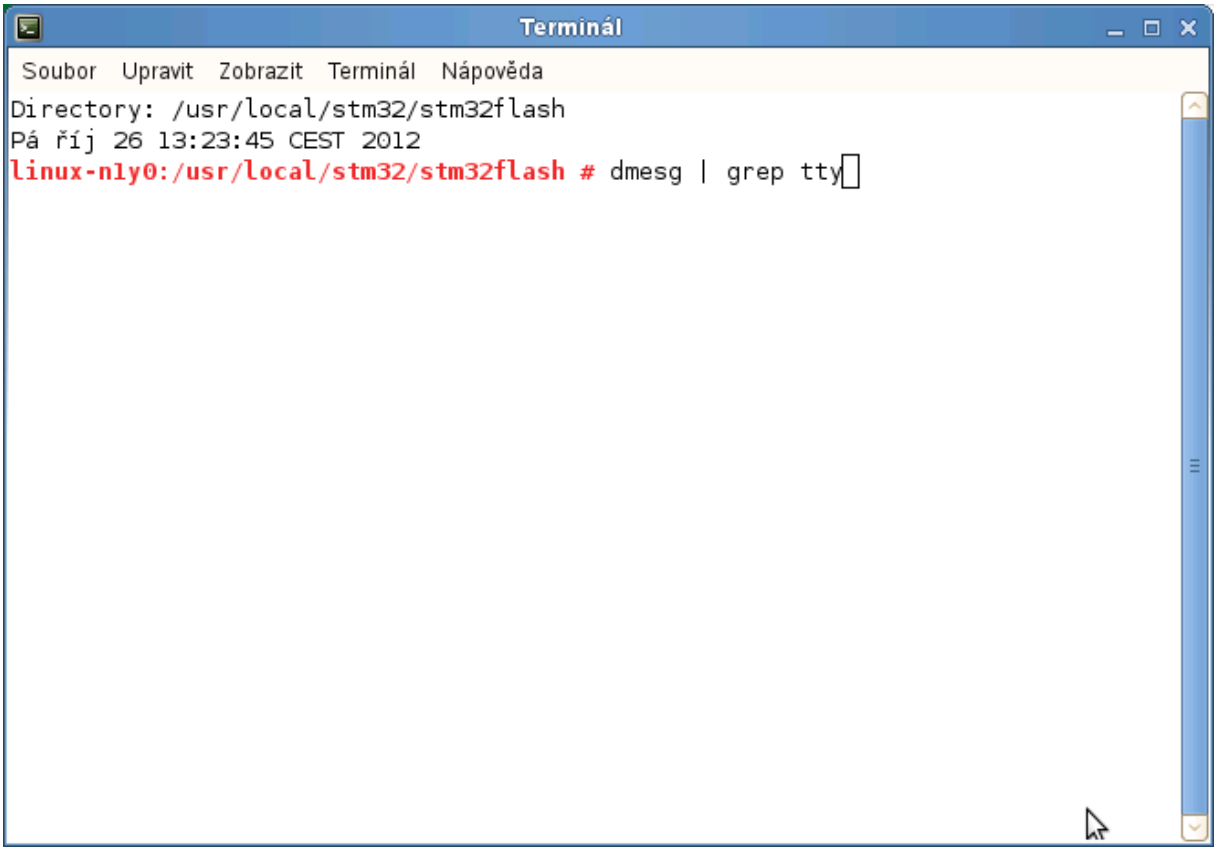


Program spustíme, dostaneme

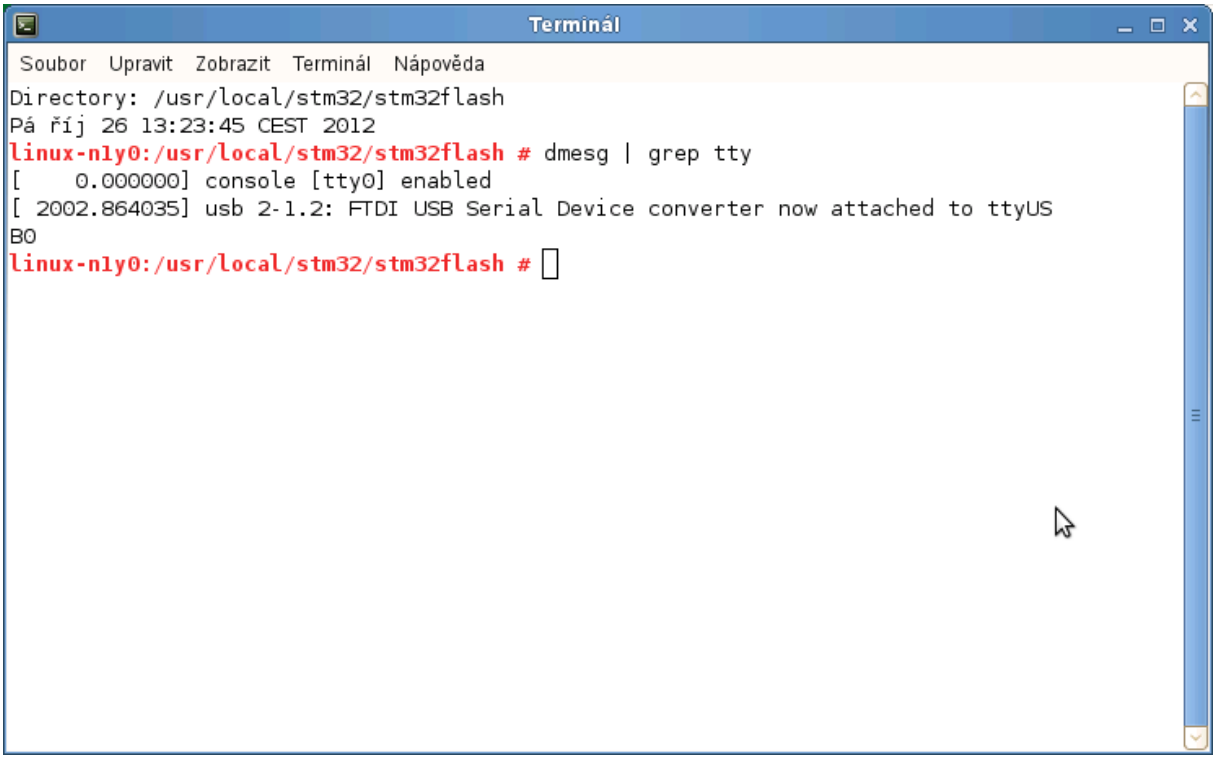


Vypsal se nám help ukazující používání programu *stm32flash*

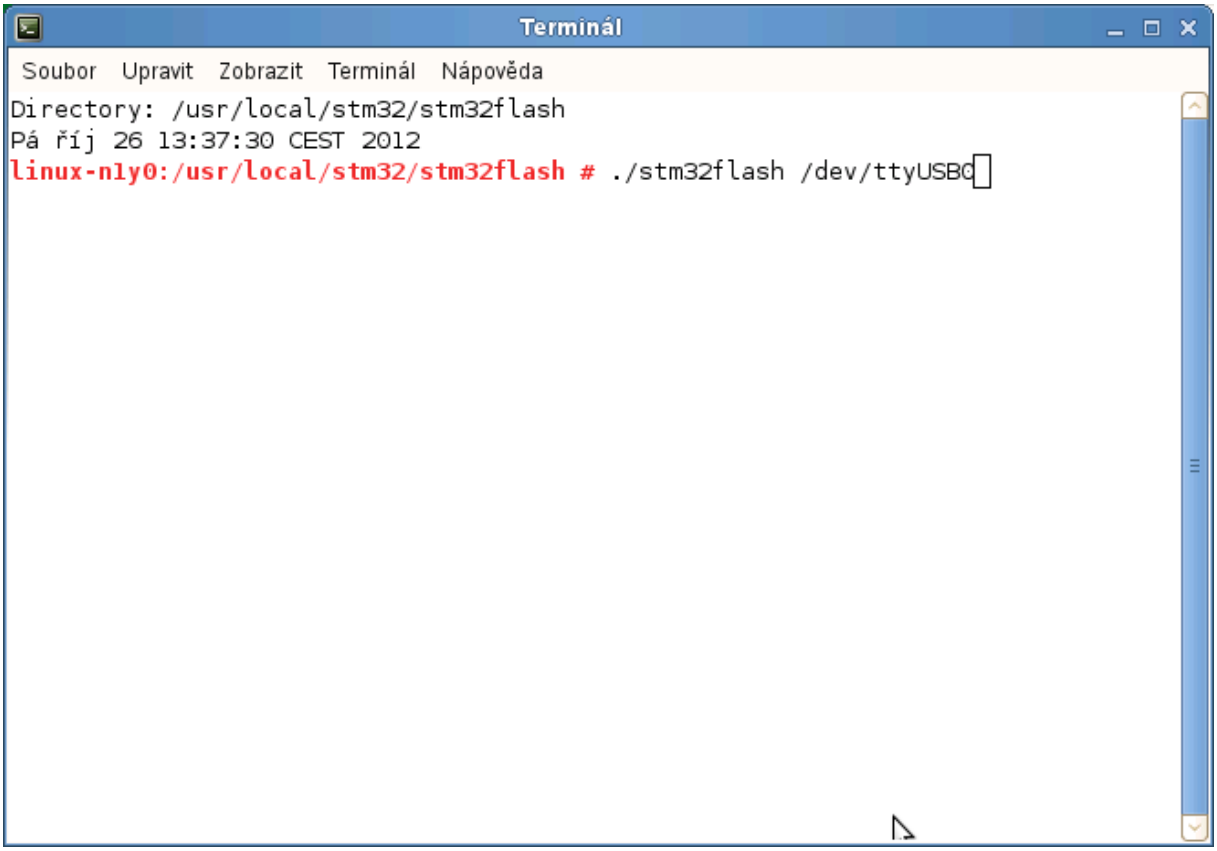
K usb PC připojíme převodník FT232R. Převědčíme se, že ho náš linux zná. V terminálu napíšeme



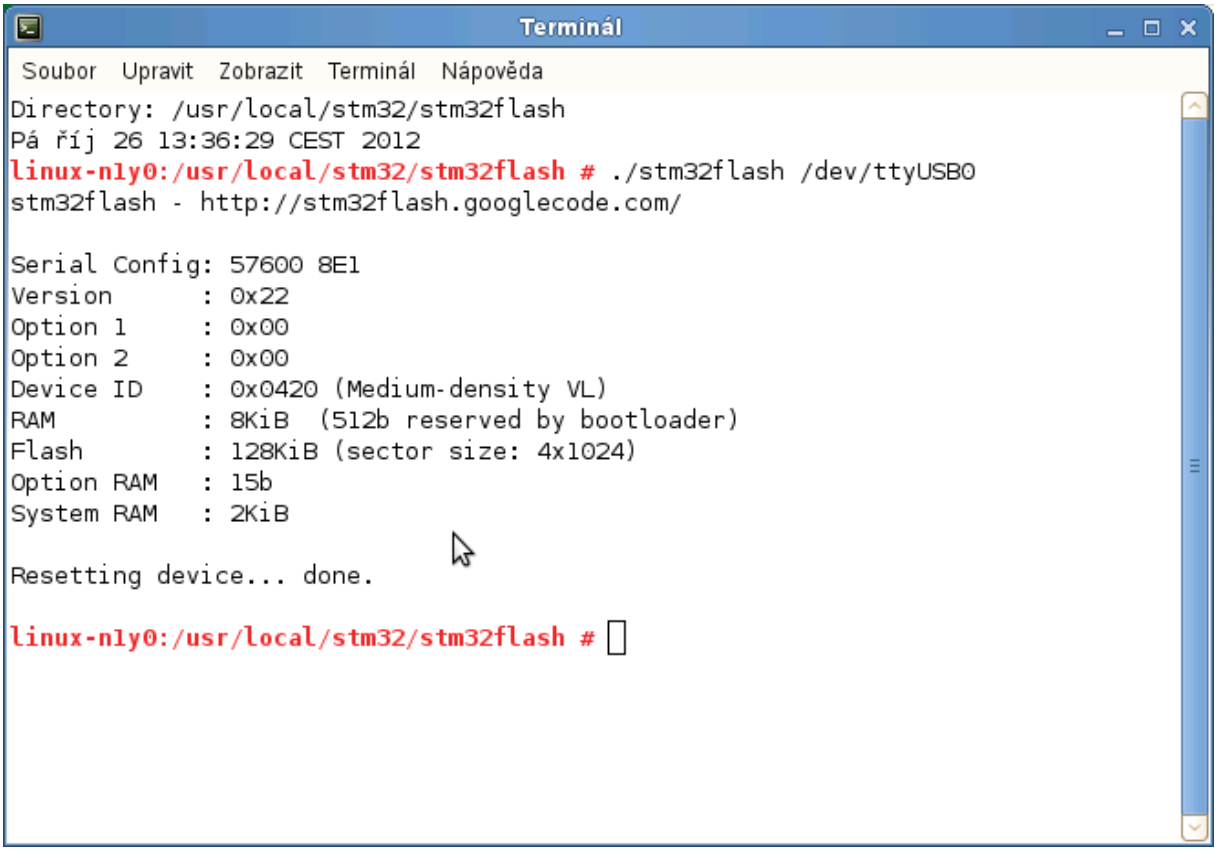
Dostaneme



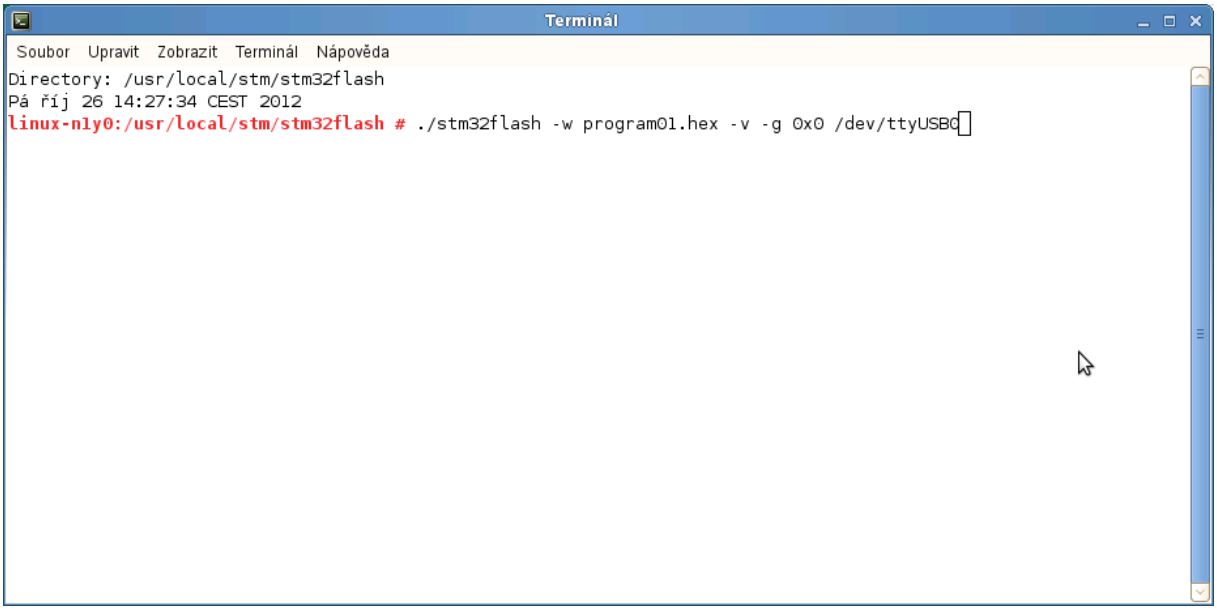
Vidíme, že je označen jako **ttyUSB0**. K tomuto převodníku připojíme onboard počítač s *STM32F100RBT6* . V terminálu napíšeme



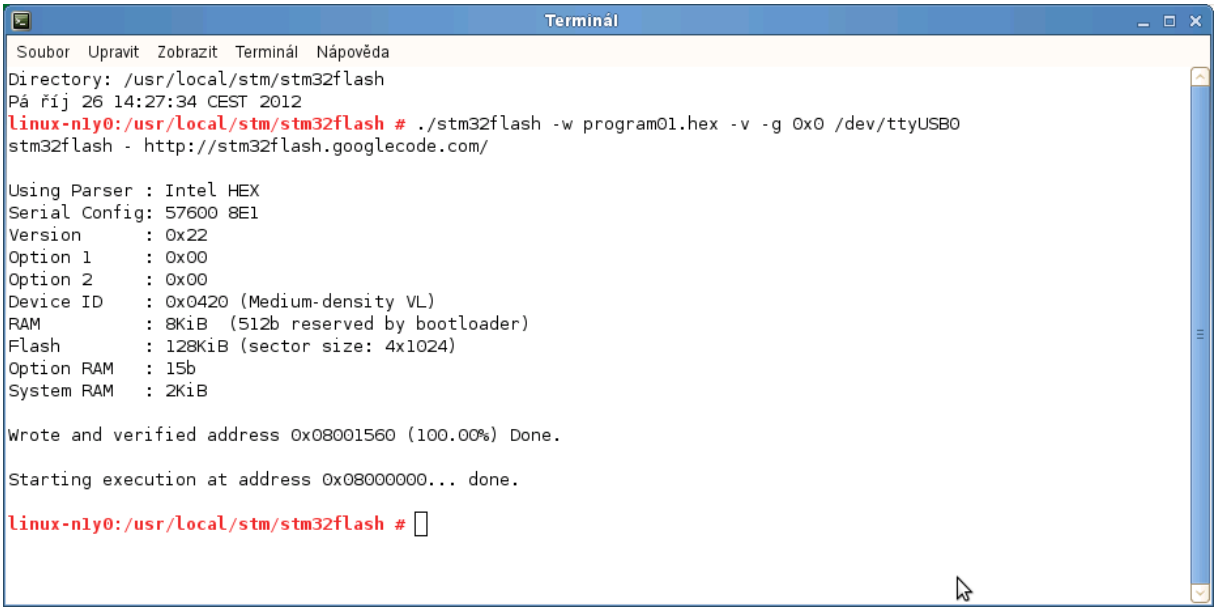
Na *STM32F100RBT6* připojíme logickou 1 na **boot0** a zresetujeme. Poté již příkaz **./stm32flash /dev/ttyUSB0 spustíme**. Dostaneme



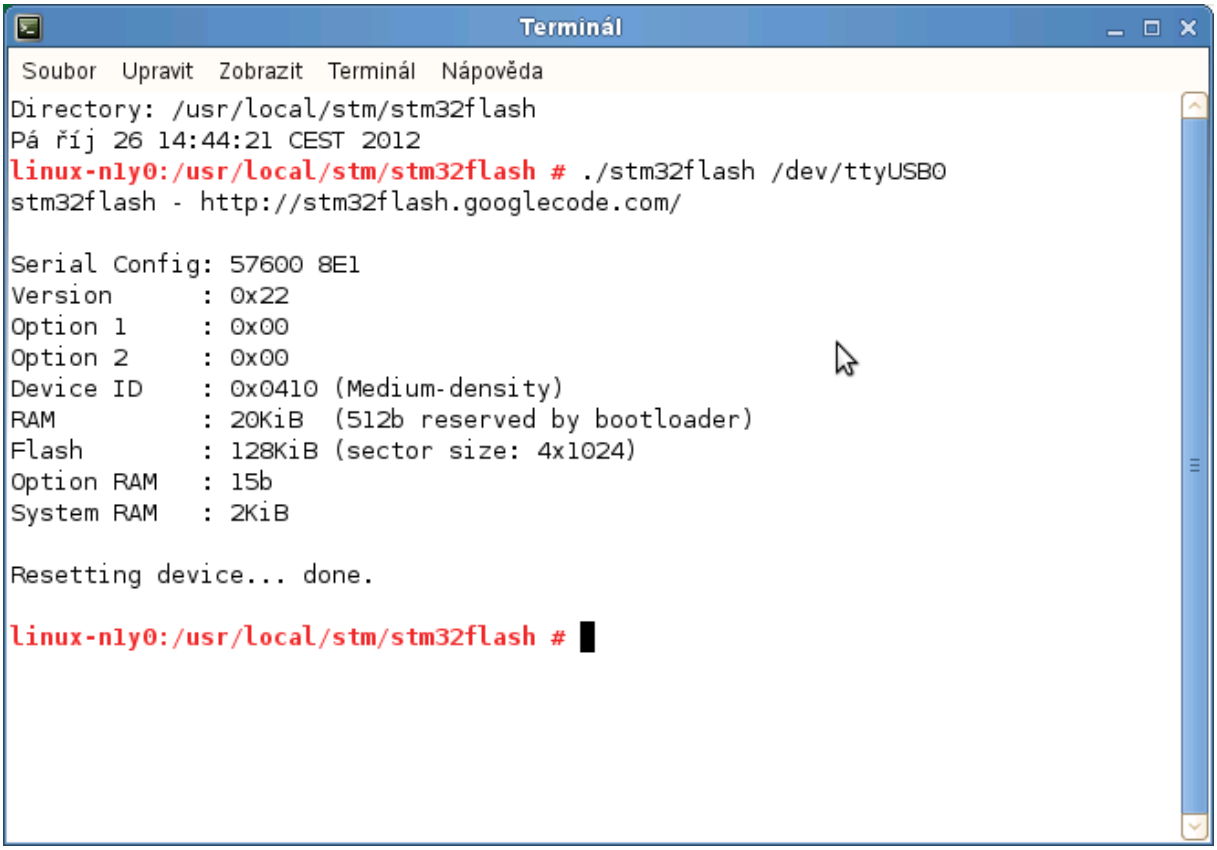
Pozn. *Device ID* 0x0420 značí *STM32F100RBT6*. Do adresáře se *stm32flash* programem umístíme soubor program01.hex který jsme získali pomocí vývojového prostředí, např. **Keil uVision4**.
V *terminálu* pak napíšeme



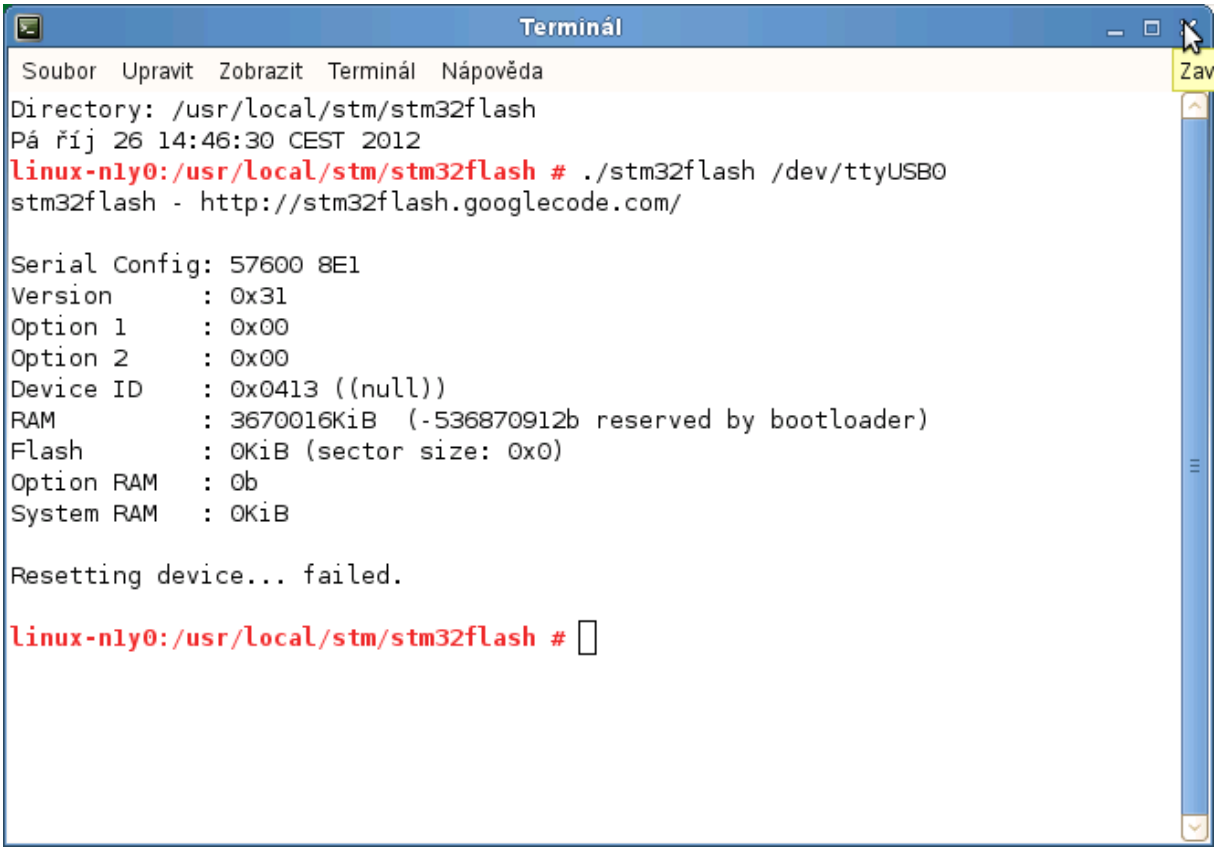
Než příkaz **./stm32flash -w program01.hex –v –g 0x0 /dev/ttzUSB0** spustíme, nezapomeneme na **boot0** dát úroveň 1 a MCU zresetovat. Po provedení tohoto příkazu máme



Obdobně bychom postupovali i u MCU *STM32F103C8T6*. Vidíme, že jeho ID je 0x0410



V případě *STM32F405RGT6* však dostaneme:

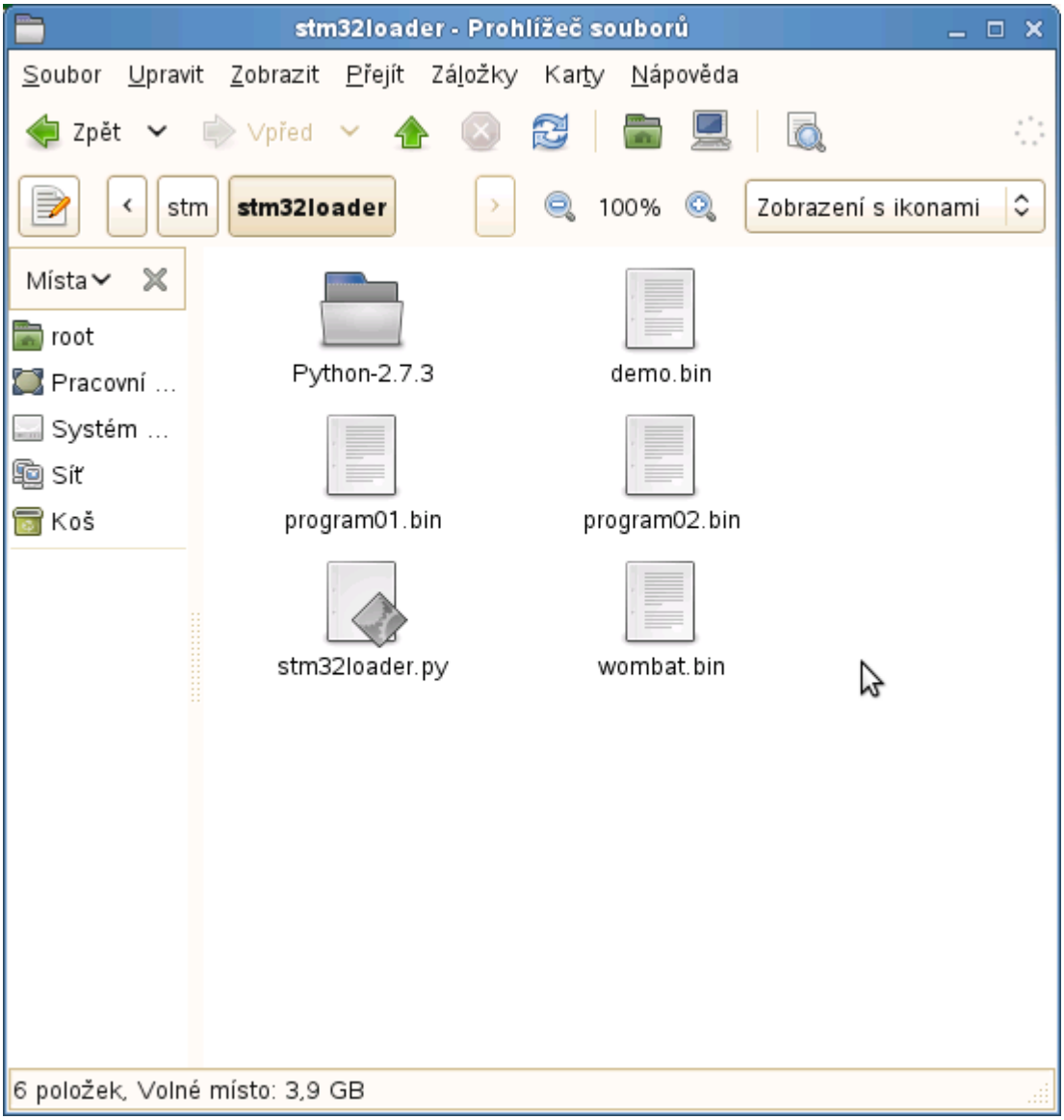


Vidíme žeID tohoto obvodu je 0x0413. Kromě toho je zřejmé, že pro *STM32F405RGT6* se *stm32flash* použít nedá. Tento obvod totiž ještě nepodporuje. Vzhledem k tomu, že k programu *stm32flash* máme zdrojové kódy, snadno najdeme v souboru **stm32.c** i tento kus kódu

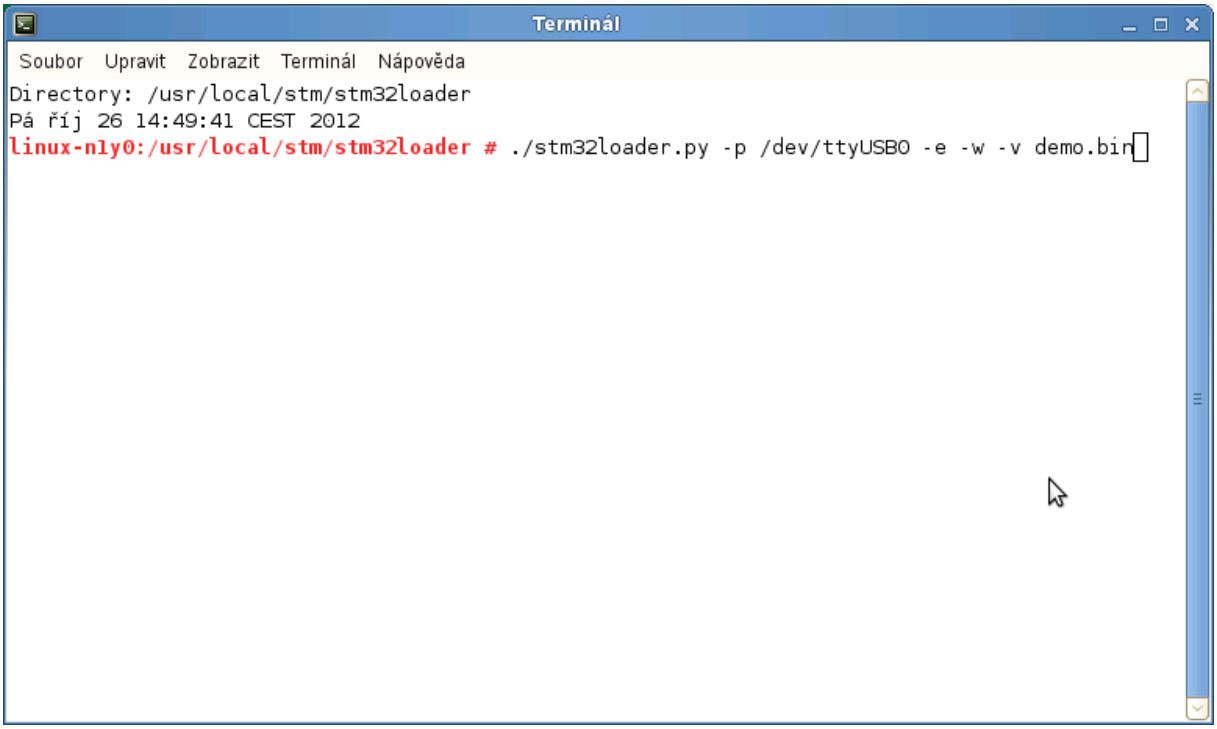
```
/* device table */
const stm32_dev_t devices[] = {
      {0x412, "Low-density"      , 0x20000200, 0x20002800, 0x08000000,
0x08008000, 4, 1024, 0x1FFFF800, 0x1FFFF80F, 0x1FFFF000, 0x1FFFF800},
      {0x410, "Medium-density"   , 0x20000200, 0x20005000, 0x08000000,
0x08020000, 4, 1024, 0x1FFFF800, 0x1FFFF80F, 0x1FFFF000, 0x1FFFF800},
      {0x414, "High-density"     , 0x20000200, 0x20010000, 0x08000000,
0x08080000, 2, 2048, 0x1FFFF800, 0x1FFFF80F, 0x1FFFF000, 0x1FFFF800},
      {0x418, "Connectivity line", 0x20001000, 0x20010000, 0x08000000,
0x08040000, 2, 2048, 0x1FFFF800, 0x1FFFF80F, 0x1FFFB000, 0x1FFFF800},
      {0x420, "Medium-density VL", 0x20000200, 0x20002000, 0x08000000,
0x08020000, 4, 1024, 0x1FFFF800, 0x1FFFF80F, 0x1FFFF000, 0x1FFFF800},
      {0x430, "XL-density"       , 0x20000800, 0x20018000, 0x08000000,
0x08100000, 2, 2048, 0x1FFFF800, 0x1FFFF80F, 0x1FFFE000, 0x1FFFF800},
      {0x0}
};
```

Vidíme, že podporuje jen MCU s ID 0x0412, 0x0410, 0x0414, 0x0418, 0x0420 a 0x0430 . Můžeme si tuto tabulku doplnit a program stm32flash znovu přeložit. Místo stm32flash můžeme použít jiný program, který i *STM32F405RGT6* podporuje. Takovým programem je např. stm32loader. Rovněž

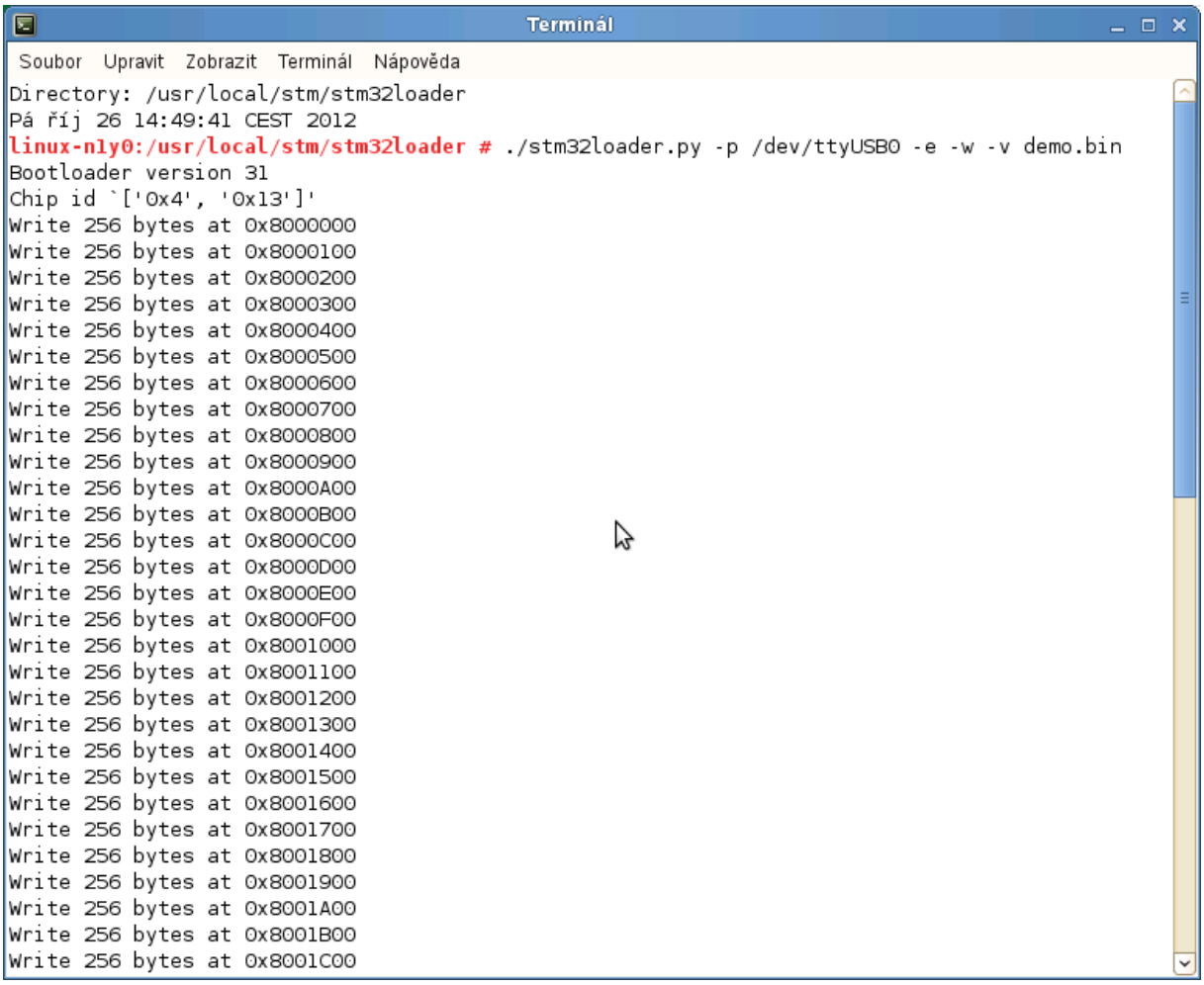
tento program je free. Protože je napsán v jazyce Python, musíme mít Python v Linuxu nainstalovaný.
Tento jazyk je často používán pro tvorbu sw pro linux pro práci s jednočipovými počítači. Proto ho mám na svém PC s Linuxem již nainstalovaný. Zbývá proto jen z https://github.com/jsnyder/stm32loader stáhnout stáhnout soubor stm32loader.py . Tento soubor umístíme do nějakého adresáře spolu s binárním(i) souborem (soubory), které chceme naprogramovat do paměti flash. (Protože **stm32flash.py** bude komunikovat přes usb, musíme mít ovšem nainstalovanou i knihovnu *PySerial.*)

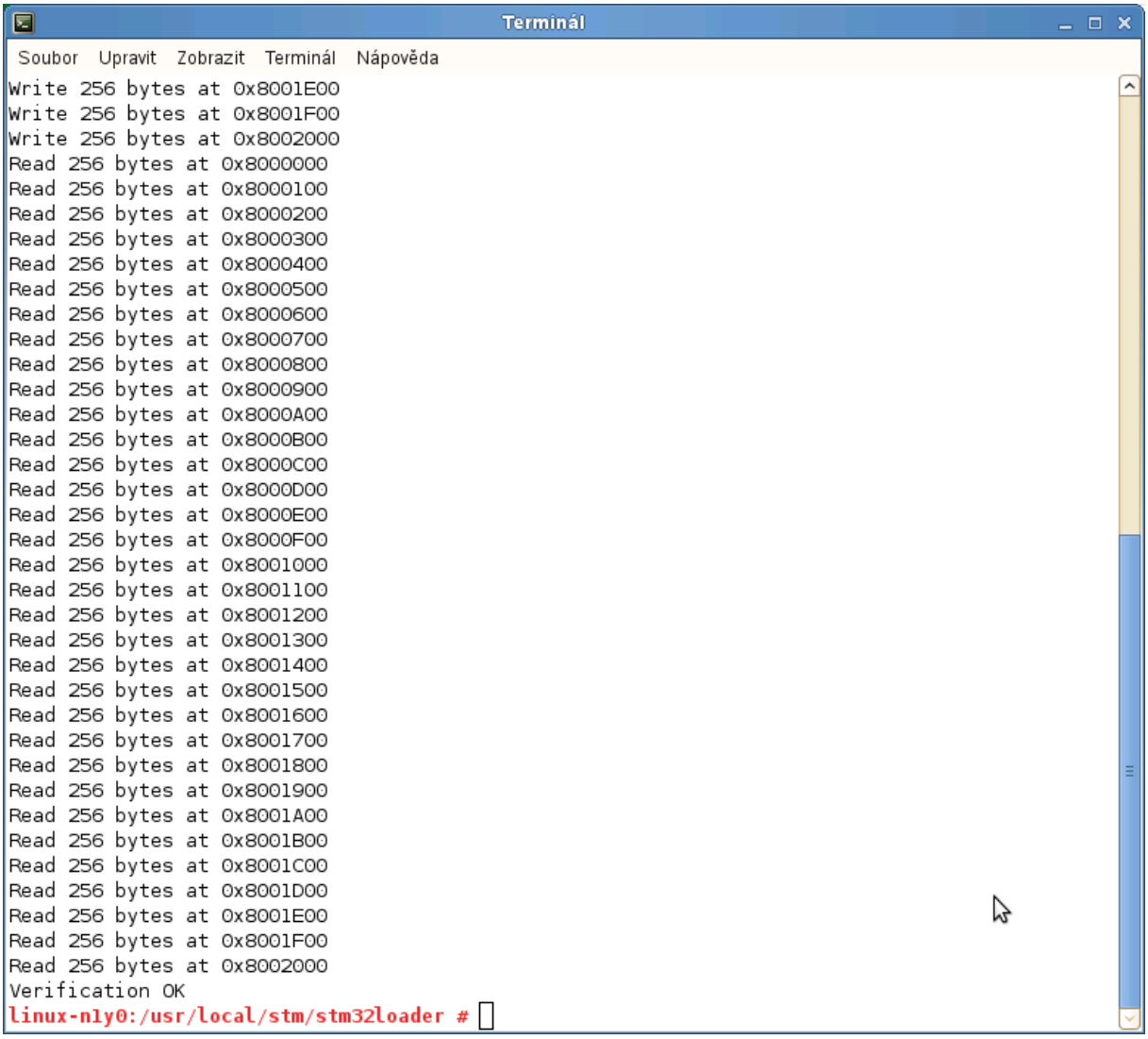


V Terminálu napíšeme



A poté, co na **boot0** dáme úroveň logické 1 a MCU zresetujeme, spustíme. Dostaneme

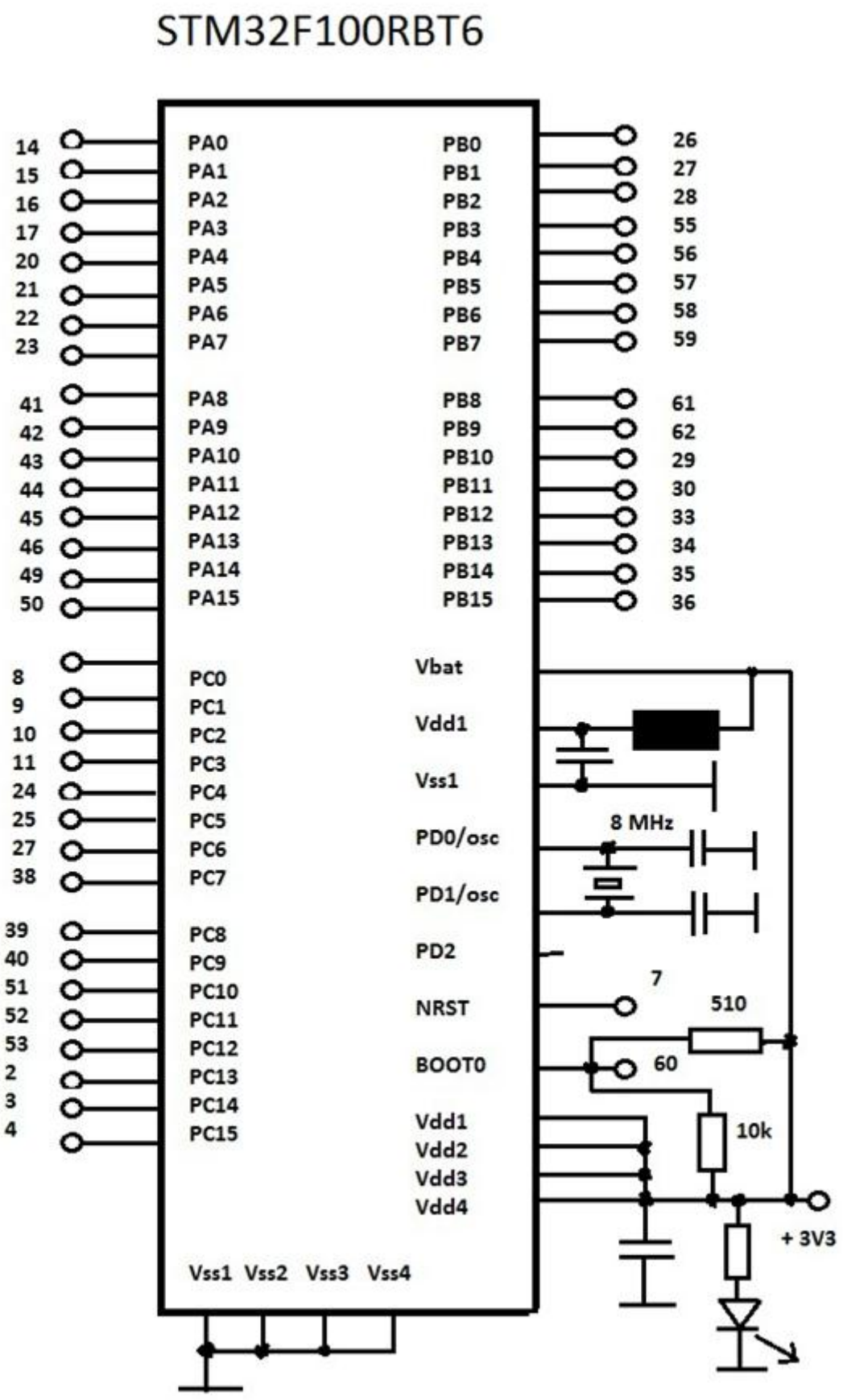




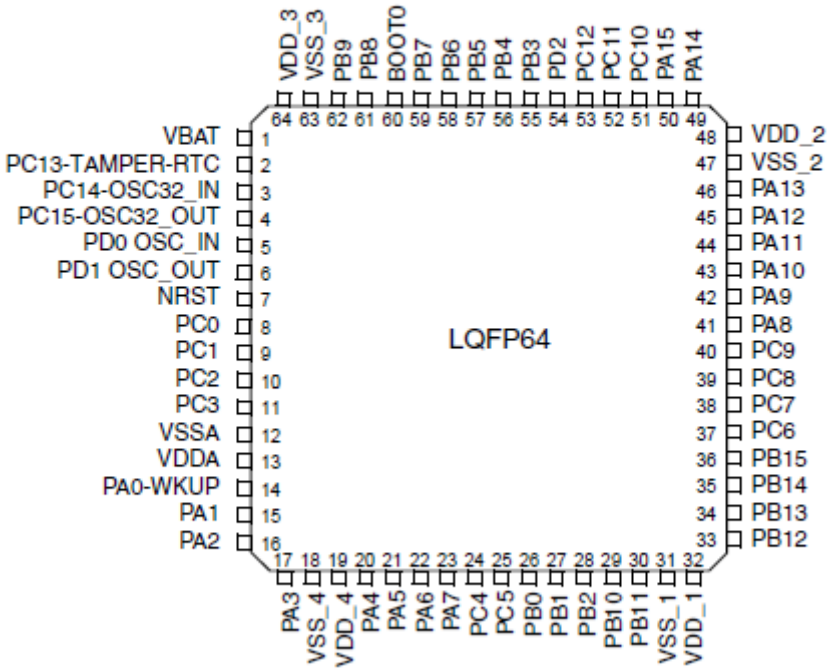
Vidíme, že naprogramování se podařilo.

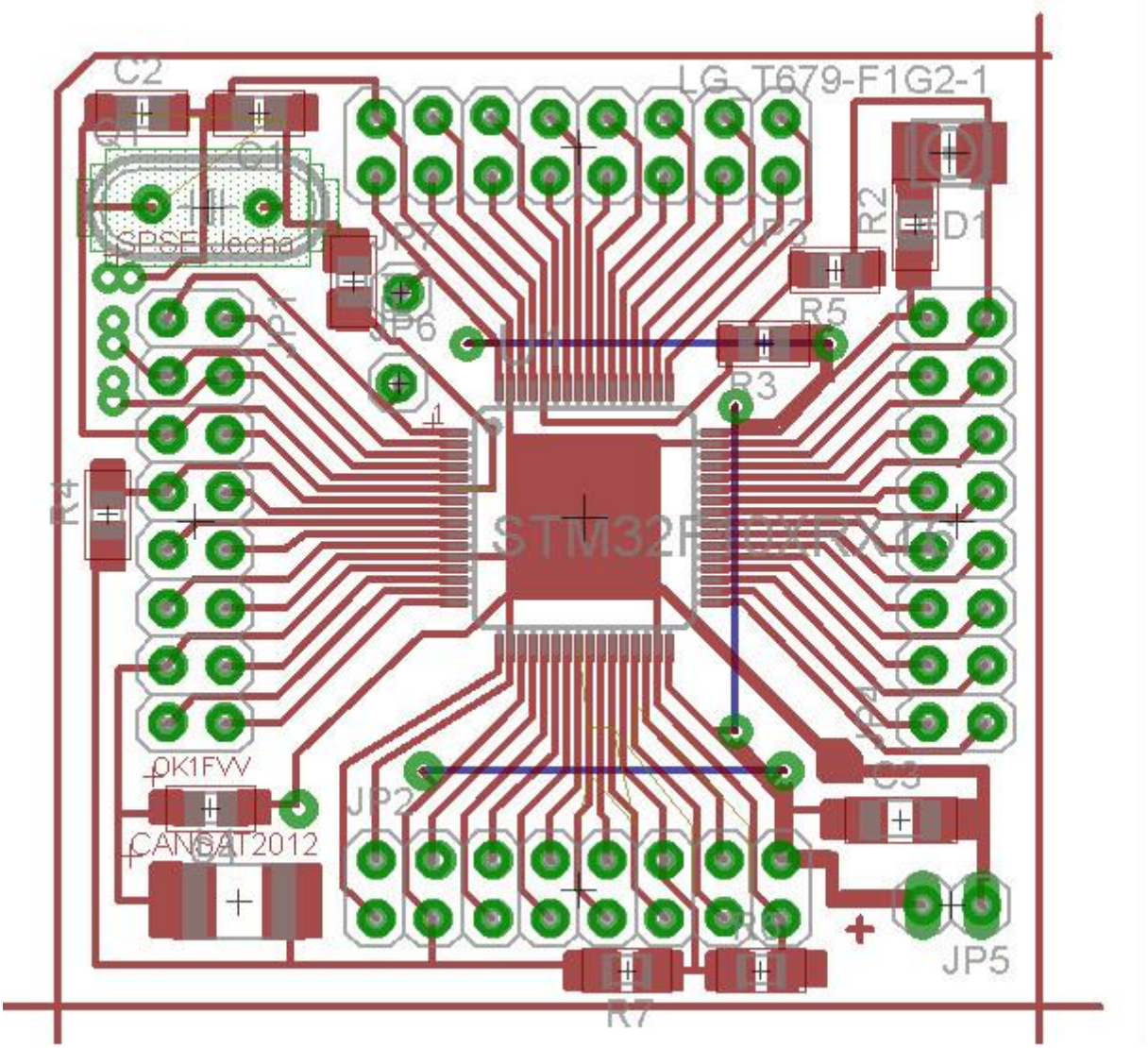
2.1.1 Konstrukční provedení onboard počítače s STM32F100RBT6

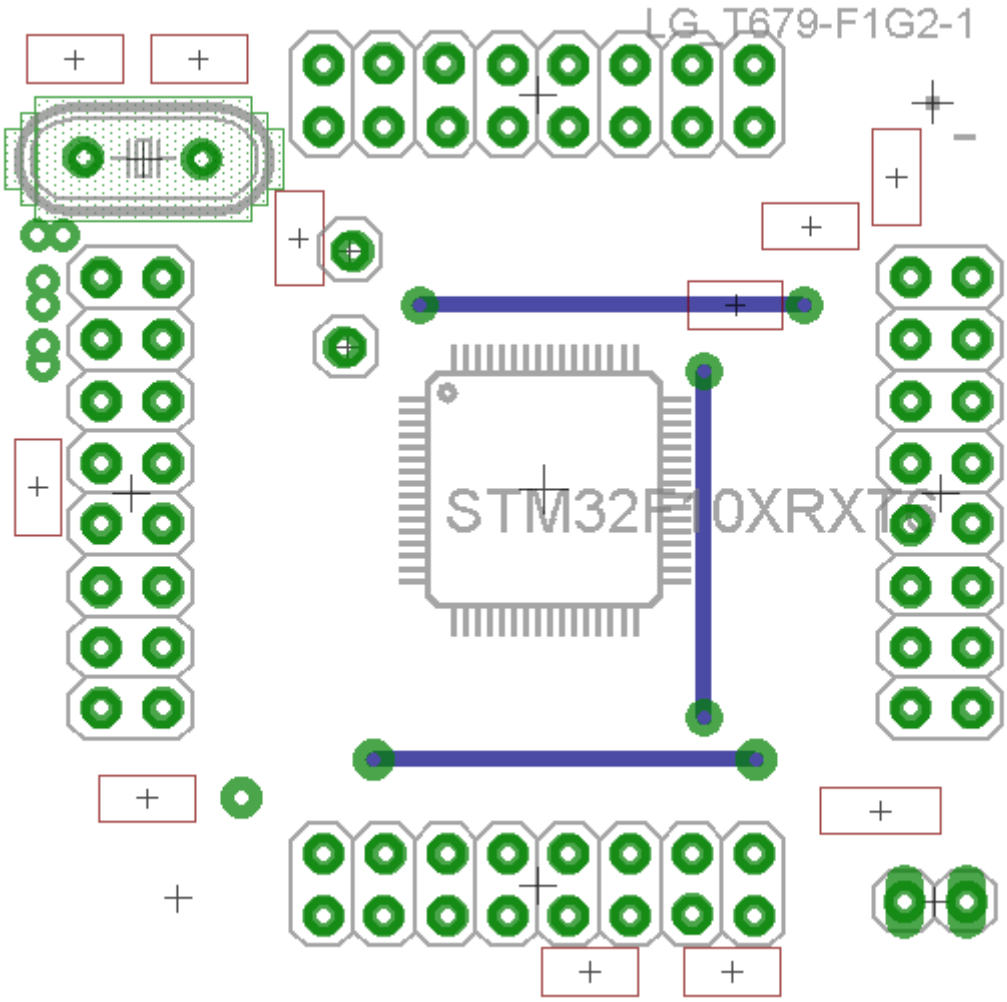
Vzhledem k jeho jednoduchosti si uvedeme jen jeho zapojení , PCB a zapojení pinů obvodu *STM32F100RBT6*:


STM32F100RBT6
14
15
16
17
20
21
22
23
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
26
27
28
55
56
57
58
59
41
42
43
44
45
46
49
50
PA8
PA9
PA10
PA11
PA12
PA13
PA14
PA15
PB8
PB9
PB10
PB11
PB12
PB13
PB14
PB15
61
62
29
30
33
34
35
36
8
9
10
11
24
25
27
38
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
Vbat
Vdd1
Vss1
PD0/osc
PD1/osc
PD2
NRST
BOOT0
Vdd1
Vdd2
Vdd3
Vdd4
8 MHz
7
510
60
10k
+ 3V3
39
40
51
52
53
2
3
4
PC8
PC9
PC10
PC11
PC12
PC13
PC14
PC15
Vss1 Vss2 Vss3 Vss4

STM32F100xx value line LQFP64 pinout

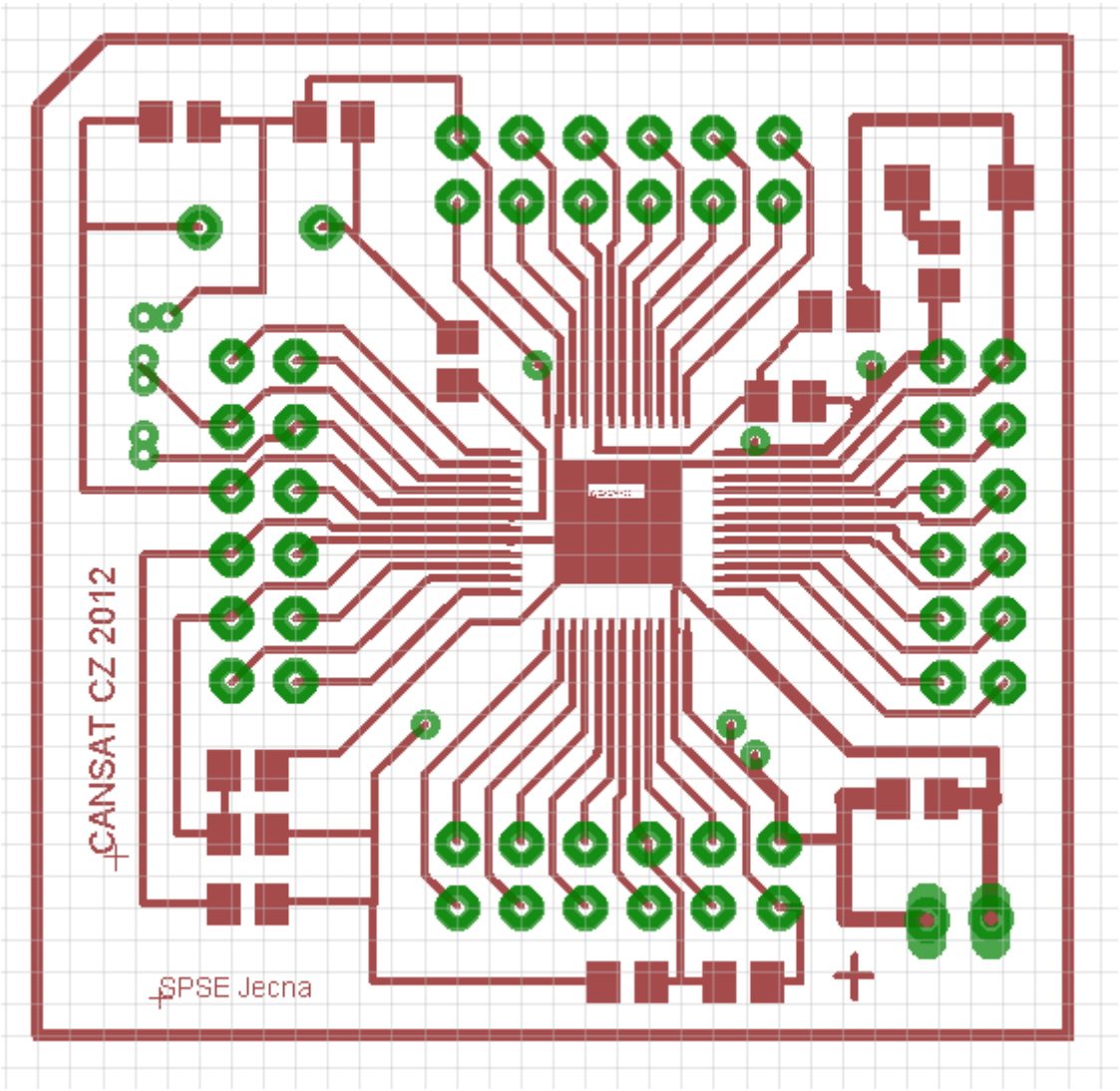


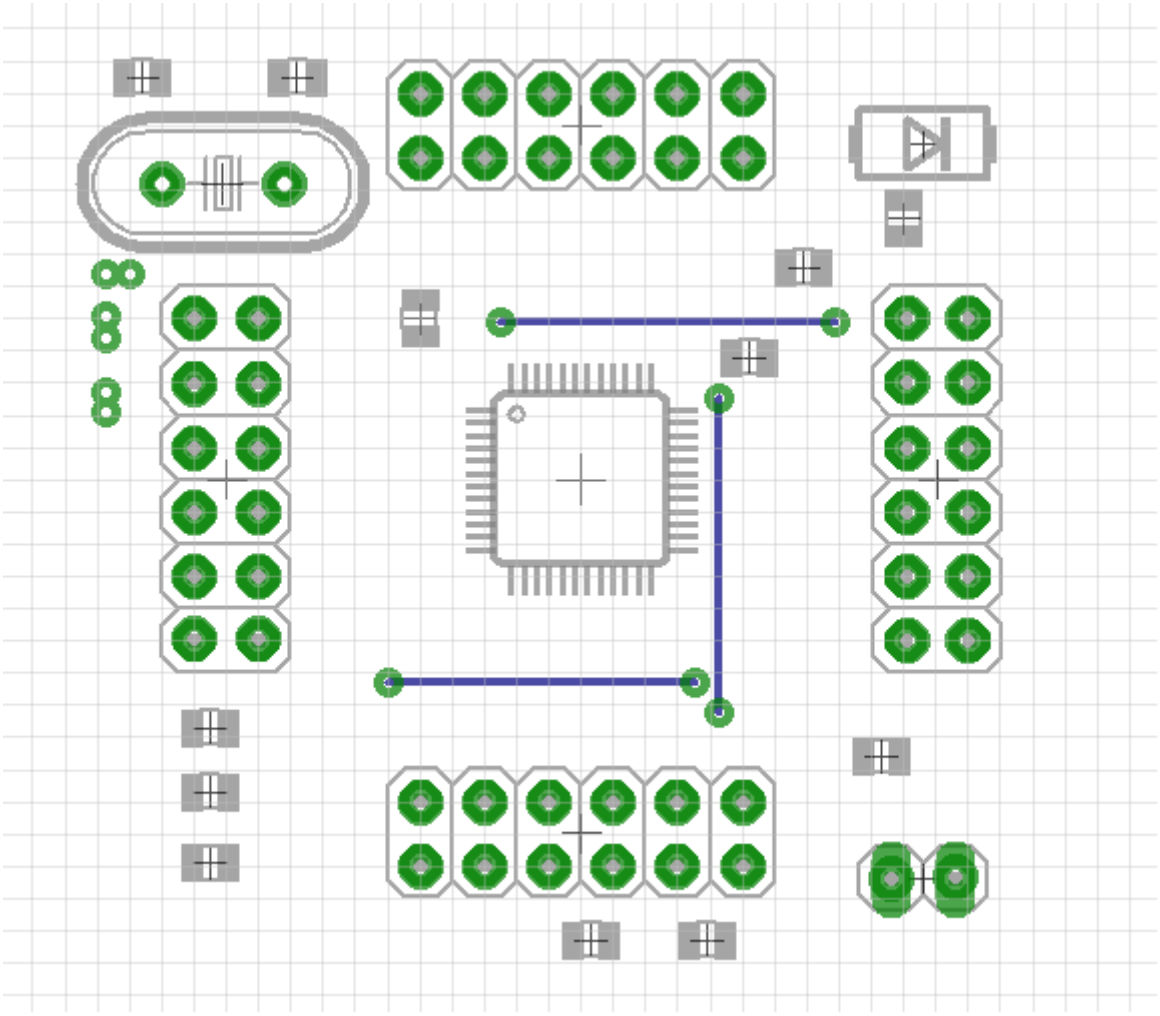

C2
LG_T679-F1G2-1
SPSE Jeena
JP7
JP6
JP3
R2
D1
R5
R3
R4
STM32F
OK1FVV
CANSAT2012
JP2
C3
JP5
R7

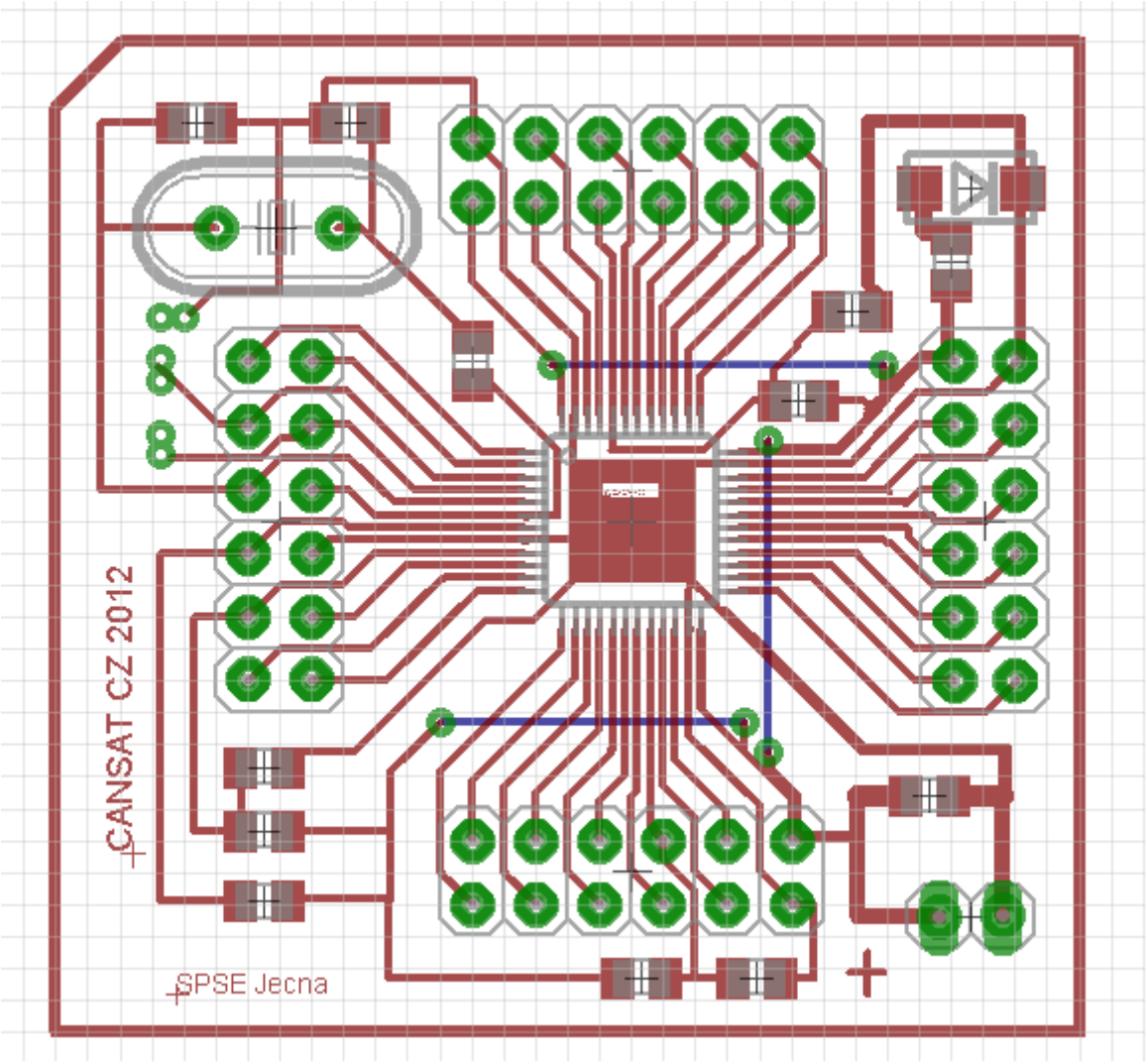


2.1.2 Konstrukční provedení onboard počítače s STM32F103C8T6

I u tohoto počítače si vzhledem k jeho jednoduchosti i uvedeme jen jeho zapojení , PCB a zapojení pinů obvodu *STM32F103C8T6* :

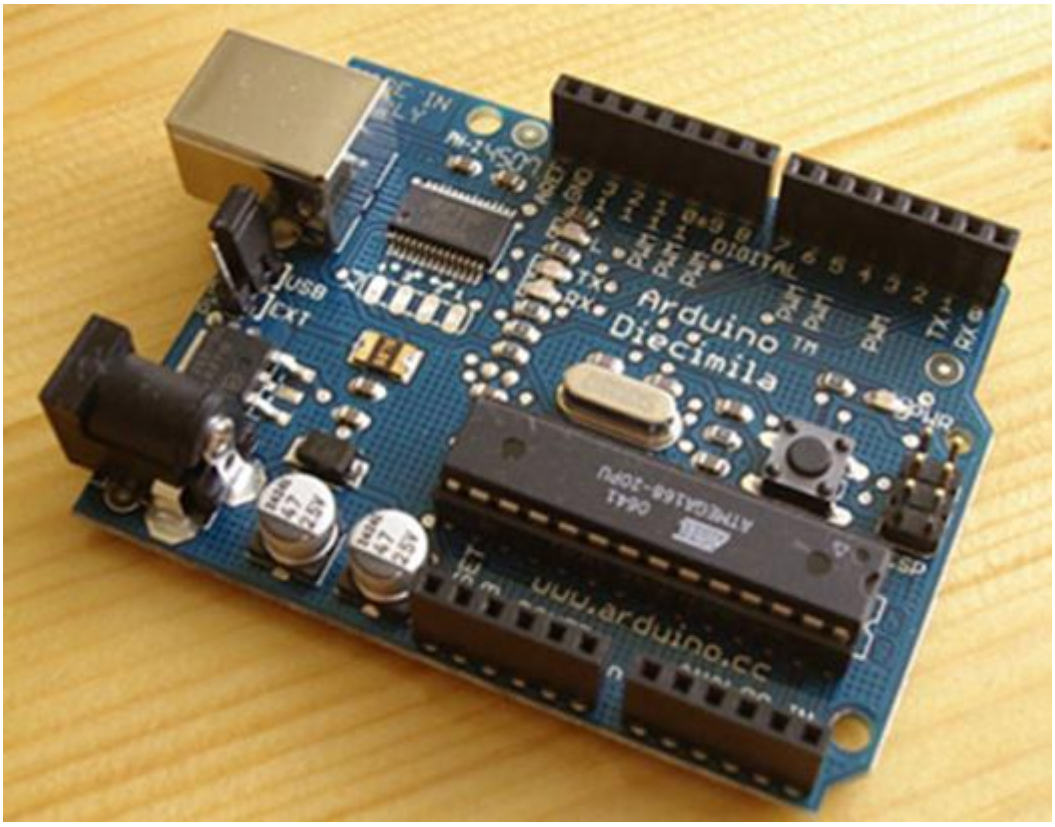

CANSAT CZ 2012
SPSE Jecna





2.1.3 Maple

Nejprve si povíme, co je *Arduino* http://www.arduino.cc/ . *Arduino* je otevřená elektronická platforma, založená na uživatelsky jednoduchém hardware a software. *Arduino* je určeno pro kutily, bastlíře, umělce, designéry - zkrátka pro každého, koho zajímá vytváření interaktivních objektů nebo prostředí.



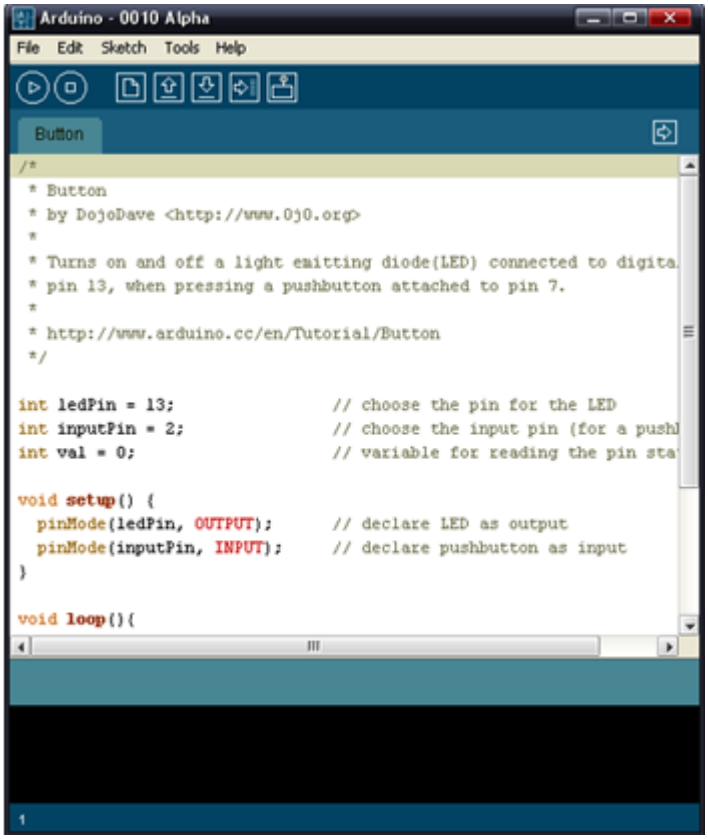
Arduino je schopné vnímat okolní prostředí pomocí vstupů z rozličných senzorů. Zároveň může ovlivňovat okolí připojenými LEDkami, motorky a dalšími výstupními periferiemi. Mikroprocesor (ATMega168 nebo ATMega328) na desce *Arduina* se programuje pomocí speciálního *Arduino* programovacího jazyku (založený na jazyku *Wiring* - podobný C) ve vlastním *Arduino* vývojovém prostředí. Projekty založené na *Arduinu* mohou jednoduše komunikovat se softwarem na stolním počítači nebo notebooku (např. Flash, Processing, MaxMSP). Desky *Arduino* je možné sestavit ručně nebo koupit již sestavené a otestované; software lze stáhnout zdarma http://www.arduino.cc/en/Main/Software . Návrhy plošného spoje je k dispozici pod otevřenou licencí, lze je tedy upravovat podle Vašich potřeb http://www.arduino.cc/en/Main/Hardware

Výhody systému Arduino jsou

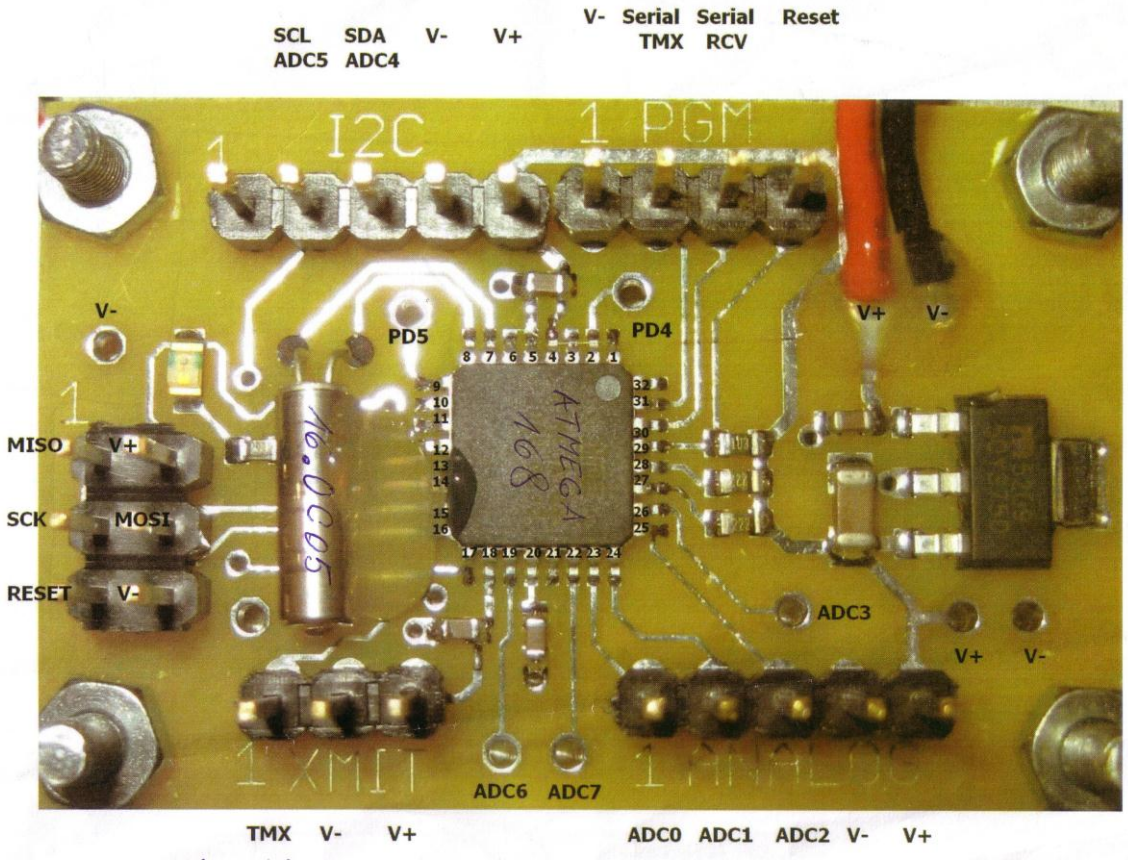
- jednoduché programování
- jednoduché zapojení
- nízká cena oproti jiným kitům
- spousta návodů
- uživatelská komunita
- platformní nezávislost (Win/Linux/MacOS/...)

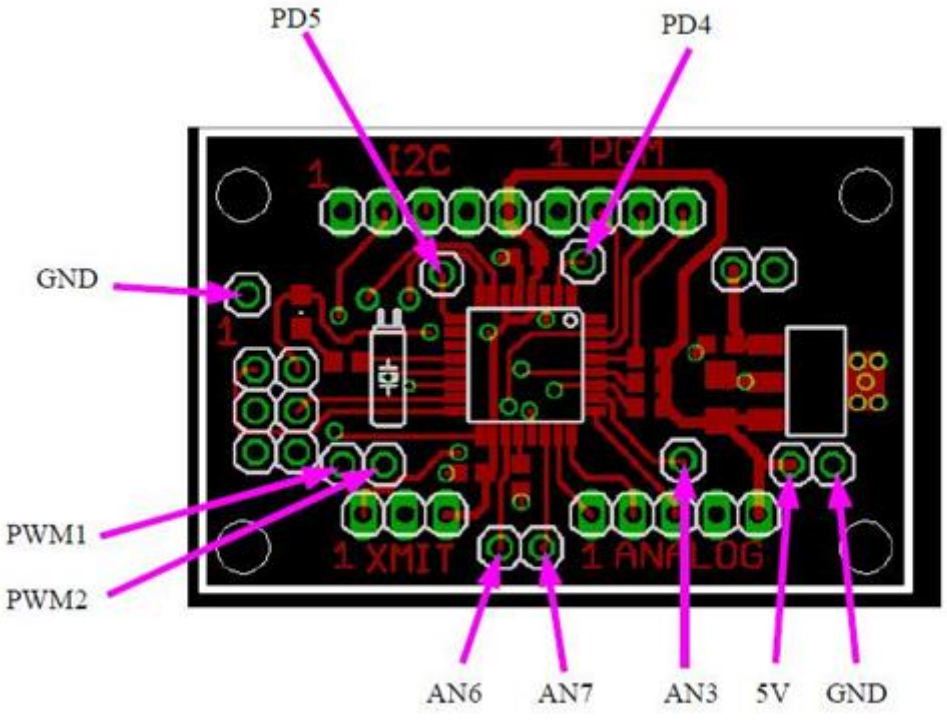
Arduino je levný, robustní vývojový kit založený na mikroprocesoru ATMega328 či ATMega168. Má 13 digitálních vstupně-výstupních pinů (z toho 6 s podporou PWM) a 6 analogových vstupů. Návrhy plošných spojů jsou k dispozici pod licencí *Attribution-ShareAlike 2.5.*

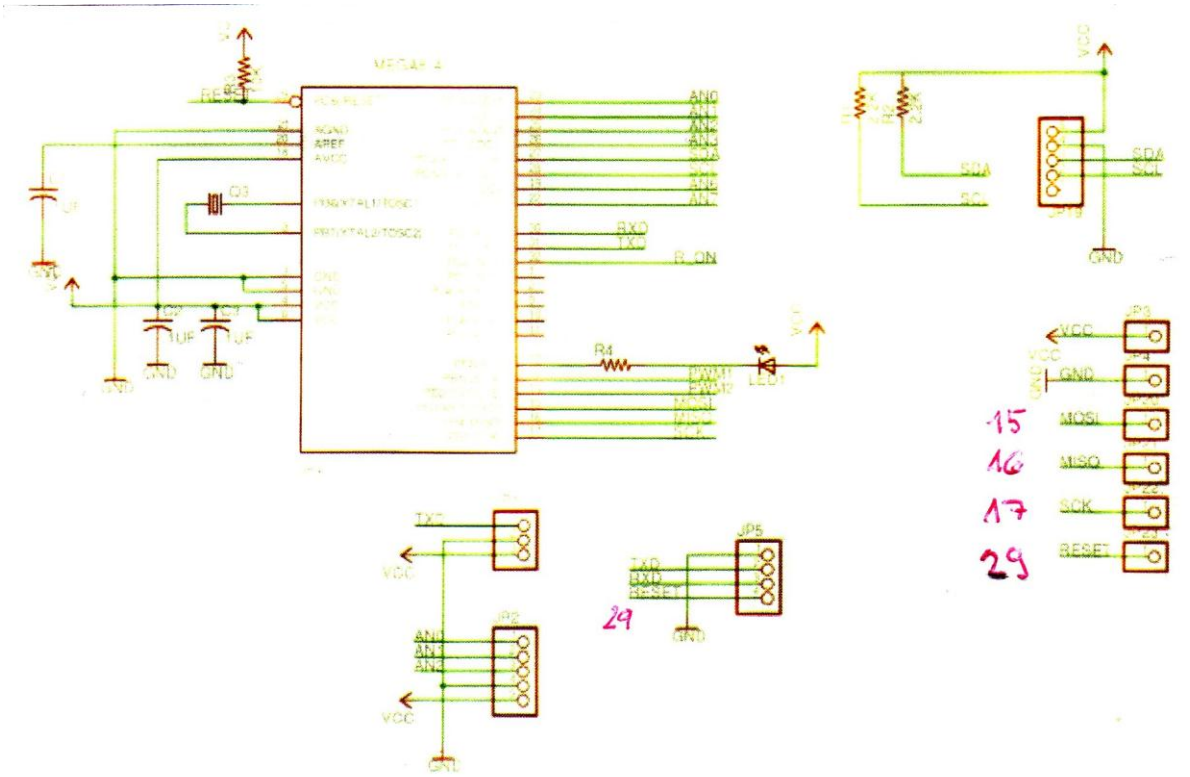
Software *Arduina* se vytváří v otevřeném vývojové prostředí , které naprosto zjednodušuje psaní kódu a nahrávání hotových programů do mikroprocesoru. Je k dispozici ve verzích pro Windows, Mac OS X a Linux. Programům pro *Arduino* se v originále říká **sketch**.

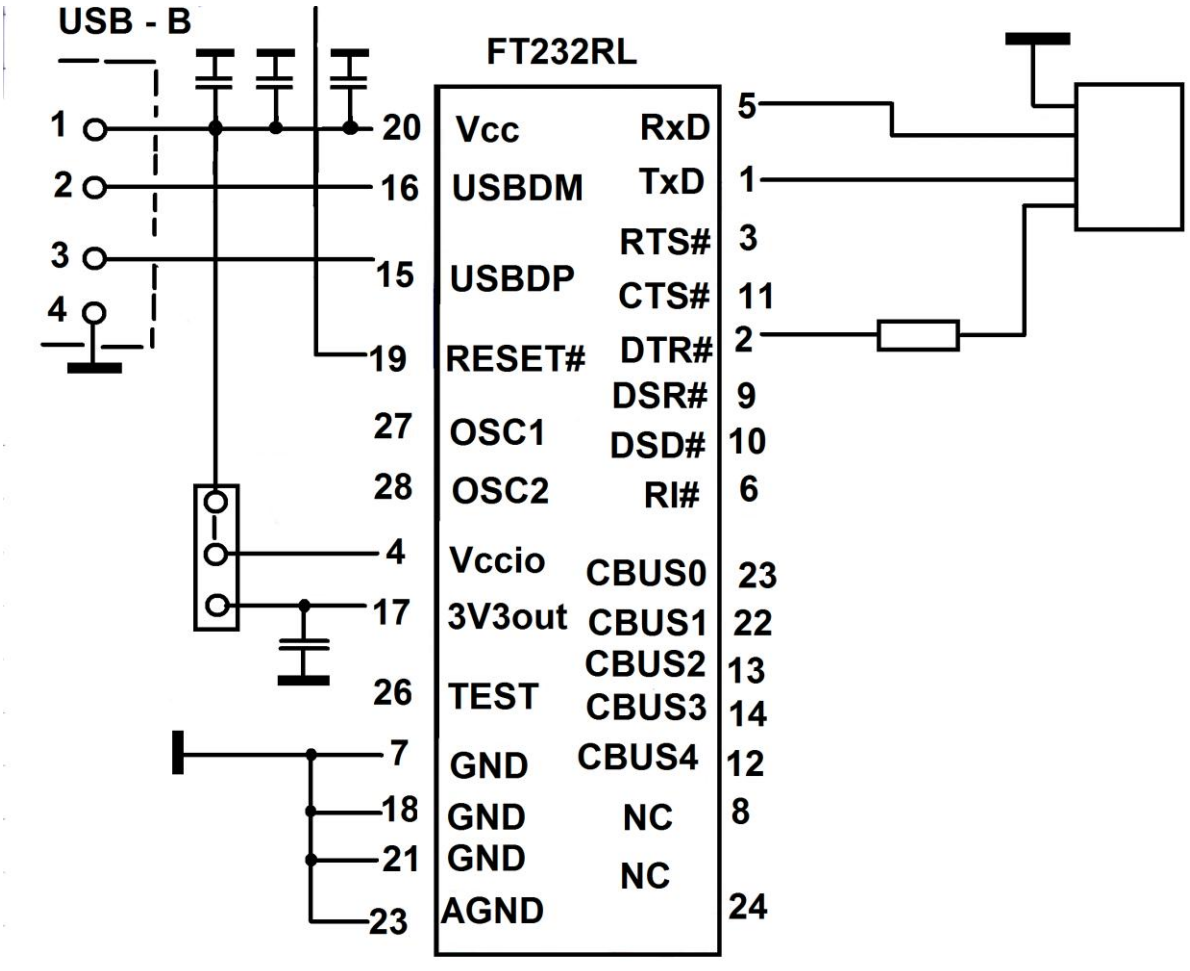


K provozování systému *Arduino* nepotřebujeme dokonce ani žádný startkit, stačí jen ATMega168 či ATMega328, do kterého nahrajeme *bootloader Arduina* (Bootloader je malý program trvale umístěný v paměti ATMega, který po restartu ATMegy kontroluje, zda přichází na vývod **RxD** z PC nějaký „sketch“ a pokud ano, tak jej nahraje do paměti a spustí. V opačném případě spustí poslední uložený „sketch“. Díky tomu lze programovat ATMegu bez speciálního programátoru. Systém *Arduino* obvykle používá propojení **RxD** ATmegy k USB konektoru PC prostřednictvím USB převodníku s FTDI obvodu FT232R). Na tom byl založen i onboard počítač stavebnic Cansatu při 1. a 2. Ročníku evropské soutěže středoškoláků CANSAT.


SCL
ADC5
SDA
ADC4
V-
V+
V-
Serial
TMX
Serial
RCV
Reset
I2C
1 PGM
V-
PD5
PD4
V+
V-
MISO
V+
SCK
MOSI
RESET
V-
ATMEGA
168
ADC3
V+
V-
1 XMIT
ADC6
ADC7
1 ANALOG
TMX
V-
V+
ADC0
ADC1
ADC2
V-
V+


PD5
PD4
I2C
1 PGM
GND
PWM1
PWM2
1 XMIT
1 ANALOG
AN6
AN7
AN3
5V
GND




USB - B
FT232RL
1
2
3
4
20
16
15
19
27
28
4
17
26
7
18
21
23
Vcc
USBDM
USBDP
RESET#
OSC1
OSC2
Vccio
3V3out
TEST
GND
GND
GND
AGND
RxD
TxD
RTS#
CTS#
DTR#
DSR#
DSD#
RI#
CBUS0
CBUS1
CBUS2
CBUS3
CBUS4
NC
NC
5
1
3
11
2
9
10
6
23
22
13
14
12
8
24

Ukázkový program v jazyce *Arduina* ze semináře **ESA Cansat** v prosinci 2011

```
void setup(){                                    //ATMega 168
  Serial.begin(38400);                           // configure the serial
port to 38400 baud
  pinMode(8,OUTPUT);                             // configure the port with
the LED
  delay(200);
  Serial.println("F8D07D");                      // Set the radio frequency
433,25 MHz  ... CZ
  delay(500);
  Serial.println("CCANSAT");                     // Set the radio ID
  delay(500);
}

void loop(){                                     // This is the main
program

//Sending seriall data
  Serial.print("S");                             //"S" is the command for
the radio to start building a package.
  Serial.print("Hello World!SPSE Jecna ");
  Serial.println();                              //Sending the package

//LED blinking
  digitalWrite(8, LOW);                          // LED ON
  delay(250);
  digitalWrite(8, HIGH);                         // LED OFF
  delay(250);
}
```

Obdobné programy v jazyce *Arduina* budeme moci vytvářet v případě našeho palubního počítače s *STM32F103RBT6* s implementací *Maple*. V této kapitole se již bude věnovat tomuto systému.
Maple může být programován v jazyce *Wiring* , stejném jako používá *Arduino*. Rovněž *IDE Maple* je z hlediska uživatele stejné jako *u Arduina*. Projektu se v těchto IDE říká *sketch* . Sestává z jednoho či několika souborů s kódem napsaným v jazyce *Wiring*, který je velice podobný jazyku C++. Hlavní rozdíl mezi jazykem *Wiring* a C++ je v tom, že v jazyce *Wiring* není nutné deklarovat globální proměnné před jejich použitím. Tudíž např. následující kód v jazyce **Wiring** je v pořádku a funkce f() vrací 5.

```
int f() {
  return g();
}

int g() {
  return 5;
}
```

Dalším rozdílem mezi jazykem **Wiring** a **C++** je to, že **Wiring** nepodporuje dynamickou alokaci paměti (**new** a **delete**).

Nejdůležitější konstrukty jazyka Wiring:

Struktury
setup()
loop()

Řídící struktury
if/else
for
switch/case
while
do...while
break
continue
return
goto

Další syntaxe
; (středník)
{} (složené závorky)
// (jednořádkový komentář)
/* */ (víceřádkový komentář)
#define
#include

Arithmetic Operators
= (přiřazení)
+ (sčítání)
- (odčítání)
* (násobení)
/ (dělení)
% (modulo)

Comparison Operators
== (rovná se)
!= (nerovná se)
< (menší než)
> (větší než)
<= (menší než nebo rovný)
>= (větší než nebo rovný)

Boolean Operators
&& (and)
|| (or)
! (not)

Pointer Operators

* dereference operator
& reference operator

Bitwise Operators
& (bitové and)
| (bitové or)
^ (bitové xor)
~ (bitové not)
<< (posun doleva)
>> (posun doprava)

Compound Operators
++ (increment)
- - (decrement)
+= (compound add)
-= (compound subtract)
*= (compound multiply)
/= (compound divide)
&= (compound bitwise and)
|= (compound bitwise or)

Keywords
stejná jako v C++

Proměnné
Constants
HIGH | LOW
INPUT | OUTPUT
true | false
Constants (integers, floating point)
Board-specific values

Data Types
void
boolean (1 byte)
char (1 byte)
unsigned char (1 byte)
byte (1 byte)
int (4 bytes)
unsigned int (4 bytes)
long (4 bytes), synonym for int
unsigned long (4 bytes), synonym for unsigned int
long long (8 bytes)
unsigned long long (8 bytes)
float (4 bytes)
double (8 bytes)
strings
arrays
enum
numeric types

Conversion
char()
byte()
int()
long()
float()
double()

Variable Scope & Qualifiers
variables, scope
static
volatile
const

Utilities
sizeof()
ASSERT()

Funkce

Digital I/O
pinMode()
digitalWrite()
digitalRead()
togglePin()
toggleLED()
isButtonPressed()
waitForButtonPress()

Analog I/O
analogRead()
pwmWrite() (analogWrite() is also available, though its use is discouraged)

Advanced I/O
tone(): TODO
noTone(): TODO
shiftOut()
pulseIn(): TODO

Time
millis()
micros()
delay()
delayMicroseconds()

Math
min()
max()
abs()

constrain()
map()
pow()
sqrt()

Trigonometry
sin()
cos()
tan()

Random Numbers
randomSeed()
random()

Bits and Bytes
lowByte()
highByte() is provided, though its use is discouraged.
bitRead()
bitWrite()
bitSet()
bitClear()
bit()

External Interrupts
Reference Page
attachInterrupt()
detachInterrupt()

Interrupts
interrupts()
noInterrupts()

Communication
SerialUSB
Serial

Nakonec si uvedeme, jak provést implementaci *Maple* na náš *STM32F103RBT6*. Z https://github.com/leaflabs/maple-bootloader stáhneme *bootloader* (přesněji řečeno jeho zdrojové kódy). Poté z nich následujícím postupem získáme bin kód, který již nahrajeme do programové flash *STM32F103RBT6*. Uvedený postup provedeme v OS Linux. Ostatně ten budeme používat i v následující kapitole 2.1.4

```
$ git clone git://github.com/leaflabs/maple-bootloader.git
$ cd maple-bootloader
$ make
$ ls -lh build/maple_boot.bin # toto je zcompilovaný bootloader binary
```

Výsledný *maple_boot.bin* nahrajeme do paměti flash postupem dle 2.1.4.2

2.1.4 Programování paměti flash STM32F1xx

2.1.4.1 Programování paměti flash

Stáhneme https://raw.github.com/leaflabs/libmaple/master/support/stm32loader.py

2.1.4.2 Programování bootloaderu Maple

Postupujeme podle http://leaflabs.com/docs/bootloader.html#flashing-a-custom-bootloader

Pod Linuxem

```
python stm32loader.py -p ser-port -evw maple_boot.bin
```

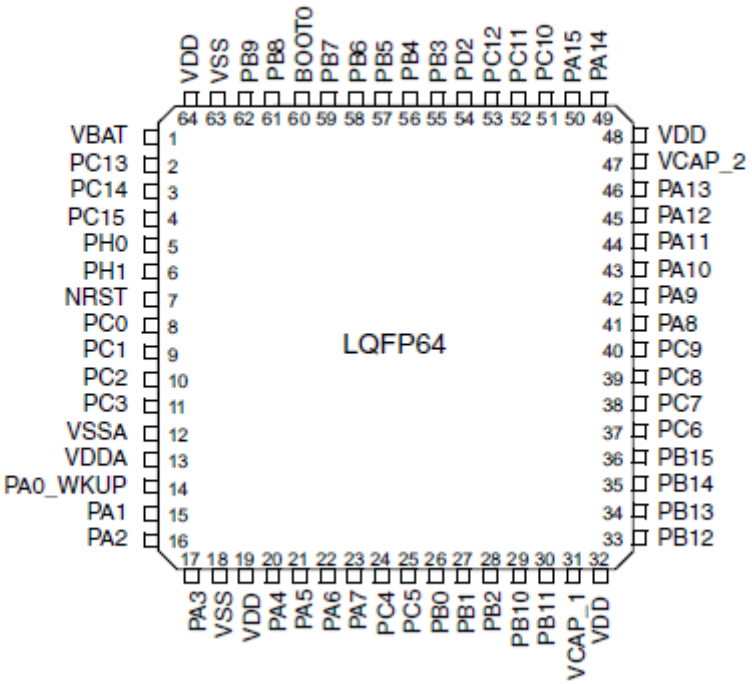
Ve windows

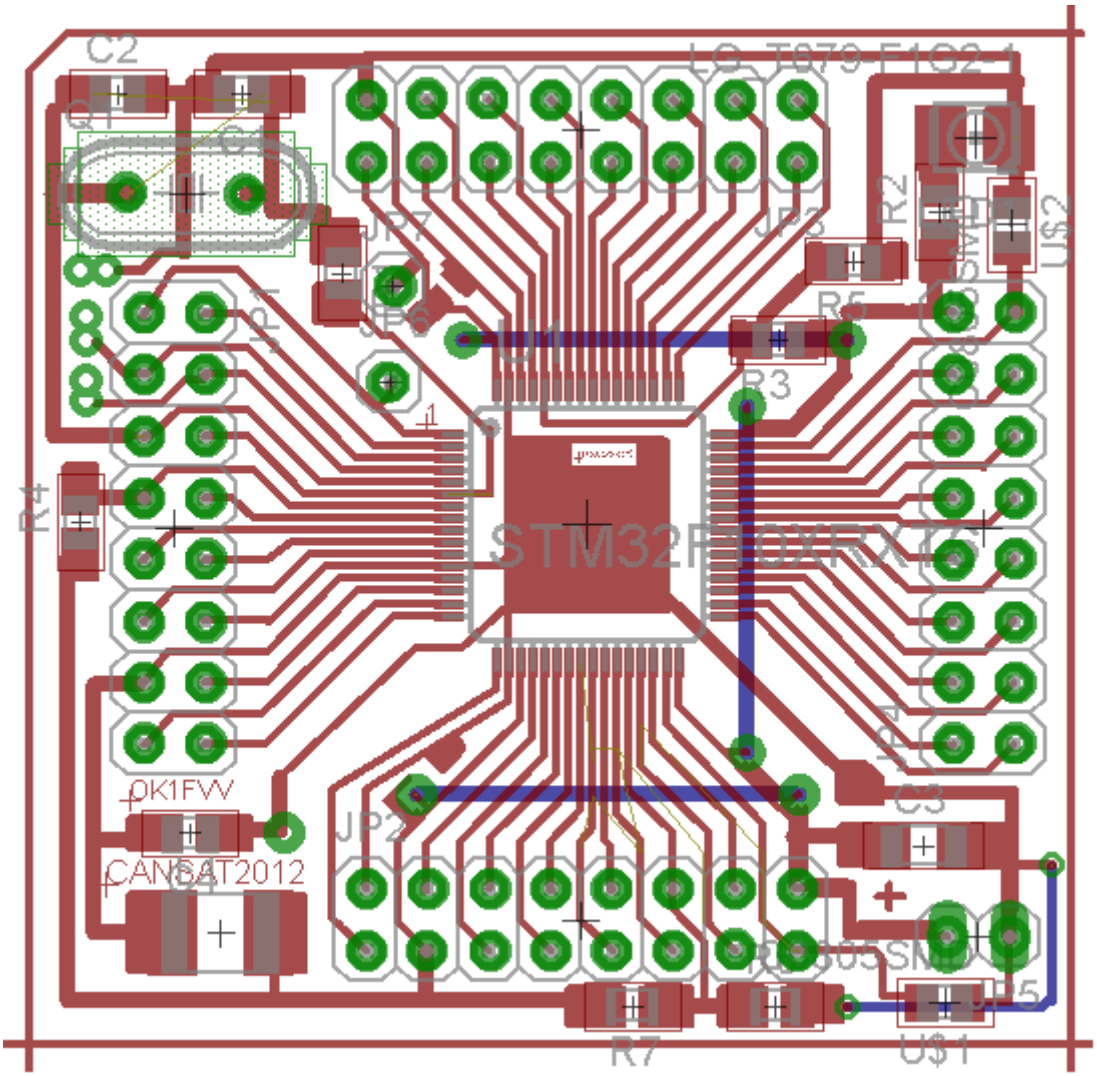
```
python.exe stm32loader.py -p ser-port -evw maple_boot.bin
```

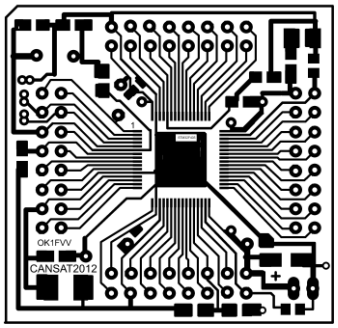
2.1.5 Onboard počítač s STM32F405RGT6,

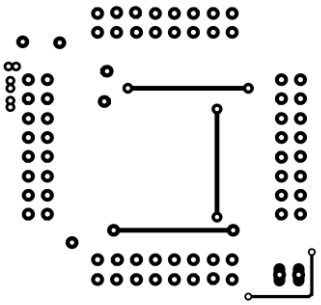
1 MB Flash 192 kB RAM 168 MHz Vvýhodou tohoto onboard počítače je, že můžeme beze zbytku použít sw startkitu *FEZ Cerberus* firmy GHI. FEZ Cerberos je založen právě na *STM32F405* http://www.ghielectronics.com/catalog/product/349 či http://cz.mouser.com/new/ghielectronics/ghielectronics-cerbuino/ a je na něm provozován micro .NET spolu s knihovnami micro .NET i dalšími od GHI. Stejný MCU je dále používán v startkitech *netduino go*, který je rovněž založen na *micro .NETu*

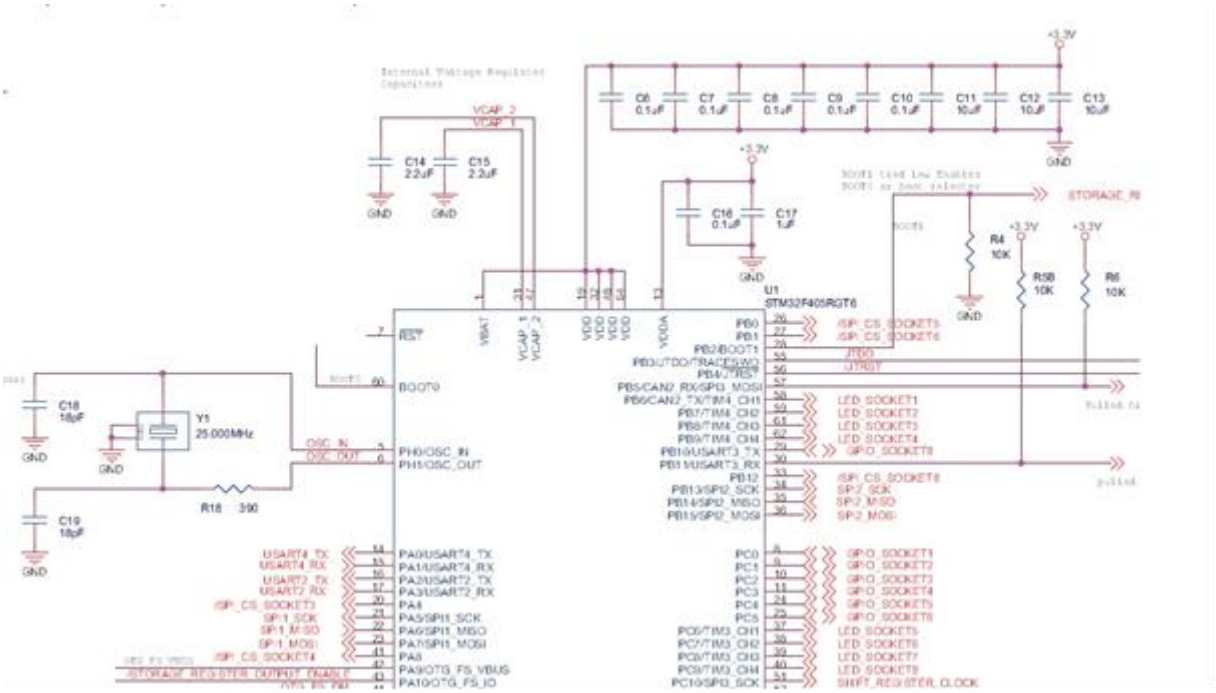
STM32F40x LQFP64 pinout




C2
C1
C4
JP7
JP6
JP3
R2
R6
R3
R4
JP1
JP2
C3
R7
U$1
U$2
OK1FVV
CANSAT2012


OK1FVV
CANSAT2012





2.1.5.1 Netduino GO a FEZ Cerberos

2.1.5.1.1 micro . NET

.NET Micro Framework je malé běhové prostředí z rodiny .NET, určené hlavně do průmyslu. S jeho pomocí se dají jednoduše vyvíjet "embedded" zařízení přímo v jazyce C#. To znamená, že programátoři, kteří mají zkušenostmi s jazykem C# a vývojovým prostředím *Microsoft Visual Studio*, mohou rychle a pohodlně tvořit programy pro malá samostatná zařízení – například funkčního robota, jednoduchý webový server... *Micro Framework* běží i na velice malém a nevýkonném hardwaru, což je jeho největší výhoda oproti Windows CE a .NET Compact Frameworku. C# jako vysoký programovací jazyk na rozdíl od assembleru masivně zvyšuje produktivitu vývojáře a usnadňuje mu zbytečnou práci.

Od verze 4.0 Microsoft uvolnil *Micro Framework* jako open-source, čímž ho učinil ještě zajímavějším. Typické zařízení s *Micro Frameworkem* má 32bitový ARM procesor. Dá se libovolně rozšířit o další periferie, jako je například dotykový barevný displej, sítová karta, slot na paměťové karty SD a MMC a mnoho dalších. Samozřejmostí je podpora USB, I2C, sběrnice 1-Wire a dalších běžných funkcí. Micro Framework ke svému běhu nepotřebuje žádný operacní systém - je posazen přímo nad hardware. Potřebuje pouze několik stovek kB RAM a nevyžaduje procesor s MMU (memory management unit). Proto může být nasazen na levném a úsporném hardwaru.

Většina firem, které vyrábějí procesory s *Micro frameworkem* nabízí také vývojové desky. Například *netduino* nebo firma GHI, která vyrábí vývojové desky FEZ. Některé vývojové desky (netduino) jsou kompatibilní s deskou *Arduino* a tím pádem nám umožňují připojit spoustu dalších tzv. „shieldů".

Co budeme potřebovat pro vývoj FEZ Cerduino?

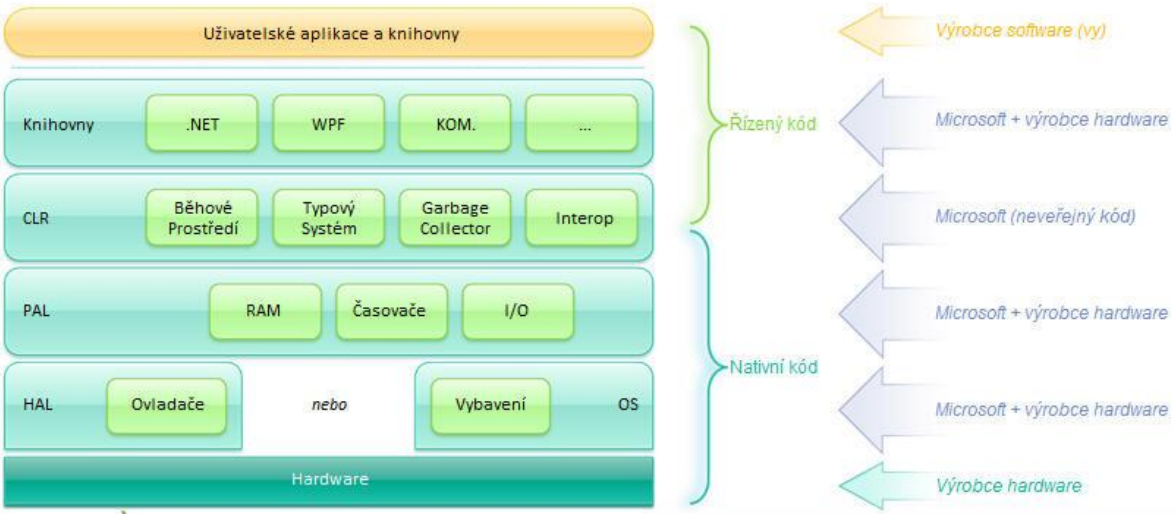
Pokud se rozhodnete vyzkoušet si vývoj Microframeworku budete potřebovat toto:

1. Microsoft Visual Studio – stačí verze Express, která je naprosto **zdarma!**
2. Microsoft .NET Micro Framework SDK
3. GHI Electronics NETMF SDK – musí být nainstalováno až po základním SDK od Microsoftu
4. Vývojovou desku FEZ Cerduino či kompatibilní zařízení.
5. Pokud budete updatovat firmware vaší desky od GHI, můžete použít doporučený TeraTerm

Co budeme potřebovat pro vývoj s Netduino GO

1. Microsoft Visual Studio – stačí verze Express, která je naprosto zdarma!
2. Microsoft .NET Micro Framework SDK
3. Netduino SDK – musí být nainstalováno až po základním SDK od Microsoftu
4. Vývojovou desku Netduino GO.

.NET Micro Framework na rozdíl od normálního .NET Frameworku nevyžaduje nosný operační systém. Zúžená verze *Common Languege Runtime* (TinyCLR) sedí přímo na hardware, takže framework je často nazýván *bootable runtime*. Runtime je nenáročný na prostředky. Stačí mu několik set kB RAM a nepotřebuje procesor s memory management unit (MMU). Tudíž .NET Micro Frameworkem může pracovat na malých a laciných 32-bitových procesorech. Architekturu .NET *Micro Frameworku* ukazuje následující obrázek:



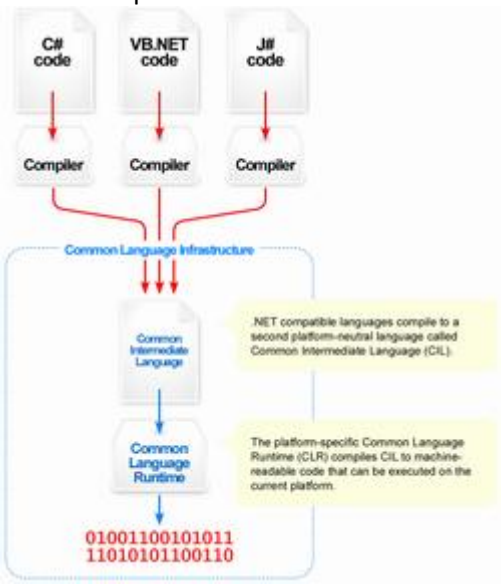
NET Micro Framework je produktově a hardwarově nezávislý systém pro běh a řízení programů určené pro Embedded aplikace napsané v řízeném kódu (managed code / managed code runtime). Nejde však o tradiční operační systém (OS) jak ho běžně známe prostřednictvím Windows CE, XP nebo Vista, který běží na vrcholu nad všemi ostatními aplikacemi. *Micro Framework* v sobě obsahuje pro embedded aplikace jen potřebnou podmnožinu funkcí plného OS, jako správu zdrojů a prostředků (resource managment), vstupů a výstupů (I/O), řízení běhu a vykonávání kódu (execution control) apod., které následně začleňuje již přímo do procesorem vykonávaného kódu (Native code). Proto programy v tomto systému již nepotřebují OS a označující se jako *bootable runtime*, tedy samobootovací. Díky tomu může být provozován i na malých zařízeních a omezenými hardwarovými

prostředky (malými RAM a Flash paměťmi, malými a levnými 32bit. CPU bez MMU), které neumožňují použití některého z OS (např. Windows XPe nebo Windows CE) či jiné softwarové podpory.

Jádrem platformy je *.NET Micro Framework Common Language Runtime* (CLR), malé optimalizované prostředí pro běh programu řízeného kódu (managed code runtime), který natahuje, spouští a řídí běh uživatelem napsaného programu. Narozdíl od běžného přímého kódu, běží řízený kód v kontextu s CLR, který mu poskytuje zaváděcí program (bootstrap), ochranu jeho běhu, záznamu a čtení dat, obsluhu přerušení (interrupt handling), zprávu paměti, procesů, úloh a zásobníku (memory, thread, process, heap managment, garbage collection) apod. Pro tento účel obsahuje řízený kód pro CLR informace o struktuře programu.

.NET Micro Frameworku CLR tak prakticky zajišťuje následující úkony:

- Minimalizaci velikosti programu => menší vytížení CPU a menší spotřebu energie
- Běh bez tradičního OS přímou integrací runtime enginu
- Zahrnutí runtime a funkčních knihoven přímo do aplikace/zařízení
- Přímé vykonávání z ROM a Flash pamětí



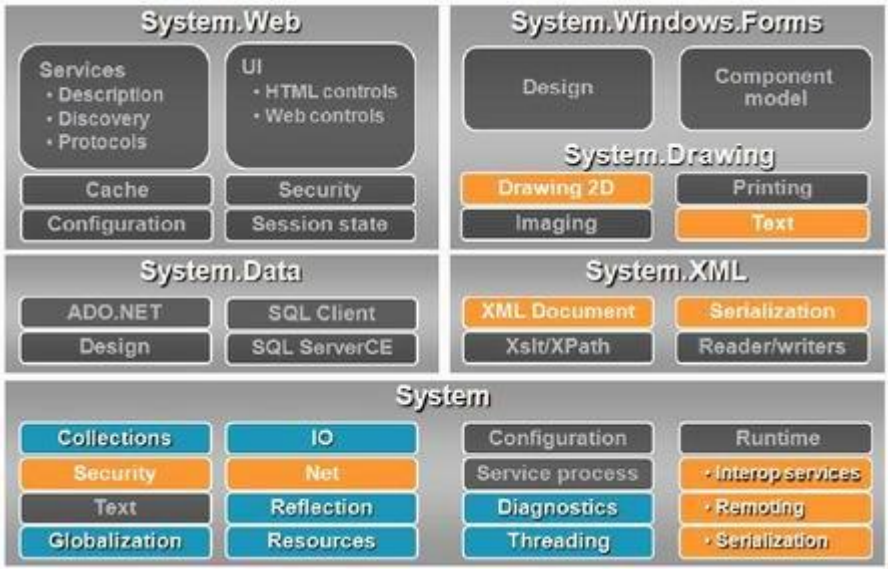
Postup překladu uživatelského programu do běžícího kódu

Základní podmínky pro bootování bez OS, navázání programu a přístup na hardware, periferie a jejich ovladače však zajišťuje vrstva HAL (Hardware Abstraction Layer). Ta na hardwaru zabere jen několik desítek kB a přizpůsobuje program konkrétnímu hardwaru (CPU), na kterém aplikace poběží. I když HAL poskytuje rozhraní k hardwaru (IRQ, časovačem, I/O), nejde o plnohodnotný kernel a pro zajištění všech potřebných funkcí (správu a řízení procesů apod.) se zde využívá propojení s CLR. Propojením HAL a CLR vzniká něco, co se dá vzdáleně označit jako do programu integrovaný OS. Zároveň však není nutné HAL použít a program spouštět na zařízeních s OS.

Z pohledu uživatele / programátora embedded aplikací přináší *.NET Micro Framework* proti jiným "klasickým" metodám navíc velkou výhodu v podobě použití Frameworks knihoven. Tak lze většinu

běžných operací realizovat voláním příslušné funkce v knihovně a není nutné ji vyvíjet. Pro potřeby a požadavky *Micro Framework* je k dispozici opět jen omezená skupina funkcí celé *.NET Framework* knihovny, protože ostatní funkce nejsou jednoduchých aplikacích potřeba.

Pomocí *.NET Micro Framework* tak lze snadno vytvořit moderní bezpečný program s vývojovými nástroji a programovacími jazyky C# a Visual Studio.NET, i pro malá a jednoduchá zařízení bez i s OS a s omezenými výpočetními prostředky bez požadavku se při programování detailně zabývat hardwarovou strukturou CPU. To zjednodušuje a urychluje vývoj aplikací.



. Porovnání podpory knihoven .NET Framework a .NET Micro Framework (barevná pole jsou podporovány Micro Framework)

Porovnání .NET Micro Framework proti OS

- Není to OS v pravém slova smyslu
- Integruje prvky OS přímo do kódu => *bootable runtime* - není nutný samostatný OS
- Malé nároky na hardware (podpora CPU bez MMU, např. typu ARM 7)
- Malé nároky na paměť (stovky kB)
- Framework technologie zajišťuje spolehlivý a bezpečný běh programu
- Univerzální programování embedded aplikace v C#, Visual Studio.NET, J# nezávisle na hardwaru
- Využití tradičních vývojových prostředků - např. Visual Studio

Parametr	.NET Micro Framework	Windows CE	Windows XPe
Použití	senzory, displeje, komunikační uzly, robotika	GPS, PDA, HMI, měř. přístroje, zprac.dat	rozsáhlé řídící a ovládací průmyslové systémy, řízení výrobních linek, složité měřící systémy
Paměťová náročnost	stovky kB	jednotky MB	desítky MB
El. spotřeba	velmi nízká - lze baterie	nízká - lze akumulátory	běžná - nutné je síťové napájení
CPU	ARM7, ARM9, CPU bez MMU	X86, MIPS, SH4, ARM, s MMU	x86 - AMD, Intel, VIA, ...
Real-Time	nepodporuje	podporuje	podporuje

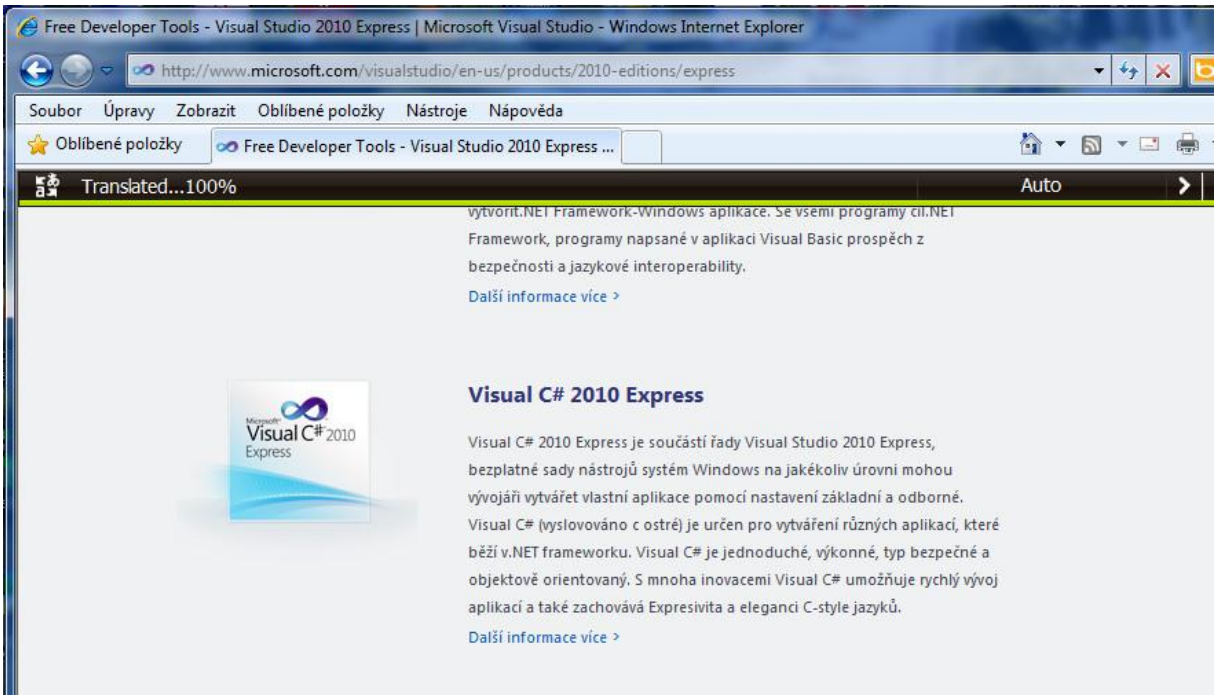
my remarks: *CanSat Book for Students* – part.3 - 2012

zprac. dat			

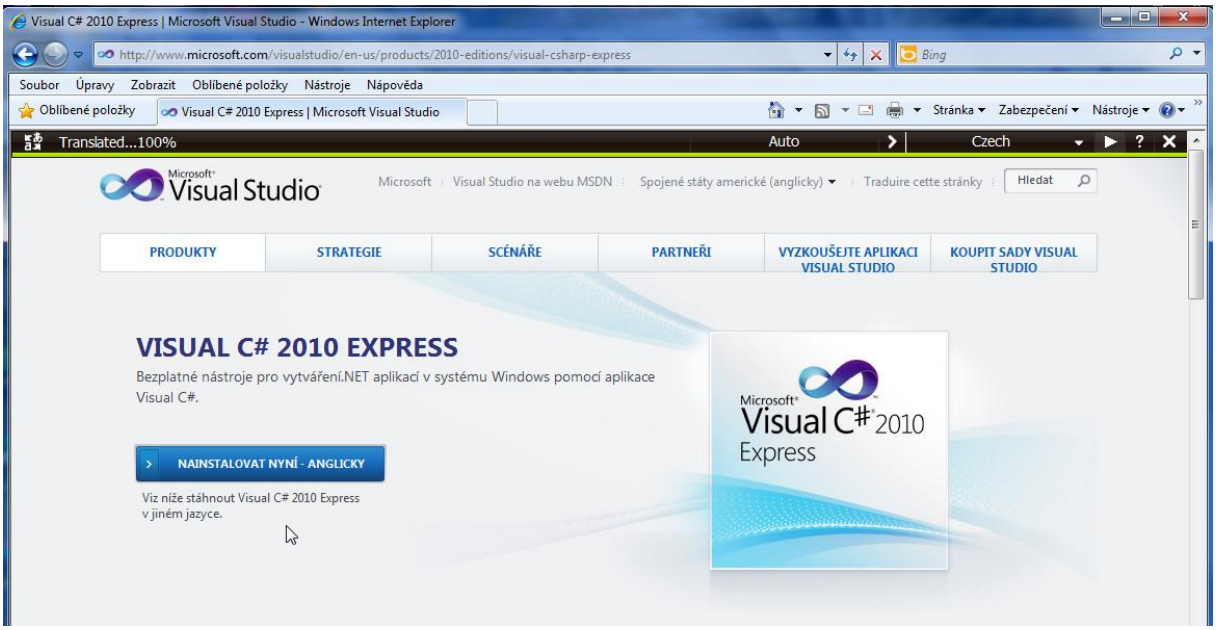
Shrnutí: .NET Micro Framework můžeme používat pro programování jednočipových (32 bitových) počítačů namísto dřívějšího programování v asembleru či C.

Instalace micro .NET

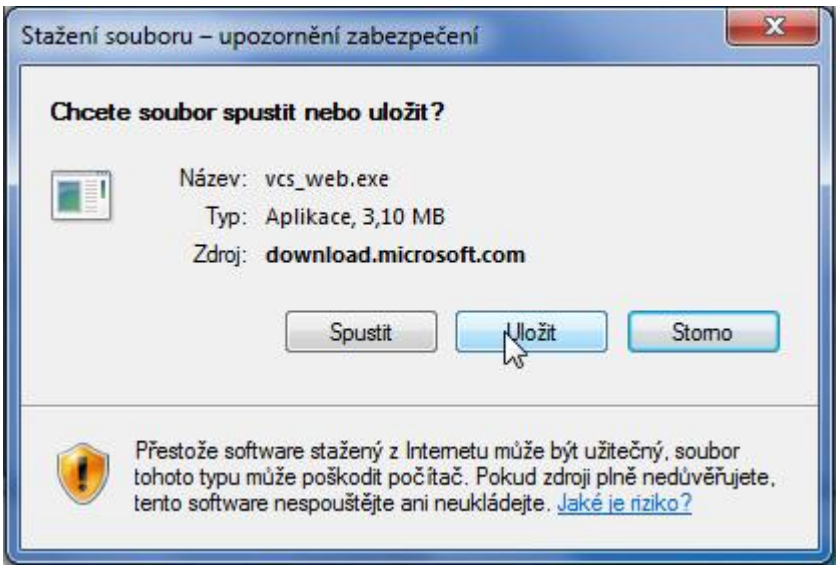
Na stránce http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express



Klikneme na **Další informace vice** , dostaneme



Klikeme na tlačítko **Nainstalovat nyní – anglicky ,** objeví se



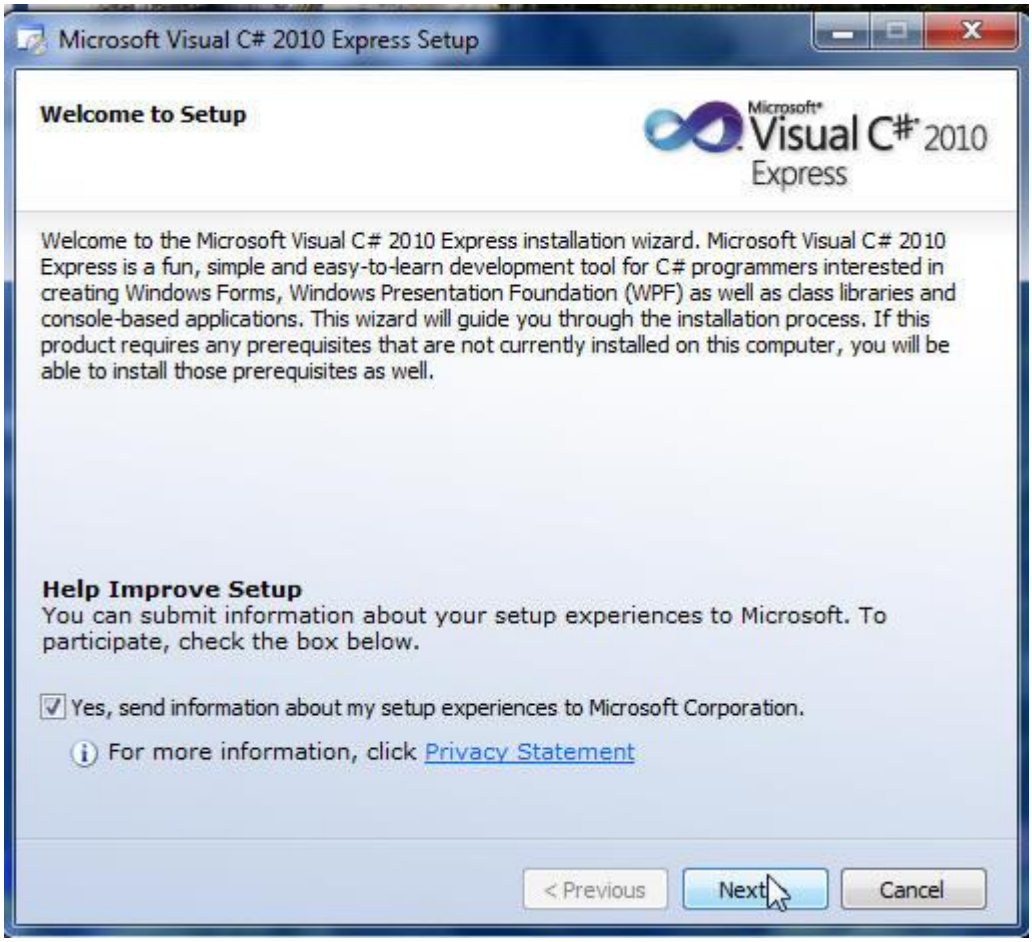
Tento soubor si uložíme někam na HDD našeho PC:

Pozn.
U *Visual C# 2010 Express* není k dispozici instalační medium (DVD či CD) na rozdíl od předchozích verzí Express Edition a k instalaci je nutné připojení k internet a kratičký (cca 3 MB) soubor vsc_web.exe, zabezpečující stahování potřebných souborů I jejich umístění na našem PC.
Aktuální verze *.NET Micro Framework* verze 4.1 je již podporována Visual Studiem 2010 a k němu již instalační DVD existuje a studentům je dostupné v rámci MSDN Academic Aliance.

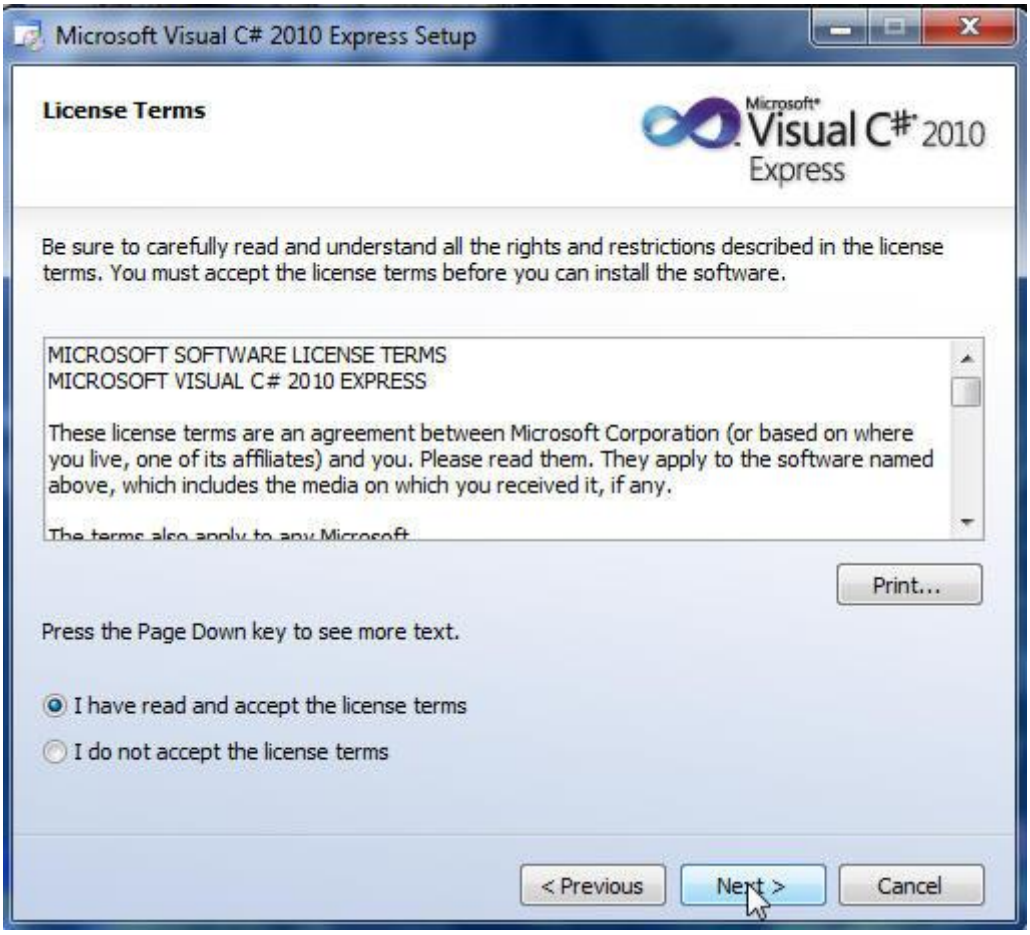
Další možností získání zdarma Visaul Studia a dalšího Microsoft software pro žáky a student všech druhů škol (zš, sš I vš) a jejich učitele pro účely souvisejí s výukou, je MS project DreamSpark. K přihlášení do tohoto projektu je však nutná karta ISIC či ITIC viz
http://www.partnershop.isicpoint.cz/microsoft-dreamspark-5

Pokračujeme v instalaci **MS Visual C# Express**, máme k dispozici

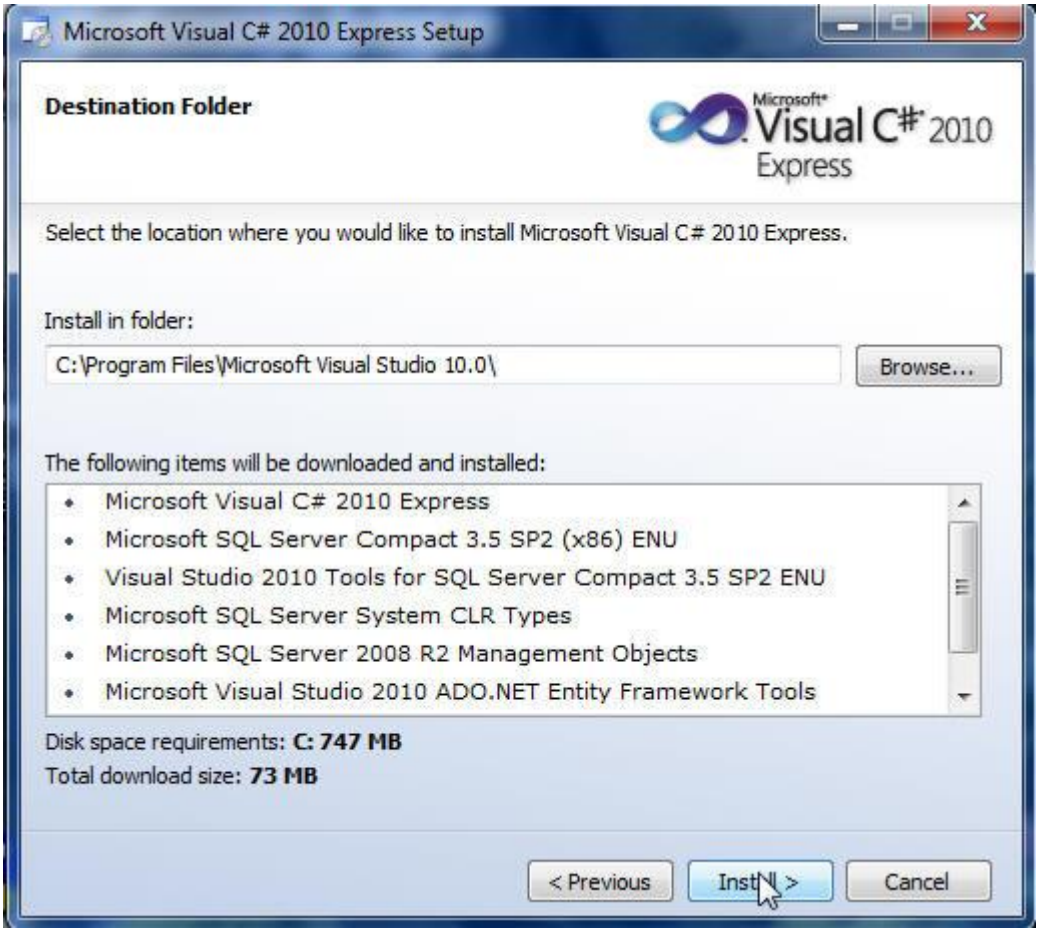
Spustíme tento prográmek, objeví se:



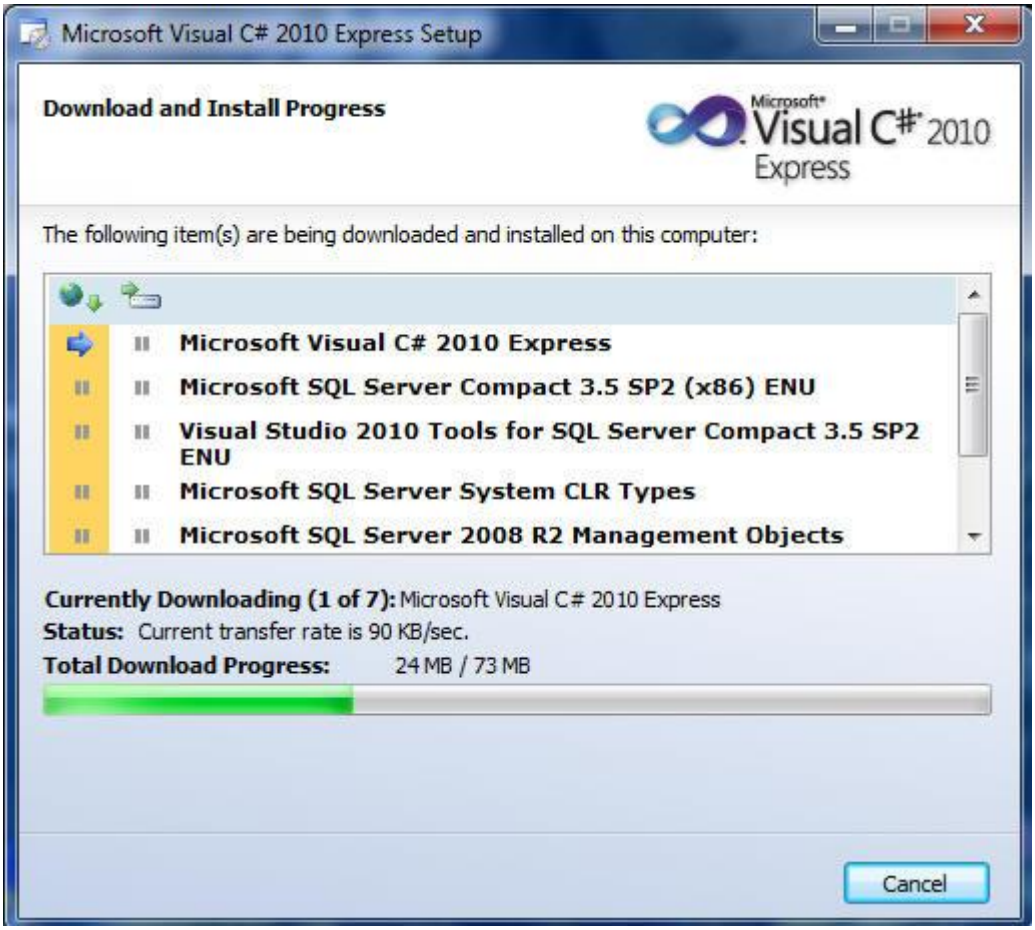
Stiskneme tlačítko **Next** a pokračujeme



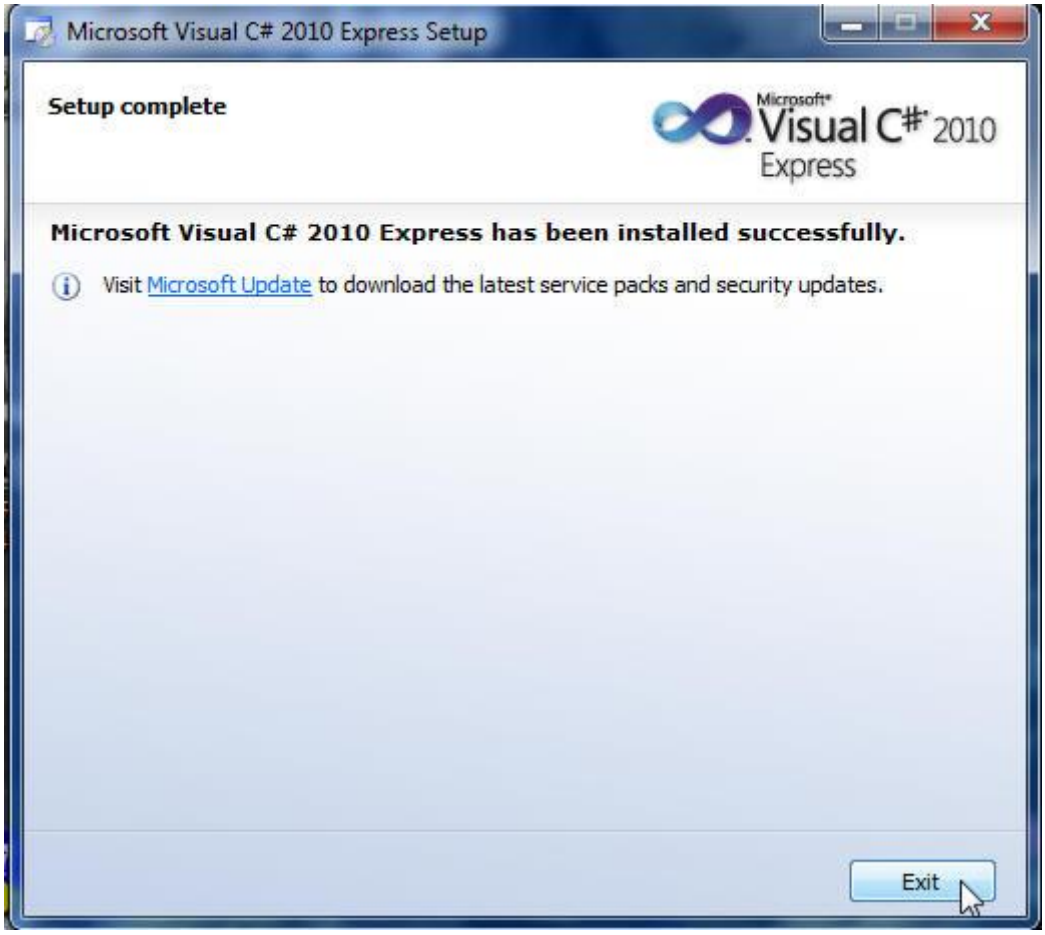
Potvrdíme souhlas s licenčními podmínkami a klikneme na tlačítko **Next**



Ponecháme nabízený sdresář pro umístění instalovaného sw a spustíme instalci stisknutím tlačítka **Install.** Začne probíhat instalace:

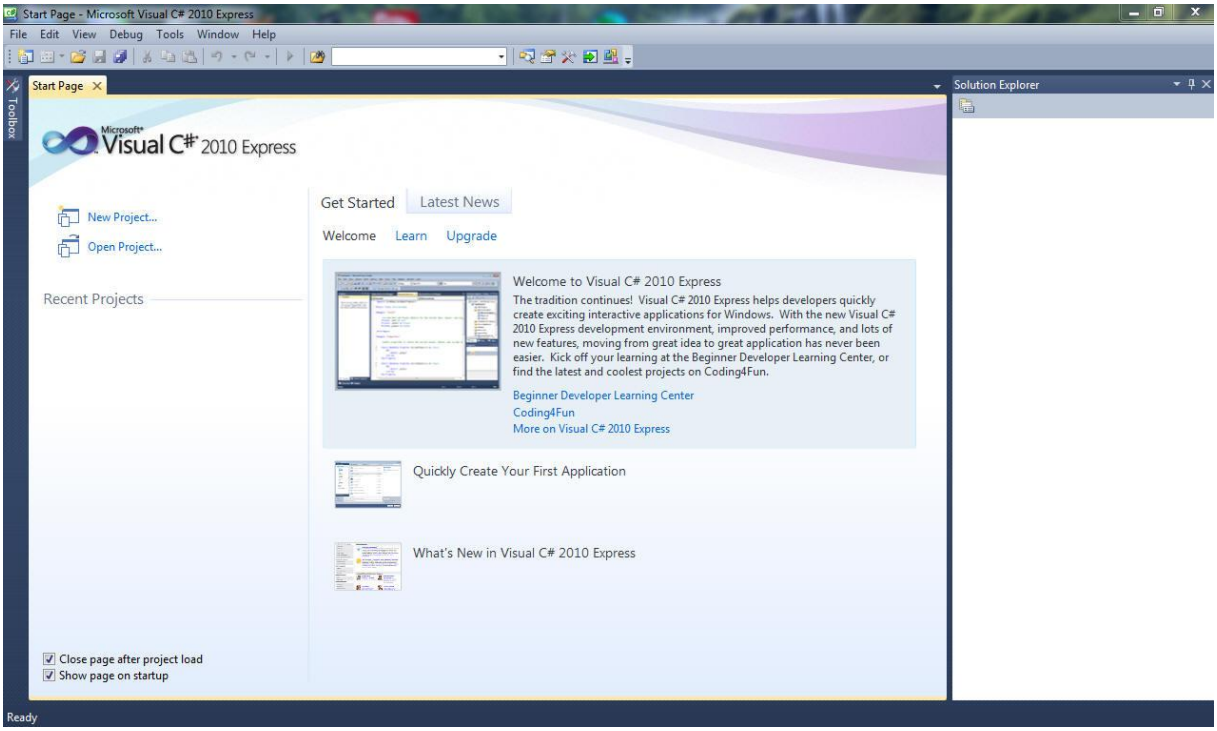


Po jejím skončení se objeví win okno:



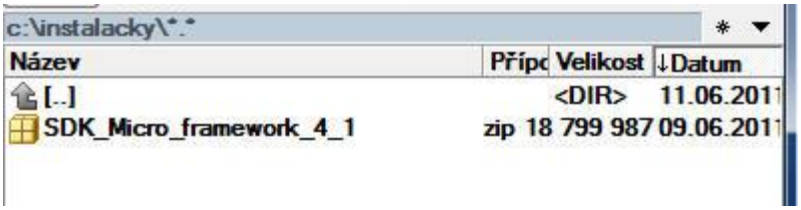
Potvrdíme tlačítkem **Exit**. MC Visual C# 2010 Express tak máme nainstalovaný

Zkusíme ho spustit (popř. ověříme jeho funkčnost pomocí tvorby nějakého program pro PC – známe z výuky na naší škole, kde v C# programujeme již od prvního ročníku)

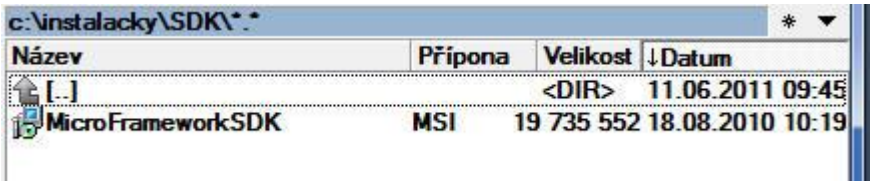


Instalace .NET Micro Framework 4.1 SDK

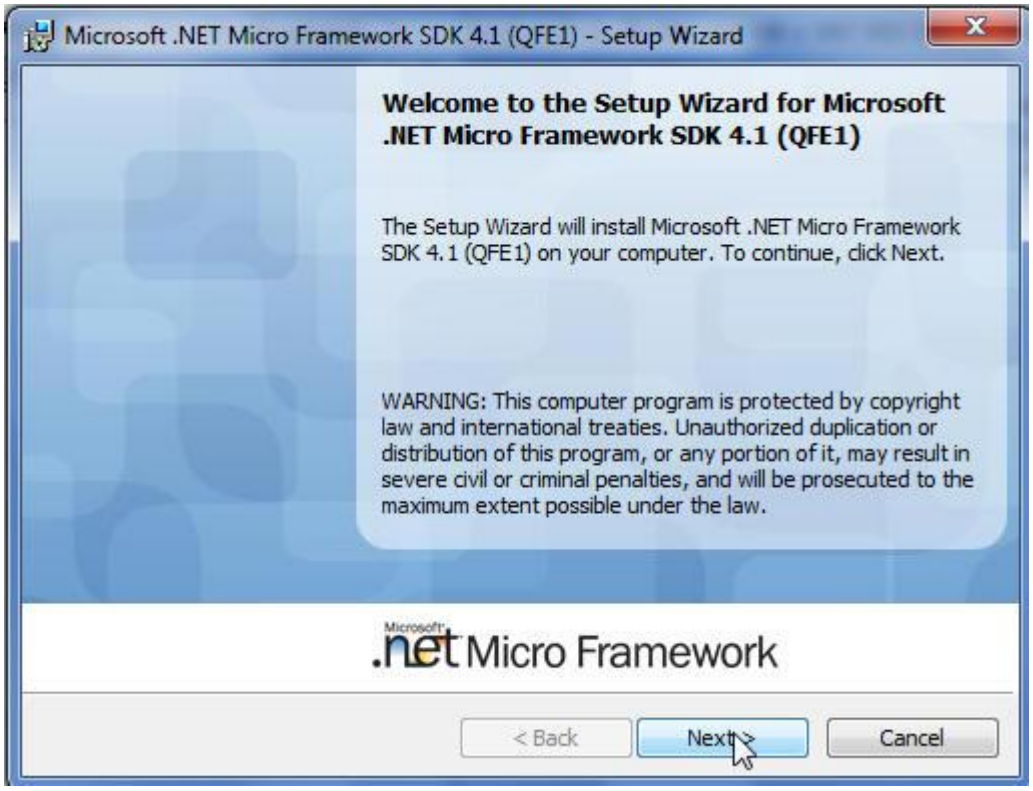
Ze stránek Microsoftu si stáhneme soubor *SDK_Micro_framework_4_1.zip* na HDD našeho PC:



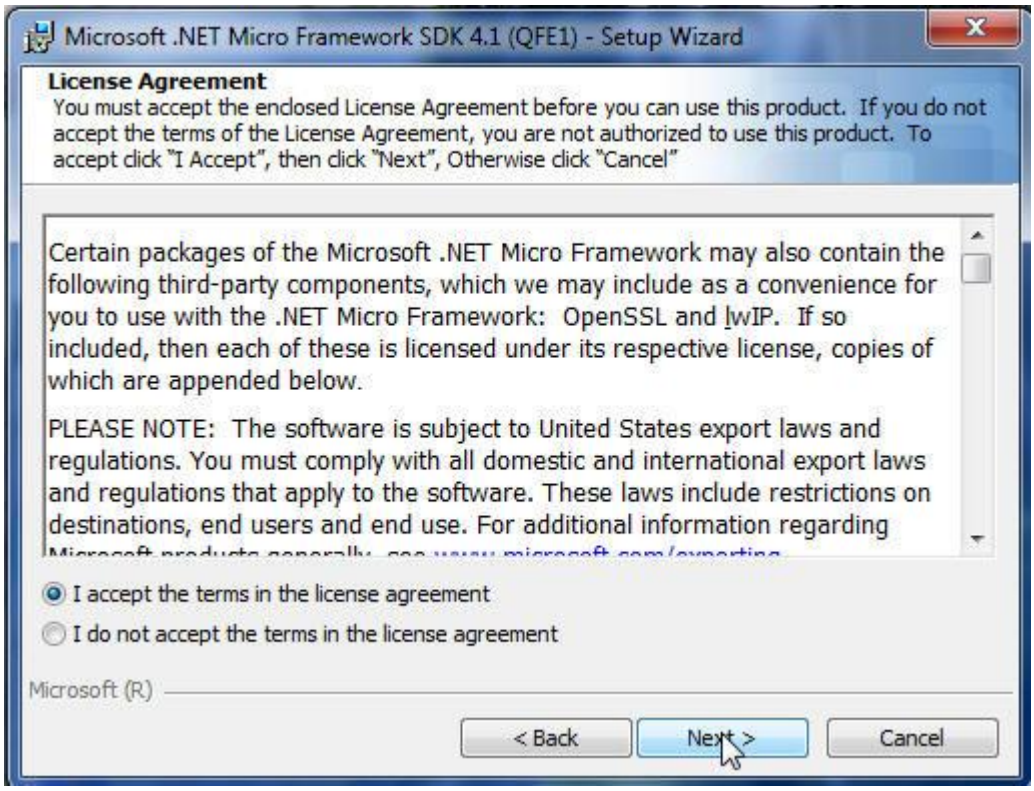
Tento sw nalezneme i na DVD příloze tohoto výukového material. Z výše uvedeného souboru již získáme (“rozzipováním”) instalační msi soubor:



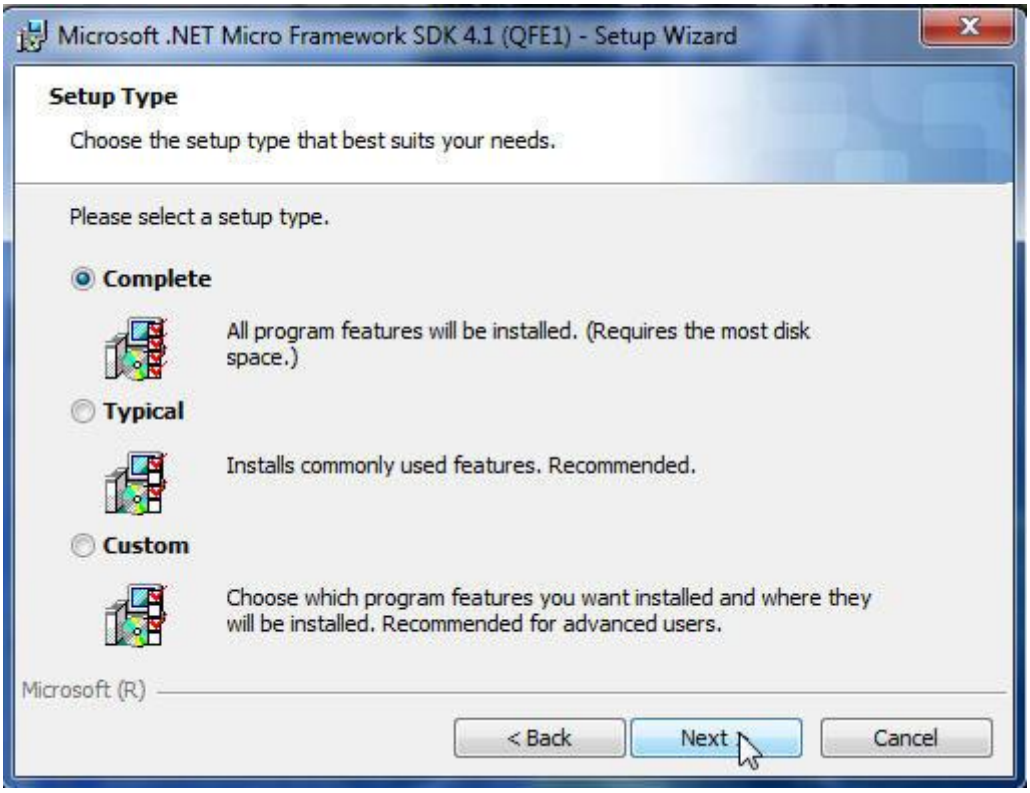
Tento soubor spustíme a poté již dostame



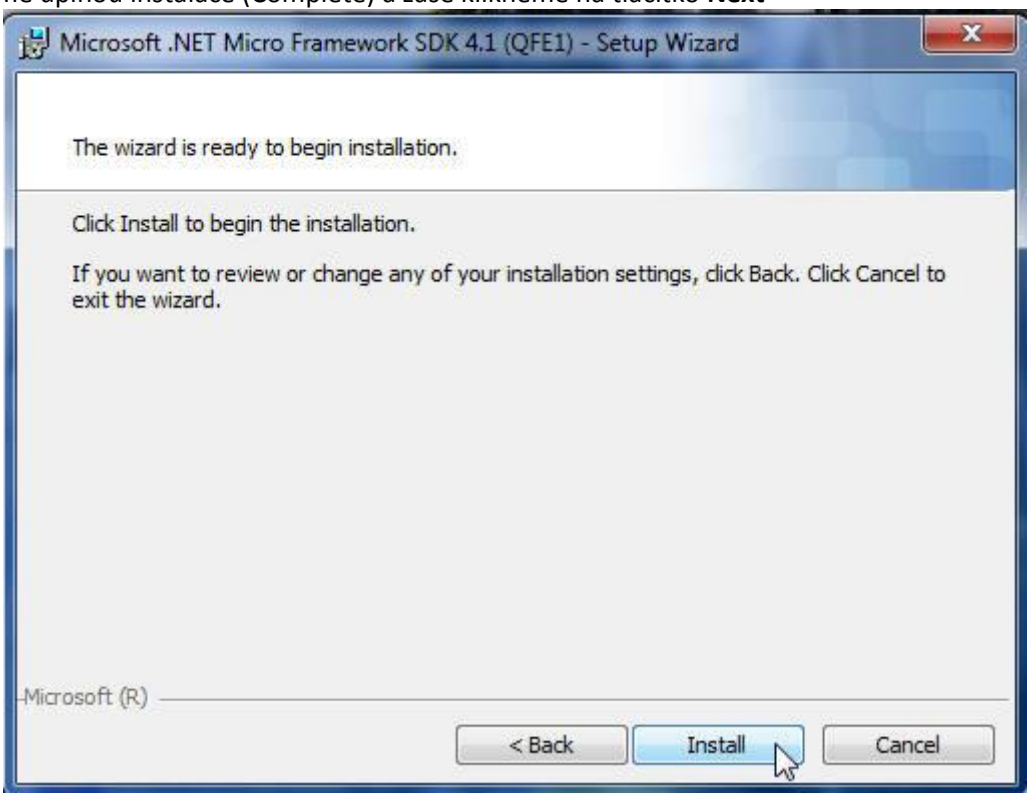
Klikneme na tlačítko **Next** a poté se objeví licenční ujednání



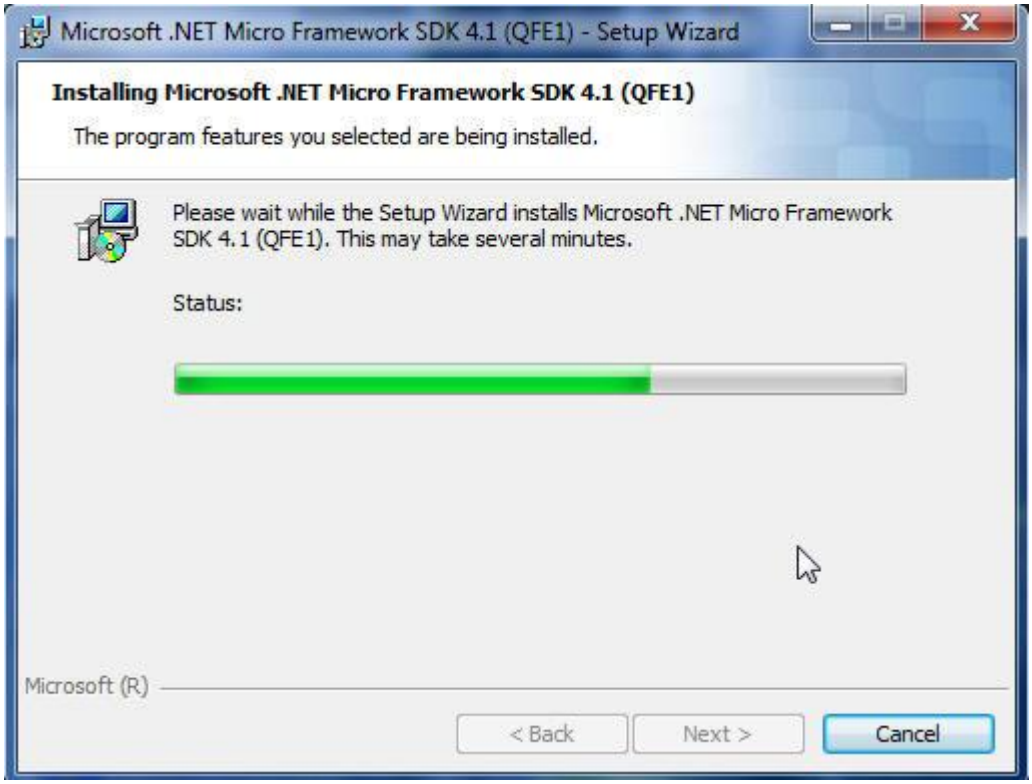
Zaškrtneme souhlas s licenčním ujednáním a opět klikneme na tlačítko **Next** , objeví se



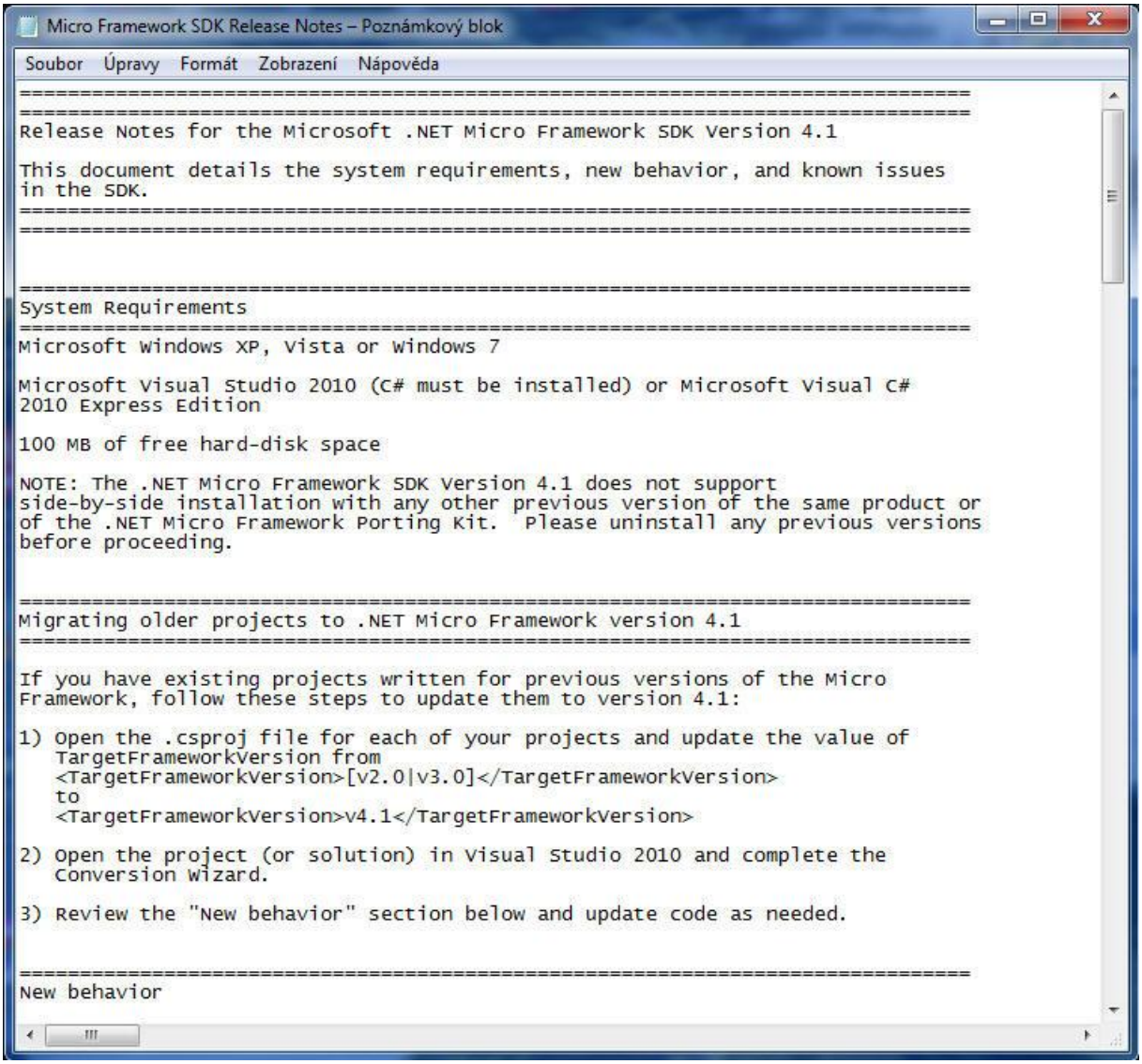
Vybereme úplnou instalace (Complete) a zase klikneme na tlačítko **Next**



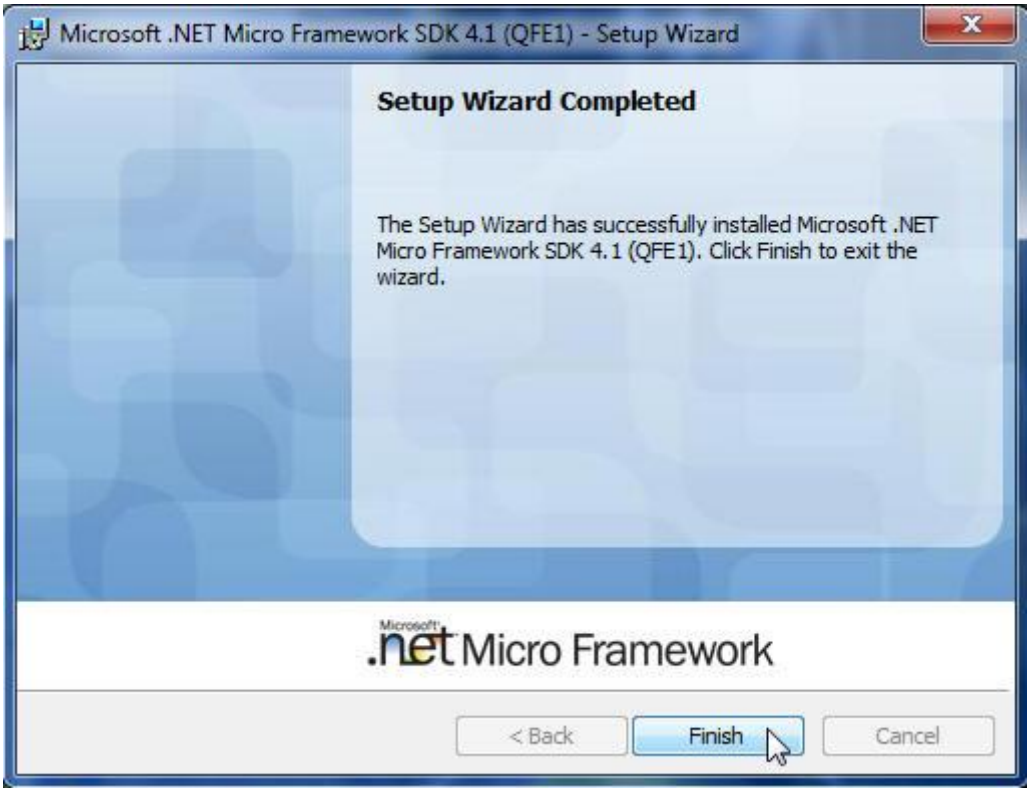
Nyní již může proběhnout vlastní instalace. Stiskneme tedy **Install .** Začne probíhat instalace:



Po dokončení instalace se zobrazí poznámky:

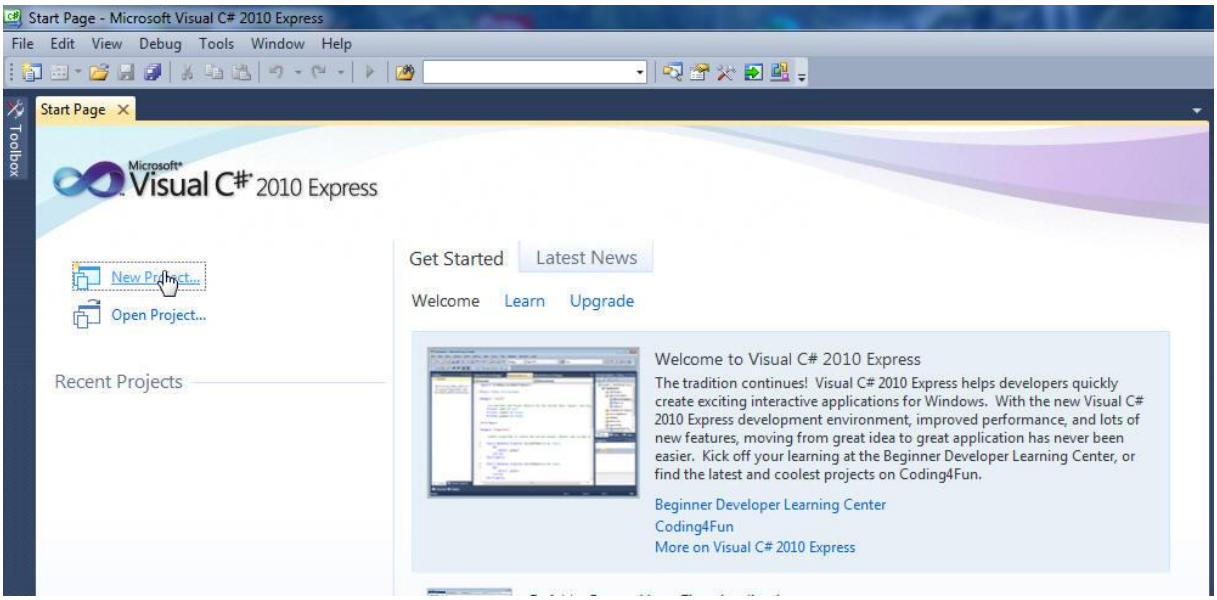


Poznámkový blok zavřeme.

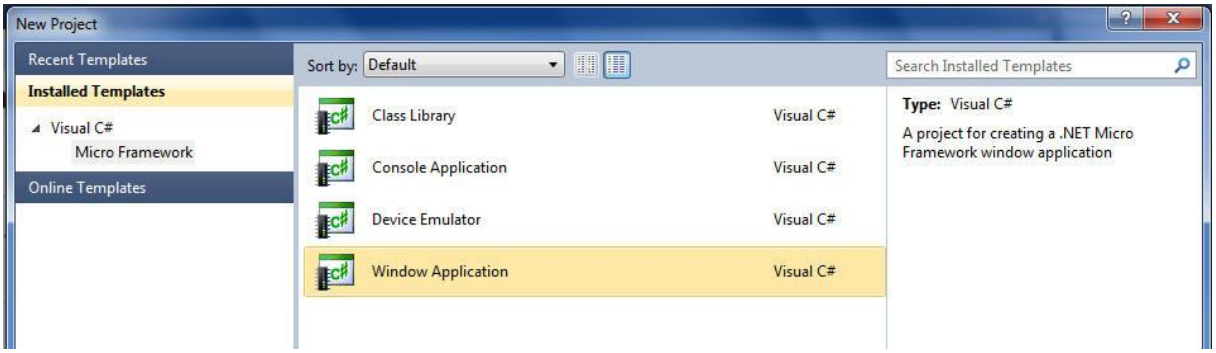


A konec instalace potvrdíme tlačítkem **Finish**. Ukážeme si, jak se projevila v našem **Visual C# 2010 Express**. Proto ho spustíme:

.



Vybereme **New Project** a dostaneme:



Vidíme, že naše vývojové prostředí již obsahuje Micro Framework se čtyřmi wizardy.

2.1.5.1.2 FEZ Cerberos

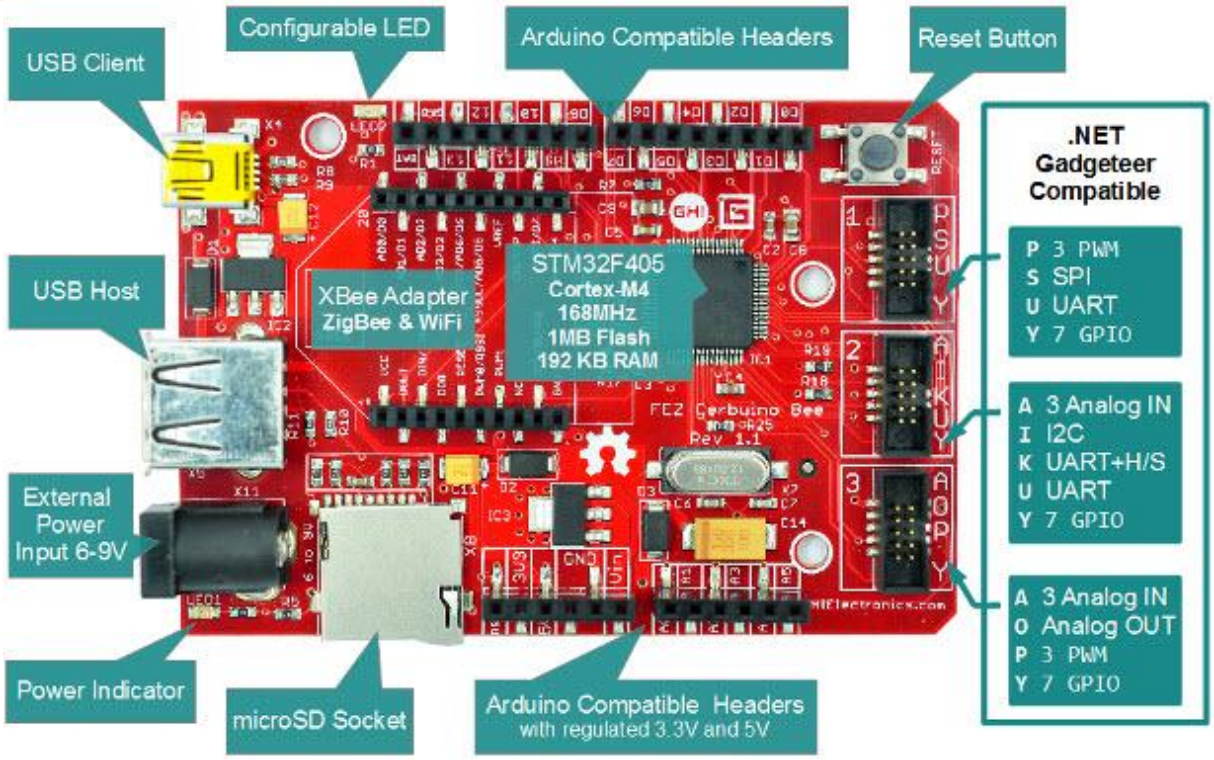
Nejprve si popíšeme tento startkit. Vzhledem ke konstrukci našeho onboard počítače budeme to vše také využít.

GHI Electronics FEZ Cerbuino Bee Mainboard

GHI Electronics FEZ Cerbuino Bee Mainboard je navržen pro vývojáře kteří potřebují laciný, *Arduino*-kompatibilní .*NET Gadgeteer*-kompatibilní startkit. Jde o 100% open-source (OSHW) řešení obsahující 168MHz *STM32F405* ARM Cortex-M4 s 1MB Flash a 192Kb RAM, dále konektor pro napájení, napěťové regulátory, MicroSD konector, USB host, a USB Client konektory. *FEZ Cerbuino Bee* umožňuje plug-and-play propojení prostřednictvím USB kabelu. Tři .NET Gadgeteer-kompatibilní sokly umožňují připojit další Gadgeteer moduly k *FEZ Cerbuino Bee* startkitu. Navíc XBee sockl automaticky přináší všechny rozšířené možnosti bezdrátového propojení včetně WiFi a ZigBee.

Vlastnosti
STM32F405 ARM Cortex-M4

3 .NET Gadgeteer kompatibilní sokly pro tyto typy:
Y, A, I, K, O, P, S, U
Arduino-kompatibní konektory (některé signály jsou sdíleny s
Gadgeteer sokly)
XBee Adapter pro ZigBee nebo WiFi XBee moduly
konfiguratelné on-board LED
Software/hardware zahrnuje (ale není omezené jen na ně):
.NET Micro Framework 4.2 (podporující C# a Visual Basic)
s FEZ Cerberus firmware
168Mhz 32bit processor s floating point instrukcemi
1MB FLASH, s 300K pro uživatelský kód
192KB RAM, 112KB pro uživatelskou haldu (heap)
plnél TCP/IP s HTTP, TCP, UDP, DHCP
podpora Ethernet s Ethernet ENC28 modulem
USB host, USB zařízení
SPI, I2C, 2 UART, CAN
9 analogových vstupů, 2 analogové výstupy
4-bit MicroSD interface
6 PWM
OneWire interface
Vestavěné hodiny reálného času (Real Time Clock -(RTC), potřebuje 32kHz krystal
RLPLite dovoluje uživateli nahrát nativní kód (C/Assembly) pro úlohy v reálném čase (RT)
FAT souborový systém



Dále si popíšeme, jak upload firmware *FEZ Cerberos* do *STM32F405* našeho onboard počítače
http://wiki.tinyclr.com/index.php?title=Firmware_Update_FEZ_Cerberus

Pokud začínáme s *Cerberus/Cerbuino* pak musíme nejprve provést několik věcí před update našeho startkitu.

Nejprve musíme nainstalovat USB ovladače pro *STM32 BOOTLOADER* a nainstalovat *DFU Tester* (v3.0.1).

Proto nejprve vyextrahujeme (unzip) instalační software z *STM_DFU.zip* který najdeme v adresáři GHI :
C:\Program Files (x86)\GHI Electronics\GHI OSHW NETMF v4.2 SDK\FEZ Cerberus\Firmware

Dále vybereme potřebnou instalačku:
DfuSe_Demo_V3.0.2_Setup.exe pro 32 bitový OS
DfuSe_Demo_V3.0.2_Setup_amd64.exe pro 64 bitový OS
a instalujeme program podle následujícího popisu.

USB ovladač musí být nainstalován spolu s *GHI NETMF v4.2* a *.NET Gadgeteer Package* instalací, jako Krok 7. Je-li to potřebné, můžeme popř. ovladače nainstalovat z menu *Package -> Installation Files*, jako **GHI NETMF USB Drivers x64** (nebo x86 v závislosti na tom, jaký máme procesor). Spuštění daného msi souboru provede instalaci ovladačů do Windows.

Nahrání Cerberus Firmware
Cerberus firmware má dvě části, *TinyBooter* (který dovoluje nastavit/změnit síťový adapter, USB jméno, atdc.) a *TinyCLR* (který je hlavním operačním systémem při umisťování C# programů).

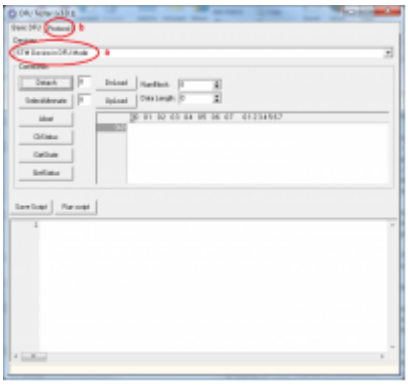
Nejprve nahrajeme *TinyBooter* podle následujících pokynů a potom nahrajeme *TinyCLR* což je popsáno po instrukcích pro nahrání *TinyBooter*.

Nahrání TinyBooter. Nahrání s použitím USB DFU

V **start** menu, klikneme na **Všechny programy**, najedeme na **STMicroelectronics->DfuSe** a vybereme **STDFU Tester.**

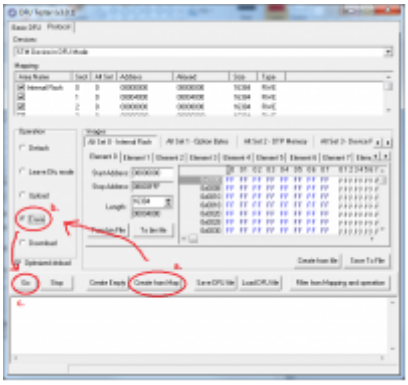
1. Při připojování startkitu Cerberus k napájení prostřednictvím USB Client DP nemo USB Client SP modulů, musíme nastavit Cerberus do režimu Boot nastavením úrovně H na pin BOOT.
To, že startkit je opravdu v režimu boot indikuje (na PC) zobrazení *STM32 BOOTLOADER* v **Zařízení a tiskárny** . Nyní již můžeme odpojit H od pinu boot. Pozn – postup dle bodu 1 musíme zopakovat pokud zresetujeme startkit před nahráním (updating/uploading) firmware.

2. Po startu programu, uvidíme, zda jsme v režimu boot tím, že je indikován mezi zařízeními (a): STM Device v DFU Mode. Je-li vše vpořádku, klikneme na Protocol tab (b) nahoře v okně.



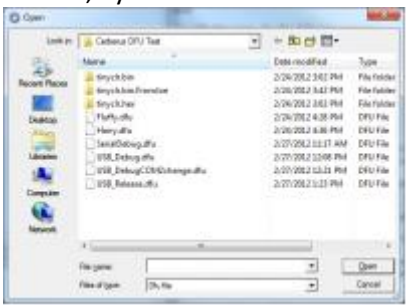
3. Erase – Pokud upgradujeme nebo měníme naš firmware musíme nejdříve vymazat Cerberus. Kvůli tomu provedeme následující kroky (pozn. Všechna data budou vymazána):

a. Klikneme na Create pomocí tlačítka Map.
b. Klikneme na radio button Erase.
c. Klikneme na Go to erase the Cerberus.

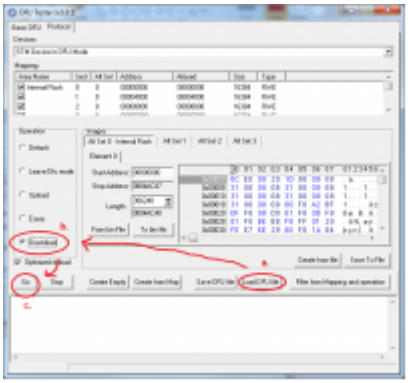


4. Download – v tomto kroku provedeme přípravu na stažení firmware pro Cerberus.

a. Klikneme na Load DFU file a tím se objeví dialogové oknor. Jděme na C:\Program Files (x86)\GHI Electronics\GHI OSHW NETMF v4.2 SDK\FEZ Cerberus\Firmware kde TinyBooter_4_2_x_x.dfu je uloženo; vybereme **tento soubor** a klikneme na **Open**.



b. Klikneme na radio button **Download**
c. Klikneme na **Go** a tím odešleme firmware do startlitu Cerberus.



Poznámka: Pokud omylem kliknete na tlačítko **Upload** místo na tlačítko **Download** tak se souibor DFU přepíše a proto budeme původní soubor DFU získat znovu z internetu.

5. Ukončíme STDFU Tester. Je-li to nutné rebootujeme Cerberus do TinyBooteru stisknutím reset tlačítka , při rozpojeném jumperu **BOOT**.

Nahrání TinyCLR (firmware)
Nahrání (TinyCLR) je už snadnější. Otevřeme MFDeploy výběrem menu **Start->Všechny Programy ->Microsoft .NET Micro Framework 4.2** a kliknutím na **Tools**, pak otevřeme **MFDeploy.exe**. Vybereme USB z drop down v sekci **Device** v horní části okna programu, a vybereme **Cerb-Family_Gadgeteer**.

Pokud Cerberus neodpovídá, podíváme se přes **Start->Zařízení a tiskárny** a přesvědčíme se, že startkit je rozeznán jako USB zařízení.

Dále vybereme pomocí Browse... and navigate poloýku C:\Program Files (x86)\GHI Electronics\GHI OSHW NETMF v4.2 SDK\FEZ Cerberus\Firmware. Vybereme adresář s firmware které potřebujeme a to buď **Ethernet** nebo **Non Ethernet**. Ve vybraném adresáři pak vybereme *Config.hex* a *Firmware.hex* a klikneme na open. Klikneme na **deploy to upload the firmware to the device**.

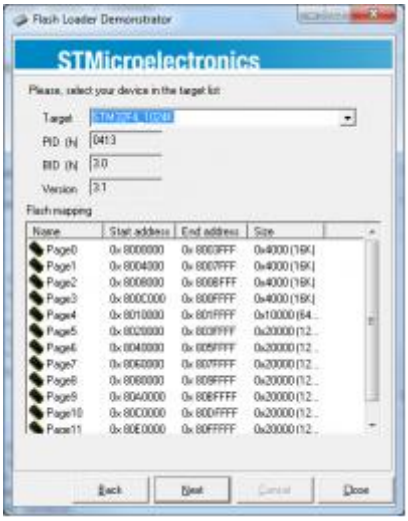
Pokud pouze updatujeme firmware tak provádíme pouze kroky následující **Load TinyCLR** a systém se automaticky uvede do *TinyBooteru*.

TinyCLRse dá updatovat použitím *STMicroelectronics' Flash Loader Demo*. Po jeho nainstalování ho spustíme. Startkit Cerberus uvedeme do režimu *Boot loader*, abychom mohli updatovat firmware. Nastavíme proto **BOOT0** pin na H při zapínání.

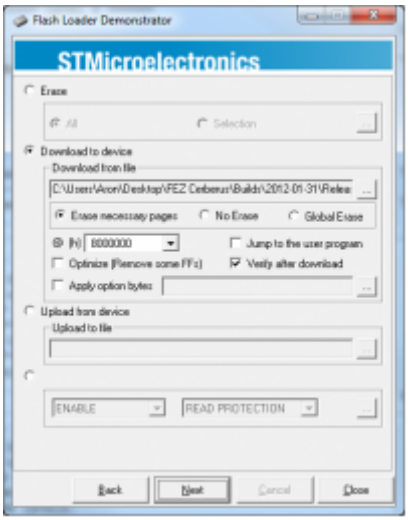
V *STMicroelectronics' Flash Loader Demo* vybereme COM port. Stiskneme tlačítko **Next**.

Dále uvidíme *dialog box* který oznamuje **Target is readable**. Klikneme na "Next".

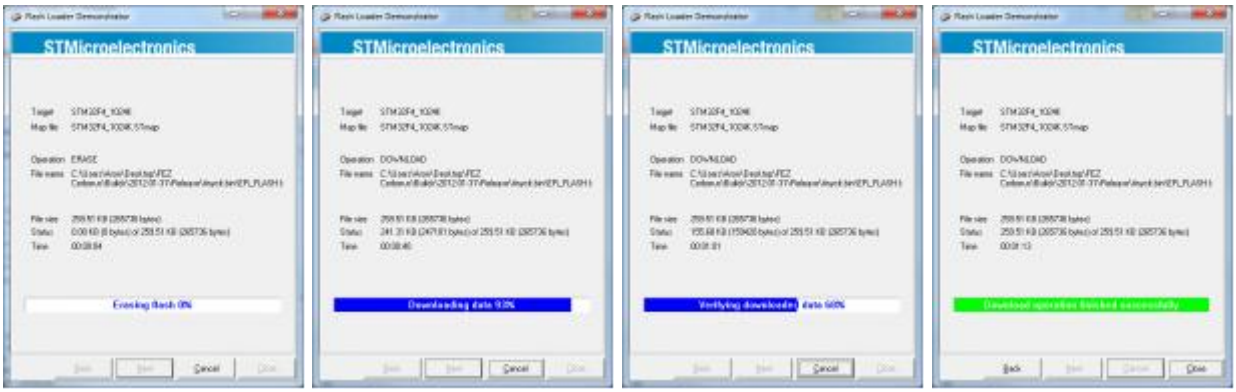
Opět stiskneme tlačítko **Next.** Dialogový box nám nabídne výběr **target**. Zvolíme *STM32F4_1024K* a stiskneme tlačítko **Next**.



Vybereme radio button **Select the Download to device**. Klikneme na ikonu **file select** na pravo od textboxu a vybereme požadovaný binární soubor. (pozn: binární soubor musí být s extenzí .bin .) Ponecháme vybráno **Erase necessary pages** a rovněž ostatní defaultní nastavení včetně adresy @ (h) 8000000, a stiskneme tlačítko **Next** .



Nyní uvidíme indikátor **progress** ukazující jak probíhá mazání, programování a verifikace sektorů a dále úspěšné zakončení.



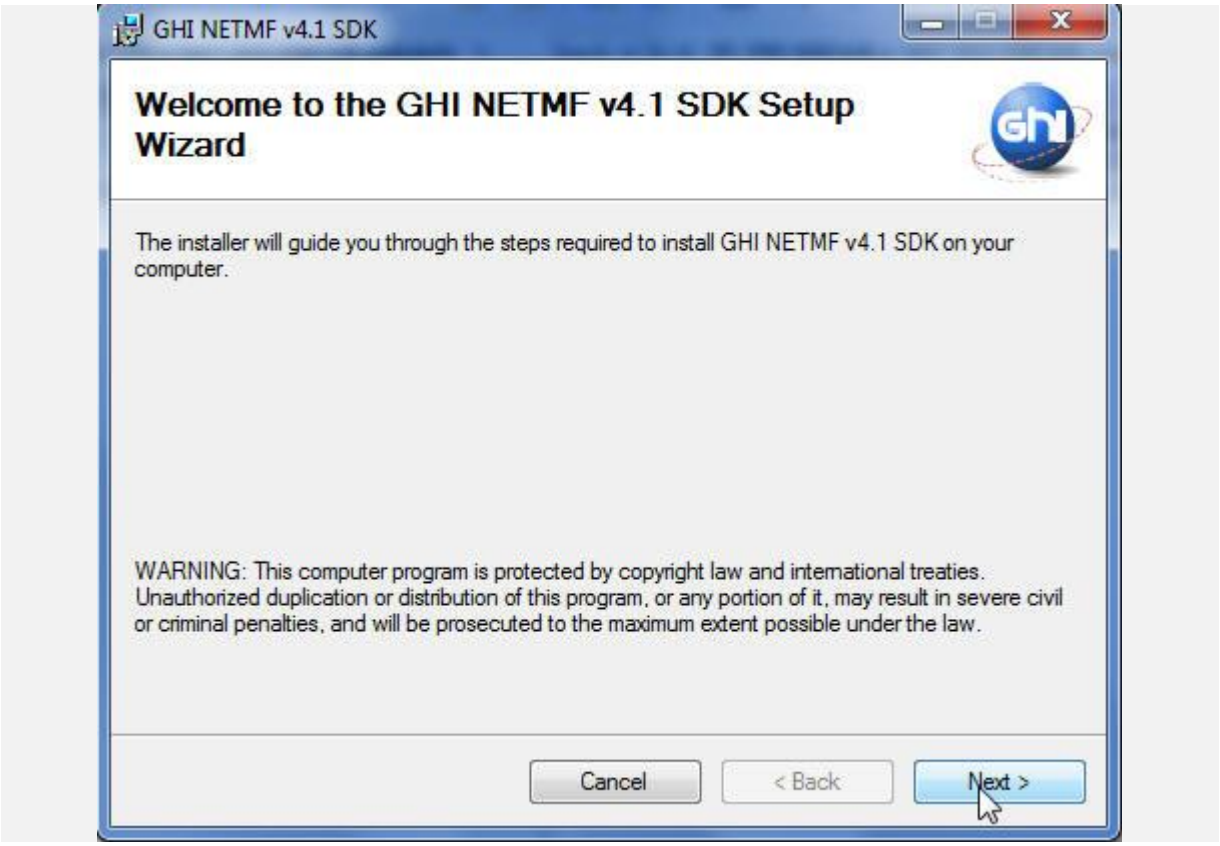
Další podrobnosti na **http://www.ghielectronics.com/support/dotnet-micro-framework**
Dále musíme základním SDK od Microsoftu doplnit *GHI Electronics NETMF SDK* :

Instalace GHI NETMF 4.1 SDK

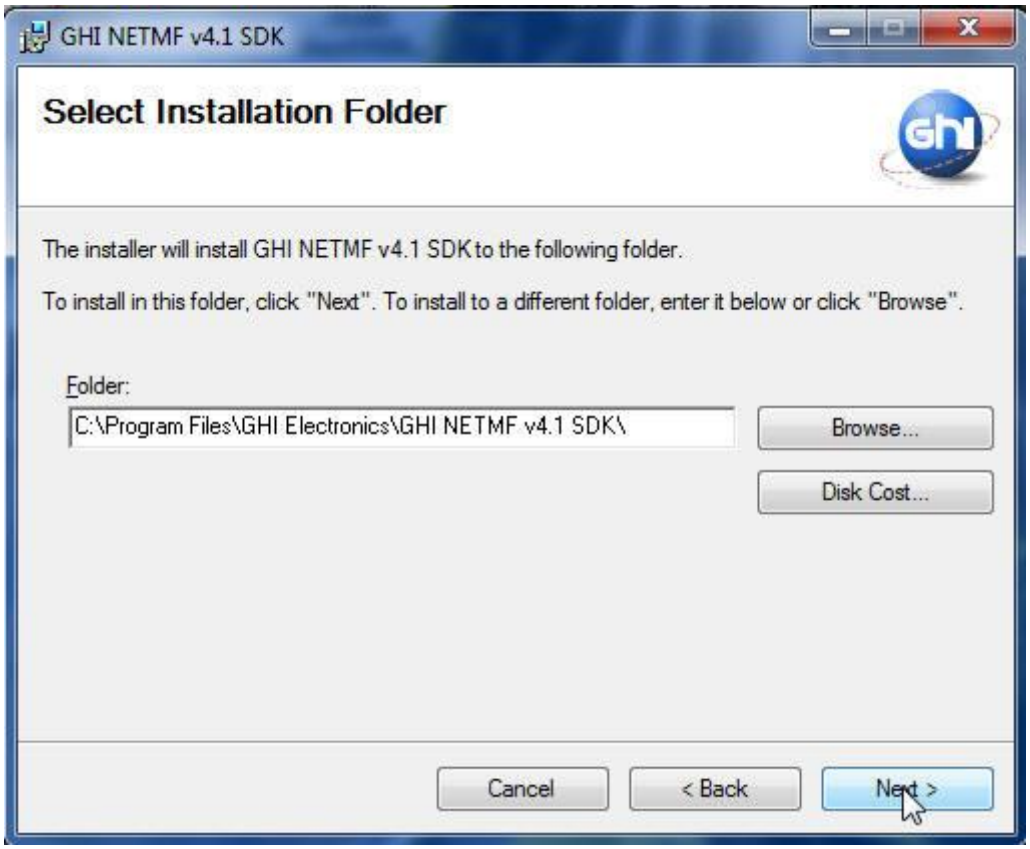
Pro práci s *FEZ Cerduino* budeme dale potřebovat nainstalovat knihovny . K jejich instalaci jsou k dispozici instalačky jako msi soubor, nebo obvyklý setup.exe:

Získáme je na firemní stránce GHI Electronics, nebo opět na DVD příloze.

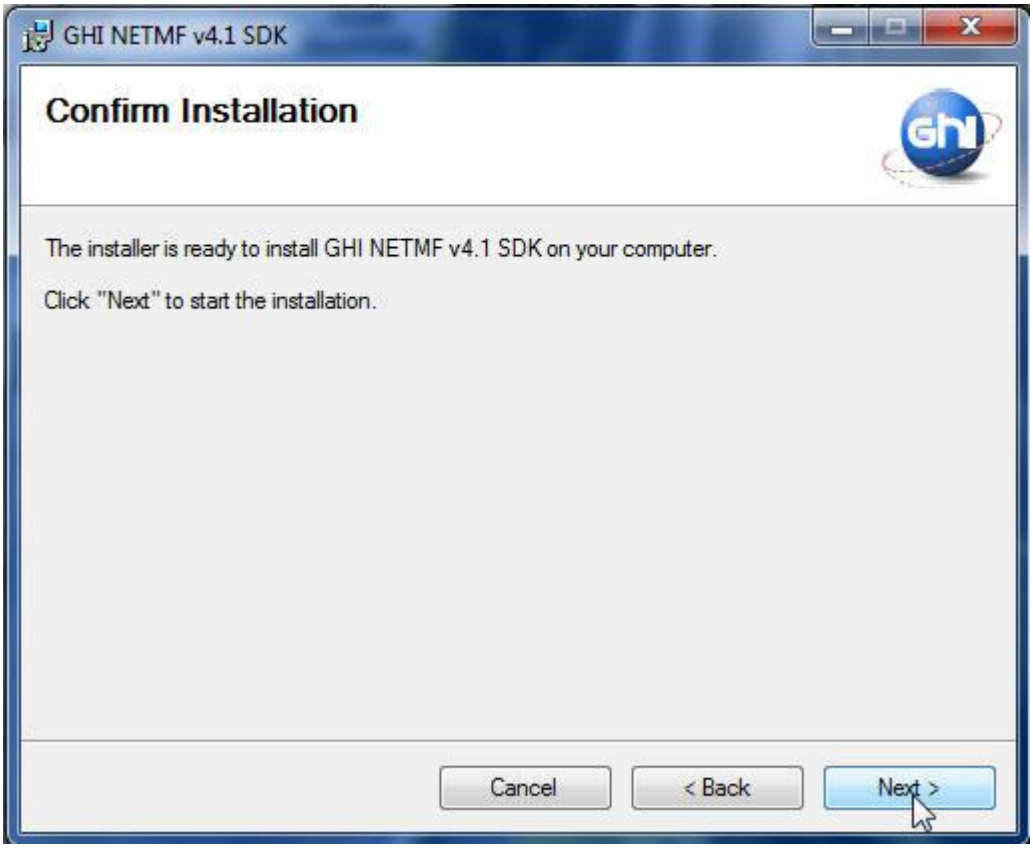
Spustíme tedy instalačky. Objeví se:



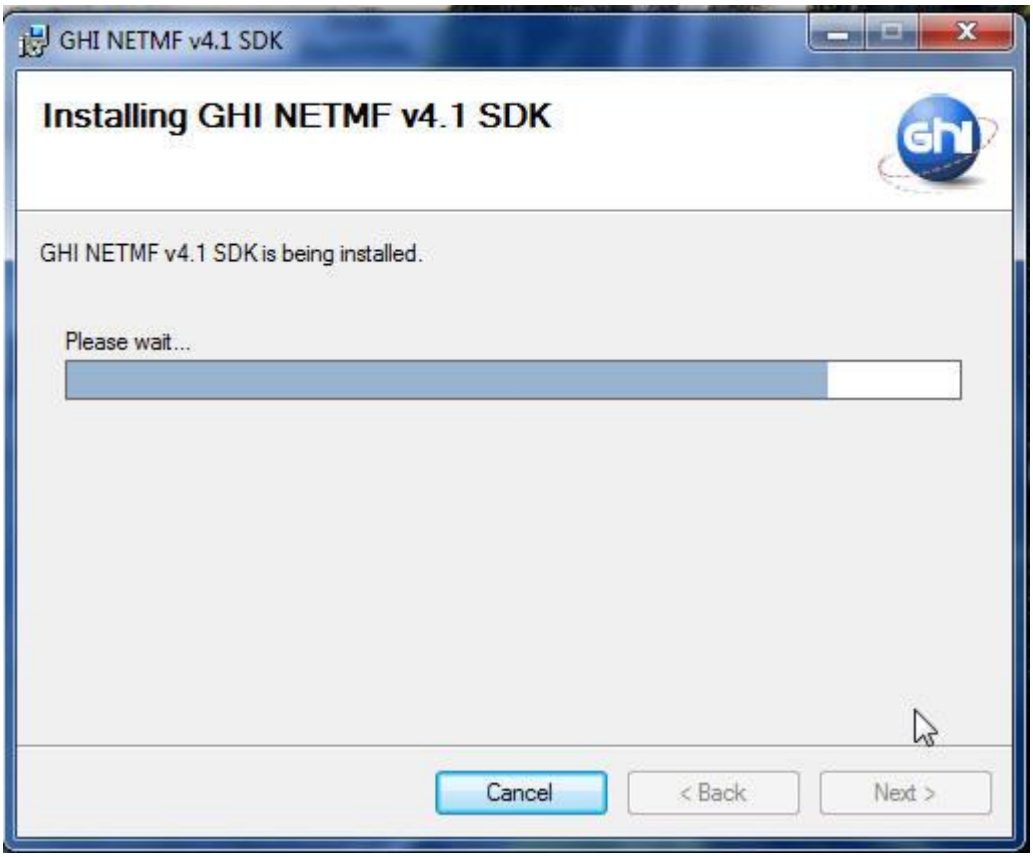
Stiskneme tlačítko **Next** a máme:



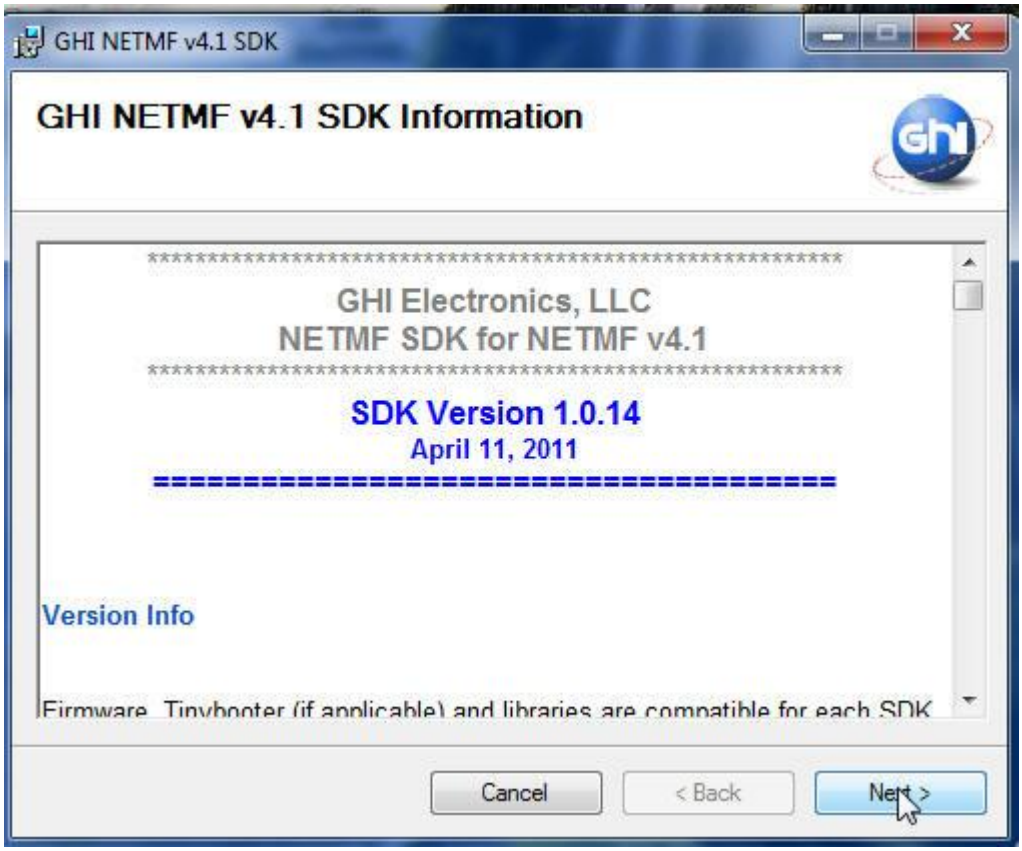
Ponecháme nabízenou složku pro sw a potvrdíme **Next**. Pokračujeme



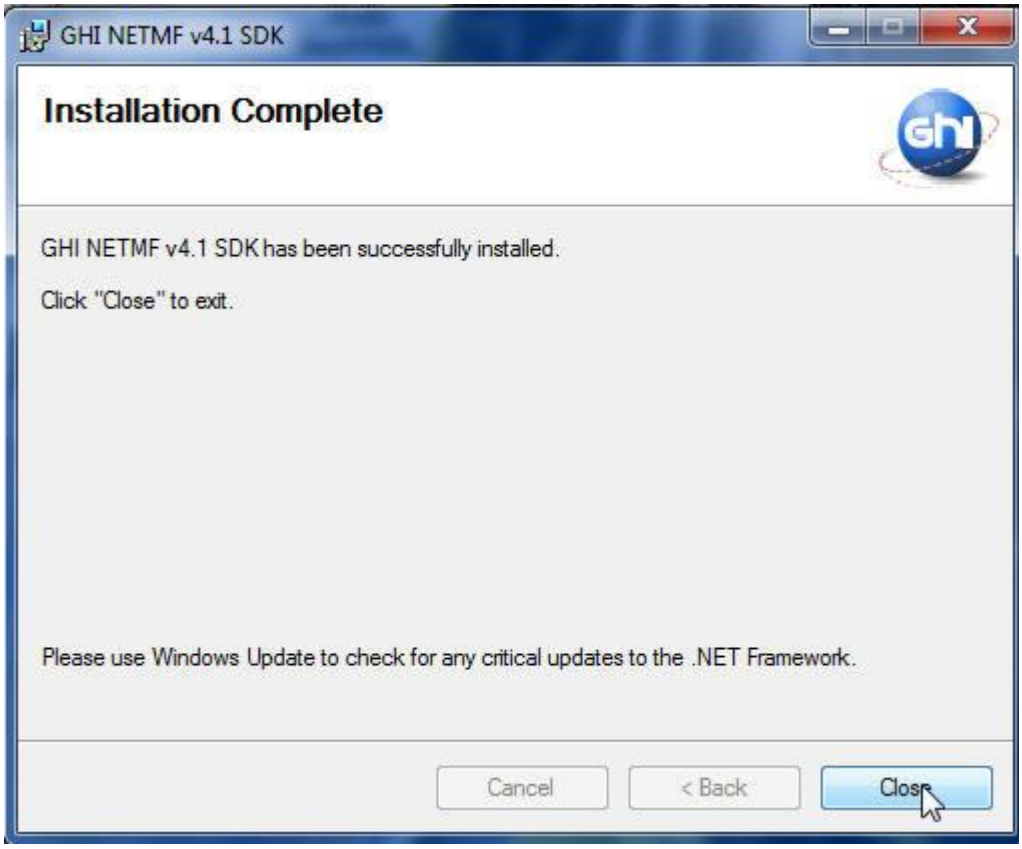
Opět tlačítko **Next**



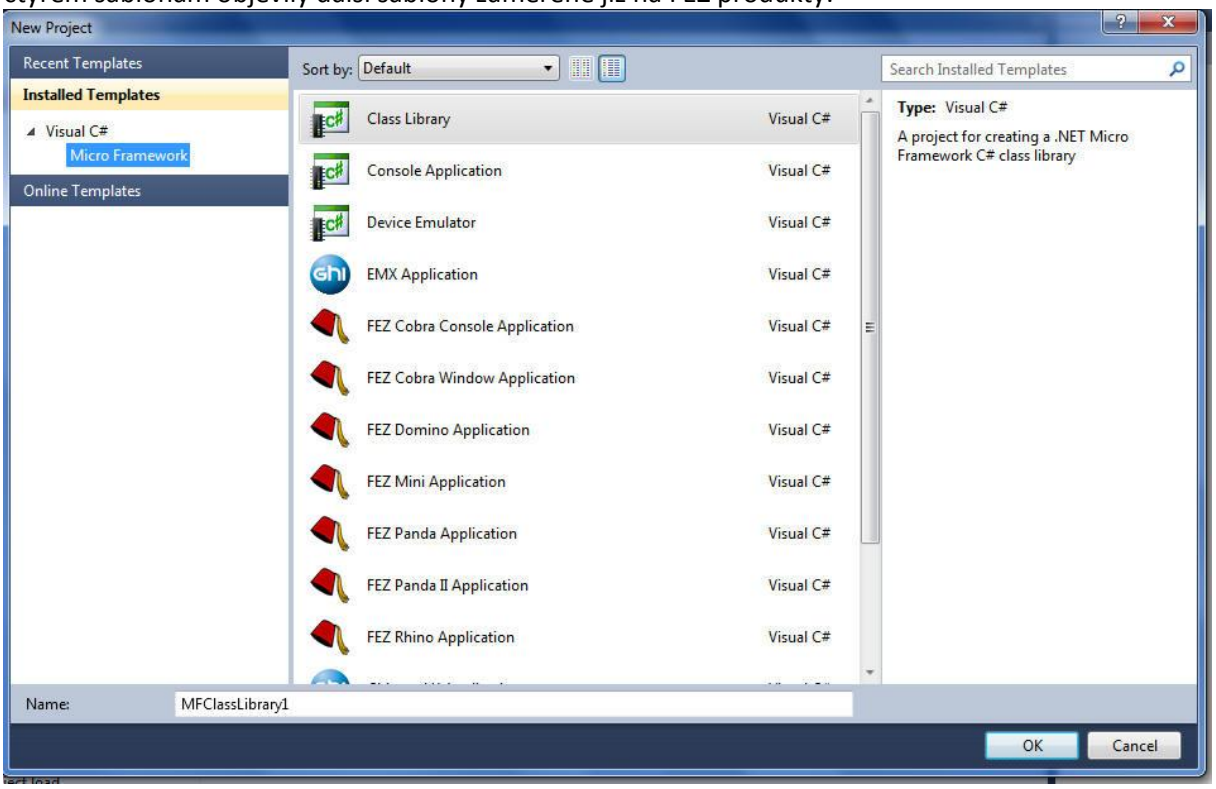
Probíhá instalace SDK



Nezbývá než opět kliknout na **Next**



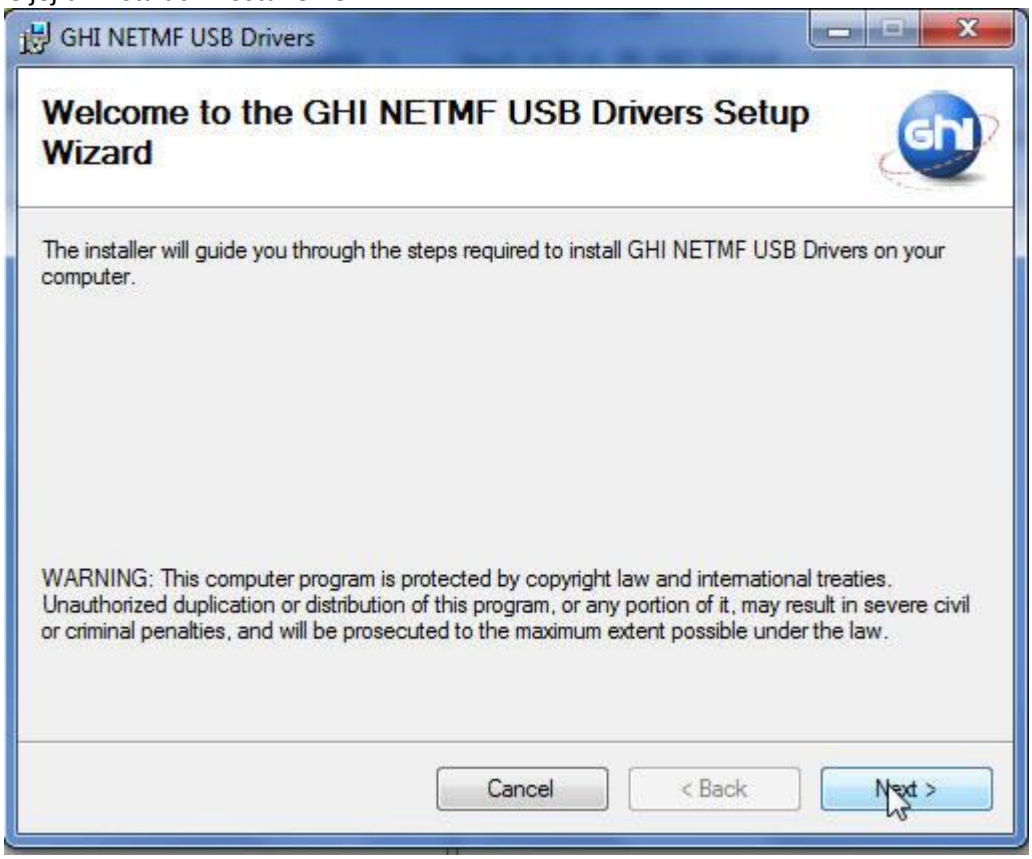
Instalace SDK byla úspěšná a tak nakonec klikneme na **Close**. Ve *Visual C# 2010* se k původním čtyřem šablonám objevily další šablony zaměřené již na FEZ produkty:



Instalace USB ovladače

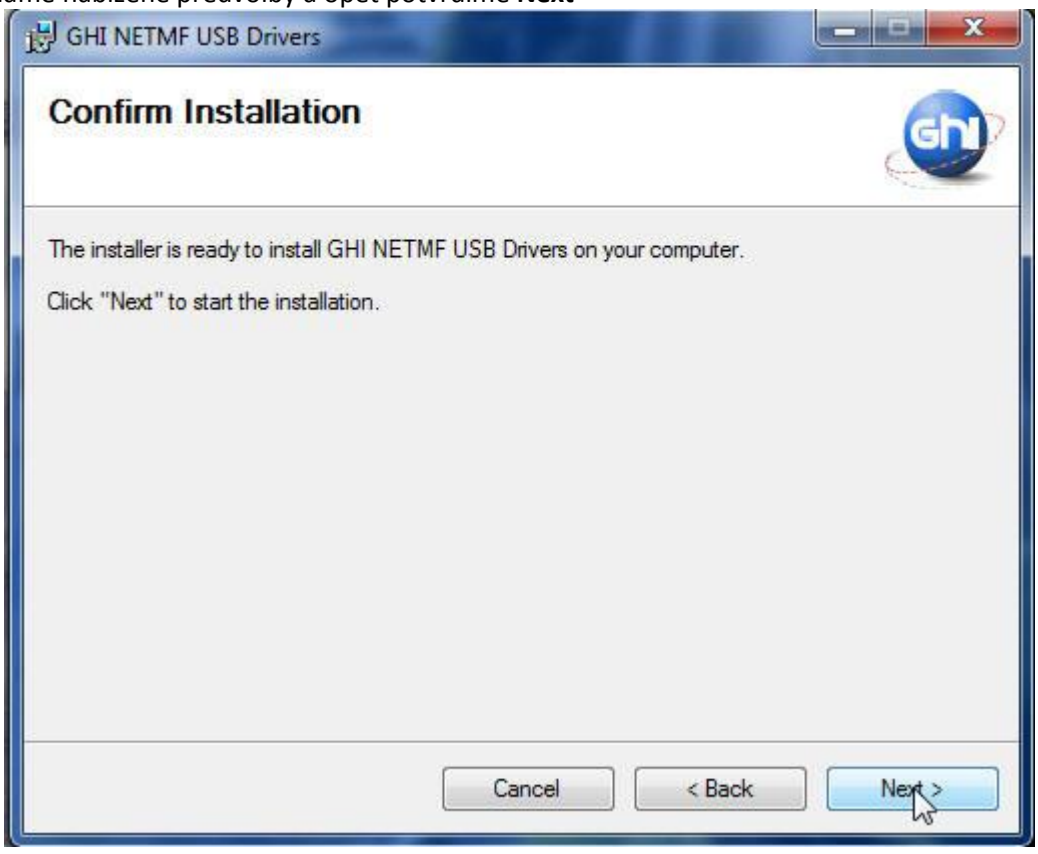
Můžeme již vytvářet *.NET Micro Framework* aplikace pro *FEZ Cerduino*, ale po jejich vytvoření je potřebujeme nějak dostat z PC do jednočipového ARM řídícího počítače robota. K tomu slouží propojení robota a PC pomocí USB kabelu. Je proto ještě potřeba nainstalovat USB ovladače. Opět je stáhneme ze stránek GHI Electronics nebo z DVD přílohy.

Spustíme jejich instalaci. Dostaneme

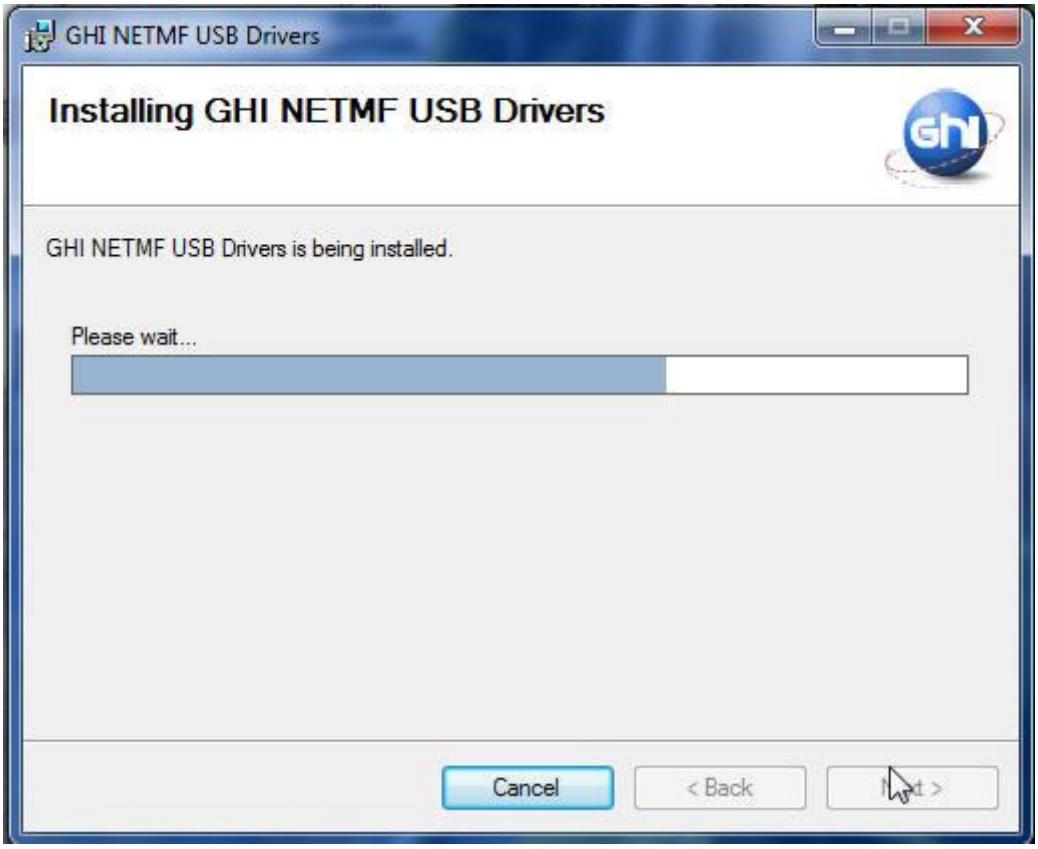


Potvrdíme **Next**

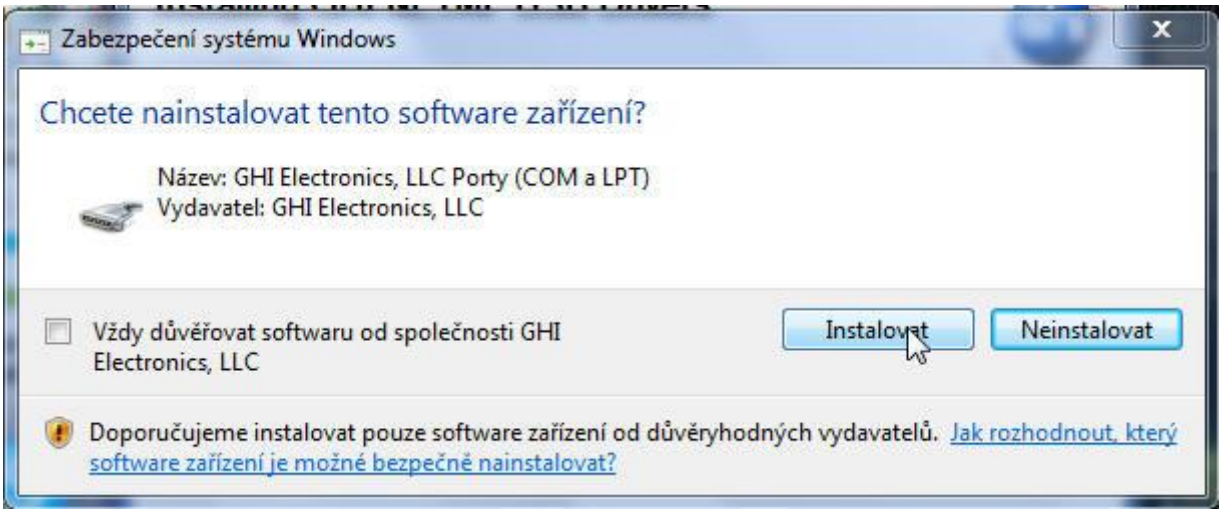
Ponecháme nabízené předvolby a opět potvrdíme **Next**



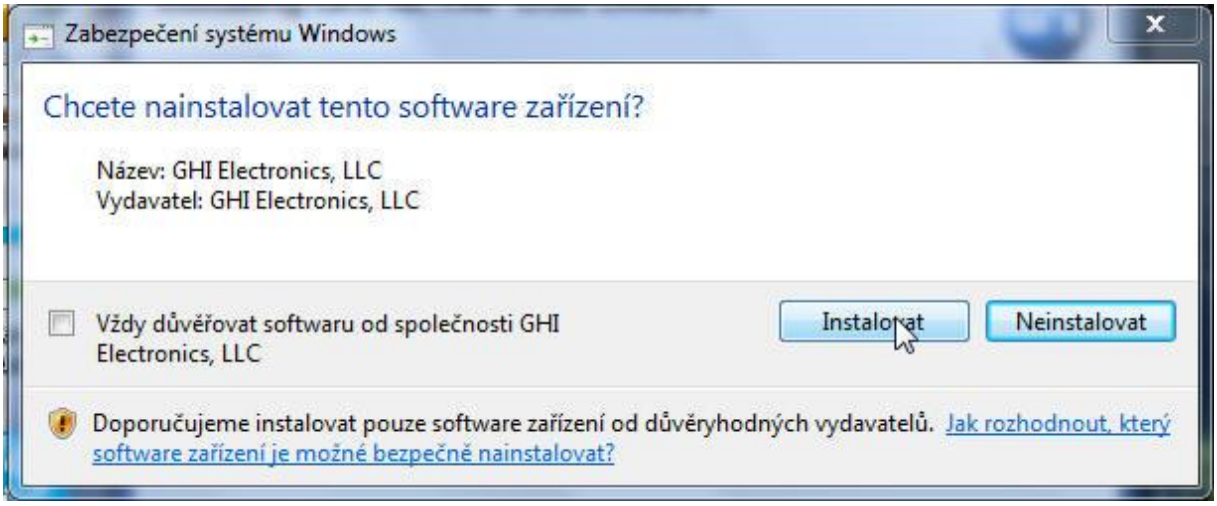
Kliknutím na **Next** již spustíme vlastní instalaci:



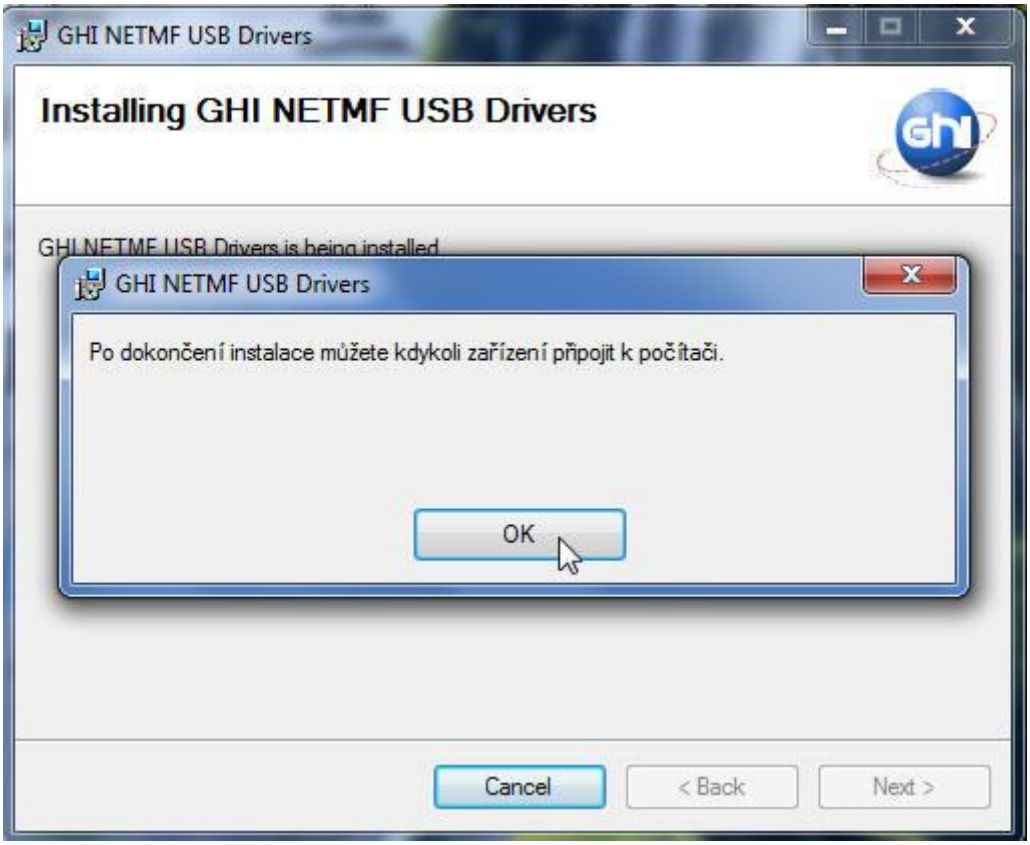
Probíhí instace USB driver, poté jsme dotázáni na instalaci dalších zařízení.



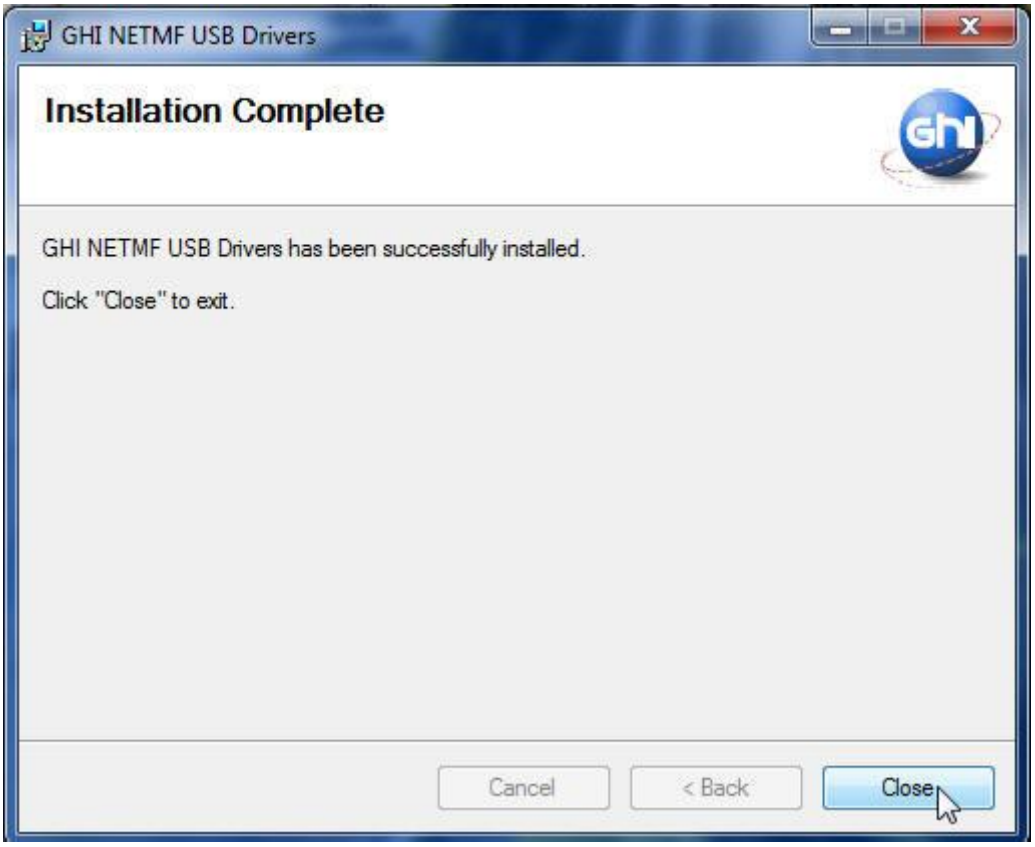
Instalaci spustíme tlačítkem **Instalovat**.



Obdobně potvrdíme tlačítkem **Instalovat** instalaci dalšího zařízení



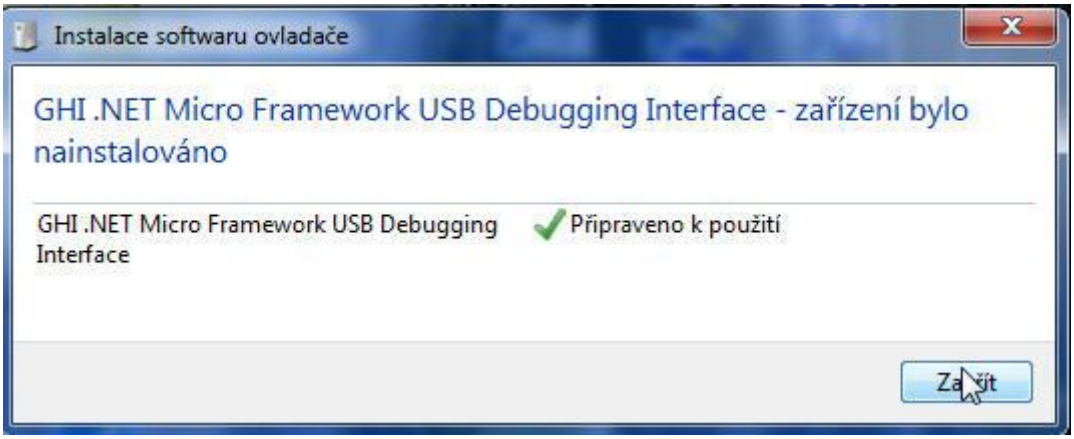
Tlačítkem **OK** potvrdíme informační okno

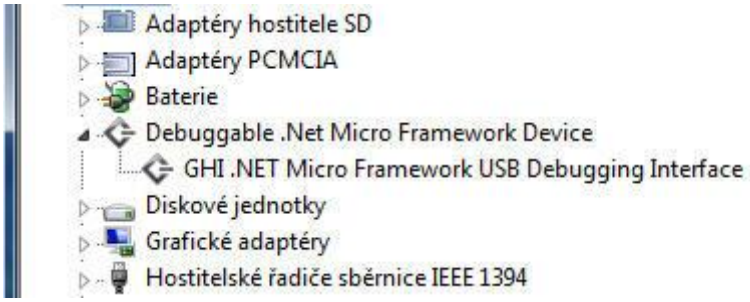


A tlačítkem **Close** potvrdíme ukončení instalace.

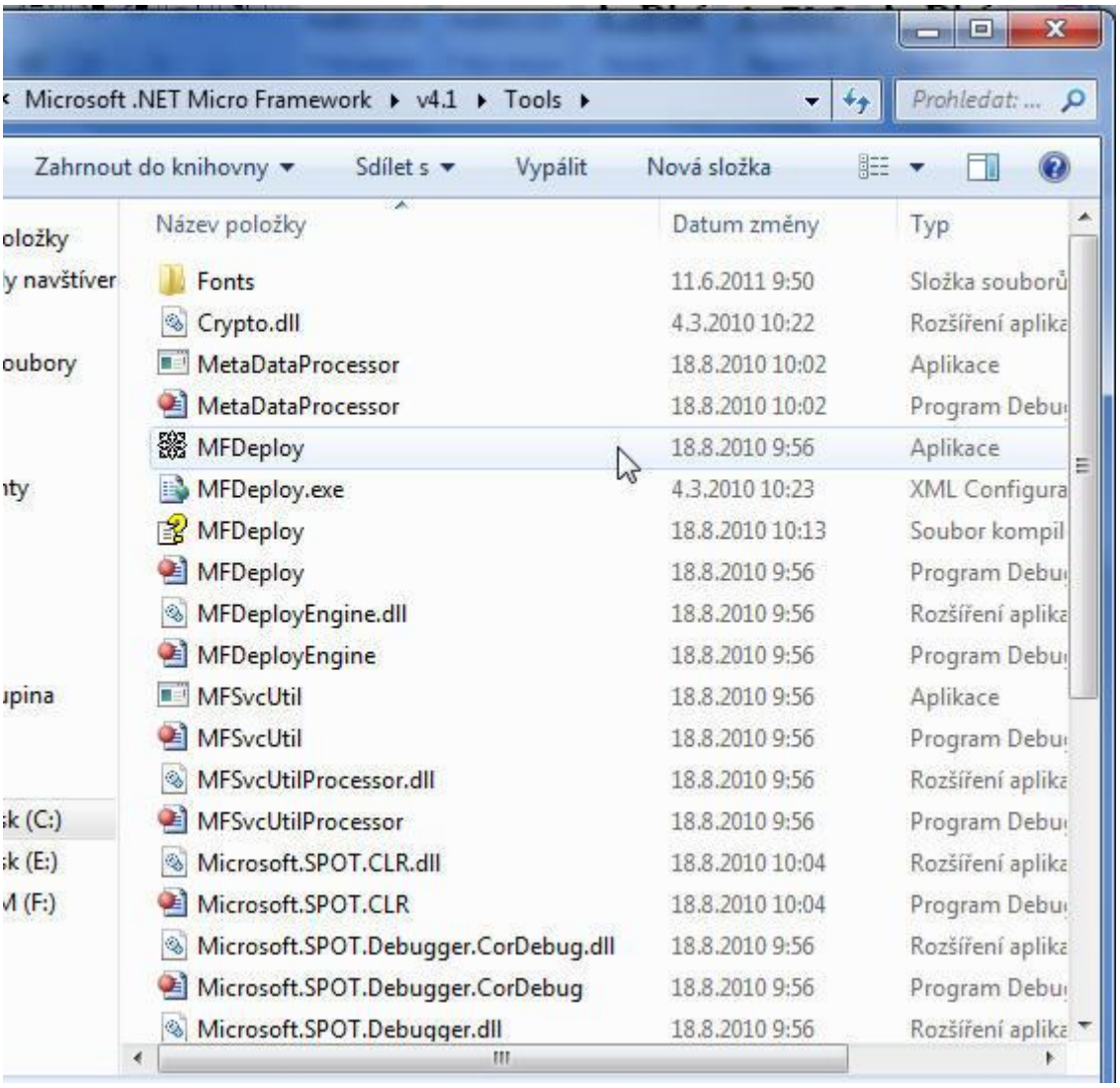
2.5 Zjištění verze firmware

Přes USB připojíme *FEZ Cerduino*. Protože jej připojujeme poprvé, nainstaluje se ovladač

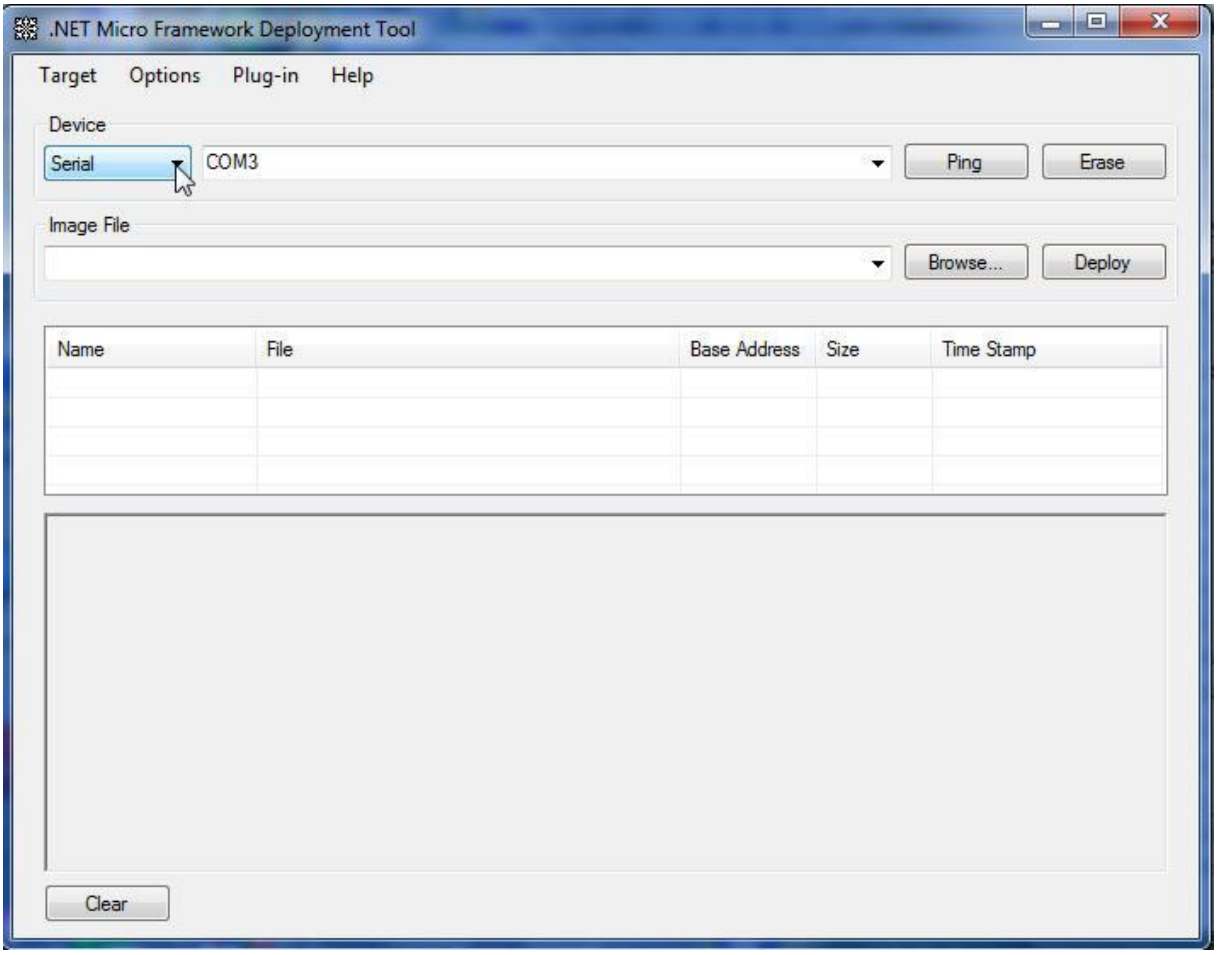




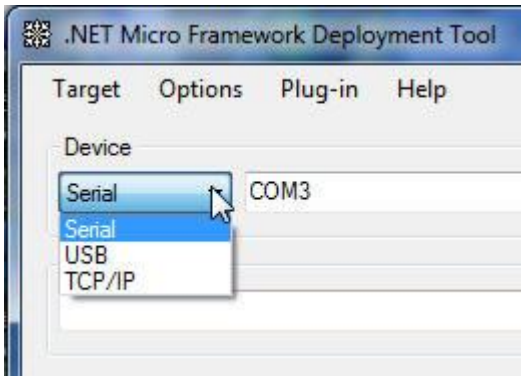
V správci hardware našeho PC vidíme, že je opravdu nainstalované. Dále ve windows uvidíme v **Start → Všechny Programy** i položku *Microsoft .NET Micro Framework 4.1*

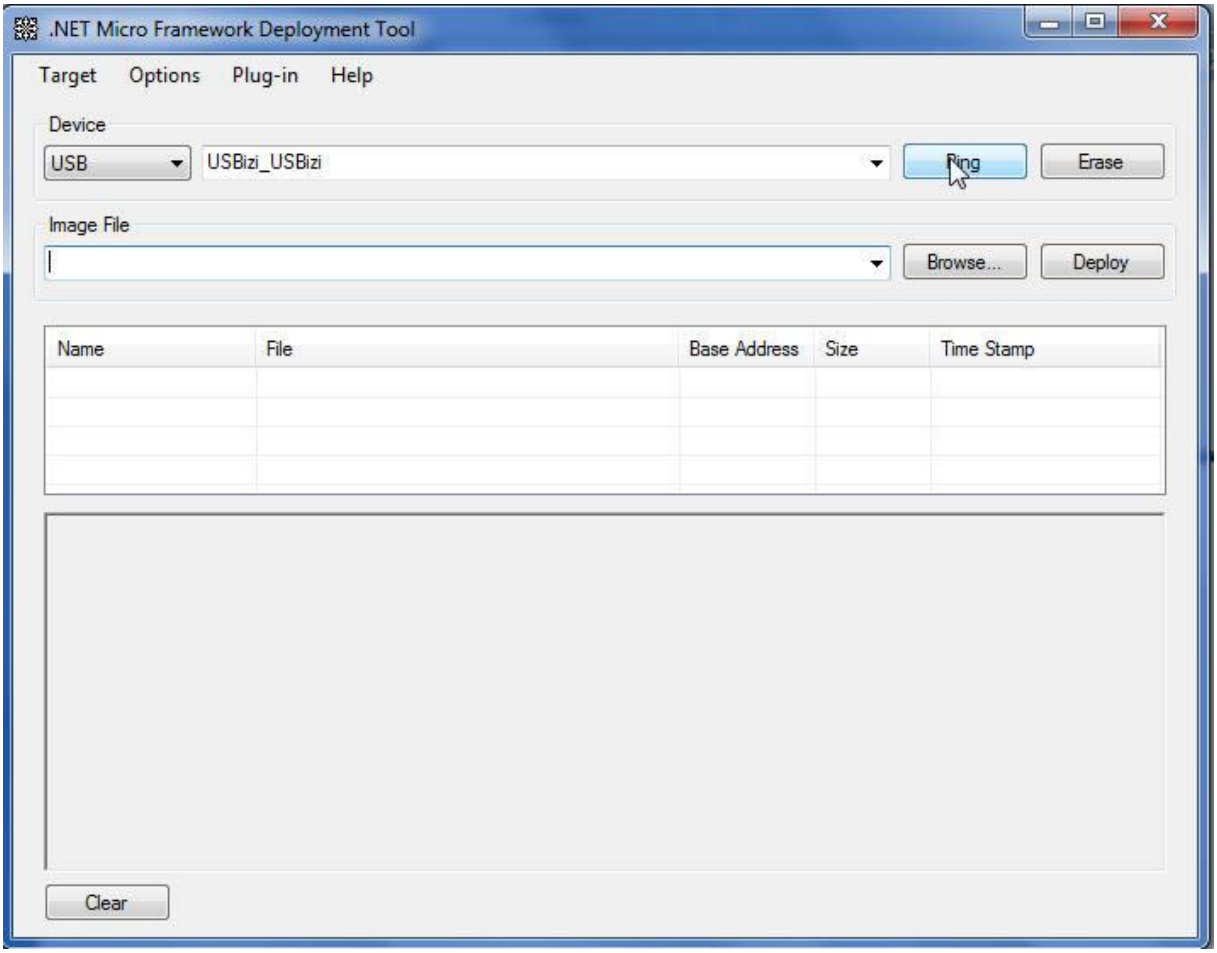


Ve složce **Tools** najdeme **MFDeploy tool**, který spustíme:

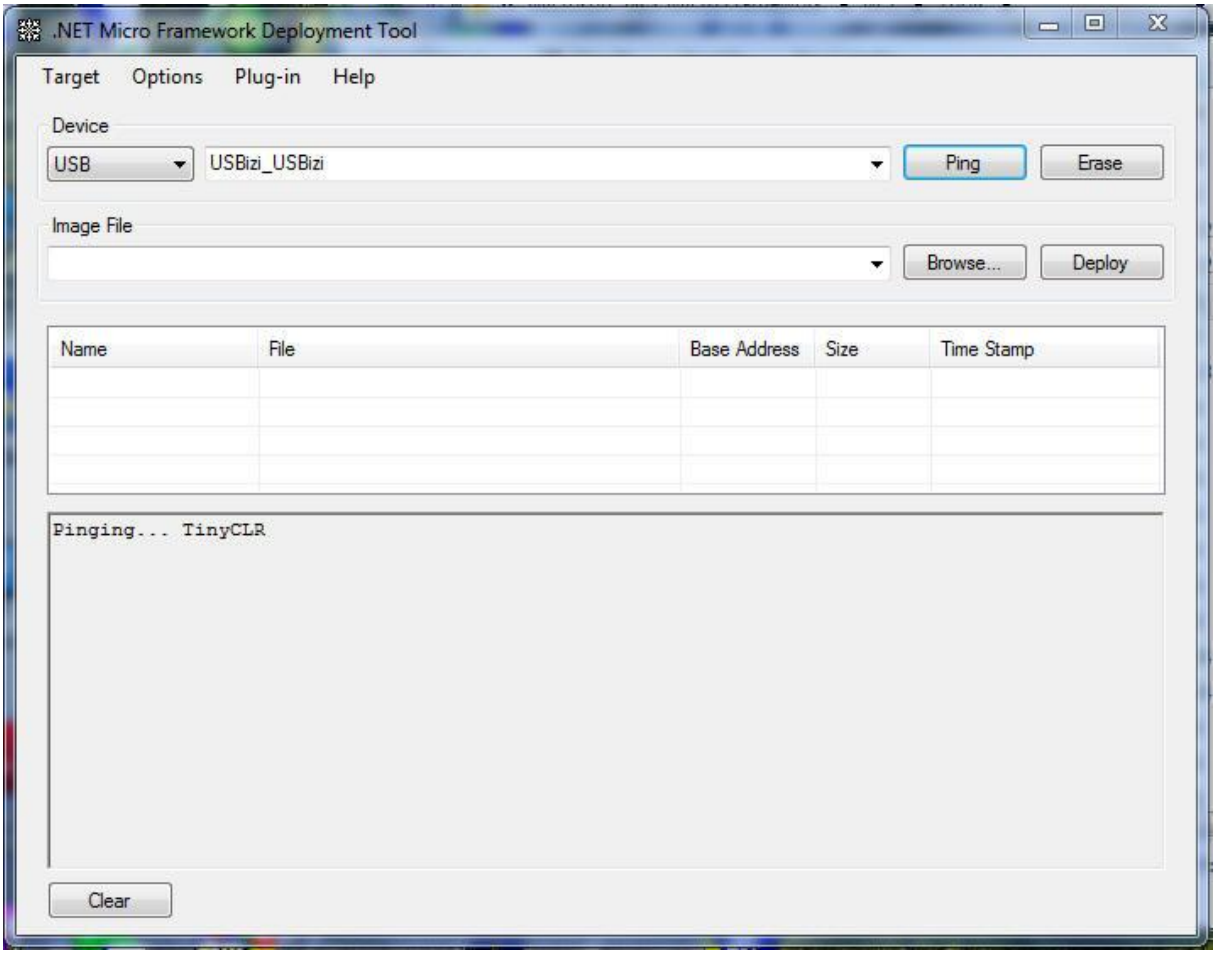


Vybereme sériový (virtuální) port, ke kterému máme připojený *FEZ Cerduino*

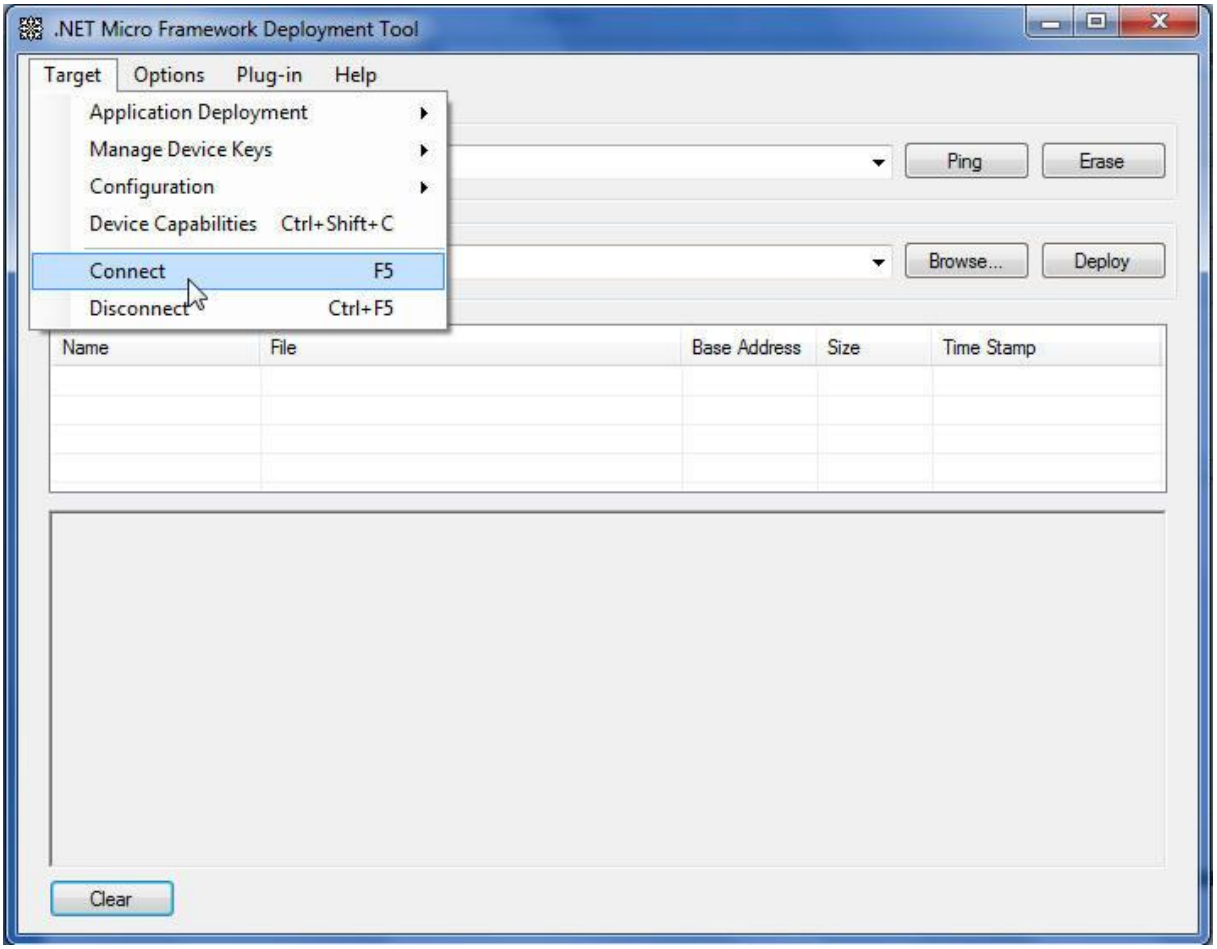




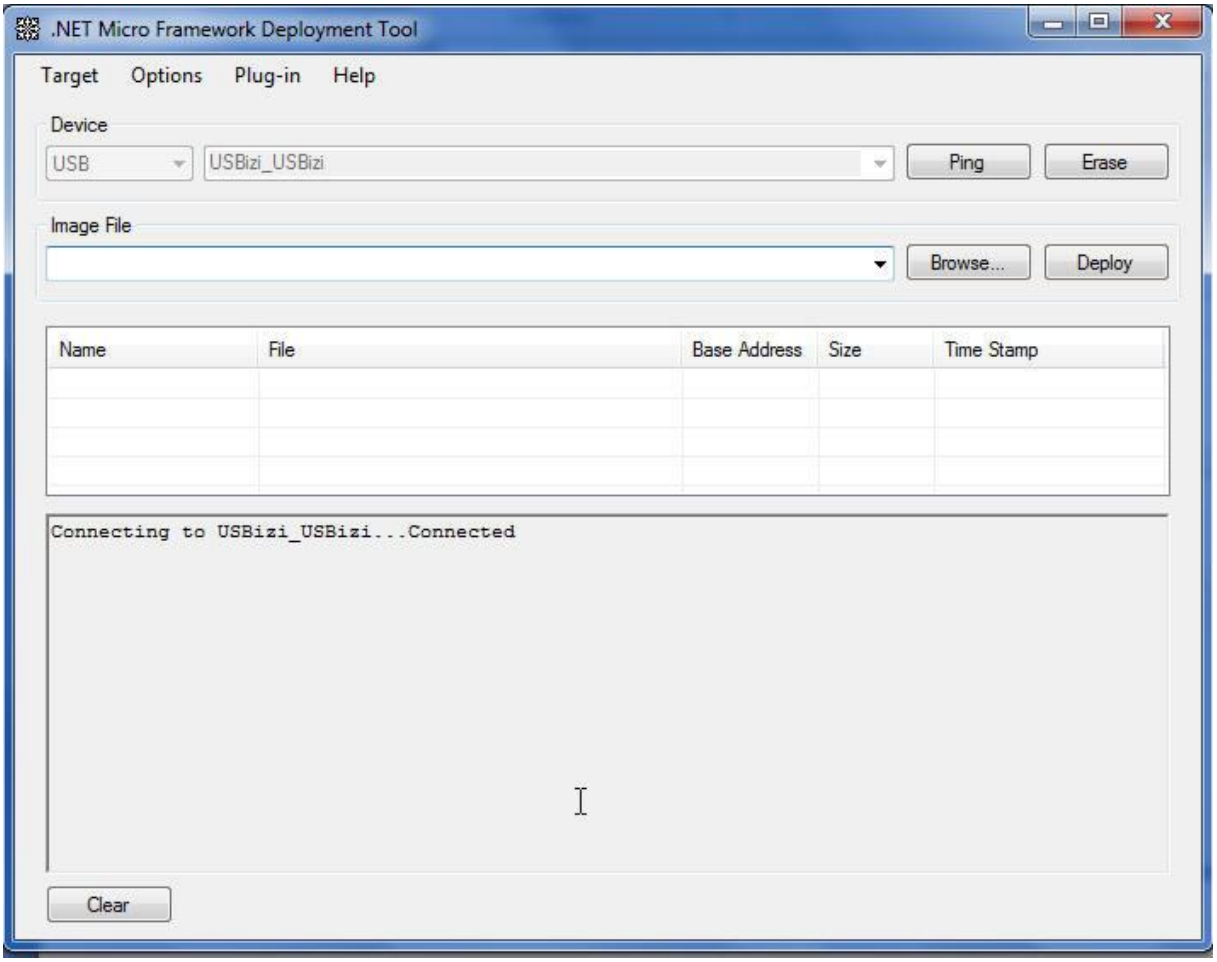
Klikneme na tlačítko **Ping**



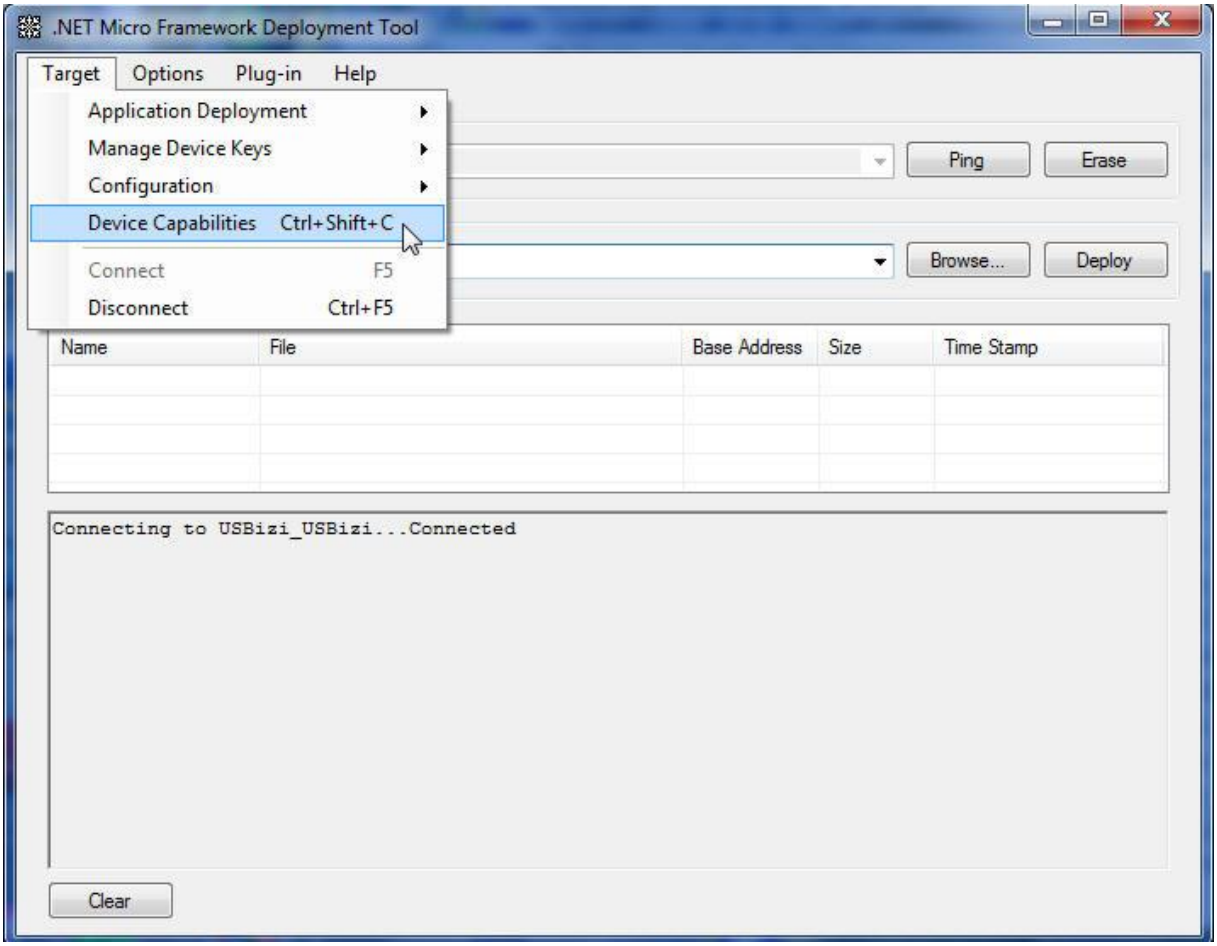
Tím se ověřuje propojení s *FEZ Cerduino*



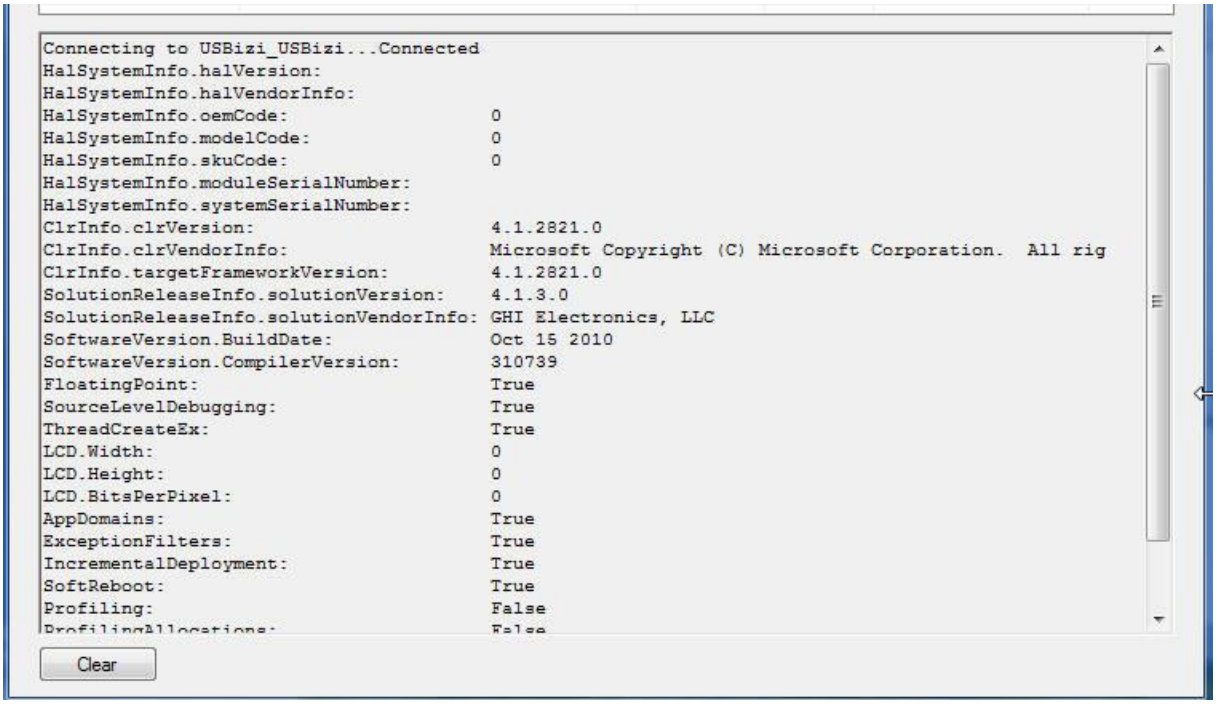
Nyní v menu vybereme **Target →Connect**



Provádí se navázání spojení. Podařilo se (*Connected*)



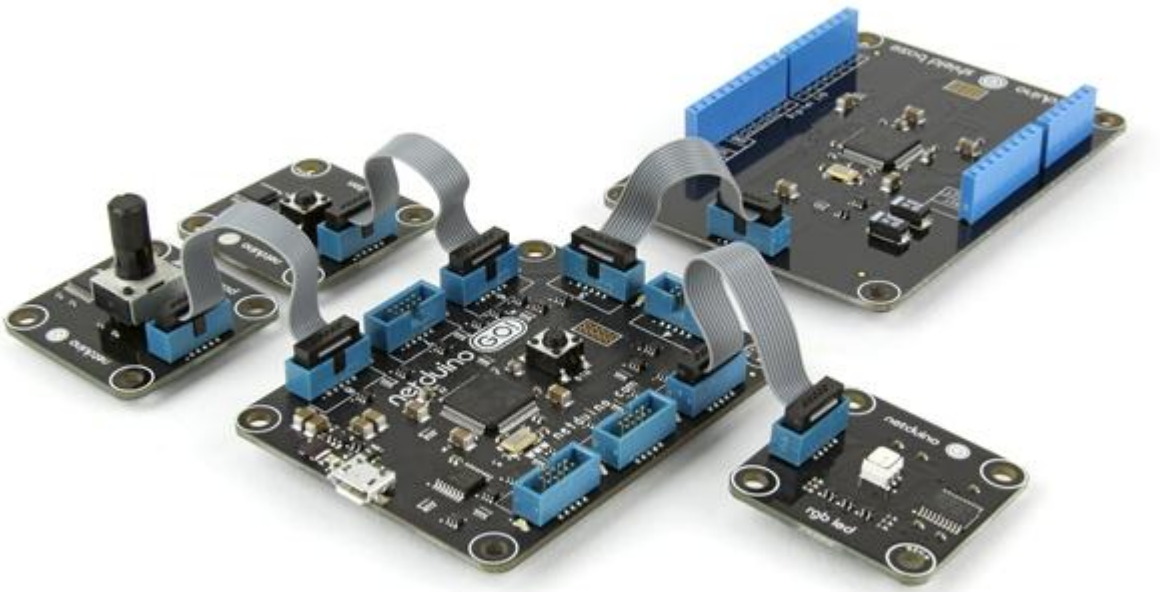
Dále v menu vybereme **Target →Device Capabilities**



Vidíme tedy, že náš FEZ_Cerduino s *.NET Micro Framework 4.1* a dále s *GHI NETMF v.4.1.3.*
Je nutné, abychom na svém PC prováděli vývoj sw právě s touto verzí.

2.1.5.1.3 Netduino

Výčet vlastností startkitu Netduino GO s *STM32F405* najdeme na :
http://netduino.com/netduinogo/specs.htm Pro ilustraci si jen ukážeme obrázky:



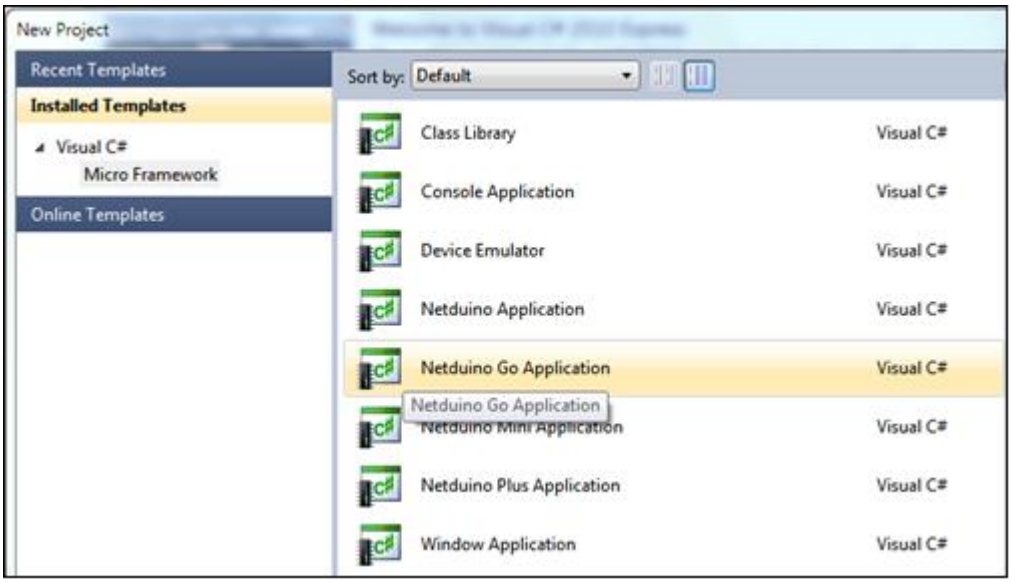
instalace a nastavení Netduina

Potřebujeme mít tyto komponenty:

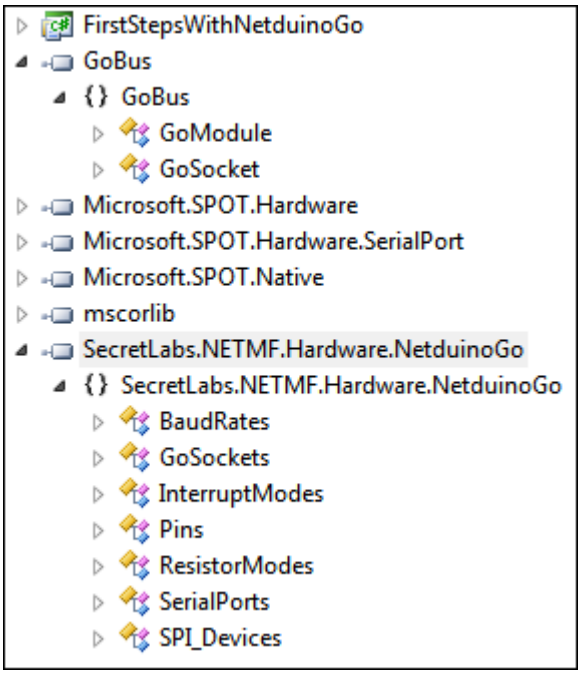
- Netduino Go či kompatibilní
- potentiometer module

- RGB LED module
- Visual Studio (C# Express stačí)
- .NET Micro Framework 4.2
- Netduino SDK pro 64-bit nebo 32-bit v závislosti na naší verzi Windows

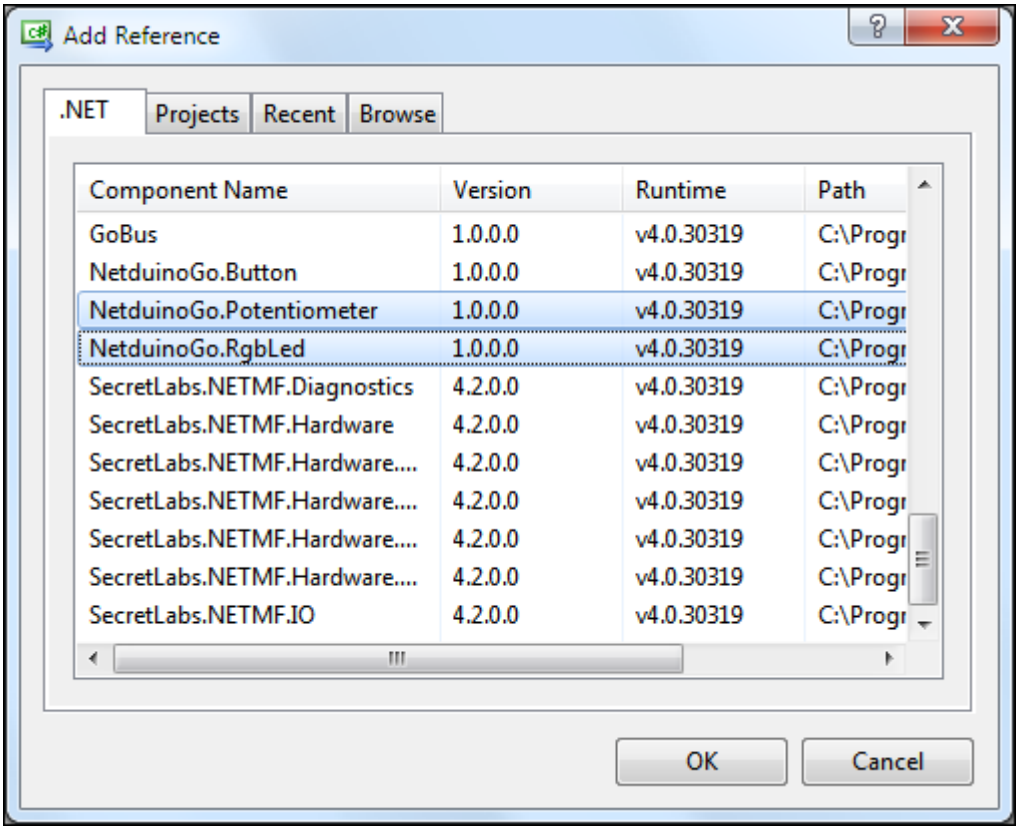
Nejprve vytvoříme nový projekt. Vybereme **Netduino Go Application** :



To vytvoří nový projekt spolu s referencemi na assemblies řídící *.NET Micro Framework*, **Go Bus** a porty *Netduina*:



Ještě přidáme podporu pro dva moduly, které chceme použít a to the **RGB LED** a **Potentiometer**. Provedeme to kliknutém pravým tlačítkem na **References** a přidáním dvou assemblies které potřebujeme :



Poté již můžeme vytvořit objekty representující naše moduly:

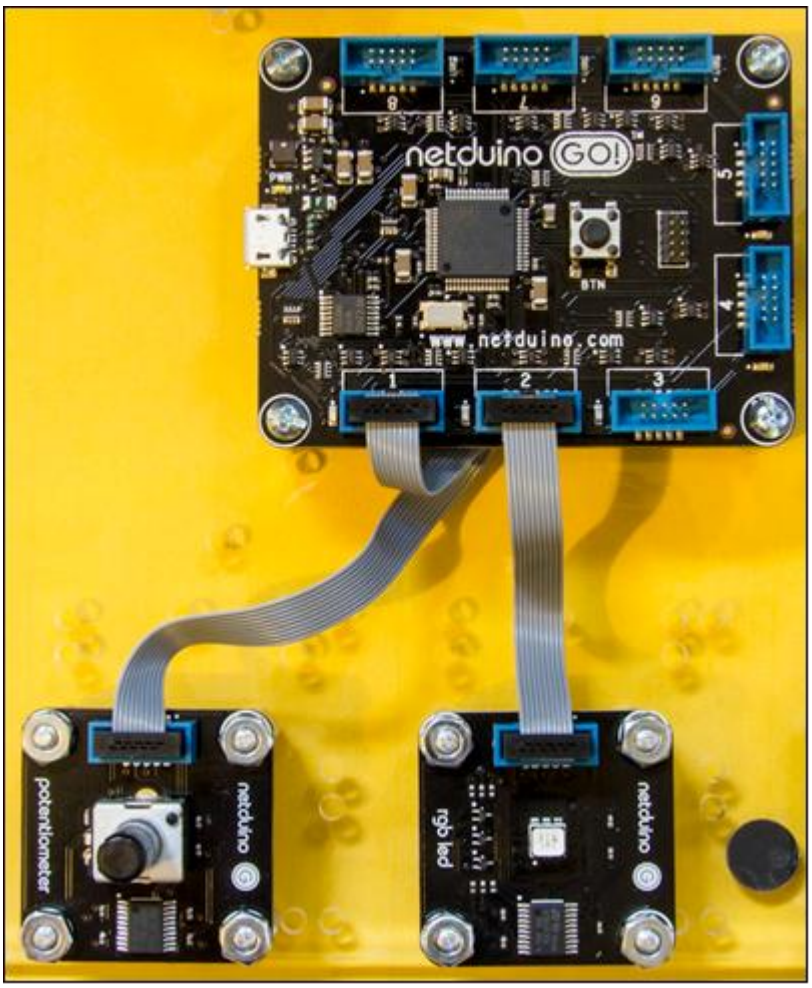
```
using System;
using NetduinoGo;

namespace FirstStepsWithNetduinoGo {
    public class Program {
        public static void Main() {
            var led = new RgbLed();
            var pot = new Potentiometer();
        }
    }
}
```

Tento kód pracuje bez toho, že bychom určovali port protože *Netduino Go* tyto informace již má. Každý konstruktor si tuto informaci bere z *GoSockets enumerace*, takže můžeme napsat:

```
using System;
using SecretLabs.NETMF.Hardware.NetduinoGo;
using NetduinoGo;

namespace FirstStepsWithNetduinoGo {
    public class Program {
        public static void Main() {
```

```
            var led = new RgbLed(GoSockets.Socket2);
            var pot = new Potentiometer(GoSockets.Socket1);
        }
    }
}
```

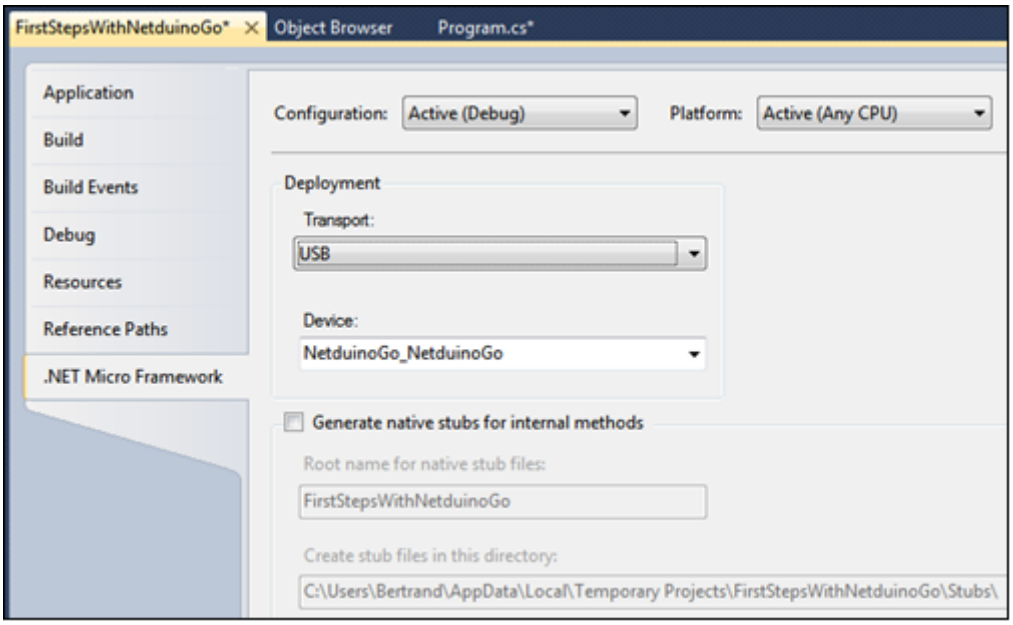
Nyní již máme objekty reprezentující naše moduly a můžeme je začít používat. Ukážeme si to na rozvícení LEDky. Hardware přitom sestavíme podle obrázku:



Mějme kód:

```
led.SetColor(255, 212, 42);
```

Abychom mohli aplikovat tento kód , musíme se nejprve ujistit že ve vlastnostech projektu je nastaveno **deploy on USB:**



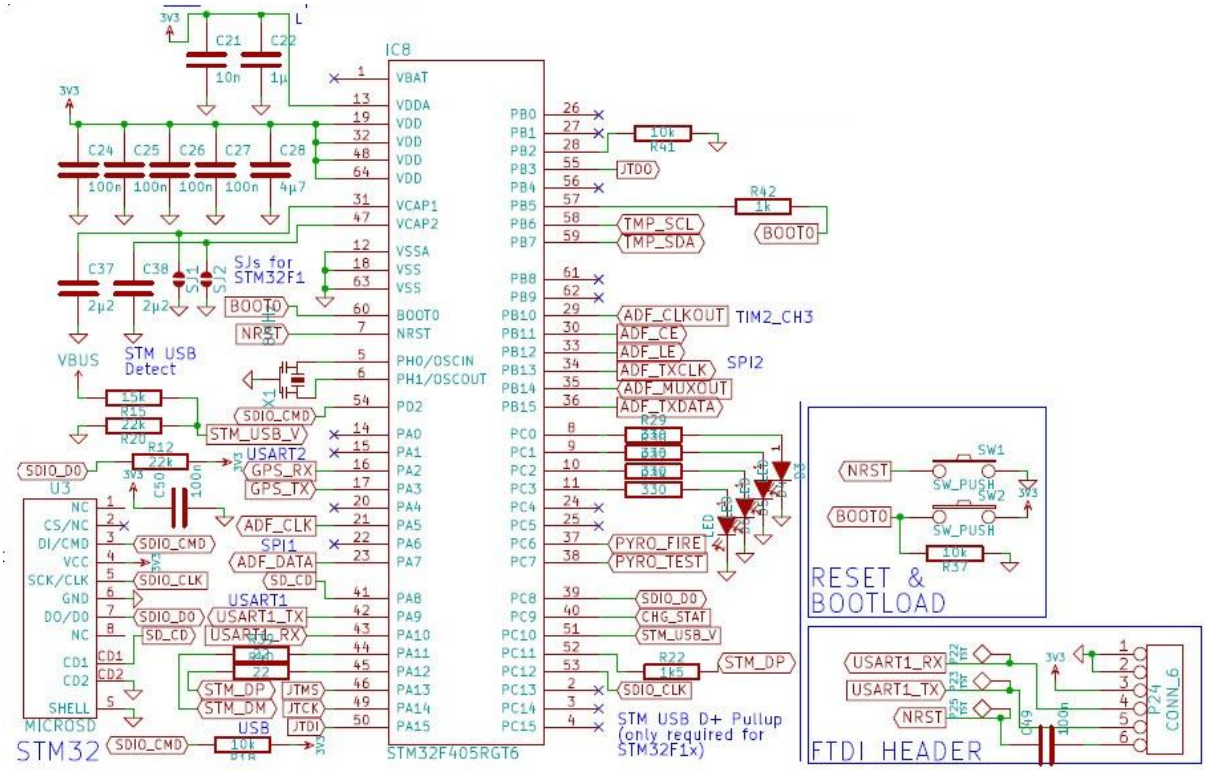
Pokud tomu tak je, můžeme stisknout **F5** a vidíme, že LED se rozsvítí…

Dále můžeme např. doplnit kód tak, že potenciometrem ovládáme jas LEDky:

```
var led = new RgbLed();
var pot = new Potentiometer();
int red = 255, green = 212, blue = 42;
while (true) {
    var intensity = pot.GetValue();
    led.SetColor(
        (byte)(red * intensity),
        (byte)(green * intensity),
        (byte)(blue * intensity));
}
```

2.1.5.2 Wombat

Uvedeme si jen schema. Projekt si podrobněji popíšeme v 4.1.2.



2.2 palubní počítač s ARM Cortex počítači řady STM32L

Nejprve si vysvětlíme, proč používat řadu STM32L. To L totiž znamená **low**, neboli obvod je optimalizován na co nejmenší spotřebu, což je vlastnost velice žádaná právě u satelitů.
Vlastnosti řady STM32L velmi stručně

- ARM Cortex™-M3 s taktem až do 32 MHz
- Energeticky ekonomický mód Run: spotřeba pod 230 µA/MHz při běhu programu z Flash
- 6 ultra-low-power módů: spotřeba až pod 270 nA ve standby režimu
- Napájecí napětí: 1.65 to 3.6 V
- Velmi nízká spotřeba potřebná pro výpočetní výkon: pod 185 µA/DMIPS
- Paměť od 32 do 384 kB Flash, až 48 kB SRAM a až 12 kB EEPROM
- Paměť Flash a EEPROM je s ECC!
- Bohatá kombinace periférií jak analogových, tak digitálních
- Pinově kompatibilní s existující rodinou STM32F

Co je ale pro nás zajímavé, je rozdíl proti verzi čipů pro obecné použití, např. STM32F100. Takže v první řadě tam naleznete USB řadič - tudíž se se svým kitem budete moci vyzkoušet i USB připojení. Dále je tam LCD řadič, tedy mcu má přímo zaintegrováno vícefázové řízení nízkopříkonového LCD displeje (řadič obsahují varianty L152 a L162). Jako jako další novinku naleznete v blokovém přehledu dva analogové komparátory. Navíc je v blokovém schématu ještě uvedena položka 3x op-amps, tj STM32L obsahuje i operační zesilovače. STM32L také na rozdíl od vyšších verzí STM32F neobsahuje CAN interface, neboť primárně není určen pro automobilový průmysl. A konečně řada STM32L16x obsahuje navrch jednotku HW šifrování pomocí AES algoritmu.

Co je ale podstatné je , že nejen některé bloky ve struktuře přibyly, ale jiné jsou přepracované a vylepšené.
Změny uvnitř struktury:

- A/D převodník je vylepšený a má jiný interface
- jednotka reálného času (RTC) je kompletně předělaná a už zahrnuje registry pro dny,měsíce,roky,hodiny, minuty - tj. už se to nemusí tak složitě přepočítávat
- Flash paměť se programuje jinak (důvodem byla úspora energie). Pro nás to nic neznamená, ST-Link funguje stále stejně, jen pokud budete chtít přímo běžícím programem zapisovat něco do flash, tak je jiný interface. Výhoda také je, že STM32L obsahuje EEPROM paměť, tedy nemusíme si jí simulovat ve Flash paměti,
- bohužel i GPIO má jiný interface, tedy budete muset změnit trochu své programy,
- RCC - jednotka hodinového taktu má nový zdroj hodin (MSI). Jde o RC oscilátor s přepínatelnými hodnotami, jde tedy zvolit několik různých frekvencí.,
- řízení spotřeby je přepracované - což je celkem pochopitelné.

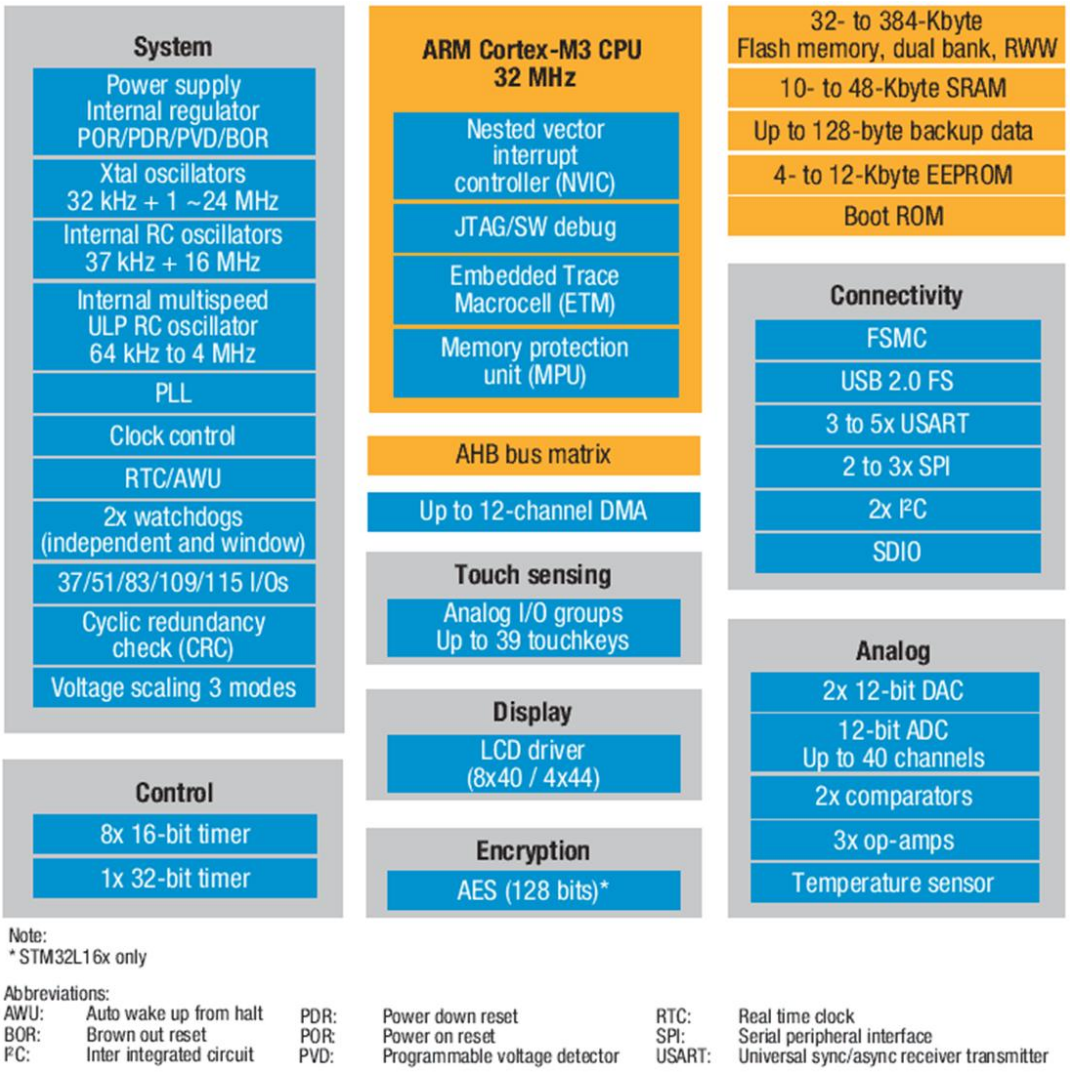
A je zde i plná pinová kompatibilita s čipy pro obecné použití, kromě Vbat, který u řady STM32F1xx slouží k zálohování chodu mcu pomocí baterie (což u řady L ztrácí význam, neboť ta je přednostně určena pro bateriové napájení) je daný pin využit jako vývod napětí pro LCD displej generovaný vnitřním kapacitním násobičem.

Snížená spotřeba

Firma ST píše o "Ultra Low Power" ale co to v praxi znamená? Čip sám je vyroben dražší technologií s nižšími svodovými proudy, tedy například při stejných podmínkách s čipem STM32F103 (program běží z flash, jádro je na 32 MHz, periférie jsou zapnuty, atd. - tj. maximální spotřeba) má L varianta cca třetinovou spotřebu. Tento fakt je sice zamaskován v datasheetu, ale můžete si dát tu práci a přepočítat si to. Proč to není napsáno natvrdo? Protože ST se chlubí mnohem nižšími spotřebami, kterých jde dosáhnout pomocí různých příkon šetřících metod. Oproti běžným F mcu umí L varianta odpojovat různé periférie nebo je používat úspornějším způsobem, pracovat na nižších frekvencích, pracovat s nižším napájecím napětím (až do 1,65 V), atd. Zkrátka šetření na mnoho způsobů je propracováno do detailů a pak je možné uvádět údaje skutečně pozoruhodné:

- běh z Flash, napájení jádra v módu Range 3 - spotřeba 230 uA/MHz
- běh z SRAM, napájení jádra v módu Range 3 - spotřeba 186 uA/MHz
- běh na 32 kHz - celková spotřeba 9uA
- Stop mód s běžícím RTC - 1.3 uA

Vnitřní strukturu STM32L blokově ukazuje následující obrázek:



Vzhledem k pinové kompatibilitě s obvody pro obecné použití STM32F můžeme pro konstrukci palubních počítačů CANSATu s STM32L použít zapojení a PCB uvedené v předchozí kapitole 2.1. Proto si v této kapitole ukážeme jen tvorbu software s STM32L. Přitom můžeme s výhodou použít startkit *STM32L-Discovery*

Vlastnosti:

- *STM32L152RBT6* microcontroller mající 128 KB Flash, 16 KB RAM, 4 KB EEPROM, v pouzdře LQFP64
- Interface ST-Link/V2 spolu s přepínači a špičkami umožňujícími použít kit jako programátor externích MCU ST-Link/V2 (s SWD spojením pro programování a debugging)
- napájení : prostřednictvím USB nebo z externího napájecího zdroje 3.3 nebo 5 V
- měření proudu IDD
- LCD
  - patice DIP28
  - 24 segmentú,

- čtyři LEDs:
  - LD1 (červená) pro USB komunikaci

- o LD2 (červená) pro indikaci 3.3 V napájení
- o dvě uživatelské LEDky, LD3 (zelená) and LD4 (modrá)

- dvě tlačítka (user a reset)
- jeden touch sensing slider a čtyři dotyková tlačítka (touch keys)
- Extension konektor pro LQFP64 I/Os pro rychlé propojení s prototypovou deskou a snadné testování

STM32L Discovery kit obsahuje LCD displej (. 6 segmentů a jeden indikátor spotřeby).

Segmenty displeje

Pod LCD displejem se nachází piny se zkratovací propojkou - po rozpojení můžete multimetrem přesně změřit spotřebu mcu ve vaší aplikaci.

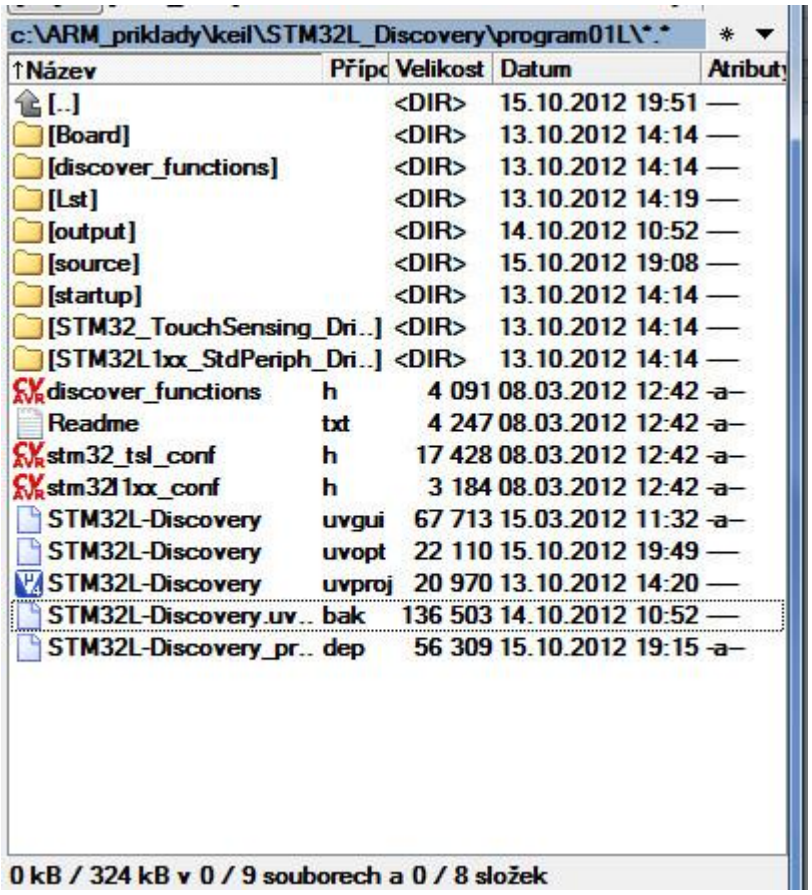
2.2.1 Začínáme programovat STM32L

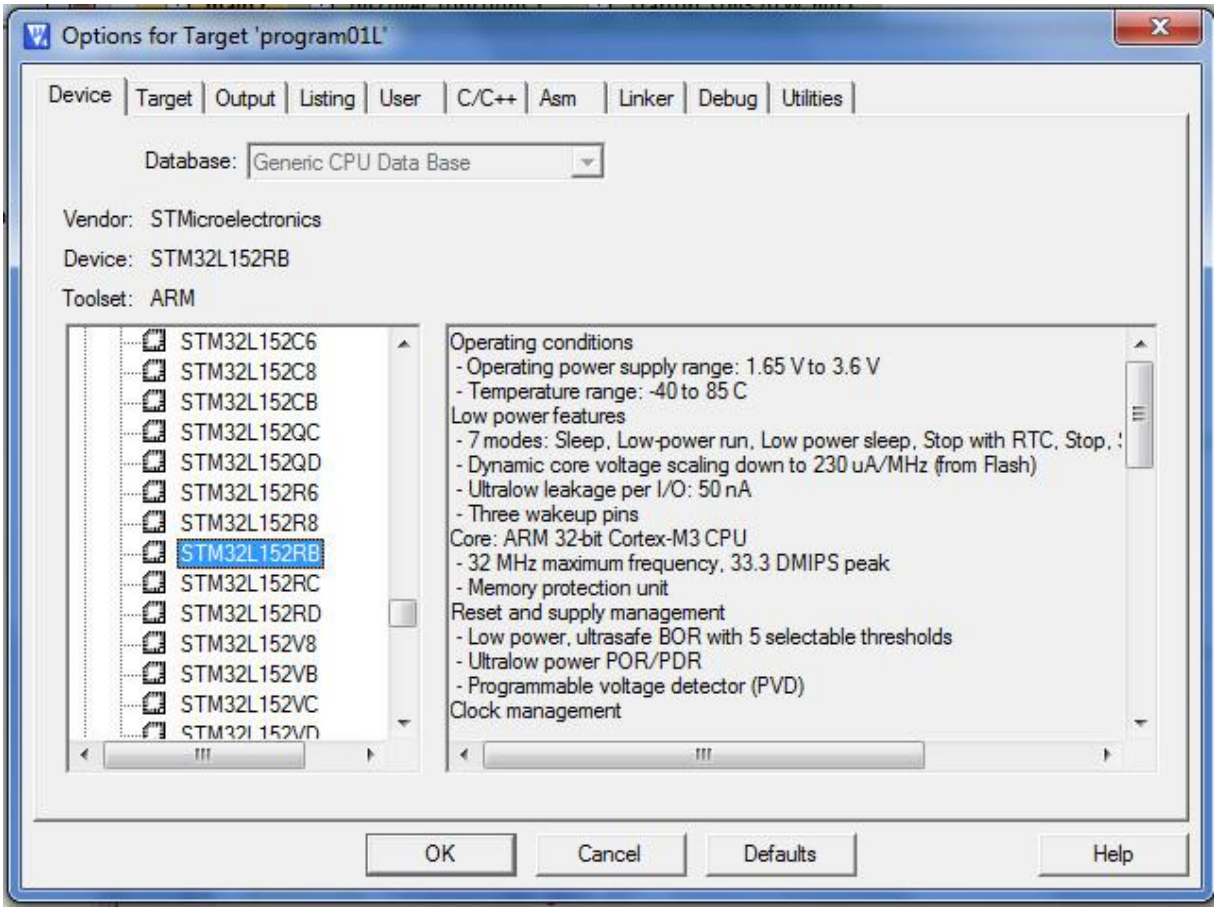
Nejlepším způsobem, jak začít s nějakou novou technologií je použít startkit, který právě pro tento účel výrobce poskytuje. Ukážeme si proto několik jednoduchých prográmků právě na výše uvedeném startkitu *STM32L-Discovery*. Ostatně stejně jsme postupovali při osvojování si základní řady STM32F1xx pomocí startkitu STM32VL-Discovery v druhém dílu skript [1.2].
Použijeme i stejné vývojové prostředí uVision Keil4 a proto teď můžeme být stručnější.

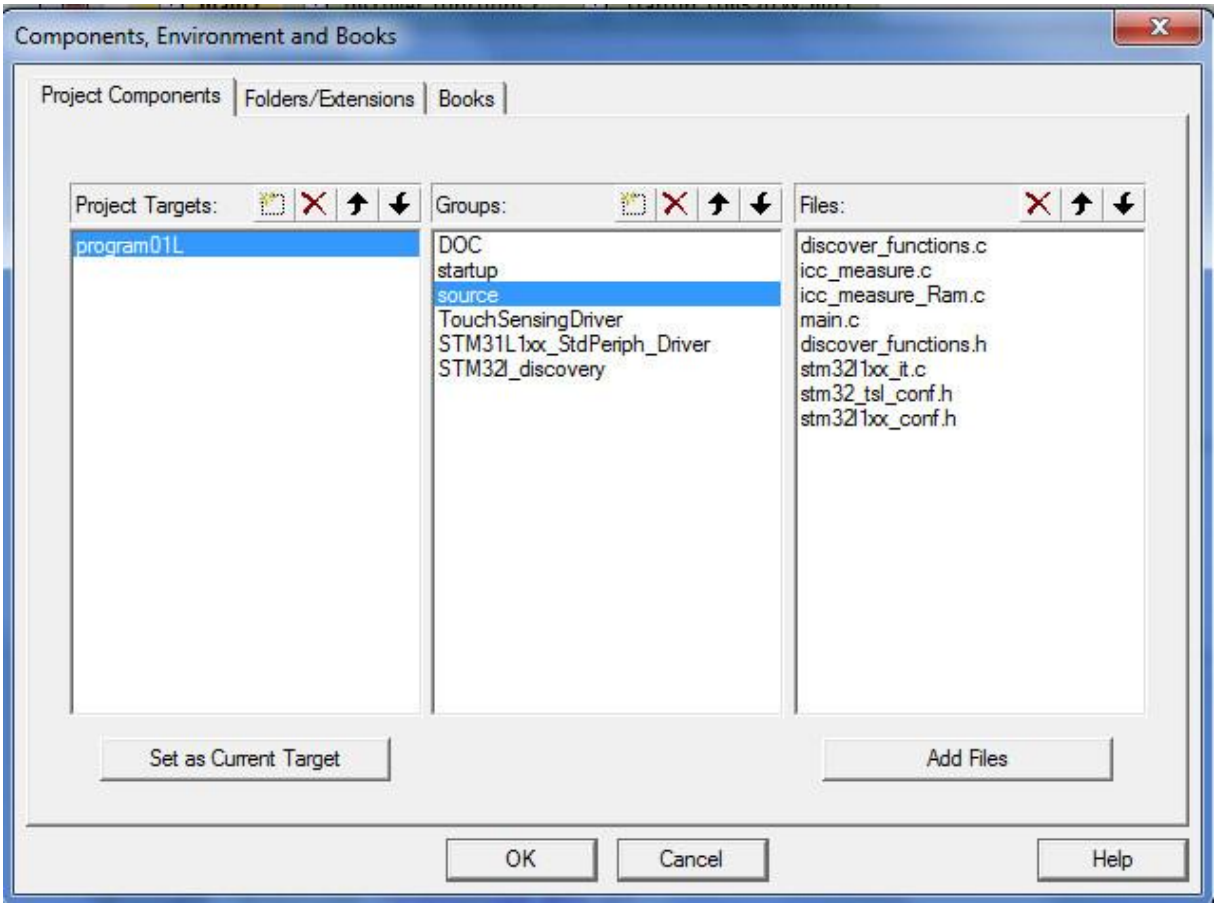
2.2.1.1 První program s STM32L-Discovery

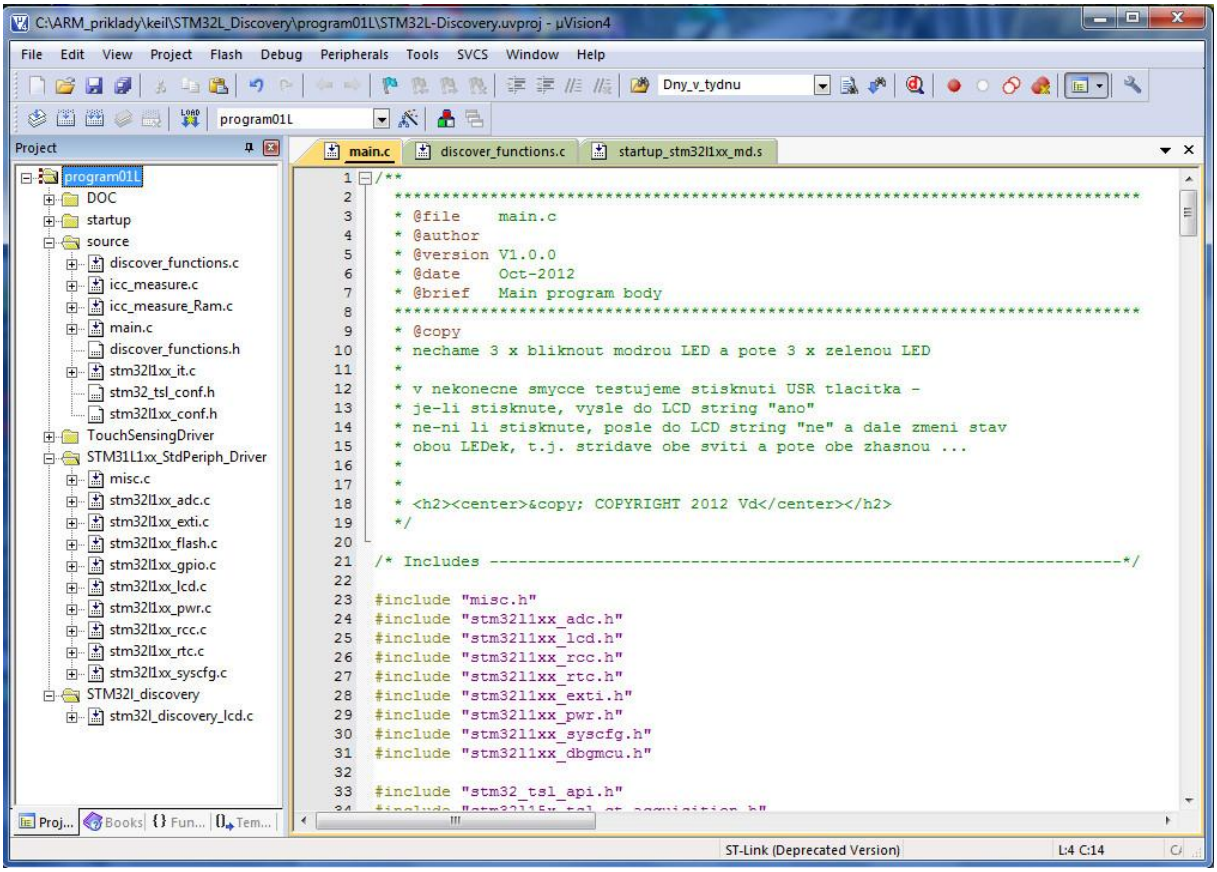
Startkit *STM32L-Discovery* obsahuje stejně jako *STM32VL-Discovery* dvě LED diody (modrou a zelenou) a uživatelské tlačítko USR. Navíc má ale i LCD displej. V prvním programu si ukážeme, jak bude vypadat kód využívající tyto periferie. Další novou periferií jsou i slider tlačítka, jejichž použití si

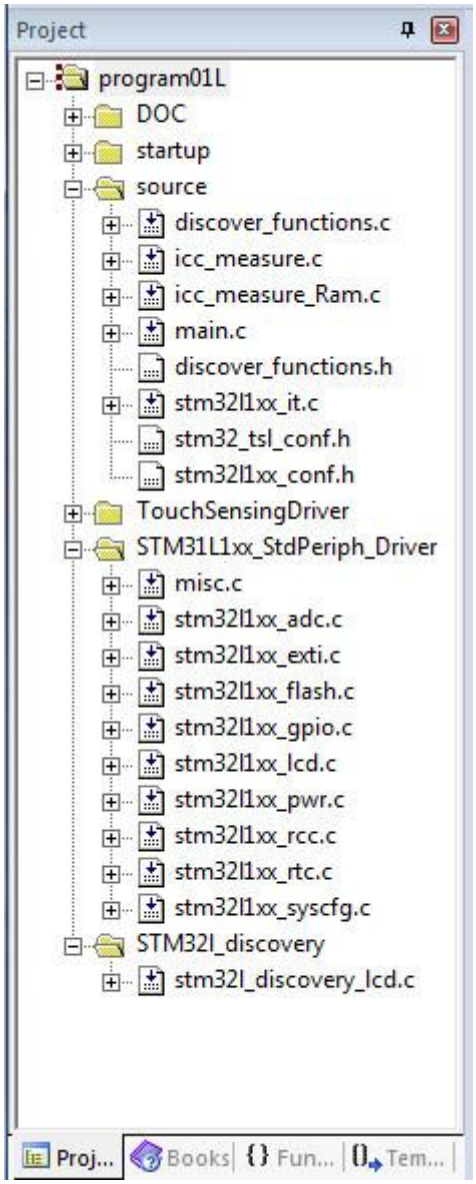
ukážeme v programech 3 a 4. Protože jsme se programováním v IDE Keil seznámili v 2. díle skript, můžeme být velice struční:




Options for Target 'program01L'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Database: Generic CPU Data Base
Vendor: STMicroelectronics
Device: STM32L152RB
Toolset: ARM
STM32L152C6
STM32L152C8
STM32L152CB
STM32L152QC
STM32L152QD
STM32L152R6
STM32L152R8
STM32L152RB
STM32L152RC
STM32L152RD
STM32L152V8
STM32L152VB
STM32L152VC
Operating conditions
- Operating power supply range: 1.65 V to 3.6 V
- Temperature range: -40 to 85 C
Low power features
- 7 modes: Sleep, Low-power run, Low power sleep, Stop with RTC, Stop,
- Dynamic core voltage scaling down to 230 uA/MHz (from Flash)
- Ultralow leakage per I/O: 50 nA
- Three wakeup pins
Core: ARM 32-bit Cortex-M3 CPU
- 32 MHz maximum frequency, 33.3 DMIPS peak
- Memory protection unit
Reset and supply management
- Low power, ultrasafe BOR with 5 selectable thresholds
- Ultralow power POR/PDR
- Programmable voltage detector (PVD)
Clock management
OK
Cancel
Defaults
Help


Components, Environment and Books
Project Components
Folders/Extensions
Books
Project Targets:
program01L
Groups:
DOC
startup
source
TouchSensingDriver
STM31L1xx_StdPeriph_Driver
STM32l_discovery
Files:
discover_functions.c
icc_measure.c
icc_measure_Ram.c
main.c
discover_functions.h
stm32l1xx_it.c
stm32_tsl_conf.h
stm32l1xx_conf.h
Set as Current Target
Add Files
OK
Cancel
Help


C:\ARM_priklady\keil\STM32L_Discovery\program01L\STM32L-Discovery.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Dny_v_tydnu
program01L
Project
program01L
DOC
startup
source
discover_functions.c
icc_measure.c
icc_measure_Ram.c
main.c
discover_functions.h
stm32l1xx_it.c
stm32_tsl_conf.h
stm32l1xx_conf.h
TouchSensingDriver
STM31L1xx_StdPeriph_Driver
misc.c
stm32l1xx_adc.c
stm32l1xx_exti.c
stm32l1xx_flash.c
stm32l1xx_gpio.c
stm32l1xx_lcd.c
stm32l1xx_pwr.c
stm32l1xx_rcc.c
stm32l1xx_rtc.c
stm32l1xx_syscfg.c
STM32l_discovery
stm32l_discovery_lcd.c
main.c
discover_functions.c
startup_stm32l1xx_md.s
1 /**
2 **
3 * @file main.c
4 * @author
5 * @version V1.0.0
6 * @date Oct-2012
7 * @brief Main program body
8 **
9 * @copy
10 * nechame 3 x bliknout modrou LED a pote 3 x zelenou LED
11 *
12 * v nekonecne smycce testujeme stisknuti USR tlacitka -
13 * je-li stisknute, vysle do LCD string "ano"
14 * ne-ni li stisknute, posle do LCD string "ne" a dale zmeni stav
15 * obou LEDek, t.j. stridave obe sviti a pote obe zhasnou ...
16 *
17 *
18 * <h2> © COPYRIGHT 2012 Vd </h2>
19 */
20
21 /* Includes --*/
22
23 #include "misc.h"
24 #include "stm32l1xx_adc.h"
25 #include "stm32l1xx_lcd.h"
26 #include "stm32l1xx_rcc.h"
27 #include "stm32l1xx_rtc.h"
28 #include "stm32l1xx_exti.h"
29 #include "stm32l1xx_pwr.h"
30 #include "stm32l1xx_syscfg.h"
31 #include "stm32l1xx_dbgmcu.h"
32
33 #include "stm32_tsl_api.h"
Proj... Books Fun... Tem...
ST-Link (Deprecated Version)
L:4 C:14




```
/**

******************************************************************************
***
  * @file    main.c
  * @author
  * @version V1.0.0
  * @date    Oct-2012
  * @brief   Main program body

******************************************************************************
***
  * @copy
  * nechame 3 x bliknout modrou LED a pote 3 x zelenou LED
  *
  * v nekonecne smycce testujeme stisknuti USR tlacitka -
```

```
  * je-li stisknute, vysle do LCD string "ano"
  * ne-ni li stisknute, posle do LCD string "ne" a dale zmeni stav
  * obou LEDek, t.j. stridave obe sviti a pote obe zhasnou ...
  *
  *
  * <h2><center>&copy; COPYRIGHT 2012 Vd</center></h2>
  */

/* Includes ------------------------------------------------------------------*/

#include "misc.h"
#include "stm32l1xx_adc.h"
#include "stm32l1xx_lcd.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_rtc.h"
#include "stm32l1xx_exti.h"
#include "stm32l1xx_pwr.h"
#include "stm32l1xx_syscfg.h"
#include "stm32l1xx_dbgmcu.h"

#include "stm32_tsl_api.h"
#include "stm32l15x_tsl_ct_acquisition.h"

#include "discover_board.h"
#include "icc_measure.h"
#include "discover_functions.h"
#include "stm32l_discovery_lcd.h"
#include "stm32_tsl_timebase.h"

static volatile uint32_t TimingDelay;
//extern unsigned char Bias_Current;
/* LCD bar graph: used for displaying active function */
//extern uint8_t t_bar[2];
/* Auto_test activation flag: set by interrupt handler if user button is
pressed for a few seconds */
//extern bool Auto_test;
//extern bool Idd_WakeUP;


/* Machine status used by main() wich indicats the active function, set by
user button in interrupt handler */
uint8_t state_machine;

uint16_t Int_CurrentSTBY;
uint32_t BTN_Get(void); //vlastni funkce testujici USR tlacitko

/*******************************************************************************
*****/
/**
  * @brief main entry point.
  * @par Parameters None
  * @retval void None
  * @par Required preconditions: None
  */

int main(void)
{
```

```
  Int_CurrentSTBY = Current_Measurement();



  PWR_PVDCmd(DISABLE);

  RCC_Configuration();

  PWR_VoltageScalingConfig(PWR_VoltageScaling_Range1);

  /* Wait Until the Voltage Regulator is ready */
  while (PWR_GetFlagStatus(PWR_FLAG_VOS) != RESET) ;

  /* Init I/O ports */
  Init_GPIOs ();

  /* Initializes ADC */
  ADC_Icc_Init();

  enableInterrupts();

  /* Warning ! in TSL Init the sysTick interrupt is setted to:
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 2000 ---> 500 µs*/

  /* Init Touch Sensing configuration */
  TSL_Init();

  sMCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sMCKeyInfo[0].Setting.b.ENABLED = 1;
  sMCKeyInfo[0].DxSGroup = 0x00;

  /* Initializes the LCD glass */
  LCD_GLASS_Init();

  /* Welcome display  - ukazka pouziti LCD*/


   // LCD_GLASS_ScrollSentence("      ** SPSE Jecna - PSS
**",1,SCROLL_SPEED);
   LCD_GLASS_DisplayString("PSS  ");

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
```

```
 Delay(500);

//bliknuti zelene LED
      GPIO HIGH(LD PORT,LD GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);

 //bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);




  while (1)
  {

      Delay(1);



 //if ((GPIOA->IDR ) != 0x0)
if (BTN_Get() != 0x0)         //test stisku tlacitka
    {//je-li stisknute tlacitko, pise se na LCD ano
      LCD_GLASS_DisplayString("Ano    ");
    }
    else
   {  //neni-li stisknute tlacitko, zmeni se stavobou LED
      GPIO_TOGGLE(LD_PORT,LD_BLUE);
      GPIO_TOGGLE(LD_PORT,LD_GREEN);
             //neni-li stisknute tlacitko, pise se na LCD ne
                   LCD_GLASS_DisplayString("Ne   ");
    }

  }
}

/**
  * @brief  Configures the different system clocks.
  * @param  None
  * @retval None
  */
void RCC_Configuration(void)
{

  /* Enable HSI Clock */
  RCC_HSICmd(ENABLE);

  /*!< Wait till HSI is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET)
  {}
```

```
  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

  RCC MSIRangeConfig(RCC MSIRange 6);

  /* Enable the GPIOs Clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOB |
RCC_AHBPeriph_GPIOC| RCC_AHBPeriph_GPIOD| RCC_AHBPeriph_GPIOE|
RCC_AHBPeriph_GPIOH, ENABLE);

  /* Enable  comparator clock LCD and PWR mngt */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_COMP | RCC_APB1Periph_LCD |
RCC_APB1Periph_PWR,ENABLE);

  /* Enable ADC clock & SYSCFG */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_SYSCFG ,
ENABLE);

  /* Allow access to the RTC */
  PWR_RTCAccessCmd(ENABLE);

  /* Reset Backup Domain */
  RCC_RTCResetCmd(ENABLE);
  RCC_RTCResetCmd(DISABLE);

  /* LSE Enable */
  RCC_LSEConfig(RCC_LSE_ON);

  /* Wait till LSE is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
  {}

  RCC_RTCCLKCmd(ENABLE);

  /* LCD Clock Source Selection */
  RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

  RCC_HSEConfig(RCC_HSE_OFF);

  if(RCC_GetFlagStatus(RCC_FLAG_HSERDY) != RESET )
  {
    while(1);
  }

}

/**
  * @brief  To initialize the I/O ports
  * @caller main
  * @param None
  * @retval None
  */
void  Init_GPIOs (void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* USER button and WakeUP button init: GPIO set in input interrupt active
mode */
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;
```

```
  /* Configure User Button pin as input */
  GPIO_InitStructure.GPIO_Pin = USER_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
  GPIO_Init(BUTTON_GPIO_PORT, &GPIO_InitStructure);

  /* Connect Button EXTI Line to Button GPIO Pin */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);

  /* Configure User Button and IDD_WakeUP EXTI line */
  EXTI_InitStructure.EXTI_Line = EXTI_Line0 ;  // PA0 for User button AND
IDD_WakeUP
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set User Button and IDD_WakeUP EXTI Interrupt to the lowest
priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn ;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure);

/* Configure the GPIO_LED pins  LD3 & LD4*/
  GPIO_InitStructure.GPIO_Pin = LD_GREEN|LD_BLUE;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(LD_PORT, &GPIO_InitStructure);
  GPIO_LOW(LD_PORT,LD_GREEN);
  GPIO_LOW(LD_PORT,LD_BLUE);

/* Counter enable: GPIO set in output for enable the counter */
  GPIO_InitStructure.GPIO_Pin = CTN_CNTEN_GPIO_PIN;
  GPIO_Init( CTN_GPIO_PORT, &GPIO_InitStructure);

/* To prepare to start counter */
  GPIO_HIGH(CTN_GPIO_PORT,CTN_CNTEN_GPIO_PIN);

/* Configure Output for LCD */
/* Port A */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
GPIO_Pin_8 | GPIO_Pin_9 |GPIO_Pin_10 |GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOA, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource15,GPIO_AF_LCD) ;
```

```
/* Configure Output for LCD */
/* Port B */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO Pin 8 | GPIO Pin 9 \
                                | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12
| GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOB, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOB, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource4,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource11,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource12,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource13,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource14,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port C*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3 | GPIO_Pin_6 \
                                | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10 |GPIO_Pin_11 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOC, &GPIO_InitStructure);


  GPIO_PinAFConfig(GPIOC, GPIO_PinSource0,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11,GPIO_AF_LCD) ;

/* ADC input */
  GPIO_InitStructure.GPIO_Pin = IDD_MEASURE  ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_Init( IDD_MEASURE_PORT, &GPIO_InitStructure);

}



/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;
```

```
  while(TimingDelay != 0);

}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{

  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }

}

uint32_t BTN_Get(void) {

 return (GPIOA->IDR & (1UL << 0));
}



#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */

  /* Infinite loop */
  while (1)
  {
  }
}

#endif

/****************** (C)  *****END OF FILE****/
```

2.2.1.2 Scrolování textu zobrazovaného LCD displejem

Tento program vznikl úpravou předchozího programu, takže budeme struční. Uvedeme zdrojový kód souboru main.c

```
/**

******************************************************************************
***
  * @file    main.c
  * @author
  * @version V1.0.0
  * @date    Oct-2012
  * @brief   Main program body

******************************************************************************
***
  * @copy
  * nechame 3 x bliknout modrou LED a pote 3 x zelenou LED
  *
  * v nekonecne smycce testujeme stisknuti USR tlacitka -
  * je-li stisknute, vysle do LCD string "ano"
  * ne-ni li stisknute, posle do LCD string "ne" a dale zmeni stav
  * obou LEDek, t.j. stridave obe sviti a pote obe zhasnou ...
  *
  *
  * <h2><center>&copy; COPYRIGHT 2012 Vd</center></h2>
  */

/* Includes ------------------------------------------------------------------
---*/

#include "misc.h"
#include "stm32l1xx_adc.h"
#include "stm32l1xx_lcd.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_rtc.h"
#include "stm32l1xx_exti.h"
#include "stm32l1xx_pwr.h"
#include "stm32l1xx_syscfg.h"
#include "stm32l1xx_dbgmcu.h"

#include "stm32_tsl_api.h"
#include "stm32l15x_tsl_ct_acquisition.h"

#include "discover_board.h"
#include "icc_measure.h"
#include "discover_functions.h"
#include "stm32l_discovery_lcd.h"
#include "stm32_tsl_timebase.h"

static volatile uint32_t TimingDelay;
//extern unsigned char Bias_Current;
/* LCD bar graph: used for displaying active function */
//extern uint8_t t_bar[2];
/* Auto_test activation flag: set by interrupt handler if user button is
pressed for a few seconds */
//extern bool Auto_test;
//extern bool Idd_WakeUP;
```

```
/* Machine status used by main() wich indicats the active function, set by
user button in interrupt handler */
uint8_t state_machine;

uint16_t Int_CurrentSTBY;
uint32_t BTN_Get(void); //vlastni funkce testujici USR tlacitko

/*******************************************************************************
*****/
/**
  * @brief main entry point.
  * @par Parameters None
  * @retval void None
  * @par Required preconditions: None
  */

int main(void)
{




  Int_CurrentSTBY = Current_Measurement();




  PWR_PVDCmd(DISABLE);

  RCC_Configuration();

  PWR_VoltageScalingConfig(PWR_VoltageScaling_Range1);

  /* Wait Until the Voltage Regulator is ready */
  while (PWR_GetFlagStatus(PWR_FLAG_VOS) != RESET) ;

  /* Init I/O ports */
  Init_GPIOs ();

  /* Initializes ADC */
  ADC_Icc_Init();

  enableInterrupts();

  /* Warning ! in TSL Init the sysTick interrupt is setted to:
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 2000 ---> 500 µs*/

  /* Init Touch Sensing configuration */
  TSL_Init();

  sMCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sMCKeyInfo[0].Setting.b.ENABLED = 1;
  sMCKeyInfo[0].DxSGroup = 0x00;

  /* Initializes the LCD glass */
  LCD_GLASS_Init();

  /* Welcome display  - ukazka pouziti LCD*/
```

```
   // LCD_GLASS_ScrollSentence("      ** SPSE Jecna - PSS
**",1,SCROLL_SPEED);
   LCD_GLASS_DisplayString("PSS  ");

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);

 //bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);




  while (1)
  {

      Delay(1);



 //if ((GPIOA->IDR ) != 0x0)
if (BTN_Get() != 0x0)         //test stisku tlacitka
    {//je-li stisknute tlacitko, pise se na LCD text, a necha se scrollovat
                  LCD_GLASS_ScrollSentence("** SPSE Jecna - PSS ** - ahoj
lidi",1,SCROLL_SPEED);

    }
    else
   {  //neni-li stisknute tlacitko, zmeni se stavobou LED
      GPIO_TOGGLE(LD_PORT,LD_BLUE);
```

```
      GPIO_TOGGLE(LD_PORT,LD_GREEN);
             //neni-li stisknute tlacitko,  LCD se vymaze
             LCD_GLASS_Clear();
                  Delay(100);

    }

  }
}

/**
  * @brief  Configures the different system clocks.
  * @param  None
  * @retval None
  */
void RCC_Configuration(void)
{

  /* Enable HSI Clock */
  RCC_HSICmd(ENABLE);

  /*!< Wait till HSI is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET)
  {}

  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

  RCC_MSIRangeConfig(RCC_MSIRange_6);

  /* Enable the GPIOs Clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOB |
RCC_AHBPeriph_GPIOC| RCC_AHBPeriph_GPIOD| RCC_AHBPeriph_GPIOE|
RCC_AHBPeriph_GPIOH, ENABLE);

  /* Enable  comparator clock LCD and PWR mngt */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_COMP | RCC_APB1Periph_LCD |
RCC_APB1Periph_PWR,ENABLE);

  /* Enable ADC clock & SYSCFG */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_SYSCFG ,
ENABLE);

  /* Allow access to the RTC */
  PWR_RTCAccessCmd(ENABLE);

  /* Reset Backup Domain */
  RCC_RTCResetCmd(ENABLE);
  RCC_RTCResetCmd(DISABLE);

  /* LSE Enable */
  RCC_LSEConfig(RCC_LSE_ON);

  /* Wait till LSE is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
  {}

  RCC_RTCCLKCmd(ENABLE);

  /* LCD Clock Source Selection */
  RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
```

```
  RCC_HSEConfig(RCC_HSE_OFF);

  if(RCC GetFlagStatus(RCC FLAG HSERDY) != RESET )
  {
    while(1);
  }

}

/**
  * @brief  To initialize the I/O ports
  * @caller main
  * @param None
  * @retval None
  */
void  Init_GPIOs (void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* USER button and WakeUP button init: GPIO set in input interrupt active
mode */
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Configure User Button pin as input */
  GPIO_InitStructure.GPIO_Pin = USER_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
  GPIO_Init(BUTTON_GPIO_PORT, &GPIO_InitStructure);

  /* Connect Button EXTI Line to Button GPIO Pin */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);

  /* Configure User Button and IDD_WakeUP EXTI line */
  EXTI_InitStructure.EXTI_Line = EXTI_Line0 ;  // PA0 for User button AND
IDD_WakeUP
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set User Button and IDD_WakeUP EXTI Interrupt to the lowest
priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn ;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure);

/* Configure the GPIO_LED pins  LD3 & LD4*/
  GPIO_InitStructure.GPIO_Pin = LD_GREEN|LD_BLUE;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(LD_PORT, &GPIO_InitStructure);
  GPIO_LOW(LD_PORT,LD_GREEN);
```

```
  GPIO_LOW(LD_PORT,LD_BLUE);

/* Counter enable: GPIO set in output for enable the counter */
  GPIO InitStructure.GPIO Pin = CTN CNTEN GPIO PIN;
  GPIO_Init( CTN_GPIO_PORT, &GPIO_InitStructure);

/* To prepare to start counter */
  GPIO_HIGH(CTN_GPIO_PORT,CTN_CNTEN_GPIO_PIN);

/* Configure Output for LCD */
/* Port A */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
GPIO_Pin_8 | GPIO_Pin_9 |GPIO_Pin_10 |GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOA, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port B */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO_Pin_8 | GPIO_Pin_9 \
                                | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12
| GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOB, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOB, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource4,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource11,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource12,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource13,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource14,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port C*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3 | GPIO_Pin_6 \
                                | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10 |GPIO_Pin_11 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOC, &GPIO_InitStructure);


  GPIO_PinAFConfig(GPIOC, GPIO_PinSource0,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6,GPIO_AF_LCD) ;
```

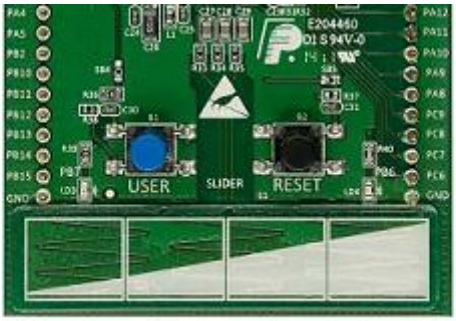
```
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11,GPIO_AF_LCD) ;

/* ADC input */
  GPIO_InitStructure.GPIO_Pin = IDD_MEASURE  ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_Init( IDD_MEASURE_PORT, &GPIO_InitStructure);

}



/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);

}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{

  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }

}

uint32_t BTN_Get(void) {

 return (GPIOA->IDR & (1UL << 0));
}



#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
```

```
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */

  /* Infinite loop */
  while (1)
  {
  }
}

#endif

/****************** (C)  *****END OF FILE****/
```

2.2.1.3 testujeme stisknuti slider tlacitek

V předchozích dvou příkladech jsme jako periferie používali dvě LED a uživatelské tlačítko USR, které jsou umístěny na tištěném spoji startkitu *STM32L-Discovery*, stejně jako tomu bylo u startkitu *STM32VL-Discovery* popisovaném v druhém díle CANSAT skript. Novinkou u *STM32L-Discovery* je LCD umístěný na jeho PCB, nicméně použití LCD jako periferie pro nás není novinkou, neboť k *STM32VL-Discovery* jsme připojovali externí LCD. K připojení externího LCD se ukázalo jako výhodné použít malý konektor na čelní straně startkitu.
Novinkou u *STM32L-Discovery* je klávesnice, kterou můžeme použít jednak jako čtyři tlačítka, což si ukážeme v této kapitole, tak jako jediné dotykové čidlo, což bude předmětem následující kapitoly 2.2.1.4.

.



Nyní se podíváme z čeho je složena a jak ji lze použít .

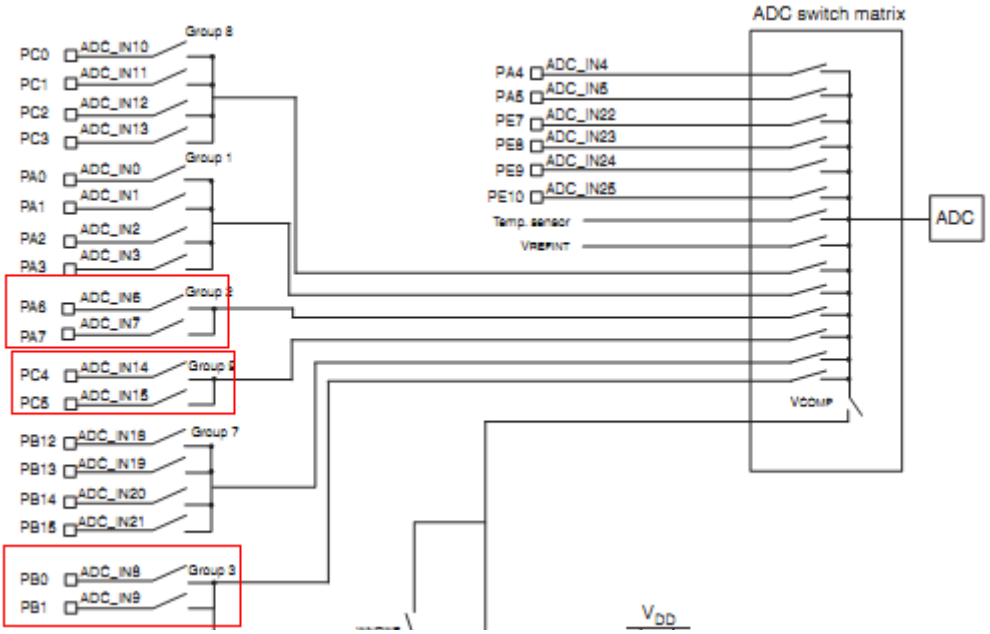
Technická stránka problému

Pro práci se sliderem se v MCU používají tři dvojice portů vstup/výstup, každá dvojice je přepínána analogovým přepínačem:

- PA6, PA7 (skupina 2)
- PC4, PC5 (skupina 9)

- PB0, PB1 (skupina 3)

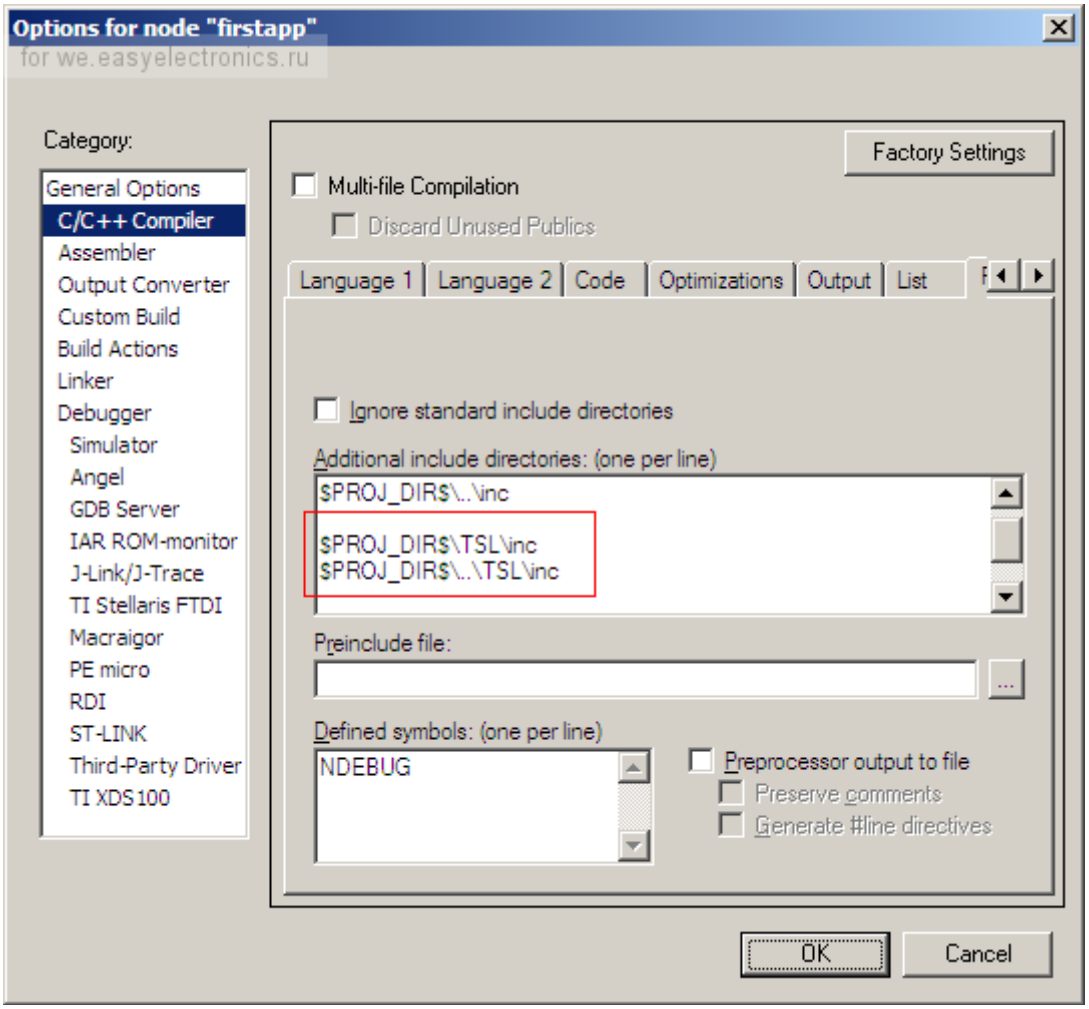
Zapojení tzv. Routing interface ukazuje násl. obrázek jemuž musíme porozumět, pokud bychom si obsluhu toutpedu chtěli naprogramovat sami (bez použití hotové knihovny):



V aplikačním listu AN2869 je podrobně je činnost toutpedu podrobně popsána včetně obrázků s průběhy signálů. Naštěstí v případě použití knihoven se tím nemusíme zabývat.

Používáme Touch-Sensing Library

Pro snadnější práci s Touchpadem poskytl výrobce, firma ST knihovnu. Po jejím stažení ze stránek ST ji nainstalujeme, přidáme hlavičkové soubory pro práci s funkcemi :



Do **main.c** přidáme hlavičkový soubor #include «stm32_tsl_api.h»,v hlavní funkci napíšeme kód inicializace TSL a provedeme ošetření stisků:

```
RCC_Configuration();
  //Inicializace portu LCD
  Init_GPIOs ();
  //nastaveni TSL тач
  TSL_Init();
  sMCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sMCKeyInfo[0].Setting.b.ENABLED = 1;
  sMCKeyInfo[0].DxSGroup = 0x00;
  //inicializace LCD
  LCD_GLASS_Init();
while(1){
  TSL_Action();
  //Funkce obsluhy slideru coby 4 různých tlačítek
```

```
  Button();
}
```

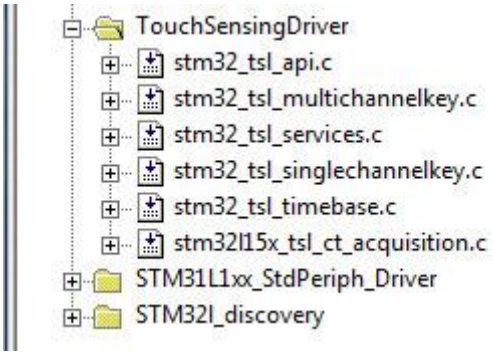
Definice pozice kvůli určení a pozice stisku:

```
#define SLIDER_DETECTED (sMCKeyInfo[0].Setting.b.DETECTED)
#define SLIDER_POSITION (sMCKeyInfo[0].UnScaledPosition)
```

Funkce zjištění čísla stisknutého tlačítka:

```
void Button(void){
  uint8_t *Message;

  //otestujeme stisk s jeho pozici
  if ((SLIDER_DETECTED)&&(TSLState == TSL_IDLE_STATE))
  {
    if( SLIDER_POSITION <= 25 )
      Message = "1";
    if( (SLIDER_POSITION > 25 ) && (SLIDER_POSITION <= 110 ))
      Message = "2";
    if( (SLIDER_POSITION > 110 ) && (SLIDER_POSITION <= 200 ))
      Message = "3";
    if( SLIDER_POSITION > 200 )
      Message = "4";
   LCD_GLASS_DisplayString(Message);
  } //dotaz na stisk
  else if((!TSL_GlobalSetting.b.CHANGED) && (TSLState == TSL_IDLE_STATE))
  {
        LCD_GLASS_DisplayString(Message);
  }
}
```

Hodnotu proměnné SLIDER_POSITION můžeme zobrazovat na displeji a tím ukazovat místo dotyku na slider. V hlavičkovém souboru **discover_functions.h** je deklarována funkce vracející právě tuto hodnotu : Slider_value(). Další podrobnosti se dají nalést v aplikačním listě AN2869.



Výsledný kód **main.c** je

```
/**

******************************************************************************
***
  * @file    main.c
  * @author  Microcontroller Division
  * @version V1.0.0
  * @date    Oct-2012
  * @brief   Main program body

******************************************************************************
***
  * @copy
  * nechame 3 x bliknout modrou LED a pote 3 x zelenou LED
  *
  * v nekonecne smycce testujeme stisknuti slider tlacitka -
  * a vysledek zobrazime na LCD
  *
  *
  *
  *
  * <h2><center>&copy; COPYRIGHT 2012 Vd</center></h2>
  */

/* Includes ------------------------------------------------------------------
---*/

#include "misc.h"
#include "stm32l1xx_adc.h"
#include "stm32l1xx_lcd.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_rtc.h"
#include "stm32l1xx_exti.h"
#include "stm32l1xx_pwr.h"
#include "stm32l1xx_syscfg.h"
#include "stm32l1xx_dbgmcu.h"

#include "stm32_tsl_api.h"
#include "stm32l15x_tsl_ct_acquisition.h"

#include "discover_board.h"
#include "icc_measure.h"
#include "discover_functions.h"
#include "stm32l_discovery_lcd.h"
```

```
#include "stm32_tsl_timebase.h"

#define SLIDER_DETECTED (sMCKeyInfo[0].Setting.b.DETECTED)
#define SLIDER_POSITION (sMCKeyInfo[0].UnScaledPosition)



static volatile uint32_t TimingDelay;
//extern unsigned char Bias_Current;
/* LCD bar graph: used for displaying active function */
//extern uint8_t t_bar[2];
/* Auto_test activation flag: set by interrupt handler if user button is
pressed for a few seconds */
//extern bool Auto_test;
//extern bool Idd_WakeUP;


/* Machine status used by main() wich indicats the active function, set by
user button in interrupt handler */
uint8_t state_machine;

uint16_t Int_CurrentSTBY;
uint32_t BTN_Get(void); //vlastni funkce testujici USR tlacitko
void Button(void);

/*******************************************************************************
*****/
/**
  * @brief main entry point.
  * @par Parameters None
  * @retval void None
  * @par Required preconditions: None
  */

int main(void)
{




  Int_CurrentSTBY = Current_Measurement();




  PWR_PVDCmd(DISABLE);

  RCC_Configuration();

  PWR_VoltageScalingConfig(PWR_VoltageScaling_Range1);

  /* Wait Until the Voltage Regulator is ready */
  while (PWR_GetFlagStatus(PWR_FLAG_VOS) != RESET) ;

  /* Init I/O ports */
  Init_GPIOs ();

  /* Initializes ADC */
  ADC_Icc_Init();
```

```
  enableInterrupts();

  /* Warning ! in TSL Init the sysTick interrupt is setted to:
  SysTick Config(RCC Clocks.HCLK Frequency / 2000 ---> 500 µs*/

  /* Init Touch Sensing configuration */
  TSL_Init();

  sMCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sMCKeyInfo[0].Setting.b.ENABLED = 1;
  sMCKeyInfo[0].DxSGroup = 0x00;

  /* Initializes the LCD glass */
  LCD_GLASS_Init();

  /* Welcome display  - ukazka pouziti LCD*/


   // LCD_GLASS_ScrollSentence("      ** SPSE Jecna - PSS
**",1,SCROLL_SPEED);
   //LCD_GLASS_DisplayString("PSS  ");

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);

 //bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);
```

```
  while (1)
  {

      TSL_Action();  //????
      Button();



  }
}

/**
  * @brief  Configures the different system clocks.
  * @param  None
  * @retval None
  */
void RCC_Configuration(void)
{

  /* Enable HSI Clock */
  RCC_HSICmd(ENABLE);

  /*!< Wait till HSI is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET)
  {}

  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

  RCC_MSIRangeConfig(RCC_MSIRange_6);

  /* Enable the GPIOs Clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOB |
RCC_AHBPeriph_GPIOC| RCC_AHBPeriph_GPIOD| RCC_AHBPeriph_GPIOE|
RCC_AHBPeriph_GPIOH, ENABLE);

  /* Enable  comparator clock LCD and PWR mngt */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_COMP | RCC_APB1Periph_LCD |
RCC_APB1Periph_PWR,ENABLE);

  /* Enable ADC clock & SYSCFG */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_SYSCFG ,
ENABLE);

  /* Allow access to the RTC */
  PWR_RTCAccessCmd(ENABLE);

  /* Reset Backup Domain */
  RCC_RTCResetCmd(ENABLE);
  RCC_RTCResetCmd(DISABLE);

  /* LSE Enable */
  RCC_LSEConfig(RCC_LSE_ON);

  /* Wait till LSE is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
  {}

  RCC_RTCCLKCmd(ENABLE);
```

```
  /* LCD Clock Source Selection */
  RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

  RCC HSEConfig(RCC HSE OFF);

  if(RCC_GetFlagStatus(RCC_FLAG_HSERDY) != RESET )
  {
    while(1);
  }

}

/**
  * @brief  To initialize the I/O ports
  * @caller main
  * @param None
  * @retval None
  */
void  Init_GPIOs (void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* USER button and WakeUP button init: GPIO set in input interrupt active
mode */
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Configure User Button pin as input */
  GPIO_InitStructure.GPIO_Pin = USER_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
  GPIO_Init(BUTTON_GPIO_PORT, &GPIO_InitStructure);

  /* Connect Button EXTI Line to Button GPIO Pin */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);

  /* Configure User Button and IDD_WakeUP EXTI line */
  EXTI_InitStructure.EXTI_Line = EXTI_Line0 ;  // PA0 for User button AND
IDD_WakeUP
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set User Button and IDD_WakeUP EXTI Interrupt to the lowest
priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn ;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure);

/* Configure the GPIO_LED pins  LD3 & LD4*/
  GPIO_InitStructure.GPIO_Pin = LD_GREEN|LD_BLUE;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
```

```
  GPIO_Init(LD_PORT, &GPIO_InitStructure);
  GPIO_LOW(LD_PORT,LD_GREEN);
  GPIO_LOW(LD_PORT,LD_BLUE);

/* Counter enable: GPIO set in output for enable the counter */
  GPIO_InitStructure.GPIO_Pin = CTN_CNTEN_GPIO_PIN;
  GPIO_Init( CTN_GPIO_PORT, &GPIO_InitStructure);

/* To prepare to start counter */
  GPIO_HIGH(CTN_GPIO_PORT,CTN_CNTEN_GPIO_PIN);

/* Configure Output for LCD */
/* Port A */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
GPIO_Pin_8 | GPIO_Pin_9 |GPIO_Pin_10 |GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOA, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port B */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO_Pin_8 | GPIO_Pin_9 \
                                | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12
| GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOB, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOB, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource4,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource11,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource12,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource13,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource14,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port C*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3 | GPIO_Pin_6 \
                                | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10 |GPIO_Pin_11 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOC, &GPIO_InitStructure);


  GPIO_PinAFConfig(GPIOC, GPIO_PinSource0,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2,GPIO_AF_LCD) ;
```

```
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11,GPIO_AF_LCD) ;

/* ADC input */
  GPIO_InitStructure.GPIO_Pin = IDD_MEASURE  ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_Init( IDD_MEASURE_PORT, &GPIO_InitStructure);

}



/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);

}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{

  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }

}

uint32_t BTN_Get(void) {

 return (GPIOA->IDR & (1UL << 0));
}


void Button(void)
{
  uint8_t *Message;    //
      Message = "0";
if ((SLIDER_DETECTED)&&(TSLState == TSL_IDLE_STATE))
      {
      if( SLIDER_POSITION <= 25 )      Message = "1";
      if( (SLIDER_POSITION > 25 ) && (SLIDER_POSITION <= 110 ))
      Message = "2";
```

```
  if( (SLIDER_POSITION > 110 ) && (SLIDER_POSITION <= 200 ))
            Message = "3";

  if( SLIDER_POSITION > 200 ) Message = "4";
     LCD_GLASS_DisplayString(Message);  } //
       else
 if((!TSL_GlobalSetting.b.CHANGED) && (TSLState == TSL_IDLE_STATE))
  {        LCD_GLASS_DisplayString(Message);  }
}









#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */

  /* Infinite loop */
  while (1)
  {
  }
}

#endif

/****************** (C)  *****END OF FILE****/
```

2.2.1.4 Slider coby čidlo polohy stisku

V předchozím programu jsme slider používali jako čtyři tlačítka a programově jsme zjišťovali jejich stisk. Nyní budeme slider používat jako čidlo pozice dotyku prstu, přičemž tento dotyk bude určen čísly 0 až 255 (0 je dotyk na levém okraji). Zároveň si v kódu ukážeme práci s řetězcem znaků, resp. převod numerické hodnoty na řetězec znaků a jeho následné zobrazení na LCD.

```
/**

*****************************************************************************
***
```

```
  * @file    main.c
  * @author
  * @version V1.0.0
  * @date    Oct-2012
  * @brief   Main program body

******************************************************************************
***
  * @copy
  * nechame 3 x bliknout modrou LED a pote 3 x zelenou LED
  *
  * v nekonecne smycce testujeme stisknuti slider tlacitka -
  * a vysledek zobrazime na LCD
  *
  *
  *
  *
  * <h2><center>&copy; COPYRIGHT 2012 Vd</center></h2>
  */

/* Includes ------------------------------------------------------------------
---*/

#include "misc.h"
#include <stddef.h>
#include <string.h>

#include "stm32l1xx_adc.h"
#include "stm32l1xx_lcd.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_rtc.h"
#include "stm32l1xx_exti.h"
#include "stm32l1xx_pwr.h"
#include "stm32l1xx_syscfg.h"
#include "stm32l1xx_dbgmcu.h"

#include "stm32_tsl_api.h"
#include "stm32l15x_tsl_ct_acquisition.h"

#include "discover_board.h"
#include "icc_measure.h"
#include "discover_functions.h"
#include "stm32l_discovery_lcd.h"
#include "stm32_tsl_timebase.h"

#define SLIDER_DETECTED (sMCKeyInfo[0].Setting.b.DETECTED)
#define SLIDER_POSITION (sMCKeyInfo[0].UnScaledPosition)



static volatile uint32_t TimingDelay;
//extern unsigned char Bias_Current;
/* LCD bar graph: used for displaying active function */
//extern uint8_t t_bar[2];
/* Auto_test activation flag: set by interrupt handler if user button is
pressed for a few seconds */
//extern bool Auto_test;
//extern bool Idd_WakeUP;
```

```
/* Machine status used by main() wich indicats the active function, set by
user button in interrupt handler */
uint8_t state_machine;

uint16_t Int_CurrentSTBY;
uint32_t BTN_Get(void); //vlastni funkce testujici USR tlacitko
void Button(void);

void itoa(uint16_t n, uint8_t s[]);
void reverse(uint8_t s[]);


/*******************************************************************************
*****/
/**
  * @brief main entry point.
  * @par Parameters None
  * @retval void None
  * @par Required preconditions: None
  */

int main(void)
{




  Int_CurrentSTBY = Current_Measurement();




  PWR_PVDCmd(DISABLE);

  RCC_Configuration();

  PWR_VoltageScalingConfig(PWR_VoltageScaling_Range1);

  /* Wait Until the Voltage Regulator is ready */
  while (PWR_GetFlagStatus(PWR_FLAG_VOS) != RESET) ;

  /* Init I/O ports */
  Init_GPIOs ();

  /* Initializes ADC */
  ADC_Icc_Init();

  enableInterrupts();

  /* Warning ! in TSL Init the sysTick interrupt is setted to:
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 2000 ---> 500 µs*/

  /* Init Touch Sensing configuration */
  TSL_Init();

  sMCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sMCKeyInfo[0].Setting.b.ENABLED = 1;
  sMCKeyInfo[0].DxSGroup = 0x00;

  /* Initializes the LCD glass */
```

```
  LCD_GLASS_Init();

  /* Welcome display  - ukazka pouziti LCD*/


   // LCD_GLASS_ScrollSentence("      ** SPSE Jecna - PSS
**",1,SCROLL_SPEED);
   //LCD_GLASS_DisplayString("PSS  ");

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);

 //bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);




  while (1)
  {

      TSL_Action();  //????
      Button();



  }
}

/**
```

```
  * @brief  Configures the different system clocks.
  * @param  None
  * @retval None
  */
void RCC_Configuration(void)
{

  /* Enable HSI Clock */
  RCC_HSICmd(ENABLE);

  /*!< Wait till HSI is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET)
  {}

  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

  RCC_MSIRangeConfig(RCC_MSIRange_6);

  /* Enable the GPIOs Clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOB |
RCC_AHBPeriph_GPIOC| RCC_AHBPeriph_GPIOD| RCC_AHBPeriph_GPIOE|
RCC_AHBPeriph_GPIOH, ENABLE);

  /* Enable  comparator clock LCD and PWR mngt */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_COMP | RCC_APB1Periph_LCD |
RCC_APB1Periph_PWR,ENABLE);

  /* Enable ADC clock & SYSCFG */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_SYSCFG ,
ENABLE);

  /* Allow access to the RTC */
  PWR_RTCAccessCmd(ENABLE);

  /* Reset Backup Domain */
  RCC_RTCResetCmd(ENABLE);
  RCC_RTCResetCmd(DISABLE);

  /* LSE Enable */
  RCC_LSEConfig(RCC_LSE_ON);

  /* Wait till LSE is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
  {}

  RCC_RTCCLKCmd(ENABLE);

  /* LCD Clock Source Selection */
  RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

  RCC_HSEConfig(RCC_HSE_OFF);

  if(RCC_GetFlagStatus(RCC_FLAG_HSERDY) != RESET )
  {
    while(1);
  }

}

/**
```

```
  * @brief  To initialize the I/O ports
  * @caller main
  * @param None
  * @retval None
  */
void  Init_GPIOs (void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* USER button and WakeUP button init: GPIO set in input interrupt active
mode */
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Configure User Button pin as input */
  GPIO_InitStructure.GPIO_Pin = USER_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
  GPIO_Init(BUTTON_GPIO_PORT, &GPIO_InitStructure);

  /* Connect Button EXTI Line to Button GPIO Pin */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);

  /* Configure User Button and IDD_WakeUP EXTI line */
  EXTI_InitStructure.EXTI_Line = EXTI_Line0 ;  // PA0 for User button AND
IDD_WakeUP
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set User Button and IDD_WakeUP EXTI Interrupt to the lowest
priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn ;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure);

/* Configure the GPIO_LED pins  LD3 & LD4*/
  GPIO_InitStructure.GPIO_Pin = LD_GREEN|LD_BLUE;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(LD_PORT, &GPIO_InitStructure);
  GPIO_LOW(LD_PORT,LD_GREEN);
  GPIO_LOW(LD_PORT,LD_BLUE);

/* Counter enable: GPIO set in output for enable the counter */
  GPIO_InitStructure.GPIO_Pin = CTN_CNTEN_GPIO_PIN;
  GPIO_Init( CTN_GPIO_PORT, &GPIO_InitStructure);

/* To prepare to start counter */
  GPIO_HIGH(CTN_GPIO_PORT,CTN_CNTEN_GPIO_PIN);

/* Configure Output for LCD */
/* Port A */
```

```
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
GPIO_Pin_8 | GPIO_Pin_9 |GPIO_Pin_10 |GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOA, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port B */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO_Pin_8 | GPIO_Pin_9 \
                                | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12
| GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOB, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOB, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource4,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource11,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource12,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource13,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource14,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port C*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3 | GPIO_Pin_6 \
                                | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10 |GPIO_Pin_11 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOC, &GPIO_InitStructure);


  GPIO_PinAFConfig(GPIOC, GPIO_PinSource0,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11,GPIO_AF_LCD) ;

/* ADC input */
  GPIO_InitStructure.GPIO_Pin = IDD_MEASURE  ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_Init( IDD_MEASURE_PORT, &GPIO_InitStructure);
```

```
}



/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);

}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{

  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }

}

uint32_t BTN_Get(void) {

 return (GPIOA->IDR & (1UL << 0));
}


void Button(void)
{

  int i = 0;
  uint8_t LCD_text[] = "        ";
  uint16_t Hodnota=0;




      //

      LCD_GLASS_Clear();

if ((SLIDER_DETECTED)&&(TSLState == TSL_IDLE_STATE))
      {


              /* hodnota z slideru */
      Hodnota = SLIDER_POSITION;
      for(i=7; i>0; i--)
```

```
          LCD_text[i] = 0x20;
      itoa(Hodnota, LCD_text);
      for(i=7; i>1; i--)
          LCD text[i] = LCD text[i-2];
      LCD_text[0] = 0x20;
      LCD_text[1] = 0x20;
      LCD_GLASS_DisplayString((uint8_t*)LCD_text);
      /* Toggle blue LED */
      GPIO_TOGGLE(LD_PORT,LD_BLUE);

      Delay(0xFF);



       } //
       else
 if((!TSL_GlobalSetting.b.CHANGED) && (TSLState == TSL_IDLE_STATE))
  {
       /* hodnota z slideru */
      //Hodnota = SLIDER_POSITION;
            Hodnota = 0;
      for(i=7; i>0; i--)
          LCD_text[i] = 0x20;
      itoa(Hodnota, LCD_text);
      for(i=7; i>1; i--)
          LCD_text[i] = LCD_text[i-2];
      LCD_text[0] = 0x20;
      LCD_text[1] = 0x20;
      LCD_GLASS_DisplayString((uint8_t*)LCD_text);
      /* Toggle blue LED */
      GPIO_TOGGLE(LD_PORT,LD_BLUE);

      Delay(0xFF);
  }
}


void itoa(uint16_t n, uint8_t s[])
{
    int i, sign;
    if ((sign = n) < 0)  /* record sign */
        n = -n;          /* make n positive */
    i = 0;
    do {       /* generate digits in reverse order */
        s[i++] = n % 10 + '0';   /* get next digit */
    } while ((n /= 10) > 0);     /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}



/**
  * @brief  Reverse string
  * @param  s = string
  * @retval : None
  */
void reverse(uint8_t s[])
```

```
{
    int i, j;
    char c;

    for (i = 0, j = strlen((char*)s)-1; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}




#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */

  /* Infinite loop */
  while (1)
  {
  }
}

#endif

/****************** (C)  *****END OF FILE****/
```

2.2.1.5 A/D převod

Tento program vznikl tak, že jsem jednoduše přepsal program pro *STM32VL-Discovery* z 2.dílu skript.

```


/**

*****************************************************************************
***
  * @file    main.c
  * @author  Vd
  * @version V1.0.0
  * @date    Oct-2012
  * @brief   Main program body
```

```
******************************************************************************
***
  * @copy
  * nechame 3 x bliknout modrou LED a pote 3 x zelenou LED
  *
  * v nekonecne smycce testujeme je-li stisknuto USR tlacitko:
  * pokud ano, tak funkcí ADC1_Read() zmerime napeti
  * na kanalu ADC1 tj pin PA5
  * nutno prepocitat - dostaneme totiz cisla v rozsahu 0 az 4095
  * tj (2 na 12) - 1
  *a zobrazime na LCD displeji
  * <h2><center>&copy; COPYRIGHT 2012 Vd</center></h2>
  */

/* Includes ------------------------------------------------------------------
---*/

#include "misc.h"
#include <stddef.h>
#include <string.h>

#include "stm32l1xx_adc.h"
#include "stm32l1xx_lcd.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_rtc.h"
#include "stm32l1xx_exti.h"
#include "stm32l1xx_pwr.h"
#include "stm32l1xx_syscfg.h"
#include "stm32l1xx_dbgmcu.h"

#include "stm32_tsl_api.h"
#include "stm32l15x_tsl_ct_acquisition.h"

#include "discover_board.h"
#include "icc_measure.h"
#include "discover_functions.h"
#include "stm32l_discovery_lcd.h"
#include "stm32_tsl_timebase.h"

static volatile uint32_t TimingDelay;
//extern unsigned char Bias_Current;
/* LCD bar graph: used for displaying active function */
//extern uint8_t t_bar[2];
/* Auto_test activation flag: set by interrupt handler if user button is
pressed for a few seconds */
//extern bool Auto_test;
//extern bool Idd_WakeUP;


/* Machine status used by main() wich indicats the active function, set by
user button in interrupt handler */
uint8_t state_machine;

uint16_t Int_CurrentSTBY;
uint32_t BTN_Get(void); //vlastni funkce testujici USR tlacitko
void itoa(uint16_t n, uint8_t s[]);
void reverse(uint8_t s[]);
void ADC_Inicializace(void);
uint16_t ADC1_Read(void);
```

```
/*******************************************************************************
*****/
/**
  * @brief main entry point.
  * @par Parameters None
  * @retval void None
  * @par Required preconditions: None
  */

int main(void)
{
  int i = 0;
  uint8_t LCD_text[] = "        ";
  uint16_t Hodnota=0;
  Int_CurrentSTBY = Current_Measurement();

  PWR_PVDCmd(DISABLE);

  RCC_Configuration();

  PWR_VoltageScalingConfig(PWR_VoltageScaling_Range1);

  /* Wait Until the Voltage Regulator is ready */
  while (PWR_GetFlagStatus(PWR_FLAG_VOS) != RESET) ;

  /* Init I/O ports */
  Init_GPIOs ();




  /* Initializes ADC */
      ADC_Inicializace();


  enableInterrupts();

  /* Warning ! in TSL Init the sysTick interrupt is setted to:
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 2000 ---> 500 µs*/

  /* Init Touch Sensing configuration */
  TSL_Init();

  sMCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sMCKeyInfo[0].Setting.b.ENABLED = 1;
  sMCKeyInfo[0].DxSGroup = 0x00;

  /* Initializes the LCD glass */
  LCD_GLASS_Init();

  /* Welcome display  - ukazka pouziti LCD*/

  LCD_GLASS_ScrollSentence((uint8_t*)"  ADC1  PSS SPSE Jecna  ",3,200);


   // LCD_GLASS_ScrollSentence("      ** SPSE Jecna - PSS
**",1,SCROLL_SPEED);
```

```
//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
Delay(500);

//bliknuti modre LED
      GPIO_HIGH(LD_PORT,LD_BLUE);
Delay(500);
      GPIO_LOW(LD_PORT,LD_BLUE);
 Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);

 //bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
Delay(500);

//bliknuti zelene LED
      GPIO_HIGH(LD_PORT,LD_GREEN);
Delay(500);
      GPIO_LOW(LD_PORT,LD_GREEN);
 Delay(500);



  while (1)
  {

      i++;
    /* Check if the User Button is pressed */
    if ((GPIOA->IDR & USER_GPIO_PIN) != 0x0)
    {
      /* ADC1 measure */
      Hodnota = ADC1_Read();
                  LCD_GLASS_Clear();
      for(i=7; i>0; i--)
          LCD_text[i] = 0x20;
      itoa(Hodnota, LCD_text);
      for(i=7; i>1; i--)
          LCD_text[i] = LCD_text[i-2];
      LCD_text[0] = 0x20;
      LCD_text[1] = 0x20;
      LCD_GLASS_DisplayString((uint8_t*)LCD_text);
      /* Toggle blue LED */
      //GPIO_TOGGLE(LD_PORT,LD_BLUE);
```

```
      GPIO_HIGH(LD_PORT,LD_BLUE);
      Delay(0xFF);
    }
    else
    {
        GPIO_LOW(LD_PORT,LD_BLUE);
        GPIO_LOW(LD_PORT,LD_GREEN);
    }




  }
}

/**
  * @brief  Configures the different system clocks.
  * @param  None
  * @retval None
  */
void RCC_Configuration(void)
{

  /* Enable HSI Clock */
  RCC_HSICmd(ENABLE);

  /*!< Wait till HSI is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET)
  {}

  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

  RCC_MSIRangeConfig(RCC_MSIRange_6);

  /* Enable the GPIOs Clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOB |
RCC_AHBPeriph_GPIOC| RCC_AHBPeriph_GPIOD| RCC_AHBPeriph_GPIOE|
RCC_AHBPeriph_GPIOH, ENABLE);

  /* Enable  comparator clock LCD and PWR mngt */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_COMP | RCC_APB1Periph_LCD |
RCC_APB1Periph_PWR,ENABLE);

  /* Enable ADC clock & SYSCFG */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_SYSCFG ,
ENABLE);

  /* Allow access to the RTC */
  PWR_RTCAccessCmd(ENABLE);

  /* Reset Backup Domain */
  RCC_RTCResetCmd(ENABLE);
  RCC_RTCResetCmd(DISABLE);

  /* LSE Enable */
  RCC_LSEConfig(RCC_LSE_ON);

  /* Wait till LSE is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
```

```
  {}

  RCC_RTCCLKCmd(ENABLE);

  /* LCD Clock Source Selection */
  RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

  RCC_HSEConfig(RCC_HSE_OFF);

  if(RCC_GetFlagStatus(RCC_FLAG_HSERDY) != RESET )
  {
    while(1);
  }

}

/**
  * @brief  To initialize the I/O ports
  * @caller main
  * @param None
  * @retval None
  */
void  Init_GPIOs (void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* USER button and WakeUP button init: GPIO set in input interrupt active
mode */
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Configure User Button pin as input */
  GPIO_InitStructure.GPIO_Pin = USER_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
  GPIO_Init(BUTTON_GPIO_PORT, &GPIO_InitStructure);

  /* Connect Button EXTI Line to Button GPIO Pin */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);

  /* Configure User Button and IDD_WakeUP EXTI line */
  EXTI_InitStructure.EXTI_Line = EXTI_Line0 ;  // PA0 for User button AND
IDD_WakeUP
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set User Button and IDD_WakeUP EXTI Interrupt to the lowest
priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn ;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure);

/* Configure the GPIO_LED pins  LD3 & LD4*/
  GPIO_InitStructure.GPIO_Pin = LD_GREEN|LD_BLUE;
```

```
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(LD_PORT, &GPIO_InitStructure);
  GPIO_LOW(LD_PORT,LD_GREEN);
  GPIO_LOW(LD_PORT,LD_BLUE);

/* Counter enable: GPIO set in output for enable the counter */
  GPIO_InitStructure.GPIO_Pin = CTN_CNTEN_GPIO_PIN;
  GPIO_Init( CTN_GPIO_PORT, &GPIO_InitStructure);

/* To prepare to start counter */
  GPIO_HIGH(CTN_GPIO_PORT,CTN_CNTEN_GPIO_PIN);

/* Configure Output for LCD */
/* Port A */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
GPIO_Pin_8 | GPIO_Pin_9 |GPIO_Pin_10 |GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOA, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port B */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO_Pin_8 | GPIO_Pin_9 \
                                | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12
| GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOB, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOB, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource4,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource11,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource12,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource13,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource14,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource15,GPIO_AF_LCD) ;

/* Configure Output for LCD */
/* Port C*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3 | GPIO_Pin_6 \
                                | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10 |GPIO_Pin_11 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOC, &GPIO_InitStructure);
```

```
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource0,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11,GPIO_AF_LCD) ;

/* ADC input */
  GPIO_InitStructure.GPIO_Pin = IDD_MEASURE  ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_Init( IDD_MEASURE_PORT, &GPIO_InitStructure);

}



/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);

}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{

  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }

}

uint32_t BTN_Get(void) {

 return (GPIOA->IDR & (1UL << 0));
}

void itoa(uint16_t n, uint8_t s[])
{
    int i, sign;
    if ((sign = n) < 0)  /* record sign */
        n = -n;          /* make n positive */
    i = 0;
```

```
    do {       /* generate digits in reverse order */
        s[i++] = n % 10 + '0';   /* get next digit */
    } while ((n /= 10) > 0);     /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}



/**
  * @brief  Reverse string
  * @param  s = string
  * @retval : None
  */
void reverse(uint8_t s[])
{
    int i, j;
    char c;

    for (i = 0, j = strlen((char*)s)-1; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}


void ADC_Inicializace(void)
{
  ADC_InitTypeDef  ADC_InitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Configure PA5 GPIO port pin in Analog input mode (trigger OFF) */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_400KHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* povolime hodiny do A/D prevodniku */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  /* de-initialize ADC */
  ADC_DeInit(ADC1);

  /*  ADC configured as follow:
    - NbrOfChannel = 1 - ADC_Channel_5
    - Mode = Single ConversionMode(ContinuousConvMode disabled)
    - Resolution = 12Bits
    - Prescaler = /1
    - sampling time 192 */

  /* ADC Configuration */
  ADC_StructInit(&ADC_InitStructure);
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
```

```
  ADC_InitStructure.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC1 regular channel5 configuration */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 1,
ADC_SampleTime_192Cycles);
  ADC_DelaySelectionConfig(ADC1, ADC_DelayLength_Freeze);

  ADC_PowerDownCmd(ADC1, ADC_PowerDown_Idle_Delay, ENABLE);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Wait until ADC1 ON status */
  while (ADC_GetFlagStatus(ADC1, ADC_FLAG_ADONS) == RESET);

}



uint16_t ADC1_Read(void)
{
  uint8_t i;
  uint16_t res;

  /* initialize result */
  res = 0;
  for(i=4; i>0; i--)
  {
  /* start ADC convertion by software */
      ADC_SoftwareStartConv(ADC1);

      /* wait until end-of-covertion */
      while( ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0 );
  /* read ADC convertion result */
      res += ADC_GetConversionValue(ADC1);
  }
  return (res>>2);
}









#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
```

```
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */

  /* Infinite loop */
  while (1)
  {
  }
}

#endif

/****************** (C)  *****END OF FILE****/
```

3. Čidla

3.1 využití čidel v satelitech CANSAT

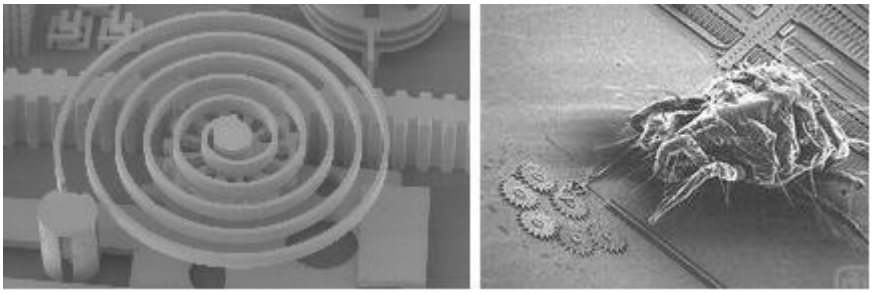
Součástí elektroniky CANSATů jsou obvykle čidla teploty, tlaku. Dále např. přijímač GPS a čidla pro el. měření dalších neelektrických veličin. Klasickými analogovými čidly stejně jako přijímačem GPS jsem se zabýval v 1.díle skript a proto se budeme v tomto materiálu zabývat čidly s digitálním výstupem.

3.2 čidla MEMS

MEMS (*Micro-Electro-Mechanical Systems*) je označení samotné technologie i produktů z ní vyplývajících.

Technologií MEMS se míní velmi sofistikované umístění elektronických, ale především mikro-mechanických prvků, na křemíkovou bázi pomocí moderních výrobních metod, které mají svůj původ ve výrobě integrovaných obvodů.

Technologie MEMS je v podstatě spojení integrovaných obvodů, mechanických elementů, senzorů, akčních členů, řídící a vyhodnocovací elektroniky na jeden křemíkový substrát prostřednictvím různých výrobních technologií. Zatímco elektronické části jsou vyráběny "tradičními" technologiemi typu CMOS, Bipolar nebo BiCMOS, mikromechanické části jsou zhotovovány prostřednictvím technologií různého selektivního leptání, nebo imple- mentováním nových vrstev. Přínosem technologie MEMS je především zmenšení rozměrů, nízká spotřeba snímačů vyráběných pomocí této technologie



Obr Křemíková pružina vyrobená technologií MEMS a porovnání některých mechanických MEMS komponent s roztočem.

Pomocí technologie MEMS lze vytvářet miniaturní až mikroskopické systémy o rozměrech několika milimetrů až mikrometrů, složené ze snímačů, převodníků, elektrických obvodů a aktuátorů, které tvoří MEMS zařízení. Mezi výhody MEMS zařízení patří malé rozměry, nízká spotřeba, vysoká

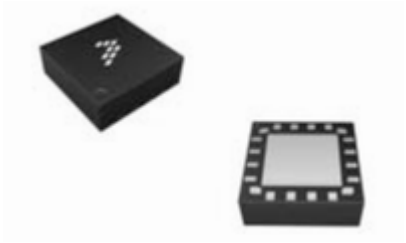
mechanická odolnost, kompaktnost, nízká cena při velkých sériích. V současnosti jsou používány tři způsoby výroby MEMS struktur - před vlastním procesem CMOS, v jeho průběhu nebo až po dokončení tohoto procesu. Celý výrobní proces se obvykle skládá ze sekvence operací, při kterých se postupně formují požadované mechanické struktury, jako jsou nosníky, ozubená kolečka, ložiska, tyčky apod. Technologie MEMS se používá například na výrobu prvků, které nachází uplatnění v automobilovém průmyslu, medicíně optoelektronice a dalších průmyslových i neprůmyslových aplikacích. Vzájemné interakce mechanické pohyblivé struktury a vyhodnocovací digitální a analogové elektroniky používají například akcelerometry, gyroskopy, oscilátory, rezonátory, optoelektronické přepínače a další zařízení.

Produkty MEMS vychází z možností MEMS technologie a jedná se především pohybové senzory (akcelerometry, gyroskopy…), ale i mikročerpadla, mikropohony, mikrocívky aj. V souvislosti s těmito produkty se hovoří o systému na čipu nebo také o inteligentním snímači, jelikož je zde přítomen jak mechanický subsystém (nutný pro transformaci fyzikální podstaty na elektrickou veličinu), tak elektronický subsystém zajišťující následné zpracování, neboli postprocessing (zesílení, saturace, filtrace aj.).

Princip MEMS si ukážeme na příkladu 3D akcelerometrů. Akcelerometry, tedy senzory zrychlení, se v dnešním robotizovaném světě využívají stále více. Proto firmy neustále vyvíjejí nové nebo vylepšují stávající funkční struktury pro vylepšení vlastností senzorů.

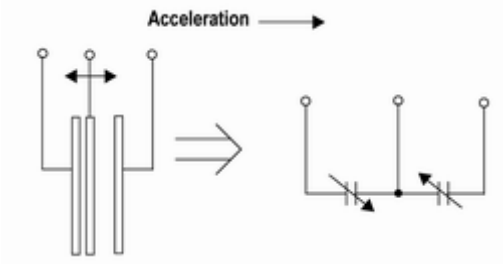
Možnosti použití 3D MEMS akcelerometrů

- Měření sklonu či naklonění ploch a předmětů
- Monitorování pohybu předmětů při přepravě
- Zabezpečovací zařízení
- Detekce a monitorování nárazů a vibrací
- Měření zrychlení - akcelerace
- Měření brždění
- Detekce a měření pádu
- Měření otřesů
- HMI rozhraní, ovládání multimediálních systémů
- Měření a předpovídání seismické aktivity
- Trakční a bezpečnostní systémy automobilů
- apod.



Mechanická struktura

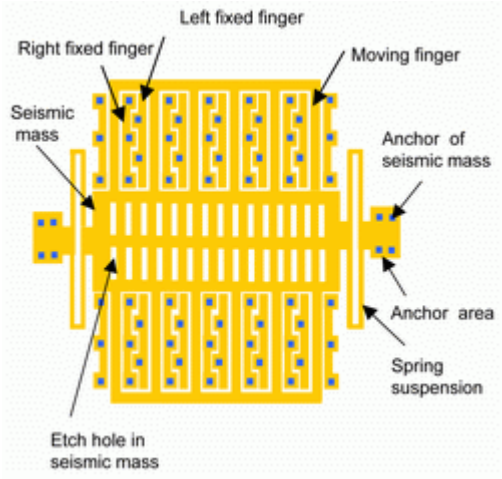
Struktura a funkce MEMS akcelerometru je založena na proměnné kapacitě tříelektrodového vzduchového kondenzátoru. Využívá se zde známé nelineární závislosti kapacity *C* na vzdálenosti elektrod kondenzátoru *d* (velikosti vzduchové mezery) dle vzorce $C = k \cdot S / d$ (k = konstanta, *S* = plocha elektrod). Pokud tedy jednu elektrodu uděláme pohyblivou a její pohyb bude závislý na působícím zrychlení, získáme kapacitní akcelerometr. A protože taková struktura pohyblivých nosníčků (elektrod) je snadno realizovatelná MEMS technologií, vznikne nám MEMS akcelerometr.



Základní princip MEMS akcelerometru

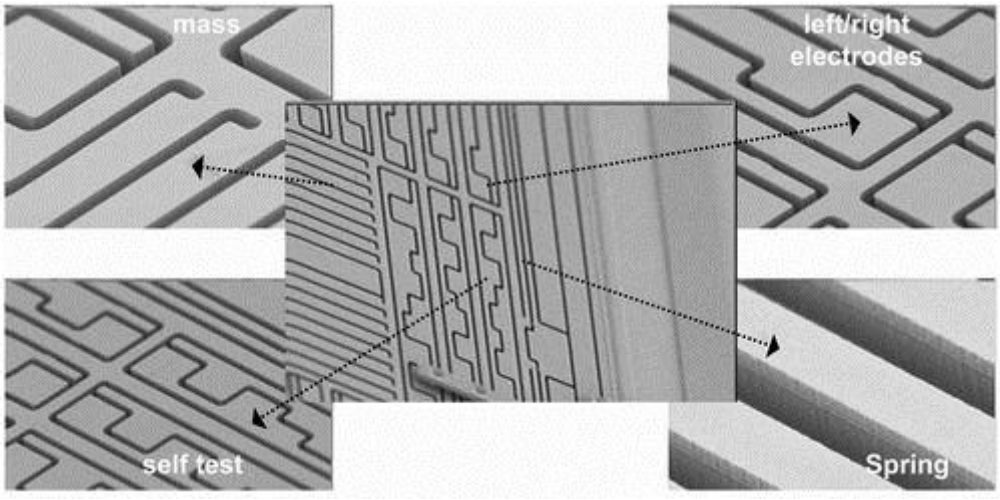
Výsledná funkční struktura ovšem není zas tak jednoduchá, jak se na první pohled zdá. Hlavní je právě zajistit lineární a dostatečně citlivý převod zrychlení na mechanický posuvný pohyb. Ten totiž určuje samotný měřící rozsah senzoru, tj. maximální a minimální měřitelné zrychlení.

Zde se vychází ze základního vztahu pro působení síly při zrychlení $F = m \cdot a$, kde F je síla vzniklá působením zrychlení a na hmotu m (Seismic mass). Síla se pak přes pružiny (Spring suspension) převádí na posuv nosníčku (Seismic mass), jejíž některé části tvoří pohyblivé elektrody vzduchového kondenzátoru (Moving finger). Jejich pozice vůči levým pevným elektrodám (Left fixed fingers) a pravým pevným elektrodám (Right fixed fingers) určuje elektronicky měřenou hodnotu kapacity takto vzniklého kondenzátoru.



Schématicky znázorněná mechanická MEMS struktura akcelerometru

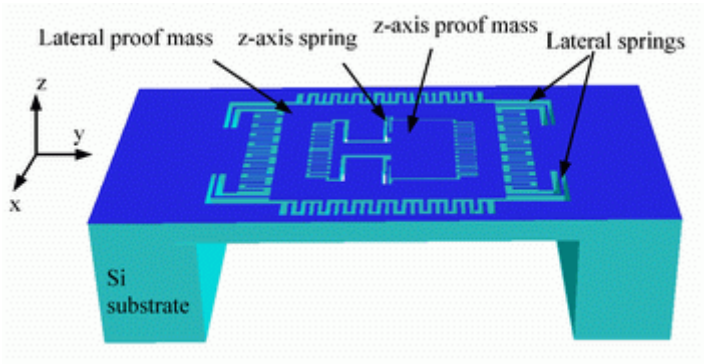
Nosníková struktura (Seismic mass) a pružiny se vyrábí leptáním polykrystalického křemíku (polysilicon), přičemž s postupným vývojem dochází k postupnému protahování pohyblivých elektrod ve směru kolmém na měřenou osu z dřívějších jednotek mikrometrů až na desítky mikrometrů. Díky tomu je možné několikanásobně protáhnout délku elektrod a tím získat lepší odstup signál/šum, menší křížovou citlivost (např. vliv zrychlení v ose X na osu Y) a hlavně odezvu na změnu velikosti zrychlení. Do budoucna se již také počítá nahrazením polySi krystalickým křemíkem.



Detailní záběry na reálné provedení jednotlivých částí struktury - pevných a pohyblivých elektrod (electrodes), detekční hmoty (mass), pružiny (sping) a samotestovací strukturu (seft test)

Výše uvedená a popsaná struktura však umožňuje měření zrychlení jen v jednom směru kolmém na pohyblivé elektrody = 1D akcelerometry. Technologicky vcelku není problém na čipu přidat další

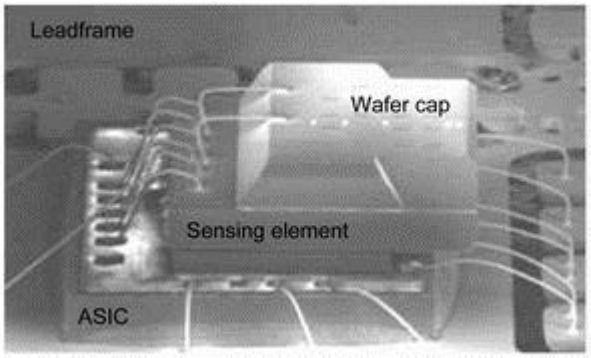
stejnou strukturu pouze proti té předchozí pootočenou o 90°. Vznikne tak 2D akcelerometr, který například měří v osách XY nebo XZ, dle natočení senzoru. Složitější je již vytvořit jednočipový 3D akcelerometr, protože se musí přidat výškově pohyblivá struktura v ose Z.



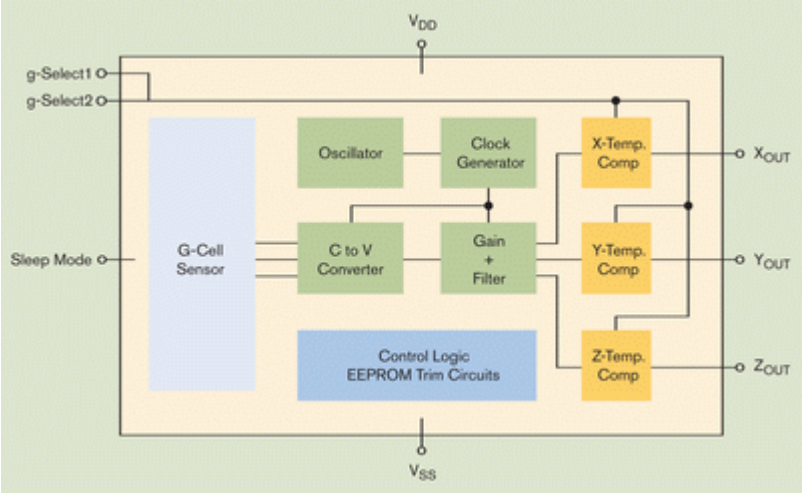
3D struktura akcelerometru (měření zrychlení v osách X, Y, Z)

Elektrická struktura

Celá ASIC elektronika, která měří změnu kapacity, převádí ji na změnu napětí a zpracovává takto získaný signál na standardní lineární napěťový výstup, je implantována pod snímacím elementem (Sensing element) překrytého krycí "kopulí" (Wafer cap). K měření kapacity se využívá metody spínaných kondenzátorů řízené číslicovou logikou a generátorem spínacího hodinového signálu. Dochází tak k převodu změny kapacity na změnu napětí. To je následně linearizováno a filtrováno opět obvody se spínanými kondenzátory a nakonec se provádí kompenzace vlivu teploty. Výsledkem je lineární, zesílený a kompenzovaný napěťový signál s definovanou převodní konstantou - citlivostí podávající informaci o kolik se musí změnit hodnota měřeného zrychlení, aby došlo ke změně výstupního napětí o 1 V (hodnota g/V). Vše se provádí zvlášť pro každou osu snímání (kanál).



Struktura spojení snímače a ASIC elektroniky

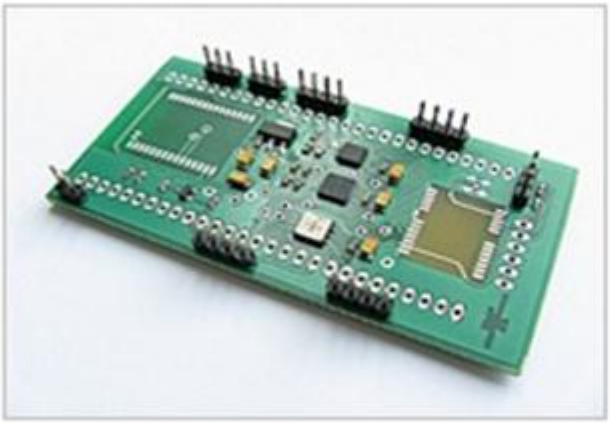


Blokové schéma elektrické struktury 3D akcelerometru

3.3 Studium čidel MEMS pomocí startkitu IMU Discovery

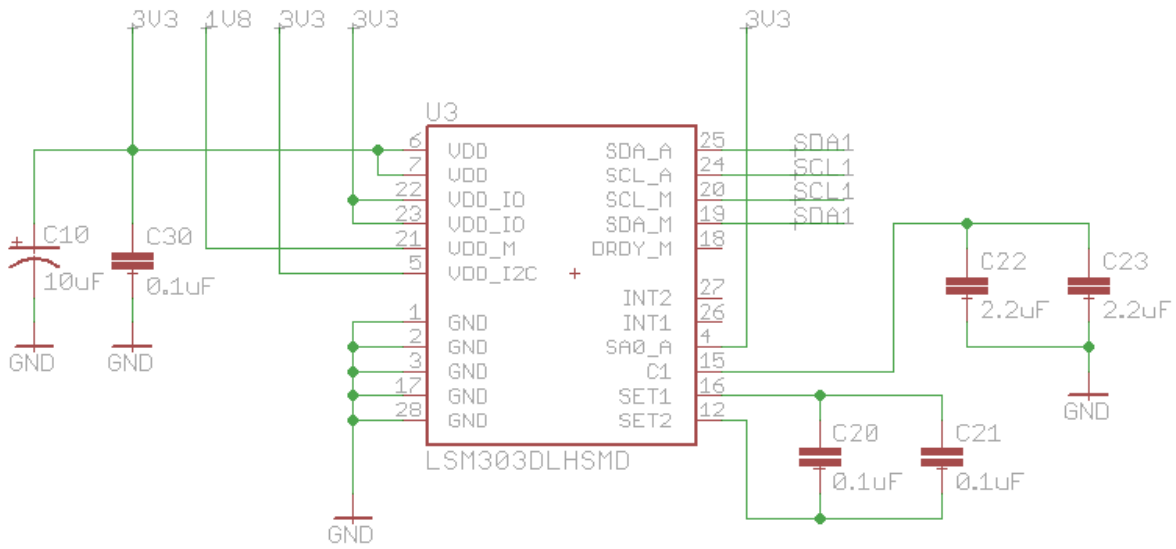
Tento startkit je konstruhován k snadnému propojení s *STM32VL discovery*, který programovat postupem uvedeným v 2. Díle skript. Proto stačí jen stručně uvést popis hw a zdrojové kódy sw.

IMU Discovery Lite

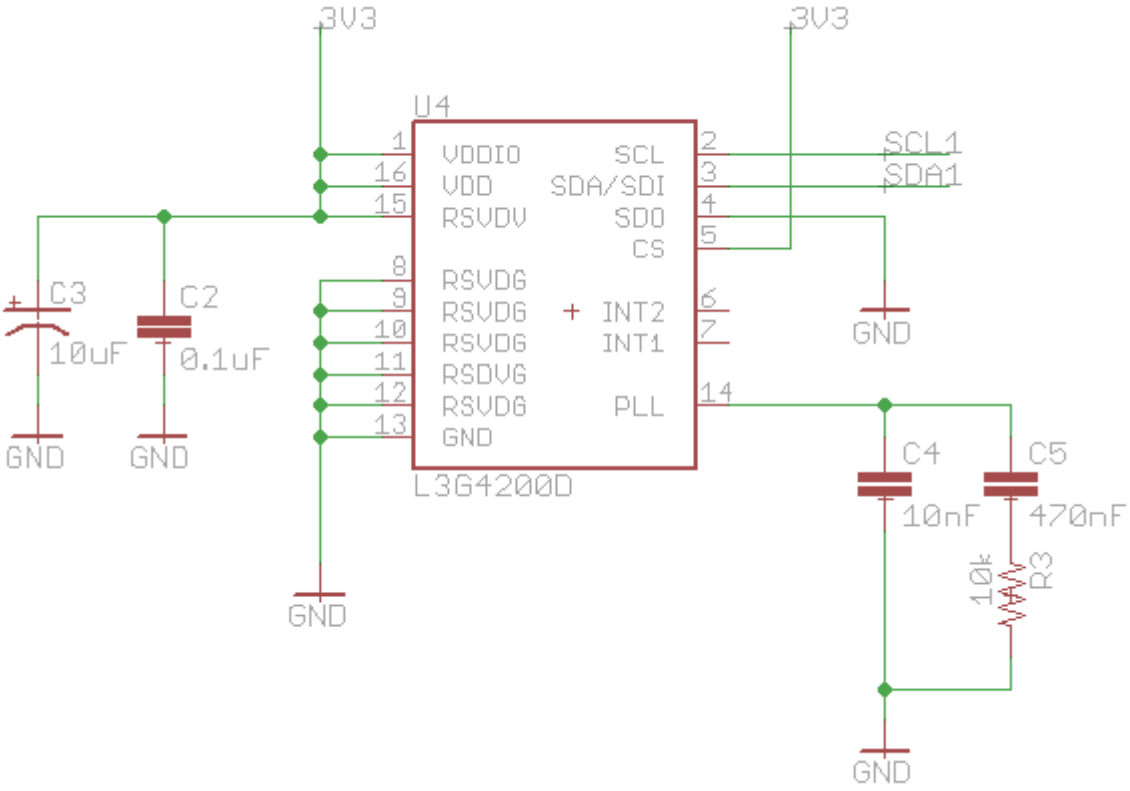


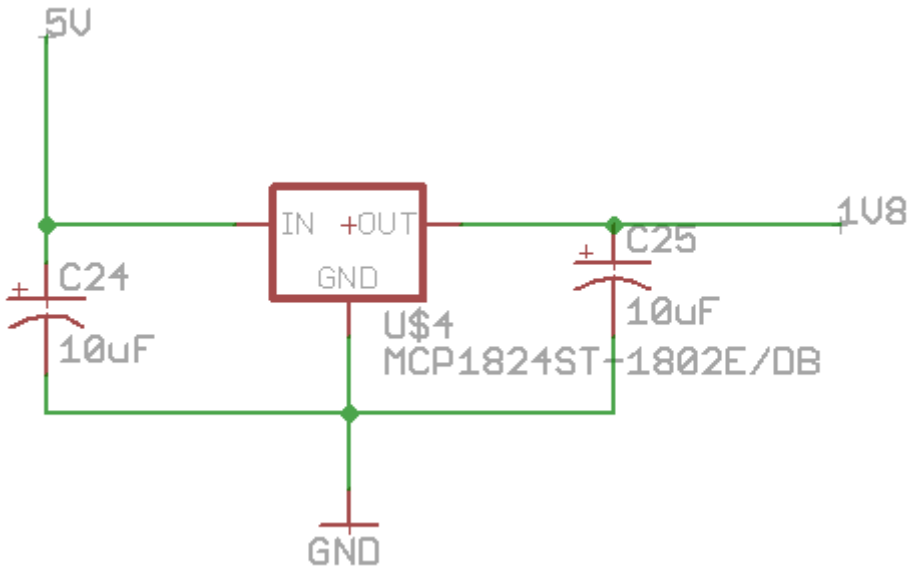
ST LSM303DLH:	3-osý akcelerometr, 3-osý magnetometr
ST L3G4200D:	3-osý gyroskop
BOSCH BMP085:	teploměr s tlakoměrem 300 až 1100 hPa

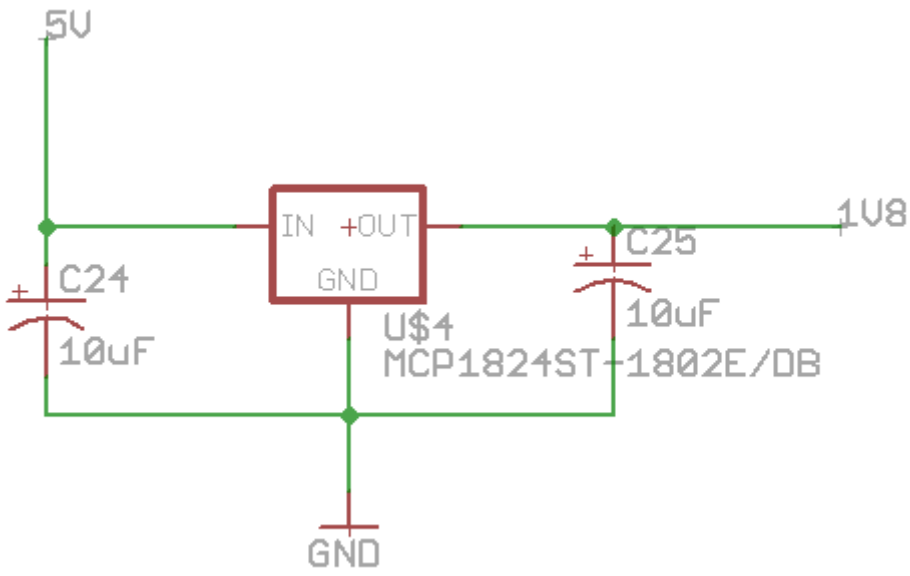
STM LSM303DLH 3-osý akcelerometr, 3-osý magnetometr



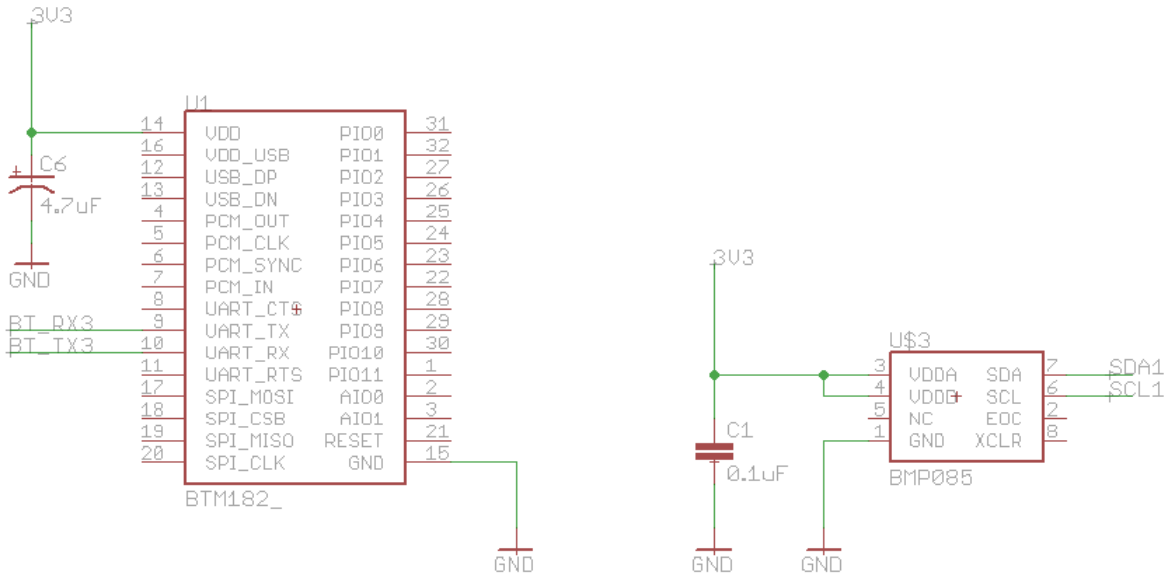
STM L3G4200D 3-osý gyroskop


3V3
3V3
U4
VDDIO
VDD
RSVDV
RSVDG
RSVDG
RSVDG
RSDVG
RSVDG
GND
SCL
SDA/SDI
SDO
CS
INT2
INT1
PLL
L3G4200D
SCL1
SDA1
GND
C3
10uF
C2
0.1uF
GND
GND
GND
C4
10nF
C5
470nF
10k
R3
GND

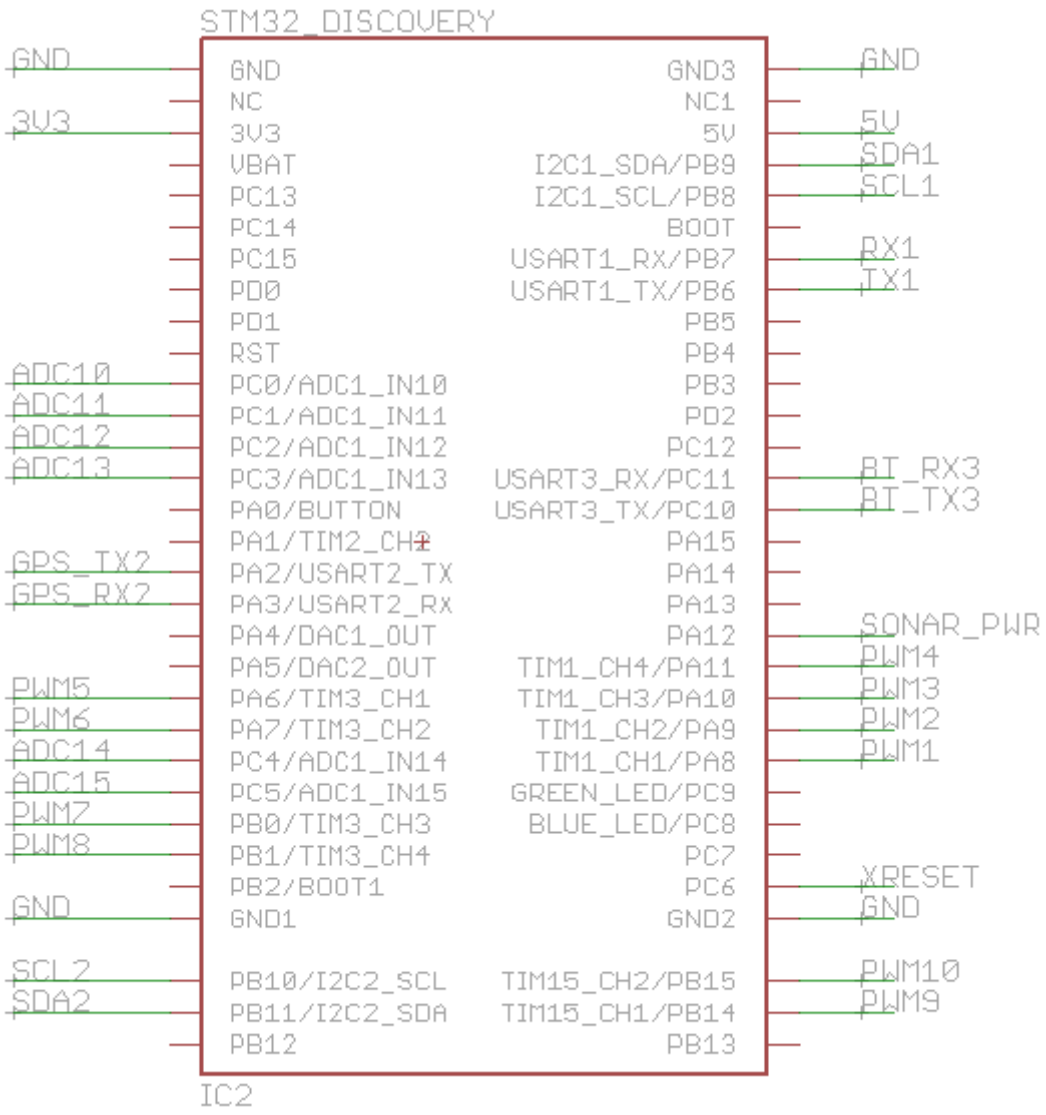

5V
IN
+OUT
GND
1V8
C24
10uF
C25
10uF
U$4
MCP1824ST-1802E/DB
GND

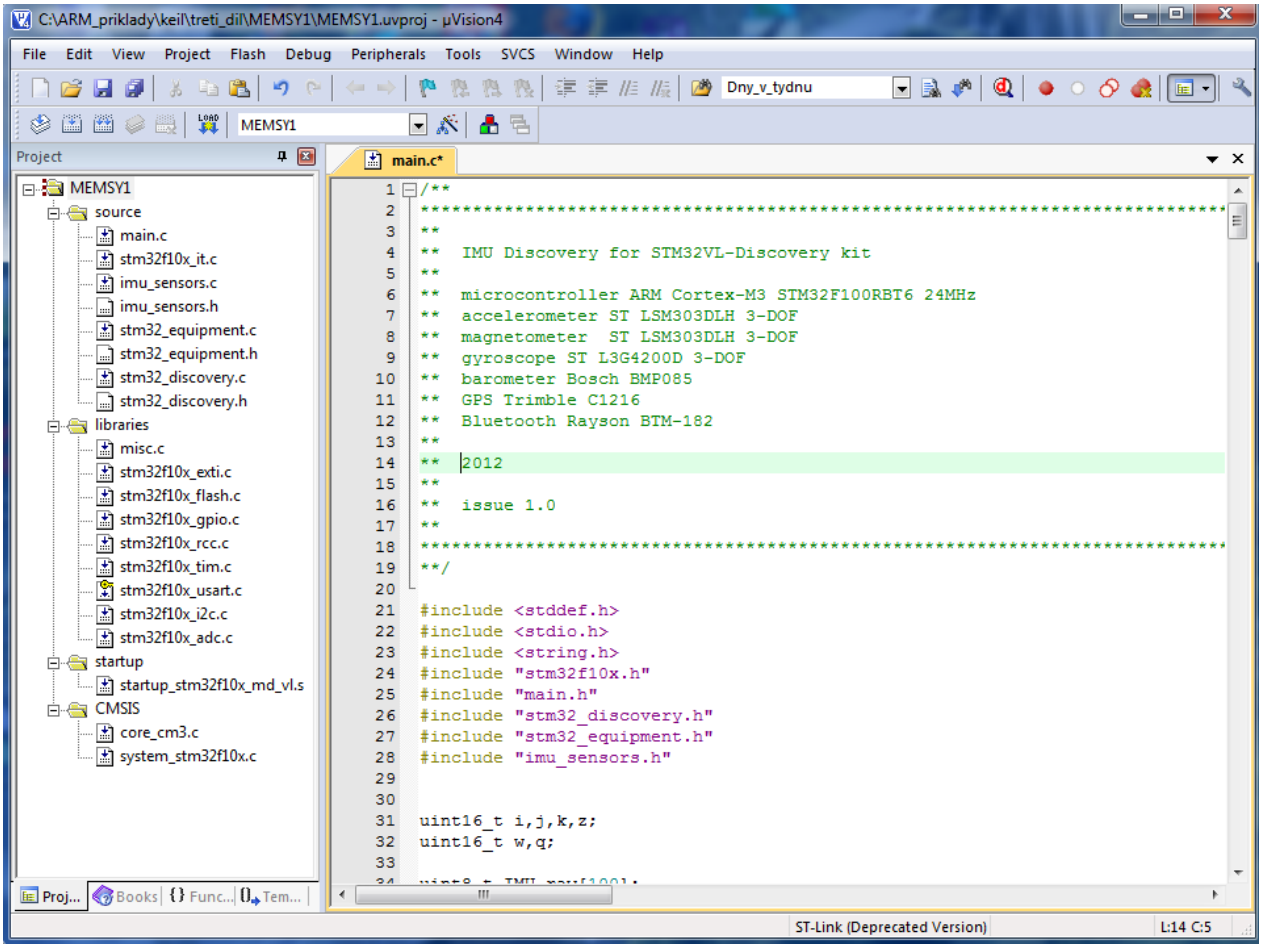


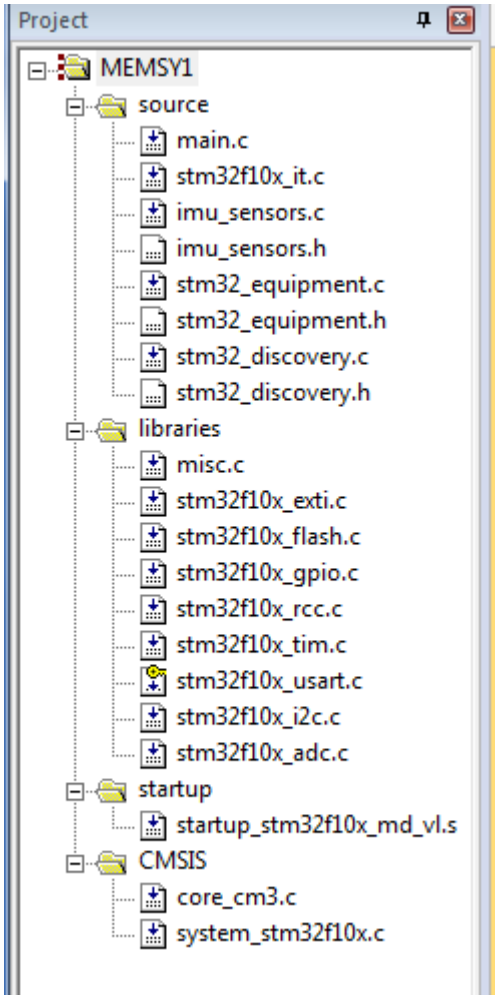
Bosch BMP085



A připojení k *STM32VL Discovery*


STM32_DISCOVERY
GND
3V3
ADC10
ADC11
ADC12
ADC13
GPS_TX2
GPS_RX2
PWM5
PWM6
ADC14
ADC15
PWM7
PWM8
GND
SCL2
SDA2
GND
NC
3V3
VBAT
PC13
PC14
PC15
PD0
PD1
RST
PC0/ADC1_IN10
PC1/ADC1_IN11
PC2/ADC1_IN12
PC3/ADC1_IN13
PA0/BUTTON
PA1/TIM2_CH2
PA2/USART2_TX
PA3/USART2_RX
PA4/DAC1_OUT
PA5/DAC2_OUT
PA6/TIM3_CH1
PA7/TIM3_CH2
PC4/ADC1_IN14
PC5/ADC1_IN15
PB0/TIM3_CH3
PB1/TIM3_CH4
PB2/BOOT1
GND1
PB10/I2C2_SCL
PB11/I2C2_SDA
PB12
GND3
NC1
5V
I2C1_SDA/PB9
I2C1_SCL/PB8
BOOT
USART1_RX/PB7
USART1_TX/PB6
PB5
PB4
PB3
PD2
PC12
USART3_RX/PC11
USART3_TX/PC10
PA15
PA14
PA13
PA12
TIM1_CH4/PA11
TIM1_CH3/PA10
TIM1_CH2/PA9
TIM1_CH1/PA8
GREEN_LED/PC9
BLUE_LED/PC8
PC7
PC6
GND2
TIM15_CH2/PB15
TIM15_CH1/PB14
PB13
GND
5V
SDA1
SCL1
RX1
TX1
BT_RX3
BT_TX3
SONAR_PWR
PWM4
PWM3
PWM2
PWM1
XRESET
GND
PWM10
PWM9
IC2


C:\ARM_priklady\keil\treti_dil\MEMSY1\MEMSY1.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Dny_v_tydnu
MEMSY1
Project
MEMSY1
source
main.c
stm32f10x_it.c
imu_sensors.c
imu_sensors.h
stm32_equipment.c
stm32_equipment.h
stm32_discovery.c
stm32_discovery.h
libraries
misc.c
stm32f10x_exti.c
stm32f10x_flash.c
stm32f10x_gpio.c
stm32f10x_rcc.c
stm32f10x_tim.c
stm32f10x_usart.c
stm32f10x_i2c.c
stm32f10x_adc.c
startup
startup_stm32f10x_md_vl.s
CMSIS
core_cm3.c
system_stm32f10x.c
Proj... Books Func... Tem...
main.c*
/**
**
** IMU Discovery for STM32VL-Discovery kit
**
** microcontroller ARM Cortex-M3 STM32F100RBT6 24MHz
** accelerometer ST LSM303DLH 3-DOF
** magnetometer ST LSM303DLH 3-DOF
** gyroscope ST L3G4200D 3-DOF
** barometer Bosch BMP085
** GPS Trimble C1216
** Bluetooth Rayson BTM-182
**
** 2012
**
** issue 1.0
**
**/
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include "stm32f10x.h"
#include "main.h"
#include "stm32_discovery.h"
#include "stm32_equipment.h"
#include "imu_sensors.h"
uint16_t i,j,k,z;
uint16_t w,q;
ST-Link (Deprecated Version)
L:14 C:5




```
/**  imu_sensors.h
**********************************
** I2C sensors
**
** STM LSM303DLH
** STM L3G4200D
** Bosch BMP085
**
**********************************
**/

#ifndef IMU_SENSORS_H_
#define IMU_SENSORS_H_

#define LSM303ACC_SLAVE_ADDRESS     0x33
#define LSM303MAG_SLAVE_ADDRESS     0x3D
#define L3G4200D_SLAVE_ADDRESS      0xD1
#define BMP085_SLAVE_ADDRESS  0xEF

/* LSM303DLH register address map */

#define CTRL_REG1_A                 0x20
#define CTRL_REG2_A                 0x21
```

```
#define CTRL_REG3_A                 0x22
#define CTRL_REG4_A                 0x23
#define CTRL_REG5_A                 0x24
#define HP_FILTER_RESET_A     0x25
#define REFERENCE_A                 0x26
#define STATUS_REG_A          0x27
#define OUT_X_L_A             0x28
#define OUT_X_H_A             0x29
#define OUT_Y_L_A             0x2A
#define OUT_Y_H_A             0x2B
#define OUT_Z_L_A             0x2C
#define OUT_Z_H_A             0x2D
#define INT1_CFG_A                  0x30
#define INT1_SOURCE_A         0x31
#define INT1_THS_A                  0x32
#define INT1_DURATION_A       0x33
#define INT2_CFG_A                  0x34
#define INT2_SOURCE_A         0x35
#define INT2_THS_A                  0x36
#define INT2_DURATION_A       0x37
#define CRA_REG_M             0x00
#define CRB_REG_M             0x01
#define MR_REG_M              0x02
#define OUT_X_H_M             0x03
#define OUT_X_L_M             0x04
#define OUT_Y_H_M             0x05
#define OUT_Y_L_M             0x06
#define OUT_Z_H_M             0x07
#define OUT_Z_L_M             0x08
#define SR_REG_Mg             0x09
#define IRA_REG_M             0x0A
#define IRB_REG_M             0x0B
#define IRC_REG_M             0x0C


/* L3G4200D register address map */

#define WHO_AM_I              0x0F
#define CTRL_REG1                   0x20
#define CTRL_REG2                   0x21
#define CTRL_REG3                   0x22
#define CTRL_REG4                   0x23
#define CTRL_REG5                   0x24
#define REFERENCE                   0x25
#define OUT_TEMP              0x26
#define STATUS_REG                  0x27
#define OUT_X_L               0x28
#define OUT_X_H               0x29
#define OUT_Y_L               0x2A
#define OUT_Y_H               0x2B
#define OUT_Z_L               0x2C
#define OUT_Z_H               0x2D
#define FIFO_CTRL_REG         0x2E
#define FIFO_SRC_REG          0x2F
#define INT1_CFG              0x30
#define INT1_SRC              0x31
#define INT1_TSH_XH           0x32
#define INT1_TSH_XL           0x33
#define INT1_TSH_YH           0x34
#define INT1_TSH_YL           0x35
```

```
#define INT1_TSH_ZH          0x36
#define INT1_TSH_ZL          0x37
#define INT1_DURATION        0x38


/* L3G4200D register address map */

#define CONTROL_REG                0xF4
#define OUT_MSB                    0xF6
#define OUT_LSB                    0xF7
#define OUT_XLSB             0xF8
#define AC1                        0xAA
#define AC2                        0xAC
#define AC3                        0xAE
#define AC4                        0xB0
#define AC5                        0xB2
#define AC6                        0xB4
#define B1                         0xB6
#define B2                         0xB8
#define MB                         0xBA
#define MC                         0xBC
#define MD                         0xBE



uint8_t LSM303ACC_Read(uint8_t);
void LSM303ACC_Write(uint8_t adresa, uint8_t data);
uint8_t LSM303MAG_Read(uint8_t adresa);
void LSM303MAG_Write(uint8_t adresa, uint8_t data);
uint8_t L3G4200D_Read(uint8_t);
void L3G4200D_Write(uint8_t adresa, uint8_t data);
uint8_t BMP085_Read_Byte(uint8_t);
uint16_t BMP085_Read(uint8_t);
int16_t BMP085_Read_Temperature(void);
uint32_t BMP085_Read_Pressure(void);
void BMP085_Write(uint8_t adresa, uint8_t data);

#endif
```

```
/** imusensors.c
************************************
** I2C sensors
**
** STM LSM303DLH
** STM L3G4200D
** Bosch BMP085
**
************************************
**/

#include "main.h"
#include "imu_sensors.h"

uint8_t Mag_MR_REG = 0;
extern uint16_t acc_x, acc_y, acc_z;
extern uint16_t mag_x, mag_y, mag_z;
extern uint16_t gyr_x, gyr_y, gyr_z;
extern uint8_t data1, data2, data3;
```

```
extern int16_t ac1, ac2, ac3, b1, b2, mb, mc, md;
extern uint16_t ac4, ac5, ac6;



uint8_t LSM303ACC_Read(uint8_t adresa)
{
        uint8_t Data = 0;
        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, LSM303ACC_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, LSM303ACC_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        Data = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);
        return(Data);
}


void LSM303ACC_Write(uint8_t adresa, uint8_t data)
{

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, LSM303ACC_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_SendData(I2C1, data);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTOP(I2C1, ENABLE);
}
```

```
uint8_t LSM303MAG_Read(uint8_t adresa)
{
        uint8_t Data = 0;
        while(I2C GetFlagStatus(I2C1, I2C FLAG BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, LSM303MAG_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, LSM303MAG_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        Data = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);
        return(Data);
}



void LSM303MAG_Write(uint8_t adresa, uint8_t data)
{

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, LSM303MAG_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_SendData(I2C1, data);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTOP(I2C1, ENABLE);
}


uint8_t L3G4200D_Read(uint8_t adresa)
{
        uint8_t Data = 0;
```

```
        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C CheckEvent(I2C1, I2C EVENT MASTER MODE SELECT));

        I2C_Send7bitAddress(I2C1, L3G4200D_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, L3G4200D_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        Data = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);

        return(Data);
}


void L3G4200D_Write(uint8_t adresa, uint8_t data)
{

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, L3G4200D_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_SendData(I2C1, data);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTOP(I2C1, ENABLE);
}

uint8_t BMP085_Read_Byte(uint8_t adresa)
{
        uint8_t Data = 0;

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
```

```
        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        Data = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);

        return(Data);
}


uint16_t BMP085_Read(uint8_t adresa)
{
        uint16_t Data = 0;

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        data1 = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);


        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
```

```
        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa+1);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        data2 = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);

        Data = data1<<8;
        Data = Data + data2;
        return(Data);
}



int16_t BMP085_Read_Temperature(void)
{
        int16_t Data = 0;

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, 0xF6);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        data1 = I2C_ReceiveData(I2C1);
```

```
        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);


        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, 0xF7);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        data2 = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);

        Data = data1<<8;
        Data = Data + data2;
        return(Data);
}

uint32_t BMP085_Read_Pressure(void)
{
        uint32_t Data = 0;

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, 0xF6);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
```

```
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        data1 = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);


        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, 0xF7);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
        data2 = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);


        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, 0xF8);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){};
```

```
        data3 = I2C_ReceiveData(I2C1);

        I2C_AcknowledgeConfig(I2C1, DISABLE);
        I2C_GenerateSTOP(I2C1, ENABLE);

        Data = data1;
        Data = Data<<8;
        Data |= data2;
        Data = Data<<8;
        Data |= data3;
        return(Data);
}


void BMP085_Write(uint8_t adresa, uint8_t data)
{
        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
        I2C_GenerateSTART(I2C1, ENABLE);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2C1, BMP085_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2C1, adresa);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_SendData(I2C1, data);
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTOP(I2C1, ENABLE);
}
```

```
// main.h
#ifndef MAIN_H_
#define MAIN_H_

#include "stm32f10x_adc.h"
#include "stm32f10x_bkp.h"
#include "stm32f10x_can.h"
#include "stm32f10x_cec.h"
#include "stm32f10x_crc.h"
#include "stm32f10x_dac.h"
#include "stm32f10x_dbgmcu.h"
#include "stm32f10x_dma.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_flash.h"
#include "stm32f10x_fsmc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_i2c.h"
#include "stm32f10x_iwdg.h"
#include "stm32f10x_pwr.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_rtc.h"
#include "stm32f10x_sdio.h"
```

```
#include "stm32f10x_spi.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_wwdg.h"
#include "misc.h"


#endif /* MAIN_H_ */
```

```
/** main.c
*******************************************************************************
**
**
**  IMU Discovery for STM32VL-Discovery kit
**
**  microcontroller ARM Cortex-M3 STM32F100RBT6 24MHz
**  accelerometer ST LSM303DLH 3-DOF
**  magnetometer  ST LSM303DLH 3-DOF
**  gyroscope ST L3G4200D 3-DOF
**  barometer Bosch BMP085
**  GPS Trimble C1216
**  Bluetooth Rayson BTM-182
**
**  2012
**
**  issue 1.0
**
*******************************************************************************
**
**/

#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include "stm32f10x.h"
#include "main.h"
#include "stm32_discovery.h"
#include "stm32_equipment.h"
#include "imu_sensors.h"


uint16_t i,j,k,z;
uint16_t w,q;

uint8_t IMU_raw[100];
uint8_t retezec[15];
uint8_t SizeOfData;
uint8_t DataReady, DataReadyRaw;
int16_t ac1, ac2, ac3, b1, b2, mb, mc, md;
uint16_t ac4, ac5, ac6;
uint8_t oss;
uint8_t registr;

int32_t x1,x2,x3,b3,b5,b6,p,t;
uint32_t b4,b7;
```

```
uint8_t data1, data2, data3, help;
uint16_t acc_x, acc_y, acc_z;
int16 t acc x sign, acc y sign, acc z sign;
float acc_x_float, acc_y_float, acc_z_float;
uint16_t mag_x, mag_y, mag_z;
int16_t mag_x_sign, mag_y_sign, mag_z_sign;
float mag_x_float, mag_y_float, mag_z_float;
uint16_t gyr_x, gyr_y, gyr_z;
int16_t gyr_x_sign, gyr_y_sign, gyr_z_sign;
float gyr_x_float, gyr_y_float, gyr_z_float;
int16_t temperature,temperature2;
uint32_t pressure;

uint8_t RxBuffer[5];
uint8_t GPS[120];
uint8_t RxBufferGPS[120];
uint8_t TxBufferGPS[10];
uint8_t RxBufferSonar[5];
uint8_t Sonar[5];

uint8_t CounterBluetooth;
__IO uint8_t TxCounter;
__IO uint8_t RxCounterGPS;
__IO uint8_t TxCounterGPS;
uint8_t NbrOfDataToTransfer;
uint8_t NbrOfDataToReadGPS;
uint8_t rec_gps_flag;
uint8_t rec_gps_byte;
uint8_t rec_gps_valid;
uint8_t rec_gps_data_ready;
uint8_t rec_gps_running;
uint8_t SizeOfSentenceGPS;

__IO uint8_t RxCounterSonar;
uint8_t NbrOfDataToReadSonar;
uint8_t rec_sonar_flag;
uint8_t rec_sonar_byte;
uint8_t rec_sonar_valid;


void GetSensors(void);
void GetSensorsRaw(void);
void GetBMP085(void);




int main(void)
{

  NVIC_Inicializace();
  //C1216_XRESET_Inicializace();
  //Maxbotix_Power_Inicializace();
  USART1_Inicializace();
  USART2_Inicializace();
  USART3_Inicializace();
  //ADC_Inicializace();
  //TIM1_Inicializace();
  //TIM3_Inicializace();
```

```
  //TIM15_Inicializace();
  I2C_Inicializace();

  STM32_Discovery_LEDInit(LED3);
  STM32_Discovery_LEDInit(LED4);
  STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
  STM32_Discovery_LEDOff(LED3);
  STM32_Discovery_LEDOff(LED4);

  //napájení 3V pro sonar Maxbotix EZ0
  GPIOA->BRR = GPIO_Pin_12;         //Zápis log.0
  for (w = 0; w < 65000; w++);
  GPIOA->BSRR = GPIO_Pin_12;  //Zápis log.1

  for (q = 0; q < 50; q++)
      for (w = 0; w < 65000; w++);


  //100ms XRESET GPS C1216
  GPIOC->BSRR = GPIO_Pin_6;         //Zápis log.1
  for (q = 0; q < 20; q++)
      for (w = 0; w < 65000; w++);
  GPIOC->BRR = GPIO_Pin_6;          //Zápis log.0
  for (q = 0; q < 20; q++)
      for (w = 0; w < 65000; w++);
  GPIOC->BSRR = GPIO_Pin_6;         //Zápis log.1



  CounterBluetooth = 0;
  NbrOfDataToTransfer = 225;
  NbrOfDataToReadGPS = 225;
  RxCounterGPS = 0;
  rec_gps_flag = 0;           // vynulování příznaku záznamu dat z GPS,
čekání na znak "$"
  rec_gps_running = 0;
  rec_gps_valid = 0;          // ukončeno načítání z GPS, data jsou platná
  rec_gps_data_ready = 0;
  SizeOfSentenceGPS = 0;

  NbrOfDataToReadSonar = 4; // počet bytů - sonar
  RxCounterSonar = 0;
  rec_sonar_flag = 0;       // vynulování příznaku záznamu dat ze sonaru,
čekání na znak "R"
  rec_sonar_valid = 0;      // ukončeno čtení ze sonaru, data jsou platná

  i = 0;
  j = 0;
  k = 0;
  DataReady = 0;
  DataReadyRaw = 0;
  data1 = 0;
  data2 = 0;
  data3 = 0;
  CounterBluetooth = 0;
  registr = 0xFF;

  // Načtení registrů z tlakoměru BMP085
  //ac1 = BMP085_Read(AC1);
  //ac2 = BMP085_Read(AC2);
```

```
  //ac3 = BMP085_Read(AC3);
  //ac4 = BMP085_Read(AC4);
  //ac5 = BMP085_Read(AC5);
  //ac6 = BMP085 Read(AC6);
  //b1 = BMP085_Read(b1);
  //b2 = BMP085_Read(b2);
  //mb = BMP085_Read(MB);
  //mc = BMP085_Read(MC);
  //md = BMP085_Read(MD);
  //oss = 3;                        // oversampling setting: ultrahigh
resolution = 3


  LSM303ACC_Write(CTRL_REG1_A, 0x27);

  LSM303MAG_Write(CRA_REG_M, 0x14);
  LSM303MAG_Write(CRB_REG_M, 0x20);
  LSM303MAG_Write(MR_REG_M, 0x00);

  L3G4200D_Write(CTRL_REG1, 0x0F);
  registr = L3G4200D_Read(CTRL_REG4);
  temperature2 = L3G4200D_Read(OUT_TEMP);


/********************** M A I N  L O O P ********************/

  while (1)
  {
      z++;
      if (z >= 100)
      {
            z = 0;
            STM32_Discovery_LEDToggle(LED3);

            //GetBMP085();
            //GetSensors();
            GetSensorsRaw();
      }


// načtení dat z GPS, věta uložena ve stringu GPS
      if (rec_gps_valid == 1)
      {
        for (i = 0; i <= SizeOfSentenceGPS; i++)
        GPS[i] = RxBufferGPS[i];
        GPS[SizeOfSentenceGPS] = 0x0A;
        rec_gps_valid = 0;
        rec_gps_data_ready = 1;
      }


// načtení dat ze sonaru, data jsou uložena ve stringu Sonar
      if (rec_sonar_valid == 1)
        {
              for (i = 0; i <= NbrOfDataToReadSonar; i++)
                    Sonar[i] = RxBufferSonar[i];
        }


// odeslání dat ze senzorů přes Bluetooth
```

```
        if (DataReadyRaw == 1)
            {
                   DataReadyRaw = 0;
                   USART ITConfig(USART3, USART IT TXE, ENABLE);  // odešle
data přes Bluetooth
            }

  }
}


/********************** E N D  O F  L O O P ********************/



  void GetSensors(void)
  {
                  acc_x = LSM303ACC_Read(OUT_X_H_A)<<8;
                  acc_x = acc_x + LSM303ACC_Read(OUT_X_L_A);
                  acc_x_sign = (int16_t)acc_x;

                  acc_y = LSM303ACC_Read(OUT_Y_H_A)<<8;
                  acc_y = acc_y + LSM303ACC_Read(OUT_Y_L_A);
                  acc_y_sign = (int16_t)acc_y;

                  acc_z = LSM303ACC_Read(OUT_Z_H_A)<<8;
                  acc_z = acc_z + LSM303ACC_Read(OUT_Z_L_A);
                  acc_z_sign = (int16_t)acc_z;


                  mag_x = LSM303MAG_Read(OUT_X_H_M)<<8;
                  mag_x = mag_x + LSM303MAG_Read(OUT_X_L_M);
                  mag_x_sign = (int16_t)mag_x;

                  mag_y = LSM303MAG_Read(OUT_Y_H_M)<<8;
                  mag_y = mag_y + LSM303MAG_Read(OUT_Y_L_M);
                  mag_y_sign = (int16_t)mag_y;

                  mag_z = LSM303MAG_Read(OUT_Z_H_M)<<8;
                  mag_z = mag_z + LSM303MAG_Read(OUT_Z_L_M);
                  mag_z_sign = (int16_t)mag_z;

                  gyr_x = L3G4200D_Read(OUT_X_H)<<8;
                  gyr_x = gyr_x + L3G4200D_Read(OUT_X_L);
                  gyr_x_sign = (int16_t)gyr_x;

                  gyr_y = L3G4200D_Read(OUT_Y_H)<<8;
                  gyr_y = gyr_y + L3G4200D_Read(OUT_Y_L);
                  gyr_y_sign = (int16_t)gyr_y;

                  gyr_z = L3G4200D_Read(OUT_Z_H)<<8;
                  gyr_z = gyr_z + L3G4200D_Read(OUT_Z_L);
                  gyr_z_sign = (int16_t)gyr_z;


                  // přepočet
                  acc_x_float = 0.00549316*acc_x_sign;  //90°/16384 =
0.00549316
                  acc_y_float = 0.00549316*acc_y_sign;
```

```
                acc_z_float = 0.00549316*acc_z_sign;

                gyr_x_float = 0.00762939*gyr_x_sign;      //250°/32768 =
0.00762939
                gyr_y_float = 0.00762939*gyr_y_sign;
                gyr_z_float = 0.00762939*gyr_z_sign;


                // vytvoření stringu pro aplikaci GLUonCONFIG,
OpenPilotGCS
                IMU_raw[0] = 'T';
                IMU_raw[1] = 'P';
                IMU_raw[2] = ';';

                j = 3;
                k = 0;
                sprintf(retezec,"%f",acc_x_float);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = ';';
                j++;


                k = 0;
                sprintf(retezec,"%f",acc_y_float);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = ';';
                j++;

                k = 0;
                sprintf(retezec,"%f",acc_z_float);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = ';';
                j++;

                k = 0;
                sprintf(retezec,"%f",gyr_x_float);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = ';';
                j++;
```

```
                k = 0;
                sprintf(retezec,"%f",gyr_y_float);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = ';';
                j++;

                k = 0;
                sprintf(retezec,"%f",gyr_z_float);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = 0x0D; j++;
                IMU_raw[j] = 0x0A; j++;

                SizeOfData = j;

                DataReady = 1;
}



void GetSensorsRaw(void)
 {
                acc_x = LSM303ACC_Read(OUT_X_H_A)<<8;
                acc_x = acc_x + LSM303ACC_Read(OUT_X_L_A);
                acc_x_sign = (int16_t)acc_x;

                acc_y = LSM303ACC_Read(OUT_Y_H_A)<<8;
                acc_y = acc_y + LSM303ACC_Read(OUT_Y_L_A);
                acc_y_sign = (int16_t)acc_y;

                acc_z = LSM303ACC_Read(OUT_Z_H_A)<<8;
                acc_z = acc_z + LSM303ACC_Read(OUT_Z_L_A);
                acc_z_sign = (int16_t)acc_z;


                mag_x = LSM303MAG_Read(OUT_X_H_M)<<8;
                mag_x = mag_x + LSM303MAG_Read(OUT_X_L_M);
                mag_x_sign = (int16_t)mag_x;

                mag_y = LSM303MAG_Read(OUT_Y_H_M)<<8;
                mag_y = mag_y + LSM303MAG_Read(OUT_Y_L_M);
                mag_y_sign = (int16_t)mag_y;

                mag_z = LSM303MAG_Read(OUT_Z_H_M)<<8;
                mag_z = mag_z + LSM303MAG_Read(OUT_Z_L_M);
                mag_z_sign = (int16_t)mag_z;

                gyr_x = L3G4200D_Read(OUT_X_H)<<8;
                gyr_x = gyr_x + L3G4200D_Read(OUT_X_L);
                gyr_x_sign = (int16_t)gyr_x;
```

```
                    gyr_y = L3G4200D_Read(OUT_Y_H)<<8;
                    gyr_y = gyr_y + L3G4200D_Read(OUT_Y_L);
                    gyr_y_sign = (int16_t)gyr_y;

                    gyr_z = L3G4200D_Read(OUT_Z_H)<<8;
                    gyr_z = gyr_z + L3G4200D_Read(OUT_Z_L);
                    gyr_z_sign = (int16_t)gyr_z;


                    // vytvoření stringu pro aplikaci GLUonCONFIG,
OpenPilotGCS
                    IMU_raw[0] = 'T';
                    IMU_raw[1] = 'R';
                    IMU_raw[2] = ';';

                    j = 3;
                    k = 0;
                    sprintf(retezec,"%i",acc_x_sign);
                    while (retezec[k] != 0)
                    {
                          IMU_raw[j] = retezec[k];
                          j++;
                          k++;
                    }
                    IMU_raw[j] = ';';
                    j++;


                    k = 0;
                    sprintf(retezec,"%i",acc_y_sign);
                    while (retezec[k] != 0)
                    {
                          IMU_raw[j] = retezec[k];
                          j++;
                          k++;
                    }
                    IMU_raw[j] = ';';
                    j++;

                    k = 0;
                    sprintf(retezec,"%i",acc_z_sign);
                    while (retezec[k] != 0)
                    {
                          IMU_raw[j] = retezec[k];
                          j++;
                          k++;
                    }
                    IMU_raw[j] = ';';
                    j++;

                    k = 0;
                    sprintf(retezec,"%i",gyr_x_sign);
                    while (retezec[k] != 0)
                    {
                          IMU_raw[j] = retezec[k];
                          j++;
                          k++;
                    }
                    IMU_raw[j] = ';';
                    j++;
```

```
                k = 0;
                sprintf(retezec,"%i",gyr_y_sign);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = ';';
                j++;

                k = 0;
                sprintf(retezec,"%i",p);
                while (retezec[k] != 0)
                {
                      IMU_raw[j] = retezec[k];
                      j++;
                      k++;
                }
                IMU_raw[j] = 0x0D; j++;
                IMU_raw[j] = 0x0A; j++;

                SizeOfData = j;

                DataReadyRaw = 1;
   }



  void GetBMP085(void)
  {
       BMP085_Write(0xF4, 0x2E);         // čtení teploty

           for (w = 0; w < 65000; w++);

     temperature = BMP085_Read_Temperature();


     BMP085_Write(0xF4, 0xF4);           // čtení tlaku
       for (q = 0; q < 5; q++)
           for (w = 0; w < 65000; w++);

     pressure = BMP085_Read_Pressure();

     // pozor! získání hodnot teploty a tlaku z BMP085 trvá
několikanásobně déle než z acc, mag a gyr



     //výpočet teploty
     x1 = (temperature - ac6)*ac5/32768;
     x2 = mc*2048/(x1+md);
     b5 = x1+x2;
     t = (b5+8)/16;

     //výpočet tlaku
     b6 = b5-4000;
     x1 = (b2*(b6*b6/4096))/2048;
```

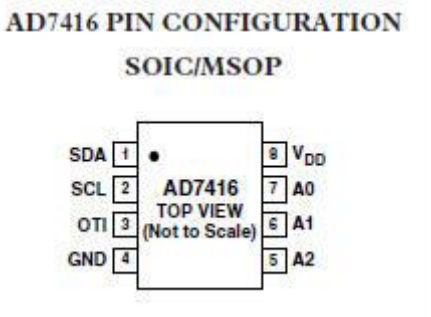
```
      x2 = ac2*b6/2048;
      x3 = x1+x2;
      b3 = (((ac1*4+x3)<<oss)+2)/4;
      x1 = ac3*b6/8192;
      x2 = (b1*(b6*b6/4096))/65536;
      x3 = ((x1+x2)+2)/4;
      b4 = ac4*(uint32_t)(x3+32768)/32768;
      b7 = ((uint32_t)pressure-b3)*(50000>>oss);
      if (b7<0x80000000)
            {p = (b7*2)/b4;}
      else
            {p = (b7/b4)*2;}
      x1 = (p/256)*(p/256);
      x1 = (x1*3038)/65536;
      x2 = (-7357*p)/65536;
      p = p+(x1+x2+3791)/16;
  }
```

```
void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
  while(1)
  {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
   __assert_func (file, line, NULL, failedexpr);
}
```

3.4 Práce s čidly pomocí startkitů STM

3.4.1 Čidlo teploty

Jde např. o STTS75 a STDS75 s rozlišením 9bitů (default) až 12bitů (rozlišení můžeme volit, lepší rozlišení je však vykoupeno delší dobou měření teploty) , STLM75 a STCM75 s rozlišením 9bitů v pouzdrech SO8



Pokud na A0,A1,A2 připojíme úroveň 0, bude bázová adresa tohoto obvodu pro zápis 0x90, pro čtení 0x91. Tento obvod, má stejnou sadu příkazů jako obvody AD7417 a AD7418, které mohou pracovat nejen jako čidla teploty, ale i jako 10bitové A/D převodníky. Proto nejprve musíme do těchto obvodů zapsat tzv. *Adress Pointer Register*. V případě, kdy požadujeme na obvodu, aby pracoval jako čidlo teploty bude tato hodnota 00000000.

Proto nejprve musí být na I2C sběrnici posloupnost následujících akcí: START, poslání 0x90 (protože budeme zapisovat), poslání 0x00 (Adres Pointer Register) a STOP. Poté již může následovat přečtení hodnoty naměřené teploty a to ve dvou bytech dat, tj. následující akce na sběrnici I2C: START, poslání 0x91 (protože budeme číst), přečtení horního byte dat (více významnější byte), přečtení dolního byte dat (méně vyznamější byte) a STOP. Tím jsme získali obsah Temperature Value Register

Table III. Temperature Value Register

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6
MSB	B8	B7	B6	B5	B4	B3	B2	B1	LSB

Informaci o teplotě s přesností 0,25 ^{o} C nese horních 10 bitů. Obsah dolních bitů D5, D4, … D0 nemá žádný význam. Následující tabulka pak ukazuje převod mezi 10bitovým datovým údajem a hodnotou teploty v ^{o} C.

Table IV. Temperature Data Format

Temperature	Digital Output
–128°C	10 0000 0000
–125°C	10 0000 1100
–100°C	10 0111 0000
–75°C	10 1101 0100
–50°C	11 0011 1000
–25°C	11 1001 1100
–0.25°C	11 1111 1111
0°C	00 0000 0000
+0.25°C	00 0000 0001
+10°C	00 0010 1000
+25°C	00 0110 0100
+50°C	00 1100 1000
+75°C	01 0010 1100
+100°C	01 1001 0000
+125°C	01 1111 0100
+127°C	01 1111 1100

Nejvíce významný bit je znaménkový – 0 je nezáporná čísla, 1 pro záporná. Pro záporná čísla je další bity v doplňkovém kódu.

Pozn. Pinově i funkčně kompatibilní (včetně I2C komunikace) s obvodem AD7416 je obvod LM75 firmy National Semiconductor. Jediný rozdíl je v tom, že LM75 měří teplotu s přesností 0,5 ^{o} C a informaci o teplotě nese 9 horních bitů Temperature Value Register.

1.12 TEMPERATURE REGISTER

(Read Only):

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	LSB	X	X	X	X	X	X	X

D0–D6: Undefined

D7–D15: Temperature Data. One LSB = 0.5°C. Two's complement format.

Algoritmus přečtění dat o teplotě z STTS75, STDS75, STLM75,STCM75 a AD7416 tedy bude

```
float ad7416_calculate()
{
int t1,t2;
float t;
i2c_start();
i2c_write( 0x90 ); // Address
i2c_write( 0x00 ); // Address pointer register
i2c_stop();

i2c_start();
i2c_write( 0x91 );
t1 = (int)i2c_read(0);  // horních osm bitů Temperature registru
t2 = (int)i2c_read(1);  // dolních osm bitů Temperature registru
i2c_stop();
t = (float)( ( (t2>>6)+(t1<<2) ) * 0.25 );
return t;
}
```

Neboli horních 8bitů posuneme o dva bity doleva, čímž dostaneme 10bitové číslo s dolními dvěma bity nulovými. Dolních 8 bitů posuneme o 6 bitů doprava. Tím přijdeme o 6dolních (nevýznamných) bitů z t2. Zůstanou nám tak 2 významné bity z t2. Po součtu již dostaneme v 10ti bitový údaj o teplotě, kde jeden bit odpovídá 0,25 ^{o} C, tak že po vynásobení 0,25 dostaneme v t hodnotu teploty ve stupních C.

Pozn.: Pro teploty větší nebo rovné nule obsahuje t1 rovnou údaj o teplotě s přesností 1 ^{o}C.

3.4.2 Čidlo tlaku

LPS331AP je kompaktní piezoresistivní čidlo absolutního tlaku (tj není rozdílový). Obsahuje monolitický citlivý prvek a IC interface převádějící informaci z tohoto prvku na digitální výstupní signal .

Figure 3. LPS331AP electrical connection

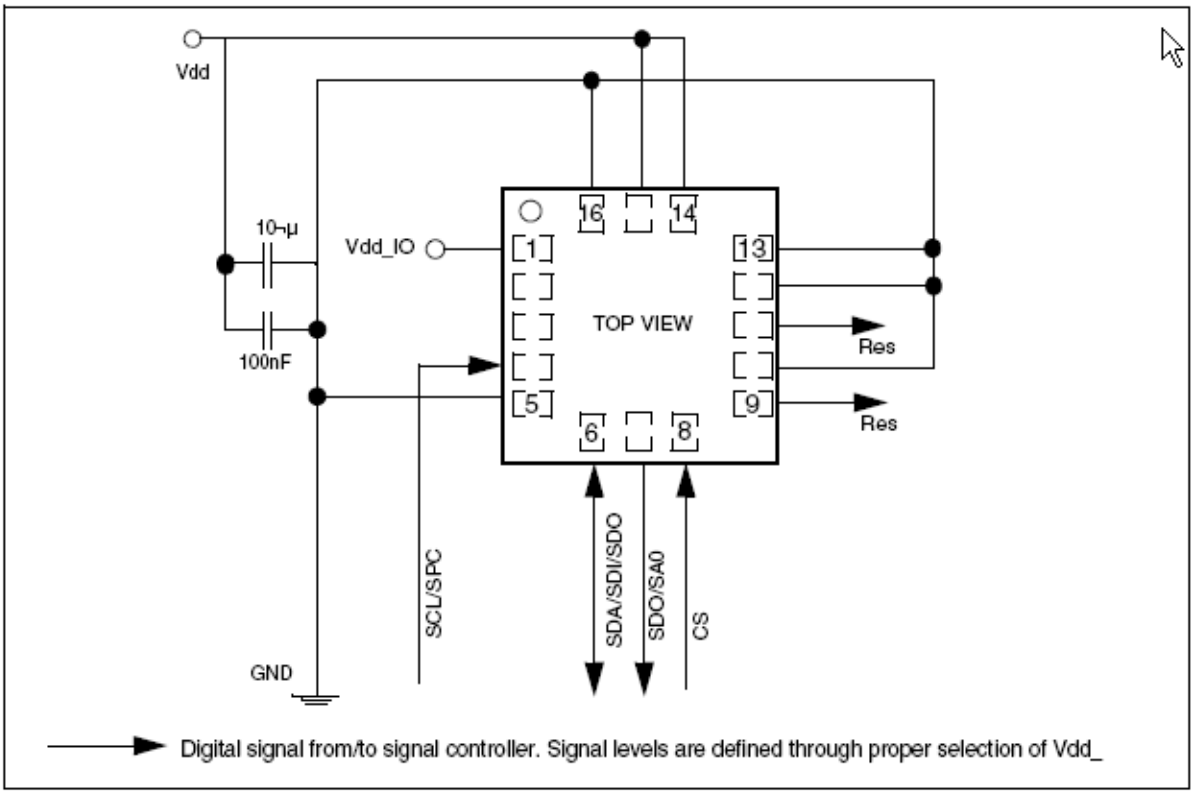
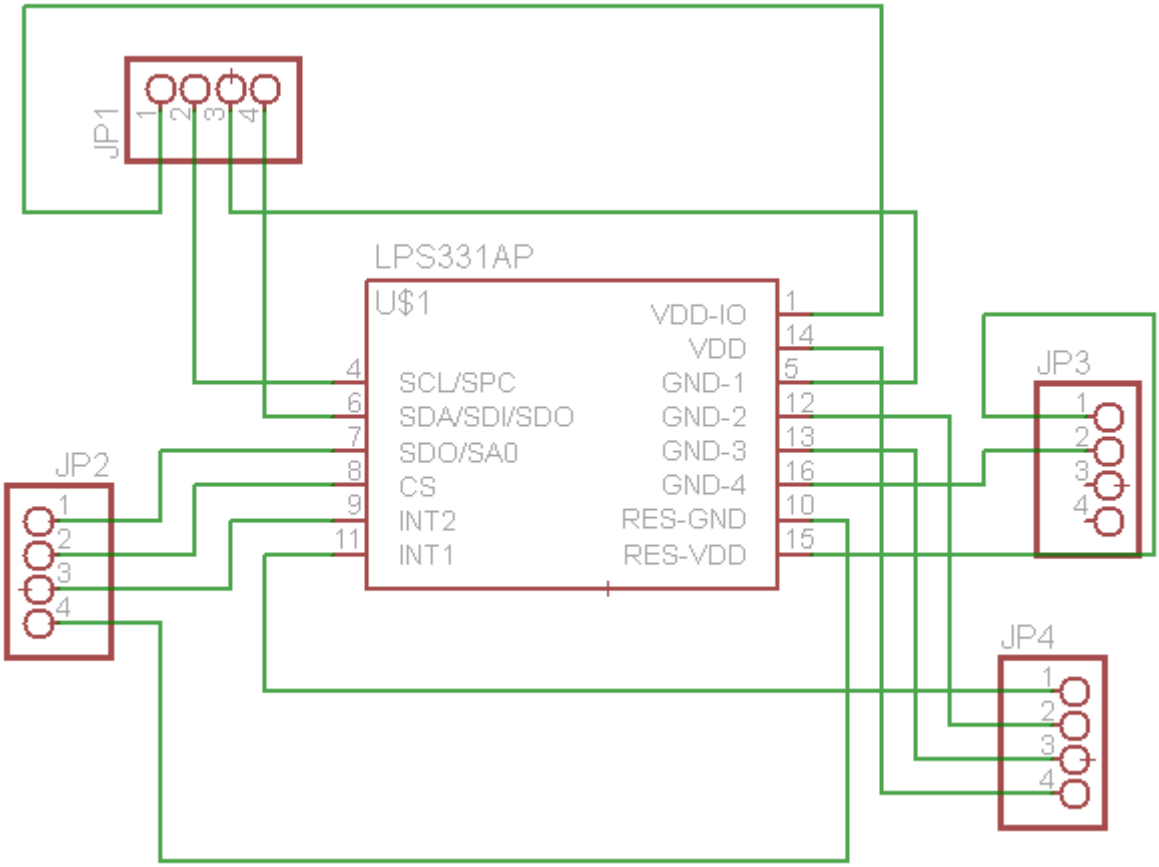
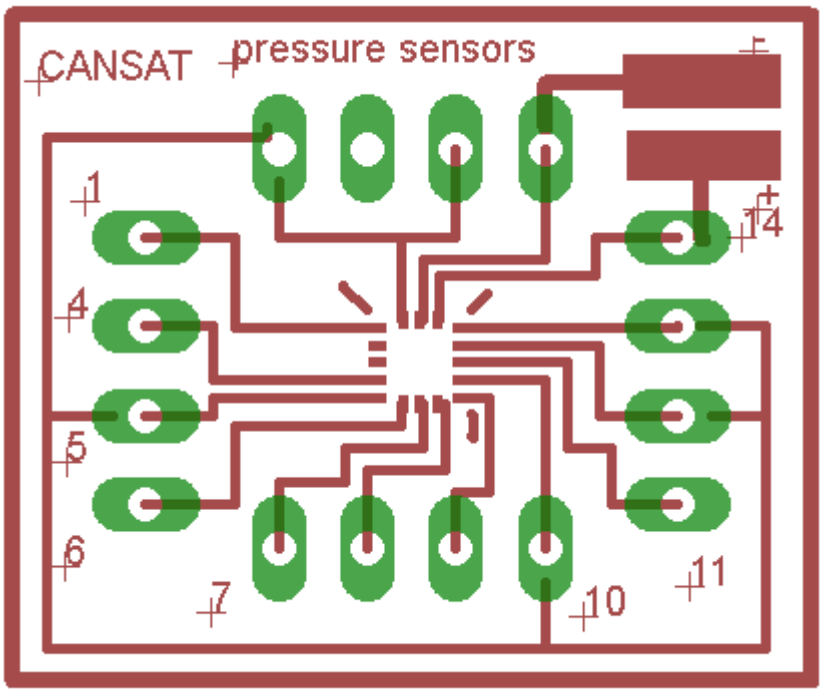


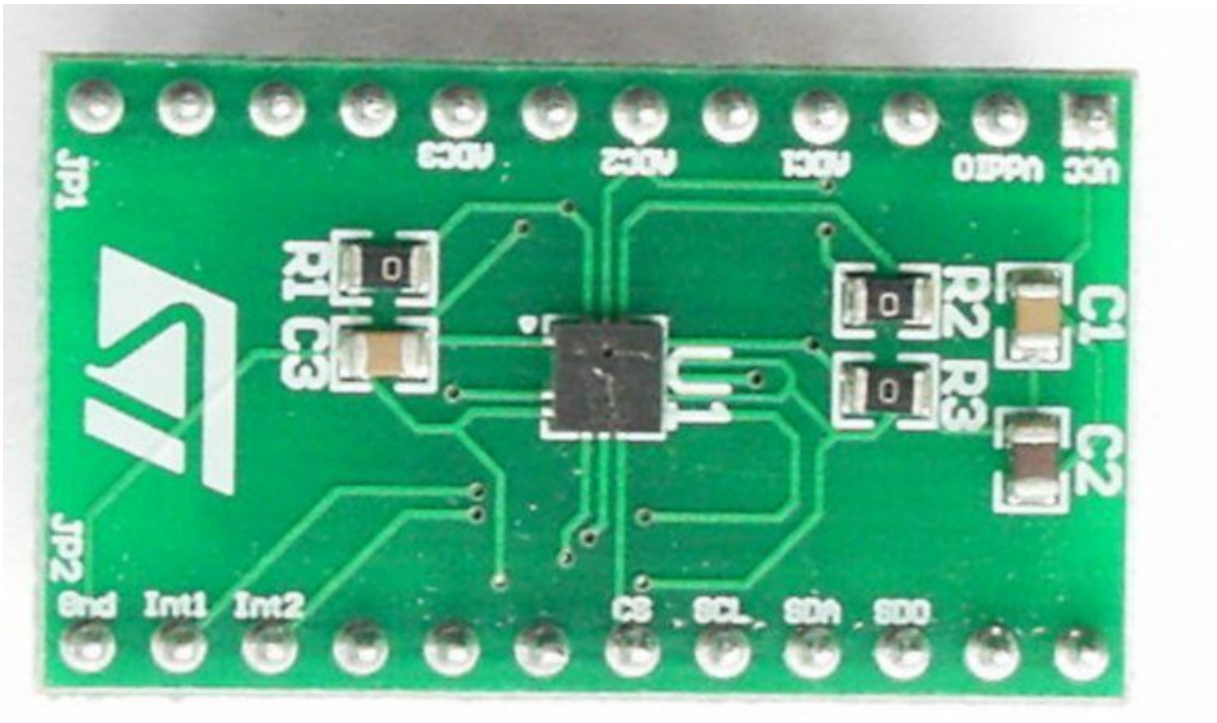
Table 9. SAD+Read/Write patterns

Command	SAD[6:1]	SAD[0] = SA0	R/W	SAD+R/W
Read	101110	0	1	10111001 (B9h)
Write	101110	0	0	10111000 (B8h)
Read	101110	1	1	10111011 (BBh)
Write	101110	1	0	10111010 (BAh)

Rovněž čidla tlaku LPS331AP lze ke sběrnici I2C připojit dvě, lišící se logickými úrovněmi na SA0


JP1
1
2
3
4
LPS331AP
U$1
VDD-IO
VDD
GND-1
GND-2
GND-3
GND-4
RES-GND
RES-VDD
SCL/SPC
SDA/SDI/SDO
SDO/SA0
CS
INT2
INT1
JP2
JP3
JP4


CANSAT
pressure sensors
1
4
5
6
7
10
11
14



Programová obsluha pro STM32F103 – soubory **LPS331AP.c** a **LPS331.h** :

```
/******************** ********************
* File Name          : lps331ap.h
* Description        : Descriptor Header for lps331ap file
* HISTORY:
*                      STM32F103
********************************************************************************
****/

/* Define to prevent recursive inclusion -------------------------------
---*/
#ifndef __LPS331AP_H
#define __LPS331AP_H

/* Includes ------------------------------------------------------------------
---*/
#include "stm32f10x.h"

/* Exported types ------------------------------------------------------------
---*/
/* Exported constants --------------------------------------------------------
---*/
#define BIT(x) ( 1<<(x) )

#define VER                             0
#define DEV                             1
#define ZOFF                            2
#define ZON                             4
#define START                           5
#define STOP                            6
#define R                               7
```

```
#define W                               8
#define DEBUG                           9
#define LIST                            10
#define SINGLE                          11
#define ECHOON                          12
#define ECHOOFF                         13
#define LISTDEV                         14
#define DBRESET                         15

#define STATUS_REG                      0x27
#define P_OUT_XL                        0x28
#define P_OUT_L                         0x29
#define P_OUT_H                         0x2a
#define T_OUT_L                         0x2b
#define T_OUT_H                         0x2c

#define REF_P_XL                        0x08
#define REF_P_L                         0x09
#define REF_P_H                         0x0a
#define REF_T_L                         0x0b
#define REF_T_H                         0x0c



#define TDA                             0x00
#define PDA                             0x01

#define CTRL_REG1                       0x20
#define CTRL_REG2                       0x21
#define CTRL_REG3                       0x22

//          INT1_SRC
#define INT_SRC_REG                     0x24
#define INT1_SRC_IA                     BIT(2)


/* Exported macro ------------------------------------------------------
---*/

/* Exported functions --------------------------------------------------
---*/
void lps331ap_AppTick(void);

#endif /* __LPS331AP_H */

/******************* LPS331.h  *****END OF FILE****/
```

```
/******************* ********************
* File Name          : lps331ap.c
*                      STM32F103
********************************************************************************
****/

/* Includes ------------------------------------------------------------
---*/
#include "lps331ap.h"
#include "led.h"
#include "command_interpreter.h"
```

```
#include "string.h"
#include "usb_lib.h"
#include "usb_desc.h"
#include "usb_pwr.h"
#include "spi_mems.h"
#include "ctype.h"
#include "utility.h"
#include "interruptHandler.h"
#include "hw_config.h"
#include "fw_selector.h"
#include <stdio.h>


/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
#define ValBit(VAR,Place)         (VAR & (1<<Place))
/* Private variables ---------------------------------------------------------*/
static uint8_t usbBuff[MAX_COMMAND_LENGTH];
static uint8_t state1 = STOP;
static uint8_t state1Old = STOP;

static uint8_t interruptOne = 0;
static uint8_t interruptTwo = 0;

static uint8_t zoffFlag = 0;
static uint8_t echoOnFlag = 0;

extern uint8_t dbSelected;

/* Private function prototypes -----------------------------------------------*/
void lps331ap_AppTick(void);
static uint8_t Decod();
static void state1Machine(void);
static void Version(void);
static void Device(void);
static void Zoff(void);
static void Zon(void);
static void Start(void);
static void Stop(void);
static void ReadReg(void);
static void WriteReg(void);
static void Debug(void);
static void InterruptReceived(uint8_t interrupt);
static void DbReset(void);

/* Private functions ---------------------------------------------------------*/


/*******************************************************************************
*****
* Function Name  : lps331ap_AppTick
* Description    : lps331ap Application Tick Function
* Input          : None
```

```
* Output          : None
* Return          : None
*******************************************************************************
****/
void lps331ap_AppTick(void) {
  EKSTM32_LEDOn(LED3);
  if(newCommandAvailable(usbBuff)) {

    Decod();
  }

  state1Machine();

}

/*******************************************************************************
*****
* Function Name  : Decod
* Description    : decode the incoming USB data
**             :  decodification of RTX vector
**           :  *start/r:            :starts sending data
**           :  *stop/r:             :stops sending data
**           :  *ver/r:              :sends firmware version number
**              :  *dev/r:             :sends device name
**           :  *wxdd/r:           :writes in register at address xx:(0xadr)
--> the value data dd:(0xdata)
**           :  *rxx/r:              :reads register at address xx:(0xadr)
**           :  *zon/r:                :activates tristate1 mode
**              :  *zoff/r:            :activates connections with devices
**
**   Realizes the command interpreter (for general purpose commands).
* Input           : None
* Output          : None
* Return          : None
*******************************************************************************
****/
static uint8_t Decod(){

  if(strcmp((char*)&usbBuff[1],"VER")==0) {
    state1Old = state1;
    state1 = VER;
  }
  else if(strcmp((char*)&usbBuff[1],"DEV")==0) {
    state1Old = state1;
    state1 = DEV;
  }
  else if(strcmp((char*)&usbBuff[1],"ZOFF")==0) {
    state1Old = state1;
    state1 = ZOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"ZON")==0) {
    state1 = ZON;
  }
  else if(strcmp((char*)&usbBuff[1],"START")==0) {
    state1Old = state1;
    state1 = START;
  }
  else if(strcmp((char*)&usbBuff[1],"STOP")==0) {
    state1Old = state1;
    state1 = STOP;
```

```
  }
  else if(strncmp((char*)&usbBuff[1],"PR", 2)==0) {
    state1Old = state1;
    state1 = R;
  }
  else if(strncmp((char*)&usbBuff[1],"PW", 2)==0) {
    state1Old = state1;
    state1 = W;
  }
  else if(strcmp((char*)&usbBuff[1],"DEBUG")==0) {
    state1 = DEBUG;
  }
  else if(strcmp((char*)&usbBuff[1],"LIST")==0) {
    state1Old = state1;
    state1 = LIST;
  }
  else if(strcmp((char*)&usbBuff[1],"SINGLE")==0) {
    state1Old = state1;
    state1 = SINGLE;
  }
  else if(strcmp((char*)&usbBuff[1],"ECHOON")==0) {
    state1Old = state1;
    state1 = ECHOON;
  }
  else if(strcmp((char*)&usbBuff[1],"ECHOOFF")==0) {
    state1Old = state1;
    state1 = ECHOOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"LISTDEV")==0) {
    state1Old = state1;
    state1 = LISTDEV;
  }
  else if(strcmp((char*)&usbBuff[1],"DBRESET")==0) {
    state1Old = state1;
    state1 = DBRESET;
  }

 return state1;
}

/*************************************************************************
*****
* Function Name  : state1Machine
* Description    : state1Machiine is called from the aAppTick main loop
*                  It calls functions that implement commands selected
*                  It implements state1 machine for commands interpreter.
*                  If Set of available commands is extended with device
specific commands
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void state1Machine(void) {

  switch(state1) {
  case VER:
    Version();
    state1 = state1Old;
    break;
```

```
  case DEV:
    Device();
    state1 = state1Old;
    break;
  case ZOFF:
    Zoff();
    state1 = STOP;
    state1Old = STOP;
    break;
  case ZON:
    Zon();
    break;
  case START:
    if(zoffFlag) {
      Start();
    }
    break;
  case STOP:
    if(zoffFlag) {
      Stop();
    }
    break;
  case R:
    if(zoffFlag) {
      ReadReg();
      state1 = state1Old;
    }
    break;
  case W:
    if(zoffFlag) {
      WriteReg();
      state1 = state1Old;
    }
    break;
  case DEBUG:
    if(zoffFlag) {
      Debug();
    }
    break;
  case LIST:
    //It Prints the list of firmwares implemented
    PrintListOfFirmware();
    state1 = state1Old;
    break;
  case SINGLE:
    if(zoffFlag) {
      Debug();
      state1 = state1Old;
    }
    break;
  case ECHOON:
      echoOnFlag = 1;
      state1 = state1Old;
    break;
  case ECHOOFF:
      echoOnFlag = 0;
      state1 = state1Old;
    break;
  case LISTDEV:
    //It Prints the list of device supported
```

```
    PrintListOfDevice();
    state1 = state1Old;
    break;
  case DBRESET:
    //unselect the db
    DbReset();
    state1 = STOP;
    break;
  }

}

/*******************************************************************************
*****
* Function Name  : Version
* Description    : Function to send Firmware version to host by USB
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Version(void) {
 //eMotion firmware version
  EmotionVer();
}

/*******************************************************************************
*****
* Function Name  : Device
* Description    : Function to send the Device Product Code
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Device(void) {
  uint8_t buffer[]="LPS331AP\r\n";

  MEMS_Print(buffer, sizeof(buffer));
}

/*******************************************************************************
*****
* Function Name  : Zoff
* Description    : It initializes the peripherals according to the device
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zoff(void) {
  EKSTM32_LEDToggle(LED2);
  RegisterInterrupt(InterruptReceived);
  SPI_Mems_Init();
  zoffFlag = 1;
}


/*******************************************************************************
*****
```

```
* Function Name  : Zon
* Description    : It DeInitializes the peripherals according to the device
*                  Now the device is disconnected from the board
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zon(void) {
  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Pin = SPI_MEMS_PIN_SCK | SPI_MEMS_PIN_MISO |
SPI_MEMS_PIN_MOSI;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(SPI_MEMS_GPIO, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = SPI_MEMS_CS;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(SPI_MEMS_CS_GPIO, &GPIO_InitStructure);

  UnRegisterInterrupt();

  EKSTM32_LEDOn(LED2);

  zoffFlag = 0;

}

/*******************************************************************************
*****
* Function Name  : Start
* Description    : Function to Send the standard output data to USB
(associated to '*start' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by acceleration, interrupt and
*                  switch status.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Start(void) {
  uint8_t dataReady = 0;
  uint8_t buffer[17];

  dataReady = SPI_Mems_Read_Reg(STATUS_REG);
  if(ValBit(dataReady, PDA)) {
    EKSTM32_LEDToggle(LED1);

    buffer[0]   = 's';
    buffer[1]   = 't';

    buffer[2]   = SPI_Mems_Read_Reg( P_OUT_XL );  //reads pressure XLSB
    buffer[3]   = SPI_Mems_Read_Reg( P_OUT_L );  //reads pressure LSB
    buffer[4]   = SPI_Mems_Read_Reg( P_OUT_H );   //reads pressure MSB
    buffer[5]   = SPI_Mems_Read_Reg( T_OUT_L );  //reads temp LSB
    buffer[6]   = SPI_Mems_Read_Reg( T_OUT_H );  //reads temp MSB
    buffer[7]   = SPI_Mems_Read_Reg( REF_P_XL );  //reads ref XL
    buffer[8]   = SPI_Mems_Read_Reg( REF_P_L );  //reads ref L
    buffer[9]   = SPI_Mems_Read_Reg( REF_P_H);    //reads ref H
```

```
    buffer[10]   = SPI_Mems_Read_Reg( REF_T_L);   //reads temp ref L
    buffer[11]   = SPI_Mems_Read_Reg( REF_T_H);   //reads temp ref H

    //interrupt 1
    if(interruptOne == 1) {
      //interruptOne = 0;
      buffer[12]   = 0x40;
    }
    else {
      buffer[12]   = 0x00;
    }

    //interrupt 2
    if(interruptTwo == 1) {
      //interruptTwo = 0;
      buffer[13]   = 0x40;
    }
    else {
      buffer[13]   = 0x00;
    }
    buffer[14] = 0;

    if(SW1_BUTTON_PIN)
      buffer[14] |= 0x01;

    if(SW2_BUTTON_PIN)
      buffer[14] |= 0x02;

    buffer[15]  = '\r';
    buffer[16]  = '\n';

      MEMS_Print(buffer, sizeof(buffer));

  }
}


/*************************************************************************
*****
* Function Name  : Stop
* Description    : It stops the state1 machine
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void Stop(void){
  state1 = STOP;
}

/*************************************************************************
*****
* Function Name  : WriteReg
* Description    : write a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void WriteReg() {
```

```
  uint8_t data = 0;
  uint8_t address = 0;

  if( isxdigit(usbBuff[3]) && isxdigit(usbBuff[4]) && isxdigit(usbBuff[5])
&& isxdigit(usbBuff[6]) ) {
    address = StringToChar(&usbBuff[3]);
    data = StringToChar(&usbBuff[5]);
    SPI_Mems_Write_Reg(address, data);
  }
  if(echoOnFlag) {
      uint8_t buffer[10];
      uint8_t response[2];

      address = StringToChar(&usbBuff[3]);

      data = SPI_Mems_Read_Reg(address);
      CharToString(data, response);

      buffer[0]  = 'P';
      buffer[1]  = 'R';
      buffer[2]  = usbBuff[3];
      buffer[3]  = usbBuff[4];
      buffer[4]  = 'h';
      buffer[5]  = response[0];
      buffer[6]  = response[1];
      buffer[7]  = 'h';
      buffer[8]  = '\r';
      buffer[9]  = '\n';

        MEMS_Print(buffer, sizeof(buffer));
  }
}

/*************************************************************************
*****
* Function Name  : ReadReg
* Description    : read a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void ReadReg(void) {
  uint8_t data = 0;
  uint8_t address = 0;
  uint8_t response[2];
  uint8_t buffer[10];

  address = StringToChar(&usbBuff[3]);
  data = SPI_Mems_Read_Reg(address);
  CharToString(data, response);

  buffer[0]  = 'P';
  buffer[1]  = 'R';
  buffer[2]  = usbBuff[3];
  buffer[3]  = usbBuff[4];
  buffer[4]  = 'h';
  buffer[5]  = response[0];
  buffer[6]  = response[1];
  buffer[7]  = 'h';
```

```
  buffer[8]  = '\r';
  buffer[9] = '\n';

  MEMS_Print(buffer, sizeof(buffer));

}

/*******************************************************************************
*****
* Function Name  : Debug
* Description    : Function to Send the debug output data to USB
(associated to '*debug' command).
*                  The system send continuosly through the USB interface
*                  a readable string containg acquired data
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Debug(void) {
  uint8_t dataReady = 0;
  uint8_t buffer[3];
  uint8_t bufferUSB[19];
  int16_t data = 0;
  int32_t dataC = 0;
  uint8_t tempDataBuffer[2];

  dataReady = SPI_Mems_Read_Reg(STATUS_REG);
  if(ValBit(dataReady, PDA)) {
    EKSTM32_LEDToggle(LED1);

    buffer[0]    = SPI_Mems_Read_Reg( P_OUT_H );    //reads pressure MSB
    buffer[1]    = SPI_Mems_Read_Reg( P_OUT_L );    //reads pressure LSB
    buffer[2]    = SPI_Mems_Read_Reg( P_OUT_XL );   //reads pressure XLSB

    tempDataBuffer[0] = SPI_Mems_Read_Reg( T_OUT_H );//reads TEMP MSB
    tempDataBuffer[1] = SPI_Mems_Read_Reg( T_OUT_L );//reads TEMP LSB

    dataC = (int32_t) ( (buffer[0]<<16) | (buffer[1]<<8) | buffer[1]);
    data = (int16_t) (dataC >> 12);
    sprintf((char*)bufferUSB,"P=%+06d ",data);

    data = (int16_t) (( (tempDataBuffer[0]<<8) | tempDataBuffer[1] ));
    sprintf((char*)&bufferUSB[9],"T=%+06d\r\n",data);

      MEMS_Print(bufferUSB, sizeof(bufferUSB));

  }
}


/*******************************************************************************
*****
* Function Name  : InterruptReceived
* Description    : Interrupt One and Two Handler
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
```

```
static void InterruptReceived(uint8_t interrupt) {
  uint8_t data;

  if(interrupt == INTERRUPTONE) { //int1
    //read pin state
    data = GPIO_ReadInputDataBit(INT1_PORT, INT1_PIN);
    if(data == 0x01)
    interruptOne = 1;
    else
      interruptOne = 0;
  }

  else if(interrupt == INTERRUPTTWO) { //int2
    //read pin state
    data = GPIO_ReadInputDataBit(INT2_PORT, INT2_PIN);
    if(data == 0x01)
    interruptTwo = 1;
    else
      interruptTwo = 0;
  }

}

/*******************************************************************************
*****
* Function Name  : DbReset
* Description    : It reset the firmware selectet.
*                : A new firmware can be now selected
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
void DbReset(void) {
  dbSelected = 0;
  Zon();
  EKSTM32_LEDOff(LED1);
  EKSTM32_LEDOff(LED2);
  EKSTM32_LEDOff(LED3);
}

/****************** LPS331.c *****END OF FILE****/
```

3.4.3 Akcelerometr

AIS326DQ 3D akcelerometr

QFPN-28 pin description

Figure 2. Pin connection

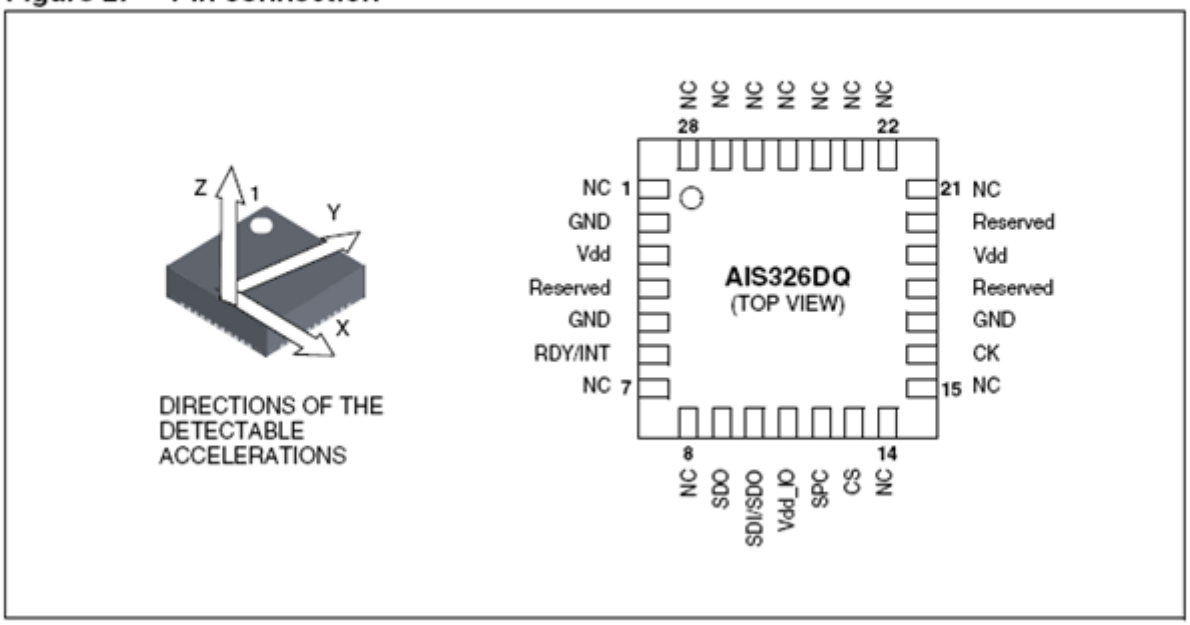
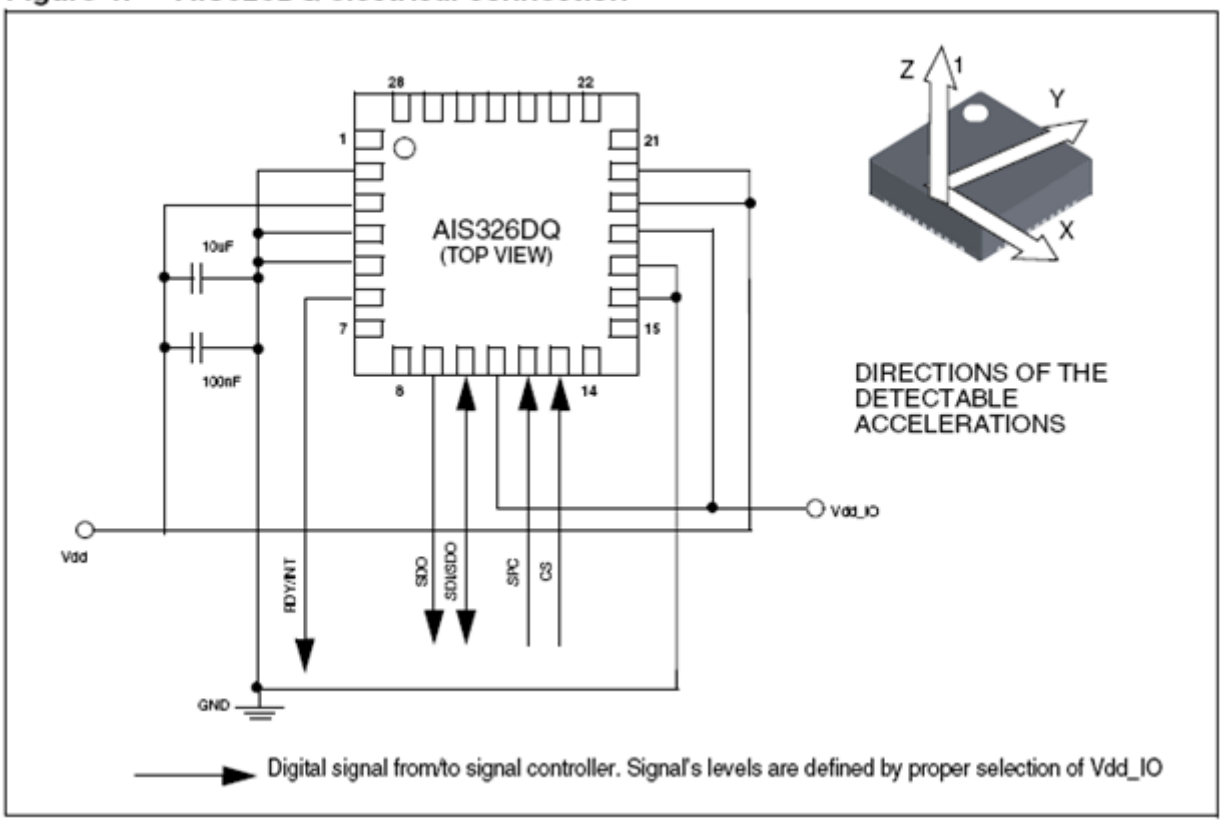
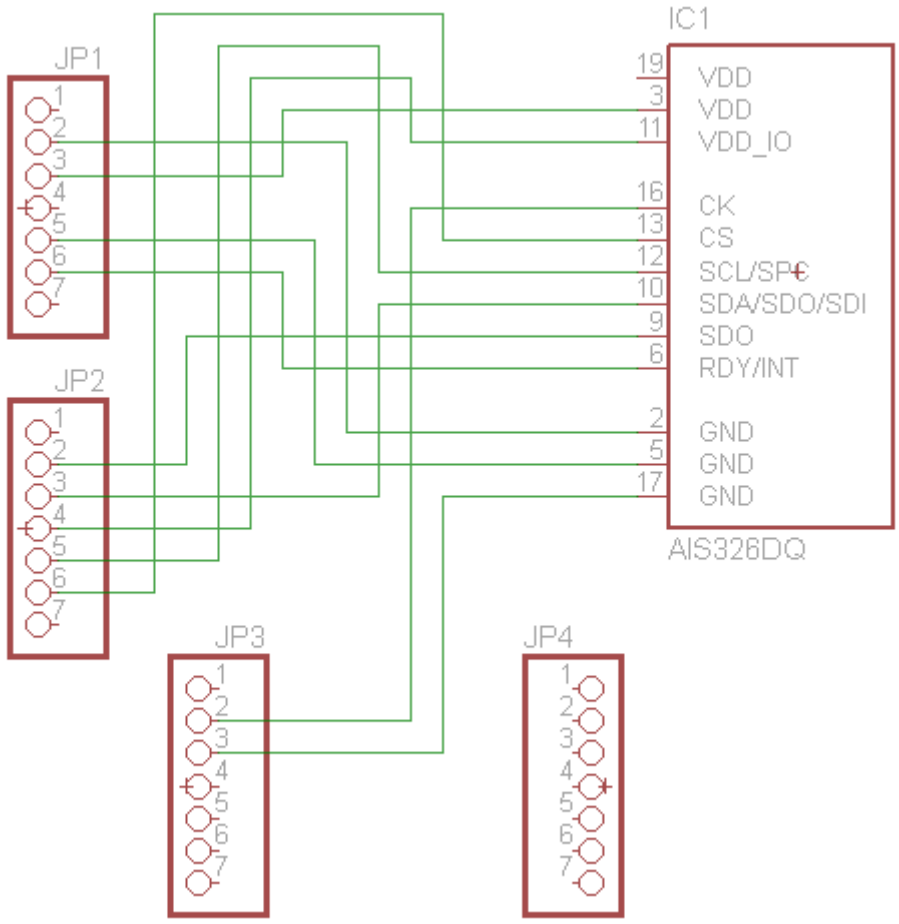


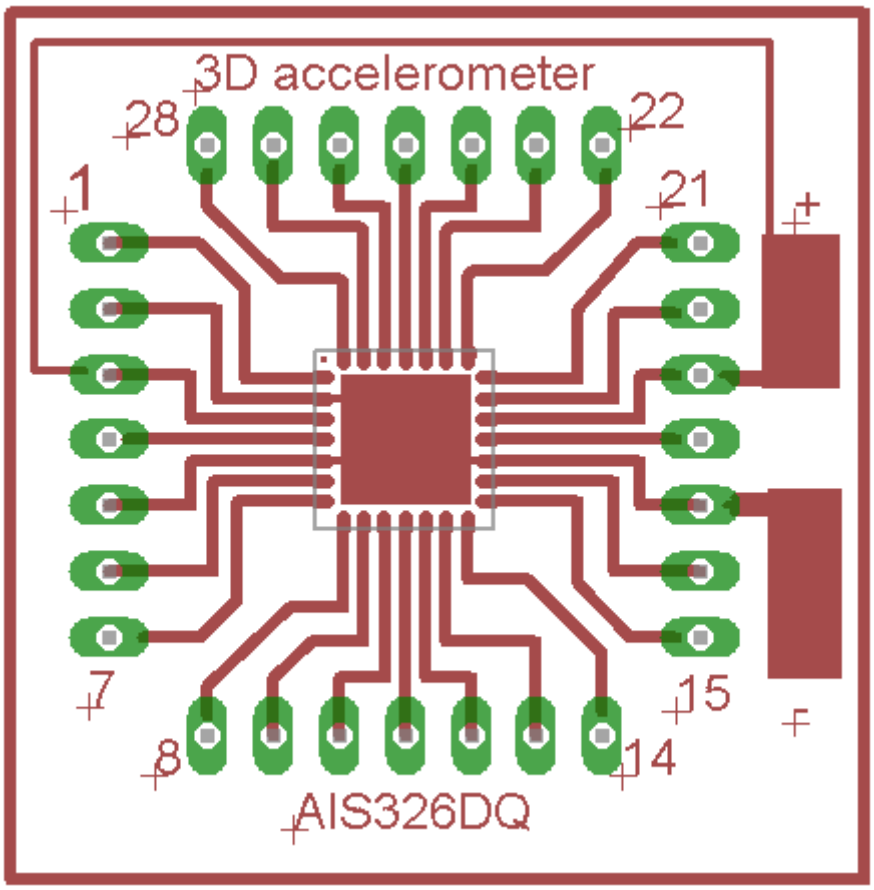
Figure 4. AIS326DQ electrical connection



I2C po START se pošle 0011101b. b je R/W

Aplikaci najdeme (včetně C kódu arduino) na http://nearfuturelaboratory.com/2006/09/22/arduino-and-the-lis3lv02dq-triple-axis-accelerometer/





Programová obsluha pro STM32F103 – soubory **AIS328DQ.c** a **AIS328DQ.h** :

```
/******************************************
* File Name          : ais328dq.h
* Description        : Descriptor Header for ais328dq file
*                      STM32F103
*
*******************************************************************************
****/

/* Define to prevent recursive inclusion -------------------------------
---*/
#ifndef __AIS328DQ_H
#define __AIS328DQ_H

/* Includes ------------------------------------------------------------------
---*/
#include "stm32f10x.h"

/* Exported types ------------------------------------------------------------
---*/
/* Exported constants --------------------------------------------------------
---*/
#define BIT(x) ( 1<<(x) )

#define VER                             0
```

```
#define DEV                             1
#define ZOFF                            2
#define ZON                             4
#define START                           5
#define STOP                            6
#define R                               7
#define W                               8
#define DEBUG                           9
#define LIST                            10
#define SINGLE                          11
#define ECHOON                          12
#define ECHOOFF                         13
#define LISTDEV                         14
#define DBRESET                         15

#define STATUS_REG                      0x27
#define OUT_X_L                         0x28
#define OUT_X_H                         0x29
#define OUT_Y_L                         0x2a
#define OUT_Y_H                         0x2b
#define OUT_Z_L                         0x2c
#define OUT_Z_H                         0x2d

#define STATUS_REG_ZYXDA                3

#define DATAREADY_BIT                   STATUS_REG_ZYXDA

#define CTRL_REG1                       0x20

#define CTRL_REG5                       0x24
#define CTRL_REG5_FIFO_EN               BIT(6)

//          INT1_SRC
#define INT1_SRC                        0x31
#define INT1_SRC_IA                     BIT(6)

//          INT2_SRC

#define INT2_SRC                        0x35
#define INT2_SRC_IA                     BIT(6)

#define CLICK_SRC                       0x39
#define CLICK_SRC_IA                    BIT(6)
/* Exported macro ------------------------------------------------------
---*/

/* Exported functions --------------------------------------------------
---*/
void ais328dq_AppTick(void);

#endif /* __AIS328DQ_H */

/****************** AIS328DQ.h  *****END OF FILE****/
```

```
/******************* ********************
* File Name          : ais328dq.c
```

```
* Description        : This file provides a set of functions needed to
manage the
*                      ais328dq adapter board.
*                     STM21F103
*
*******************************************************************************
****/

/* Includes ------------------------------------------------------------------
---*/
#include "ais328dq.h"
#include "led.h"
#include "command_interpreter.h"
#include "string.h"
#include "usb_lib.h"
#include "usb_desc.h"
#include "usb_pwr.h"
#include "spi_mems.h"
#include "ctype.h"
#include "utility.h"
#include "interruptHandler.h"
#include "hw_config.h"
#include "fw_selector.h"
#include <stdio.h>


/* Private typedef -----------------------------------------------------------
---*/
/* Private define ------------------------------------------------------------
---*/
/* Private macro -------------------------------------------------------------
---*/
#define ValBit(VAR,Place)         (VAR & (1<<Place))
/* Private variables ---------------------------------------------------------
---*/
static uint8_t usbBuff[MAX_COMMAND_LENGTH];
static uint8_t state = STOP;
static uint8_t stateOld = STOP;

static uint8_t interruptOne = 0;
static uint8_t interruptTwo = 0;

static uint8_t zoffFlag = 0;
static uint8_t echoOnFlag = 0;

extern uint8_t dbSelected;

/* Private function prototypes -----------------------------------------------
---*/
void ais328dq_AppTick(void);
static uint8_t Decod();
static void StateMachine(void);
static void Version(void);
static void Device(void);
static void Zoff(void);
static void Zon(void);
static void Start(void);
static void Stop(void);
static void ReadReg(void);
static void WriteReg(void);
```

```
static void Debug(void);
static void InterruptReceived(uint8_t interrupt);
static void DbReset(void);

/* Private functions ---------------------------------------------------------*/

/*******************************************************************************
* Function Name  : ais328dq_AppTick
* Description    : ais328dq Application Tick Function
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void ais328dq_AppTick(void) {
  EKSTM32_LEDOn(LED3);
  if(newCommandAvailable(usbBuff)) {

    state = Decod();
  }

  StateMachine();

}

/*******************************************************************************
* Function Name   : Decod
* Description     : decode the incoming USB data
**                      :  decodification of RTX vector
**                      :  *start/r:            :starts sending data
**                      :  *stop/r:             :stops sending data
**                      :  *ver/r:              :sends firmware version
number
**                      :  *dev/r:              :sends device name
**                      :  *wxxdd/r:            :writes in register at
address xx:(0xadr) --> the value data dd:(0xdata)
**                      :  *rxx/r:              :reads register at address
xx:(0xadr)
**                      :  *zon/r:              :activates tristate mode
**                      :  *zoff/r:             :activates connections with
devices
**
**   Realizes the command interpreter (for general purpose commands).
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
static uint8_t Decod(){

  if(strcmp((char*)&usbBuff[1],"VER")==0) {
    stateOld = state;
    state = VER;
  }
  else if(strcmp((char*)&usbBuff[1],"DEV")==0) {
    stateOld = state;
    state = DEV;
```

```
  }
  else if(strcmp((char*)&usbBuff[1],"ZOFF")==0) {
    stateOld = state;
    state = ZOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"ZON")==0) {
    state = ZON;
  }
  else if(strcmp((char*)&usbBuff[1],"START")==0) {
    stateOld = state;
    state = START;
  }
  else if(strcmp((char*)&usbBuff[1],"STOP")==0) {
    stateOld = state;
    state = STOP;
  }
  else if(usbBuff[1] == 'R') {
    stateOld = state;
    state = R;
  }
  else if(usbBuff[1] == 'W') {
    stateOld = state;
    state = W;
  }
  else if(strcmp((char*)&usbBuff[1],"DEBUG")==0) {
    state = DEBUG;
  }
  else if(strcmp((char*)&usbBuff[1],"LIST")==0) {
    stateOld = state;
    state = LIST;
  }
  else if(strcmp((char*)&usbBuff[1],"SINGLE")==0) {
    stateOld = state;
    state = SINGLE;
  }
  else if(strcmp((char*)&usbBuff[1],"ECHOON")==0) {
    stateOld = state;
    state = ECHOON;
  }
  else if(strcmp((char*)&usbBuff[1],"ECHOOFF")==0) {
    stateOld = state;
    state = ECHOOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"LISTDEV")==0) {
    stateOld = state;
    state = LISTDEV;
  }
  else if(strcmp((char*)&usbBuff[1],"DBRESET")==0) {
    stateOld = state;
    state = DBRESET;
  }

 return state;
}

/*****************************************************************************
*****
* Function Name  : StateMachine
* Description    : StateMachiine is called from the aAppTick main loop
*                  It calls functions that implement commands selected
```

```
*                  It implements state machine for commands interpreter.
*                  If Set of available commands is extended with device
specific commands
* Input           : None
* Output          : None
* Return          : None
*******************************************************************************
****/
static void StateMachine(void) {

  switch(state) {
  case VER:
    Version();
    state = stateOld;
    break;
  case DEV:
    Device();
    state = stateOld;
    break;
  case ZOFF:
    Zoff();
    state = STOP;
    stateOld = STOP;
    break;
  case ZON:
    Zon();
    break;
  case START:
    if(zoffFlag) {
      Start();
    }
    break;
  case STOP:
    if(zoffFlag) {
      Stop();
    }
    break;
  case R:
    if(zoffFlag) {
      ReadReg();
      state = stateOld;
    }
    break;
  case W:
    if(zoffFlag) {
      WriteReg();
      state = stateOld;
    }
    break;
  case DEBUG:
    if(zoffFlag) {
      Debug();
    }
    break;
  case LIST:
    //It Prints the list of firmwares implemented
    PrintListOfFirmware();
    state = stateOld;
    break;
  case SINGLE:
```

```
    if(zoffFlag) {
      Debug();
      state = stateOld;
    }
    break;
  case ECHOON:
      echoOnFlag = 1;
      state = stateOld;
    break;
  case ECHOOFF:
      echoOnFlag = 0;
      state = stateOld;
    break;
  case LISTDEV:
    //It Prints the list of device supported
    PrintListOfDevice();
    state = stateOld;
    break;
  case DBRESET:
    //unselect the db
    DbReset();
    state = STOP;
    break;
  }

}

/**************************************************************************
*****
* Function Name  : Version
* Description    : Function to send Firmware version to host by USB
* Input          : None
* Output         : None
* Return         : None
**************************************************************************
****/
static void Version(void) {
 //eMotion firmware version
  EmotionVer();
}

/**************************************************************************
*****
* Function Name  : Device
* Description    : Function to send the Device Product Code
* Input          : None
* Output         : None
* Return         : None
**************************************************************************
****/
static void Device(void) {
  uint8_t buffer[]="AIS328DQ \r\n";

  MEMS_Print(buffer, sizeof(buffer));
}

/**************************************************************************
*****
* Function Name  : Zoff
* Description    : It initializes the peripherals according to the device
```

```
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zoff(void) {
  EKSTM32_LEDToggle(LED2);
  RegisterInterrupt(InterruptReceived);
  SPI_Mems_Init();
  zoffFlag = 1;
}


/*******************************************************************************
*****
* Function Name  : Zon
* Description    : It DeInitializes the peripherals according to the device
*                  Now the device is disconnected from the board
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zon(void) {
  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Pin = SPI_MEMS_PIN_SCK | SPI_MEMS_PIN_MISO |
SPI_MEMS_PIN_MOSI;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(SPI_MEMS_GPIO, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = SPI_MEMS_CS;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(SPI_MEMS_CS_GPIO, &GPIO_InitStructure);

  UnRegisterInterrupt();

  EKSTM32_LEDOn(LED2);

  zoffFlag = 0;

}

/*******************************************************************************
*****
* Function Name  : Start
* Description    : Function to Send the standard output data to USB
(associated to '*start' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by acceleration, interrupt and
*                  switch status.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Start(void) {

  uint8_t dataReady = 0;
  uint8_t buffer[13];
```

```
  dataReady = SPI_Mems_Read_Reg(STATUS_REG);
  if(ValBit(dataReady, DATAREADY_BIT)) {
    //blink the led
    EKSTM32_LEDToggle(LED1);

    buffer[0]   = 's';
    buffer[1]   = 't';
    buffer[2]   = SPI_Mems_Read_Reg( OUT_X_H );//reads X MSB
    buffer[3]   = SPI_Mems_Read_Reg( OUT_X_L );//reads X LSB
    buffer[4]   = SPI_Mems_Read_Reg( OUT_Y_H );//reads Y MSB
    buffer[5]   = SPI_Mems_Read_Reg( OUT_Y_L );//reads Y LSB
    buffer[6]   = SPI_Mems_Read_Reg( OUT_Z_H );//reads Z MSB
    buffer[7]   = SPI_Mems_Read_Reg( OUT_Z_L );//reads Z LSB

    //interrupt 1
    if(interruptOne == 1) {
      //interruptOne = 0;
      buffer[8]   = 0x40;
    }
    else {
      buffer[8]   = 0x00;
    }

    //interrupt 2
    if(interruptTwo == 1) {
      //interruptTwo = 0;
      buffer[9]   = 0x40;
    }
    else {
      buffer[9]   = 0x00;
    }

    buffer[10] = 0;

    if(SW1_BUTTON_PIN)
      buffer[10] |= 0x01;

    if(SW2_BUTTON_PIN)
      buffer[10] |= 0x02;

    buffer[11]  = '\r';
    buffer[12]  = '\n';

      MEMS_Print(buffer, sizeof(buffer));
  }

}

/*******************************************************************************
*****
* Function Name  : Stop
* Description    : It stops the state machine
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Stop(void){
  state = STOP;
```

```
}

/*******************************************************************************
*****
* Function Name  : WriteReg
* Description    : write a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void WriteReg() {
  uint8_t data = 0;
  uint8_t address = 0;

  if( isxdigit(usbBuff[2]) && isxdigit(usbBuff[3]) && isxdigit(usbBuff[4])
&& isxdigit(usbBuff[5]) ) {
    address = StringToChar(&usbBuff[2]);
    data = StringToChar(&usbBuff[4]);
    SPI_Mems_Write_Reg(address, data);
  }
  if(echoOnFlag) {
      uint8_t buffer[9];
      uint8_t response[2];

      address = StringToChar(&usbBuff[2]);

      data = SPI_Mems_Read_Reg(address);
      CharToString(data, response);

      buffer[0]  = 'R';
      buffer[1]  = usbBuff[2];
      buffer[2]  = usbBuff[3];
      buffer[3]  = 'h';
      buffer[4]  = response[0];
      buffer[5]  = response[1];
      buffer[6]  = 'h';
      buffer[7]  = '\r';
      buffer[8]  = '\n';

        MEMS_Print(buffer, sizeof(buffer));
  }
}

/*******************************************************************************
*****
* Function Name  : ReadReg
* Description    : read a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void ReadReg(void) {
  uint8_t data = 0;
  uint8_t address = 0;
  uint8_t response[2];
  uint8_t buffer[9];

  address = StringToChar(&usbBuff[2]);
```

```
  data = SPI_Mems_Read_Reg(address);
  CharToString(data, response);

  buffer[0]  = 'R';
  buffer[1]  = usbBuff[2];
  buffer[2]  = usbBuff[3];
  buffer[3]  = 'h';
  buffer[4]  = response[0];
  buffer[5]  = response[1];
  buffer[6]  = 'h';
  buffer[7]  = '\r';
  buffer[8] = '\n';

  MEMS_Print(buffer, sizeof(buffer));
}

/*******************************************************************************
*****
* Function Name  : Debug
* Description    : Function to Send the debug output data to USB
(associated to '*debug' command).
*                  The system send continuosly through the USB interface
*                  a readable string containg acquired data
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Debug(void) {
  uint8_t buffer[25];
  uint8_t dataReady = 0;
  int16_t data = 0;

  dataReady = SPI_Mems_Read_Reg(STATUS_REG);

  if(ValBit(dataReady, DATAREADY_BIT)) {
    //blink the led
    EKSTM32_LEDToggle(LED1);

    data = (int16_t) ( (SPI_Mems_Read_Reg(OUT_X_H)<<8) |
SPI_Mems_Read_Reg(OUT_X_L) );
    data>>=4;
    sprintf((char*)buffer,"X=%+05d ",data);

    data = (int16_t) ( (SPI_Mems_Read_Reg(OUT_Y_H)<<8) |
SPI_Mems_Read_Reg(OUT_Y_L) );
    data>>=4;
    sprintf((char*)&buffer[8],"Y=%+05d ",data);

    data = (int16_t) ( (SPI_Mems_Read_Reg(OUT_Z_H)<<8) |
SPI_Mems_Read_Reg(OUT_Z_L) );
    data>>=4;
    sprintf((char*)&buffer[16],"Z=%+05d\r\n",data);

      MEMS_Print(buffer, sizeof(buffer));
  }

}
```

```
/*******************************************************************************
*****
* Function Name  : InterruptReceived
* Description    : Interrupt One and Two Handler
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void InterruptReceived(uint8_t interrupt) {
  uint8_t data;

  if(interrupt == INTERRUPTONE) { //int1
    //read pin state
    data = GPIO_ReadInputDataBit(INT1_PORT, INT1_PIN);
    if(data == 0x01)
    interruptOne = 1;
    else
      interruptOne = 0;
  }

  else if(interrupt == INTERRUPTTWO) { //int2
    //read pin state
    data = GPIO_ReadInputDataBit(INT2_PORT, INT2_PIN);
    if(data == 0x01)
    interruptTwo = 1;
    else
      interruptTwo = 0;
  }

}

/*******************************************************************************
*****
* Function Name  : DbReset
* Description    : It reset the firmware selectet.
*                : A new firmware can be now selected
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
void DbReset(void) {
  dbSelected = 0;
  Zon();
  EKSTM32_LEDOff(LED1);
  EKSTM32_LEDOff(LED2);
  EKSTM32_LEDOff(LED3);
}

/****************** AIS328DQ.c *****END OF FILE****/
```

3.4.4 Gyroskop

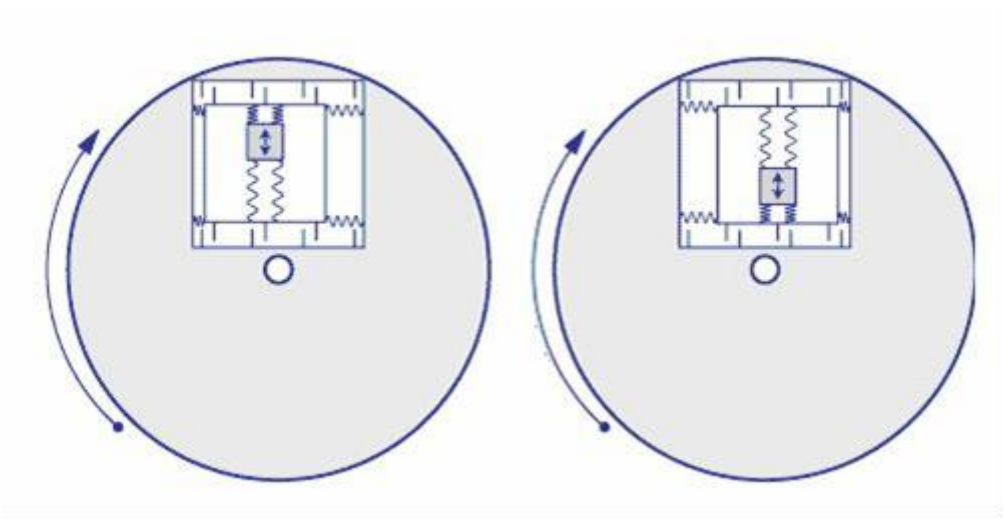
Gyroskopy jsou snímače, které se používají k stanovení úhlové rychlosti a natočení. Gyroskopy lze rozdělit dle použitého fyzikálního principu na tyto skupiny :

- Mechanické
- Kvantové
- Jaderné
- Elektrické
- Optické

Pro naše účely přicházejí v úvahu především MEMS gyroskopy
Stejně jako v případě MEMS akcelerometrů obsahují MEMS gyroskopy mimo samotného snímače i celou škálu vyhodnocovacích, řídicích obvodů a logiky. Výstupní signál je pak analogový nebo digitální. Rotaci je možné typicky měřit vzhledem k jedné ze tří os z, y, x . Gyroskopy vyráběné jako integrované MEMS obvody pracující na principu Coriolisovy síly, umějí měřit pouze v jednom směru, a to je směr kolmý na plochu obvodu. Pro jiné směry je nutné zajistit správné natočení a umístění součástky. Coriolisova síla je takzvaná virtuální síla, která působí na libovolný hmotný objekt , či předmět, který se pohybuje rychlostí vr v soustavě, která rotuje kolem osy úhlovou rychlostí ωr . Coriolisova síla působí na každý hmotný objekt na zemi.

$$Fc = m.vr \times \omega r$$

m … hmotnost (kg)
vr … rychlost (m . s-1)
ωr … úhlová rychlost (rad . s-1)



Příklad funkce struktury snímače gyroskopu při rotaci
Provedení samotného MEMS snímače vypadá tak, že základ tvoří rezonující struktura upevněná v rámu, která se vlivem vlastní mechanické rezonance, zde reprezentované pružinami, pohybuje v uvedeném směru kolmém na směr otáčení (viz obr). Přitom vzniká Coriolisova síla úměrná úhlové rychlosti otáčení, která stlačí vnější pružiny rámu a způsobí vzájemný posuv měřících plošek, které mají funkci elektrod vzduchových kondenzátorů. Výstupem je tedy změna kapacity úměrná úhlové rychlosti. Stejného účinku se pak využívá i v mechanických gyroskopech. Zde při pohybu objektu, upevněného na pružinách uvnitř rámu, směrem ven, působí na něho Coriolisova síla směrem doleva, při opačném směru pohybu objektu pak působí směrem doprava. Protože velikost a směr této síly je

úměrný i velikosti úhlové rychlosti a směru otáčení, lze tento systém s úspěchem využít pro jejich měření.

L3DG20 gyroskop

Figure 2. Pin connection

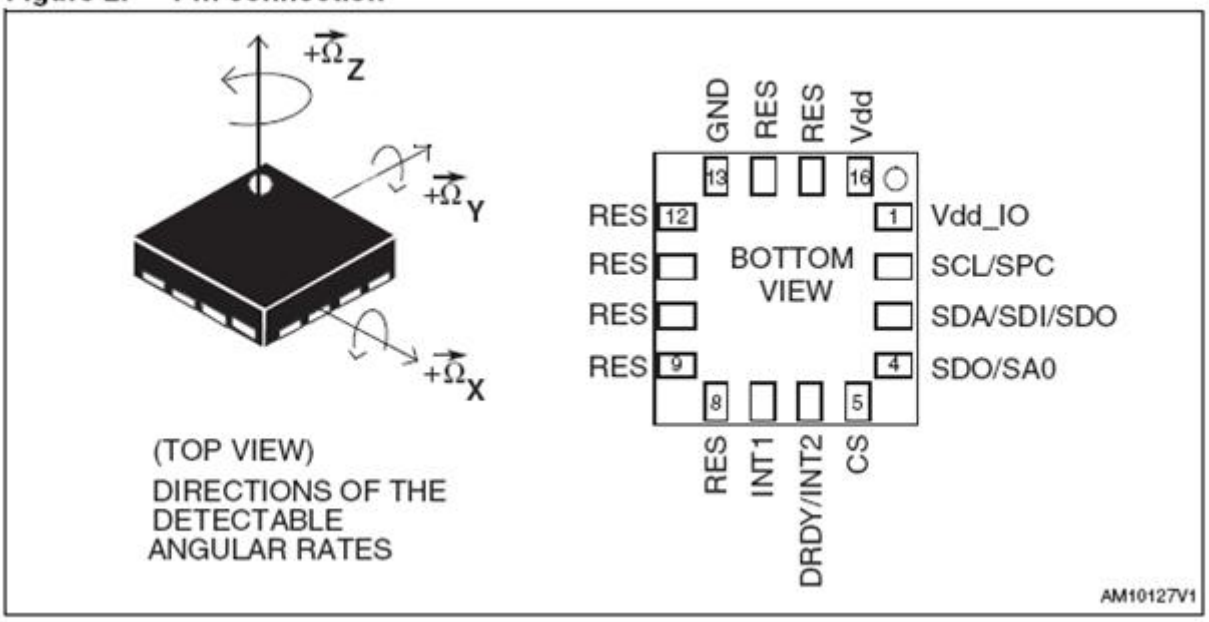


Figure 5. L3GD20 electrical connections and external component values

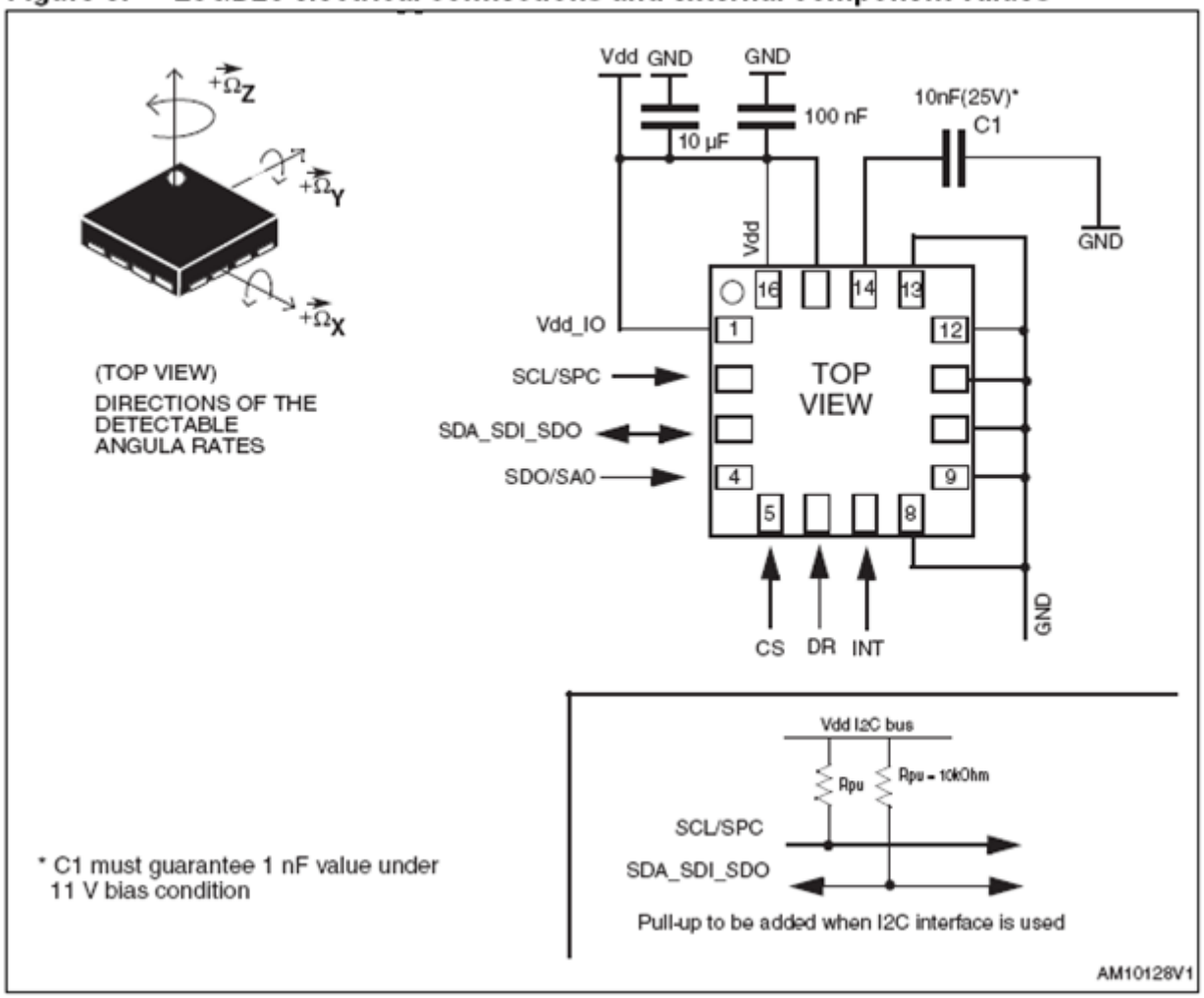
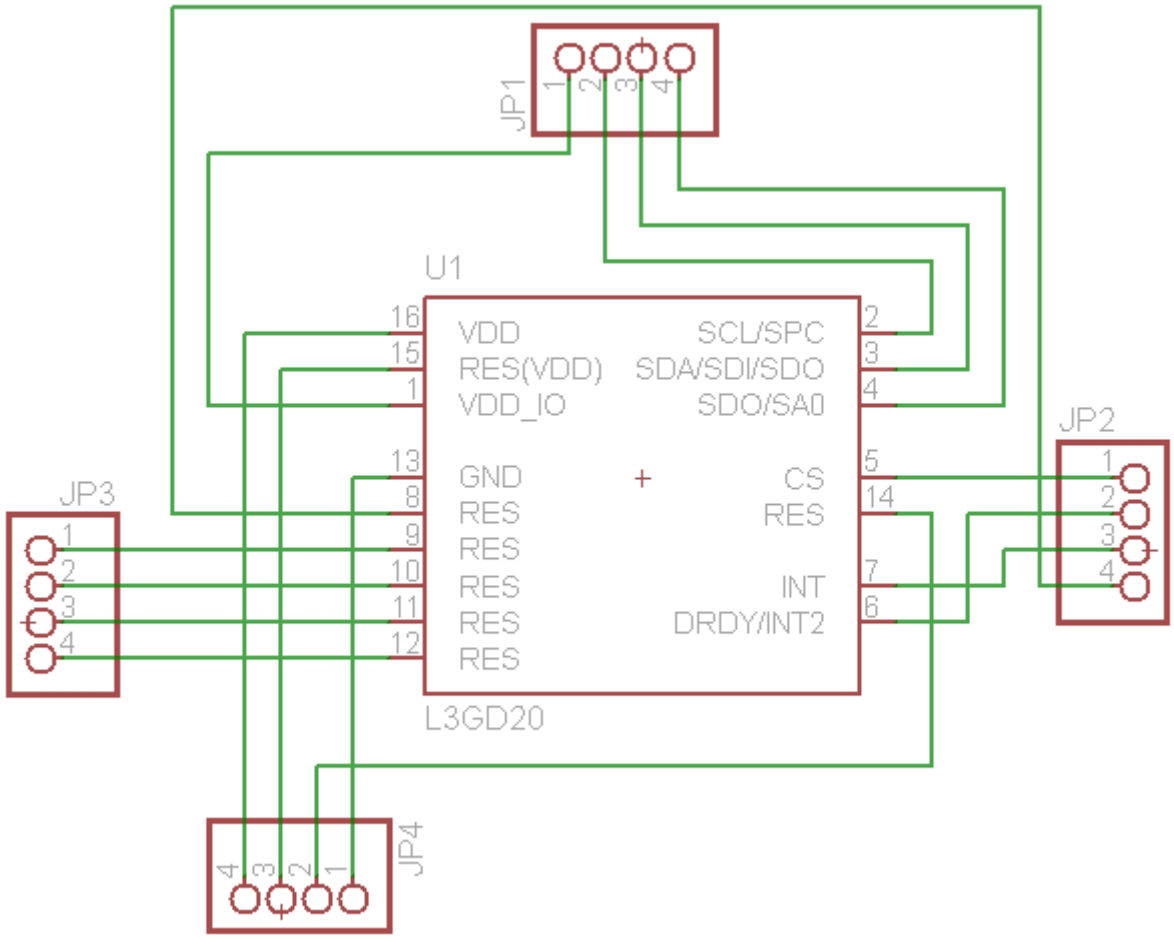
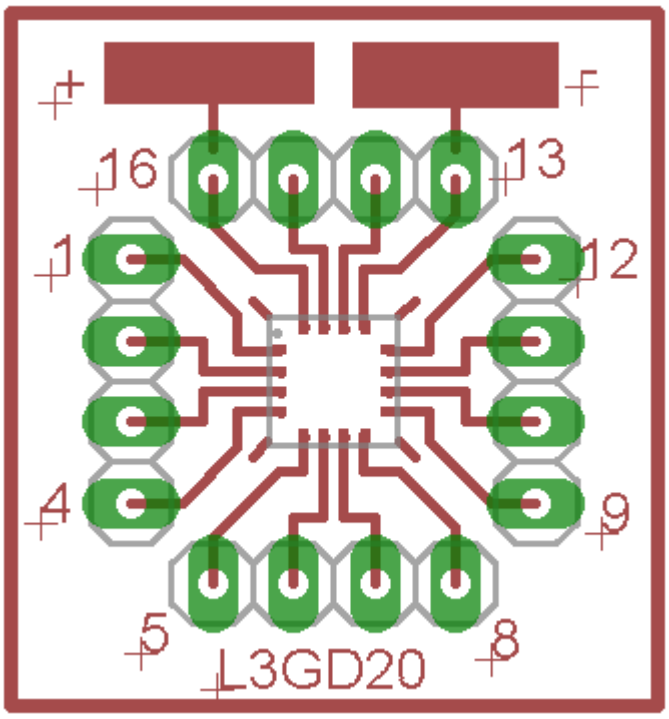


Table 11. SAD+read/write patterns

Command	SAD[6:1]	SAD[0] = SDO	R/W	SAD+R/W
Read	110101	0	1	11010101 (D1h)
Write	110101	0	0	11010100 (D0h)
Read	110101	1	1	11010111 (D3h)
Write	110101	1	0	11010110 (D2h)

Pozn. Obvod má pin SA0 umožňující připojení dvou obvodů L3DG20 k jedné I2C sběrnici s tím, že jednomu obvodu se na SA0 připojí 0 a druhému 1. To, s kterým z těchto obvodů probíhá komunikace se určuje adresou (SAD) viz tabulka11 výše.


JP1
1
2
3
4
U1
16
15
1
VDD
RES(VDD)
VDD_IO
SCL/SPC
SDA/SDI/SDO
SDO/SA0
2
3
4
13
8
9
10
11
12
GND
RES
RES
RES
RES
RES
CS
RES
INT
DRDY/INT2
5
14
7
6
JP2
1
2
3
4
JP3
1
2
3
4
JP4
4
3
2
1
L3GD20


16
13
1
12
4
9
5
8
L3GD20

Programová obsluha pro STM32F103 – soubory **L3GD20.c** a **L3GD20.h** :

```
/******************** (C) COPYRIGHT 2012 STMicroelectronics
********************
* File Name          : l3gd20.h
* Description        : Descriptor Header for l3gd20 file
*                      STM32F103
*
*******************************************************************************
****/
/* Define to prevent recursive inclusion -------------------------------
---*/
#ifndef __L3GD20_H
#define __L3GD20_H

/* Includes ------------------------------------------------------------------
---*/
#include "stm32f10x.h"

/* Exported types ------------------------------------------------------------
---*/
/* Exported constants --------------------------------------------------------
---*/

#define BIT(x) ( 1<<(x) )

#define VER                             0
#define DEV                             1
#define ZOFF                            2
#define ZON                             4
#define START                           5
#define STOP                            6
#define R                               7
#define W                               8
#define DEBUG                           9
// FIFO CONFIGURATIONS //
#define FIFO_BYPASS                     10
#define FIFO_STREAM                     11
#define FIFO_MODE                   12
#define FIFO_STREAM_TO_FIFO             13
#define FIFO_BYPASS_TO_STREAM           14
#define FIFO_READ                   15
#define FIFO_RUN                    16
#define LIST                            17
#define SINGLE                          18
#define ECHOON                          19
#define ECHOOFF                         20
#define LISTDEV                         21
#define DBRESET                         22


#define OUT_P_L                     0x28
#define OUT_P_H                     0x29
#define OUT_R_L                     0x2A
#define OUT_R_H                     0x2B
#define OUT_Y_L                     0x2C
#define OUT_Y_H                     0x2D
```

```
#define DATAREADY_BIT   STATUS_REG_ZYXDA


#define WHO AM I                        0x0F // device identification
register - default value
#define I_AM_L3GD20                     0xD3


//          Control Register 1
#define CTRL_REG1             0x20
#define CTRL_REG1_DEFAULT               (BIT(2)|BIT(1)|BIT(0))


//          Control Register 2
#define CTRL_REG2             0x21
#define CTRL_REG2_DEFAULT               0


//          Control Register 3
#define CTRL_REG3             0x22
#define CTRL_REG3_DEFAULT               0


//          Control Register 4
#define CTRL_REG4             0x23
#define CTRL_REG4_DEFAULT               0


//          Control Register 5
#define CTRL_REG5             0x24

#define CTRL_REG5_DEFAULT               0

#define CTRL_REG5_FIFO_EN           BIT(6)
#define CTRL_REG5_STOP_ON_WTM       BIT(5)


//          STATUS REGISTER
#define STATUS_REG                  0x27

#define STATUS_REG_ZYXOR            7   // 1    :     new data set has over
written the previous one
                                  // 0    :     no overrun has occurred
(default)
#define STATUS_REG_ZOR              6   // 0    :     no overrun has
occurred (default)
                                  // 1    :     new Z-axis data has over
written the previous one
#define STATUS_REG_YOR              5   // 0    :     no overrun has
occurred (default)
                                  // 1    :     new Y-axis data has over
written the previous one
#define STATUS_REG_XOR              4   // 0    :     no overrun has
occurred (default)
                                  // 1    :     new X-axis data has over
written the previous one
#define STATUS_REG_ZYXDA            3   // 0    :     a new set of data is
not yet avvious one
                                            // 1     :     a new set of
data is available
```

```
#define STATUS_REG_ZDA              2   // 0    :     a new data for the Z-
Axis is not availvious one
                                        // 1     :     a new data for
the Z-Axis is available
#define STATUS_REG_YDA              1   // 0    :     a new data for the Y-
Axis is not avail
                                        // 1     :     a new data for
the Y-Axis is available
#define STATUS_REG_XDA              0   // 0    :     a new data for the X-
Axis is not avail


//          INT_SRC
#define INT_SRC                     0x31
#define INT_SRC_IA                    BIT(6)


//          FIFO_REGISTERS
#define FIFO_CONTROL                0x2E
#define FIFO_CONTROL_WTMSAMP        (BIT(4) | BIT(3) | BIT(2) | BIT(1) |
BIT(0))

#define DEF_FIFO_CTRL_BYPASS        0x00
#define DEF_FIFO_CTRL_FIFO_MODE           BIT(5)
#define DEF_FIFO_CTRL_STREAM        BIT(6)
#define DEF_FIFO_CTRL_STREAM_TO_FIFO      (BIT(6) | BIT(5))
#define DEF_FIFO_CTRL_BYPASS_TO_STREAM    BIT(7)

#define FIFO_SOURCE                   0x2F
#define FIFO_SOURCE_SAMPLES         (BIT(4) | BIT(3) | BIT(2) | BIT(1) |
BIT(0))
#define FIFO_SOURCE_EMPTY           BIT(5)
#define FIFO_SOURCE_OVRN            BIT(6)
#define FIFO_SOURCE_WTM             BIT(7)

/* Exported macro ------------------------------------------------------
---*/

/* Exported functions --------------------------------------------------
---*/
void l3gd20_AppTick(void);

#endif /* __L3GD20_H */

/****************** l3gd20.h *****END OF FILE****/
```

```
/*******************  *******************
* File Name          : L3GD20.c
* Description        : This file provides a set of functions needed to
manage the
*                      l3GD20 adapter board.
*                      STM32F103
*
*******************************************************************************
****/
```

```
/* Includes ------------------------------------------------------------
---*/
#include "l3gd20.h"
#include "led.h"
#include "command_interpreter.h"
#include "string.h"
#include "usb_lib.h"
#include "usb_desc.h"
#include "usb_pwr.h"
#include "spi_mems.h"
#include "ctype.h"
#include "utility.h"
#include "interruptHandler.h"
#include "hw_config.h"
#include "fw_selector.h"
#include <stdio.h>
#include <string.h>


/* Private typedef -----------------------------------------------------
---*/
/* Private define ------------------------------------------------------
---*/
/* Private macro -------------------------------------------------------
---*/
#define ValBit(VAR,Place)         (VAR & (1<<Place))
/* Private variables ---------------------------------------------------
---*/
static uint8_t usbBuff[MAX_COMMAND_LENGTH];
static uint8_t state = STOP;
static uint8_t stateOld = STOP;

static uint8_t interruptOne = 0;
static uint8_t interruptTwo = 0;

static uint8_t zoffFlag = 0;
static uint8_t echoOnFlag = 0;

extern uint8_t dbSelected;

// *** FIFO VARIABLES *** //
#define FIFO_DEPTH 32

static char FIFO_Src = 0, Old_FIFO_Src = 0;
static char FIFO_Ovr = 0;
static char FIFO_interrupt = 0;
static char FIFO_count = 0;
static char FIFO_case = STOP;
static unsigned char state_machine = 0;
static unsigned char Power_Mode;
static char Fifo_depth = 32;
// *** END FIFO VARIABLES *** //

/* Private function prototypes -----------------------------------------
---*/
void l3g4200d_AppTick(void);
static uint8_t Decod();
static void StateMachine(void);
static void Version(void);
static void Device(void);
```

```
static void Zoff(void);
static void Zon(void);
static void Start(void);
static void Stop(void);
static void ReadReg(void);
static void WriteReg(void);
static void Debug(void);
static void InterruptReceived(uint8_t interrupt);
static void FifoBypass(void);
static void FifoStream(void);
static void FifoMode(void);
static void FifoRun(void);
static void FifoRead(void);
static void SendFiFoSrc(void);
static void SendFiFoDataVector(void);
static void FifoStreamToFifo(void);
static void FifoBypassToStream(void);
static void DbReset(void);

/* Private functions ---------------------------------------------------
---*/


/*******************************************************************************
*****
* Function Name  : l3gd20_AppTick
* Description    : gyro 3D Application Tick Function
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
void l3gd20_AppTick(void) {
  EKSTM32_LEDOn(LED3);
  if(newCommandAvailable(usbBuff)) {

    state = Decod();
  }
  StateMachine();
}

/*******************************************************************************
*****
* Function Name   : Decod
* Description     : decode the incoming USB data
**                      :  decodification of RTX vector
**                      :  *start/r:           :starts sending data
**                      :  *stop/r:            :stops sending data
**                      :  *ver/r:             :sends firmware version
number
**                      :  *dev/r:             :sends device name
**                      :  *wxxdd/r:           :writes in register at
address xx:(0xadr) --> the value data dd:(0xdata)
**                      :  *rxx/r:             :reads register at address
xx:(0xadr)
**                      :  *zon/r:             :activates tristate mode
**                      :  *zoff/r:            :activates connections with
devices
**
**   Realizes the command interpreter (for general purpose commands).
```

```
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static uint8_t Decod(void){

  if(strcmp((char*)&usbBuff[1],"VER")==0) {
    stateOld = state;
    state = VER;
  }
  else if(strcmp((char*)&usbBuff[1],"DEV")==0) {
    stateOld = state;
    state = DEV;
  }
  else if(strcmp((char*)&usbBuff[1],"ZOFF")==0) {
    stateOld = state;
    state = ZOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"ZON")==0) {
    state = ZON;
  }
  else if(strcmp((char*)&usbBuff[1],"START")==0) {
    stateOld = state;
    state = START;
  }
  else if(strcmp((char*)&usbBuff[1],"STOP")==0) {
    stateOld = state;
    state = STOP;
  }
  else if(strncmp((char*)&usbBuff[1],"GR", 2)==0) {
    stateOld = state;
    state = R;
  }
  else if(strncmp((char*)&usbBuff[1],"GW", 2)==0) {
    stateOld = state;
    state = W;
  }
  else if(strcmp((char*)&usbBuff[1],"DEBUG")==0) {
    state = DEBUG;
  }
  /* FIFO Instructions */
  else if(strcmp((char*)&usbBuff[1],"GFIFORST")==0) {
    state = FIFO_BYPASS;
    FIFO_case = STOP;
  }
  else if(strcmp((char*)&usbBuff[1],"GFIFOMDE")==0) {
    state = FIFO_BYPASS;
    FIFO_case = FIFO_MODE;
  }
  else if(strcmp((char*)&usbBuff[1],"GFIFOSTR")==0) {
    state = FIFO_BYPASS;
    FIFO_case = FIFO_STREAM;
  }
  else if(strcmp((char*)&usbBuff[1],"GFIFOSTF")==0) {
    state = FIFO_BYPASS;
    FIFO_case = FIFO_STREAM_TO_FIFO;
  }
  else if(strcmp((char*)&usbBuff[1],"GFIFOBTS")==0) {
    state = FIFO_BYPASS;
```

```
    FIFO_case = FIFO_BYPASS_TO_STREAM;
  }
  else if(strcmp((char*)&usbBuff[1],"LIST")==0) {
    stateOld = state;
    state = LIST;
  }
  else if(strcmp((char*)&usbBuff[1],"SINGLE")==0) {
    stateOld = state;
    state = SINGLE;
  }
  else if(strcmp((char*)&usbBuff[1],"ECHOON")==0) {
    stateOld = state;
    state = ECHOON;
  }
  else if(strcmp((char*)&usbBuff[1],"ECHOOFF")==0) {
    stateOld = state;
    state = ECHOOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"LISTDEV")==0) {
    stateOld = state;
    state = LISTDEV;
  }
  else if(strcmp((char*)&usbBuff[1],"DBRESET")==0) {
    stateOld = state;
    state = DBRESET;
  }

 return state;
}


/*************************************************************************
*****
* Function Name  : StateMachine
* Description    : StateMachiine is called from the aAppTick main loop
*                  It calls functions that implement commands selected
*                  It implements state machine for commands interpreter.
*                  If Set of available commands is extended with device
specific commands
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void StateMachine(void) {

  switch(state) {
  case VER:
    Version();
    state = stateOld;
    break;
  case DEV:
    Device();
    state = stateOld;
    break;
  case ZOFF:
    Zoff();
    state = STOP;
    stateOld = STOP;
    break;
```

```
  case ZON:
    Zon();
    state = STOP;
    stateOld = STOP;
    break;
  case START:
    if(zoffFlag) {
      Start();
    }
    break;
  case STOP:
    if(zoffFlag) {
      Stop();
    }
    break;
  case R:
    if(zoffFlag) {
      ReadReg();
      state = stateOld;
    }
    break;
  case W:
    if(zoffFlag) {
      WriteReg();
      state = stateOld;
    }
    break;
  case DEBUG:
    if(zoffFlag) {
      Debug();
    }
    break;
  case FIFO_BYPASS:
    if(zoffFlag) {
      FifoBypass();
    }
    break;
  case FIFO_STREAM:
    if(zoffFlag) {
      FifoStream();
    }
    break;
  case FIFO_MODE:
    if(zoffFlag) {
      FifoMode();
    }
    break;
  case FIFO_RUN:
    if(zoffFlag) {
      FifoRun();
    }
    break;
  case FIFO_READ:
    if(zoffFlag) {
      FifoRead();
    }
    break;
  case FIFO_STREAM_TO_FIFO:
    if(zoffFlag) {
      FifoStreamToFifo();
```

```
    }
    break;
  case FIFO_BYPASS_TO_STREAM:
    if(zoffFlag) {
      FifoBypassToStream();
    }
    break;
  case LIST:
    //It Prints the list of firmwares implemented
    PrintListOfFirmware();
    state = stateOld;
    break;
  case SINGLE:
    if(zoffFlag) {
      Debug();
      state = stateOld;
    }
    break;
  case ECHOON:
      echoOnFlag = 1;
      state = stateOld;
    break;
  case ECHOOFF:
      echoOnFlag = 0;
      state = stateOld;
    break;
  case LISTDEV:
    //It Prints the list of device supported
    PrintListOfDevice();
    state = stateOld;
    break;
  case DBRESET:
    //unselect the db
    DbReset();
    state = STOP;
    break;
  }

}

/**************************************************************************
*****
* Function Name  : Version
* Description    : Function to send Firmware version to host by USB
* Input          : None
* Output         : None
* Return         : None
**************************************************************************
****/
static void Version(void) {
 //eMotion firmware version
  EmotionVer();
}

/**************************************************************************
*****
* Function Name  : Device
* Description    : Function to send the Device Product Code
* Input          : None
* Output         : None
```

```
* Return         : None
*******************************************************************************
****/
static void Device(void) {
  uint8_t buffer[]="L3GD20\r\n";

  MEMS_Print(buffer, sizeof(buffer));

}

/*******************************************************************************
*****
* Function Name  : Zoff
* Description    : It initializes the peripherals according to the device
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zoff(void) {
  EKSTM32_LEDOn(LED2);
  RegisterInterrupt(InterruptReceived);
  SPI_Mems_Init();
  zoffFlag = 1;
}


/*******************************************************************************
*****
* Function Name  : Zon
* Description    : It DeInitializes the peripherals according to the device
*                  Now the device is disconnected from the board
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zon(void) {
  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Pin = SPI_MEMS_PIN_SCK | SPI_MEMS_PIN_MISO |
SPI_MEMS_PIN_MOSI;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(SPI_MEMS_GPIO, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = SPI_MEMS_CS;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(SPI_MEMS_CS_GPIO, &GPIO_InitStructure);

  UnRegisterInterrupt();

  EKSTM32_LEDOff(LED2);

  zoffFlag = 0;

}


/*******************************************************************************
*****
```

```
* Function Name  : Start
* Description    : Function to Send the standard output data to USB
(associated to '*start' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by acceleration, interrupt and
*                  switch status.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Start(void) {

  uint8_t dataReady = 0;
  uint8_t buffer[13];

  dataReady = SPI_Mems_Read_Reg(STATUS_REG);
  if(ValBit(dataReady, DATAREADY_BIT)) {
    EKSTM32_LEDToggle(LED1);

    buffer[0]   = 's';
    buffer[1]   = 't';
    buffer[3]   = SPI_Mems_Read_Reg( OUT_P_L );//reads X LSB
    buffer[2]   = SPI_Mems_Read_Reg( OUT_P_H );//reads X MSB
    buffer[5]   = SPI_Mems_Read_Reg( OUT_R_L );//reads Y LSB
    buffer[4]   = SPI_Mems_Read_Reg( OUT_R_H );//reads Y MSB
    buffer[7]   = SPI_Mems_Read_Reg( OUT_Y_L );//reads Z LSB
    buffer[6]   = SPI_Mems_Read_Reg( OUT_Y_H );//reads Z MSB


    //interrupt 1
    if(interruptOne == 1) {
      //interruptOne = 0;
      buffer[8]   = 0x40;
    }
    else {
      buffer[8]   = 0x00;
    }

    //interrupt 2
    if(interruptTwo == 1) {
      //interruptTwo = 0;
      buffer[9]   = 0x40;
    }
    else {
      buffer[9]   = 0x00;
    }

    buffer[10] = 0;

    if(SW1_BUTTON_PIN)
      buffer[10] |= 0x01;

    if(SW2_BUTTON_PIN)
      buffer[10] |= 0x02;

    buffer[11]  = '\r';

    buffer[12]  = '\n';
```

```
      MEMS_Print(buffer, sizeof(buffer));
  }


}


/*******************************************************************************
*****
* Function Name  : Stop
* Description    : It stops the state machine
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Stop(void){
  state = STOP;
}


/*******************************************************************************
*****
* Function Name  : WriteReg
* Description    : write a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void WriteReg(void) {
  uint8_t data = 0;
  uint8_t address = 0;


    if( isxdigit(usbBuff[3]) && isxdigit(usbBuff[4]) &&
isxdigit(usbBuff[5]) && isxdigit(usbBuff[6]) ) {
      address = StringToChar(&usbBuff[3]);
      data = StringToChar(&usbBuff[5]);
      SPI_Mems_Write_Reg(address, data);
    }

    if(echoOnFlag) {
      uint8_t buffer[10];
      uint8_t response[2];

      address = StringToChar(&usbBuff[3]);

      data = SPI_Mems_Read_Reg(address);
      CharToString(data, response);

      buffer[0]  = 'G';
      buffer[1]  = 'R';
      buffer[2]  = usbBuff[3];
      buffer[3]  = usbBuff[4];
      buffer[4]  = 'h';
      buffer[5]  = response[0];
      buffer[6]  = response[1];
      buffer[7]  = 'h';
      buffer[8]  = '\r';
```

```
      buffer[9]  = '\n';

        MEMS_Print(buffer, sizeof(buffer));
  }
}

/*************************************************************************
*****
* Function Name  : ReadReg
* Description    : read a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void ReadReg(void) {
  uint8_t data = 0;
  uint8_t address = 0;
  uint8_t response[2];
  uint8_t buffer[10];

  address = StringToChar(&usbBuff[3]);
  data = SPI_Mems_Read_Reg(address);
  CharToString(data, response);

  buffer[0]  = 'G';
  buffer[1]  = 'R';
  buffer[2]  = usbBuff[3];
  buffer[3]  = usbBuff[4];
  buffer[4]  = 'h';
  buffer[5]  = response[0];
  buffer[6]  = response[1];
  buffer[7]  = 'h';
  buffer[8]  = '\r';
  buffer[9] = '\n';

  MEMS_Print(buffer, sizeof(buffer));
}

/*************************************************************************
*****
* Function Name  : Debug
* Description    : Function to Send the debug output data to USB
(associated to '*debug' command).
*                  The system send continuosly through the USB interface
*                  a readable string containg acquired data
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void Debug(void) {
  uint8_t buffer[28];
  uint8_t dataReady = 0;
  int16_t data;

  dataReady = SPI_Mems_Read_Reg(STATUS_REG);


  if(ValBit(dataReady, DATAREADY_BIT)) {
```

```
    EKSTM32_LEDToggle(LED1);

    data = (int16_t) ( (SPI_Mems_Read_Reg(OUT_P_H)<<8) |
SPI Mems Read Reg(OUT P L) );
    sprintf((char*)buffer,"P=%+06d ",data);

    data = (int16_t) ( (SPI_Mems_Read_Reg(OUT_R_H)<<8) |
SPI_Mems_Read_Reg(OUT_R_L) );
    sprintf((char*)&buffer[9],"R=%+06d ",data);

    data = (int16_t) ( (SPI_Mems_Read_Reg(OUT_Y_H)<<8) |
SPI_Mems_Read_Reg(OUT_Y_L) );
    sprintf((char*)&buffer[18],"Y=%+06d\r\n",data);


      MEMS_Print(buffer, sizeof(buffer));
  }

}

/*******************************************************************************
*****
* Function Name  : InterruptReceived
* Description    : Interrupt One and Two Handler
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void InterruptReceived(uint8_t interrupt) {
  uint8_t data;

  if(interrupt == INTERRUPTONE) { //int1
    //read pin state
    data = GPIO_ReadInputDataBit(INT1_PORT, INT1_PIN);
    if(data == 0x01)
    interruptOne = 1;
    else
      interruptOne = 0;
  }

  else if(interrupt == INTERRUPTTWO) { //int2
    //read pin state
    data = GPIO_ReadInputDataBit(INT2_PORT, INT2_PIN);
    if(data == 0x01)
    interruptTwo = 1;
    else
      interruptTwo = 0;
  }

}


/*******************************************************************************
*****
* Function Name  : FifoStreamToFifo
* Description    : Fifo stream Function
* Input          : None
* Output         : None
* Return         : None
```

```
*****************************************************************************
****/
static void FifoStreamToFifo(void) {

  switch (state_machine) {

  case 0:
    // Clear Interrupt if LIR enabled
    SPI_Mems_Read_Reg(INT_SRC);
    // Interrupt spend 1/2 samples to go low!!!

    SPI_Mems_Write_Reg(CTRL_REG5,
(SPI_Mems_Read_Reg(CTRL_REG5)|CTRL_REG5_FIFO_EN));
    SPI_Mems_Write_Reg(FIFO_CONTROL, ((SPI_Mems_Read_Reg(FIFO_CONTROL) &
FIFO_CONTROL_WTMSAMP)|DEF_FIFO_CTRL_STREAM_TO_FIFO));

    // Send FIFO Source
    SendFiFoSrc();

    state_machine = 1;

    break;

  case 1:
    FIFO_Src = SPI_Mems_Read_Reg(FIFO_SOURCE);
    FIFO_Ovr = FIFO_Src & FIFO_SOURCE_OVRN;

    // If FIFO Overrun and Interrupt
    if (FIFO_Ovr) {
      // Interrupt selection
      FIFO_interrupt = SPI_Mems_Read_Reg(INT_SRC) & INT_SRC_IA;

      if (FIFO_interrupt) {
        state_machine = 2;

      }
    }

    FIFO_Src = SPI_Mems_Read_Reg(FIFO_SOURCE);

    // If new sample in FIFO
    if (FIFO_Src != Old_FIFO_Src) {
      // Send FIFO Source
      SendFiFoSrc();
        Old_FIFO_Src = FIFO_Src;
    }

    break;

  case 2:
   // FIFO needs 1 ODR after TRIGGER
   // ODR 800Hz is selected in order to have fast ODR generation.
   // After 1 ODR FIFO is stopped.
   Power_Mode = SPI_Mems_Read_Reg(CTRL_REG1);
   SPI_Mems_Write_Reg(CTRL_REG1, 0xFF);   // 800Hz
   wait(1000);
   SPI_Mems_Write_Reg(CTRL_REG1, Power_Mode); // Restore previous ODR

   // Set next FIFO case
   FIFO_case = FIFO_STREAM_TO_FIFO;
```

```
   state = FIFO_READ;

   // reset samples counter
   FIFO count=0;
   state_machine = 3;
   break;

  case 3:
   state = STOP;
   break;
  default: break;

  }

}


/**************************************************************************
*****
* Function Name  : FifoBypassToStream
* Description    :
* Input          : None
* Output         : None
* Return         : None
**************************************************************************
****/
static void FifoBypassToStream(void) {

  // Clear Interrupt if LIR enabled
  SPI_Mems_Read_Reg(INT_SRC);
  // Interrupt spend 1/2 samples to go low!!!

  SPI_Mems_Write_Reg(CTRL_REG5,
(SPI_Mems_Read_Reg(CTRL_REG5)|CTRL_REG5_FIFO_EN));
  SPI_Mems_Write_Reg(FIFO_CONTROL, ((SPI_Mems_Read_Reg(FIFO_CONTROL) &
FIFO_CONTROL_WTMSAMP)|DEF_FIFO_CTRL_BYPASS_TO_STREAM));

  // Send FIFO Source
  SendFiFoSrc();

  // In FIFO MODE, case FIFO STREAM is executed only one time!!
  state = FIFO_RUN;

}


/**************************************************************************
*****
* Function Name  : FifoBypass
* Description    :
* Input          : None
* Output         : None
* Return         : None
**************************************************************************
****/
static void FifoBypass(void) {


  // Clear Interrupt if LIR enabled
  while(SPI_Mems_Read_Reg(INT_SRC) & INT_SRC_IA );
```

```
  // Reset FIFO_STOP_Trigger State Machine
  state_machine = 0;

  // BYPASS Mode
  SPI_Mems_Write_Reg(FIFO_CONTROL, ((SPI_Mems_Read_Reg(FIFO_CONTROL) &
FIFO_CONTROL_WTMSAMP)|DEF_FIFO_CTRL_BYPASS));

  // Disable FIFO
  SPI_Mems_Write_Reg(CTRL_REG5,
(SPI_Mems_Read_Reg(CTRL_REG5)&(~CTRL_REG5_FIFO_EN)));

  // Send FIFO Source
  SendFiFoSrc();

  if (SPI_Mems_Read_Reg(CTRL_REG5)& CTRL_REG5_STOP_ON_WTM) {
    Fifo_depth = (SPI_Mems_Read_Reg(FIFO_CONTROL) & FIFO_CONTROL_WTMSAMP) +
1;
  }

  else {
    Fifo_depth = 32;
  }

  // Return to previous FIFO CASE
  state = FIFO_case;

}


/**************************************************************************
*****
* Function Name  : FifoStream
* Description    :
* Input          : None
* Output         : None
* Return         : None
**************************************************************************
****/
static void FifoStream(void) {

  SPI_Mems_Write_Reg(CTRL_REG5,
(SPI_Mems_Read_Reg(CTRL_REG5)|CTRL_REG5_FIFO_EN));
  SPI_Mems_Write_Reg(FIFO_CONTROL, ((SPI_Mems_Read_Reg(FIFO_CONTROL) &
FIFO_CONTROL_WTMSAMP)|DEF_FIFO_CTRL_STREAM));

  // Send FIFO Source
  SendFiFoSrc();

  // In FIFO MODE, case FIFO STREAM is executed only one time!!
  state = FIFO_RUN;

}


/**************************************************************************
*****
* Function Name  : FifoMode
* Description    :
* Input          : None
```

```
* Output          : None
* Return          : None
*******************************************************************************
****/
static void FifoMode(void) {

  SPI_Mems_Write_Reg(CTRL_REG5,
(SPI_Mems_Read_Reg(CTRL_REG5)|CTRL_REG5_FIFO_EN));
  SPI_Mems_Write_Reg(FIFO_CONTROL, ((SPI_Mems_Read_Reg(FIFO_CONTROL) &
FIFO_CONTROL_WTMSAMP)|DEF_FIFO_CTRL_FIFO_MODE));

  // Send FIFO Source
  SendFiFoSrc();

  state = FIFO_RUN;
}



/*******************************************************************************
*****
* Function Name  : FifoRun
* Description    :
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void FifoRun(void) {

  FIFO_Src = SPI_Mems_Read_Reg(FIFO_SOURCE);
  FIFO_Ovr = FIFO_Src & FIFO_SOURCE_OVRN;

  // If new sample in FIFO
  if (FIFO_Src != Old_FIFO_Src) {
    // Send FIFO Source
    SendFiFoSrc();
      Old_FIFO_Src = FIFO_Src;
  }

  // if FIFO is full, read FIFO content
  if (FIFO_Ovr) {
    // Set next FIFO case
    FIFO_case = FIFO_RUN;
    state = FIFO_READ;

    // reset samples counter
    FIFO_count = 0;
  }
}

/*******************************************************************************
*****
* Function Name  : FifoRead
* Description    :
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
```

```
static void FifoRead(void) {
  //FIFO READ : The system send 32 accelerations samples
  //through the USB interface
  //(FIFO content)

  if (FIFO_count < Fifo_depth) {
    SendFiFoDataVector();
      FIFO_count++;
  }
  else
    state = FIFO_case; // Return to previous FIFO CASE

}



/*************************************************************************
*****
* Function Name  : FifoRead
* Description    : Function to Send the standard output data to USB
(associated to '*FIFO' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by, interrupt,
*                  FIFO_CONTROL and FIFO_SOURCE
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void SendFiFoSrc(void) {
  uint8_t FIFO_Vector[14];

  FIFO_Vector[0] = 's';
  FIFO_Vector[1] = 't';

  FIFO_Vector[2] = 0;
  FIFO_Vector[3] = 0;
  FIFO_Vector[4] = 0;
  FIFO_Vector[5] = 0;
  FIFO_Vector[6] = 0;
  FIFO_Vector[7] = 0;

  //interrupt 1
  if(interruptOne == 1) {
    FIFO_Vector[8] = 0x40;
  }
  else {
    FIFO_Vector[8] = 0x00;
  }

  //interrupt 1
  if(interruptTwo == 1) {
    FIFO_Vector[9] = 0x40;
  }
  else {
    FIFO_Vector[9] = 0x00;
  }

  FIFO_Vector[10] = SPI_Mems_Read_Reg(FIFO_CONTROL);
  FIFO_Vector[11] = SPI_Mems_Read_Reg(FIFO_SOURCE);
```

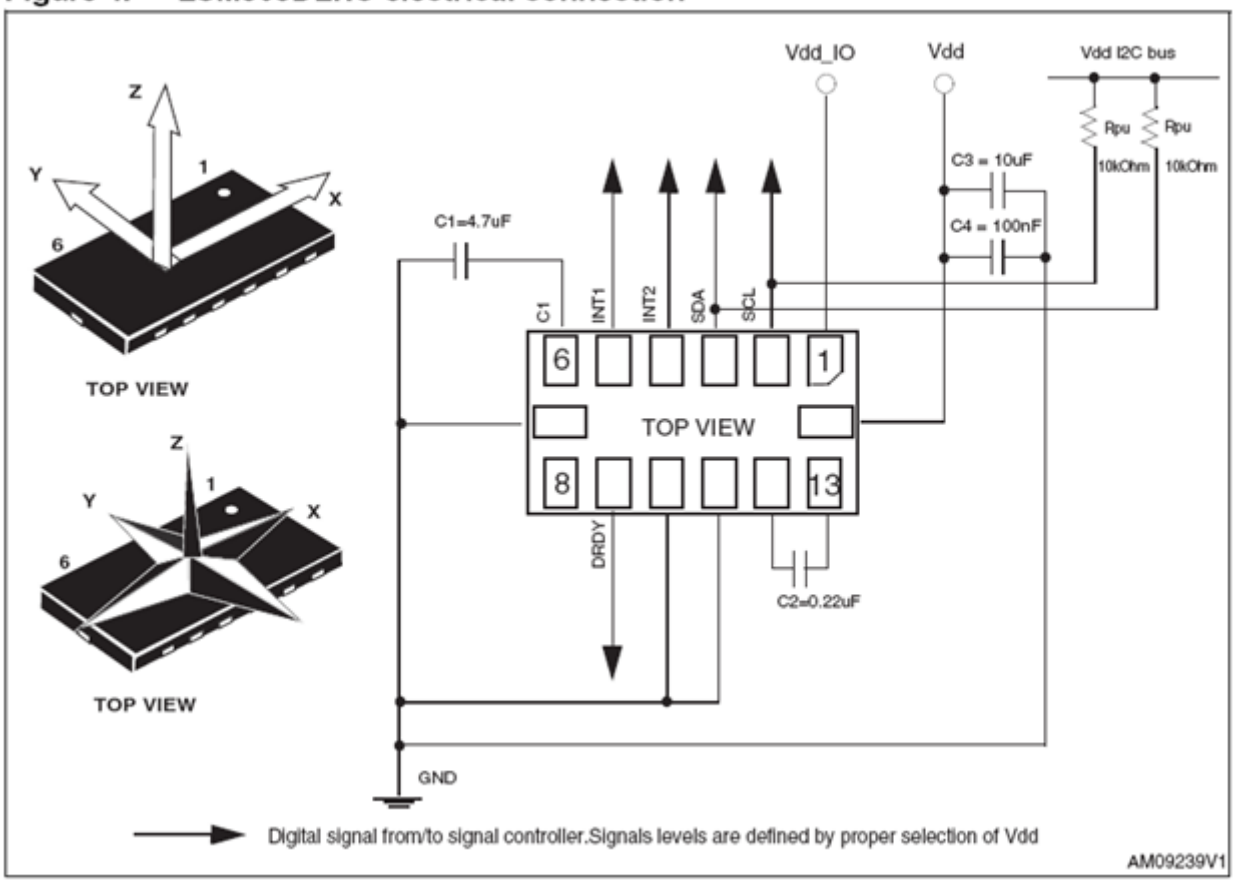
```
  FIFO_Vector[12] = '\r';
  FIFO_Vector[13] = '\n';

  MEMS_Print(FIFO_Vector, sizeof(FIFO_Vector));

  Delay(1);
}


/*******************************************************************************
*****
* Function Name  : SendFiFoDataVector
* Description    : Function to Send the standard output data to USB
(associated to '*FIFO' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by, FIFO Data, interrupt,
*                  FIFO_CONTROL and FIFO_SOURCE
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void SendFiFoDataVector(void) {
  uint8_t FIFO_Data_Vector[14];


  FIFO_Data_Vector[0] = 's';
  FIFO_Data_Vector[1] = 't';


  FIFO_Data_Vector[3] = SPI_Mems_Read_Reg( OUT_P_L );//reads X LSB
  FIFO_Data_Vector[2] = SPI_Mems_Read_Reg( OUT_P_H );//reads X MSB

  FIFO_Data_Vector[5] = SPI_Mems_Read_Reg( OUT_R_L );//reads Y LSB
  FIFO_Data_Vector[4] = SPI_Mems_Read_Reg( OUT_R_H );//reads Y MSB

  FIFO_Data_Vector[7] = SPI_Mems_Read_Reg( OUT_Y_L );//reads Z LSB
  FIFO_Data_Vector[6] = SPI_Mems_Read_Reg( OUT_Y_H );//reads Z MSB

  //interrupt 1
  if(interruptOne == 1) {
    FIFO_Data_Vector[8] = 0x40;
  }
  else {
    FIFO_Data_Vector[8] = 0x00;
  }

  //interrupt 1
  if(interruptTwo == 1) {
    FIFO_Data_Vector[9] = 0x40;
  }
  else {
    FIFO_Data_Vector[9] = 0x00;
  }

  FIFO_Data_Vector[10] = SPI_Mems_Read_Reg(FIFO_CONTROL);
  FIFO_Data_Vector[11] = SPI_Mems_Read_Reg(FIFO_SOURCE);

  FIFO_Data_Vector[12] = '\r';
```

```
  FIFO_Data_Vector[13] = '\n';

  MEMS_Print(FIFO_Data_Vector, sizeof(FIFO_Data_Vector));

  Delay(1);
}

/*******************************************************************************
*****
* Function Name  : DbReset
* Description    : It reset the firmware selectet.
*                : A new firmware can be now selected
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
void DbReset(void) {
  dbSelected = 0;
  Zon();
  EKSTM32_LEDOff(LED1);
  EKSTM32_LEDOff(LED2);
  EKSTM32_LEDOff(LED3);
}


/****************** L3GD20.c *****END OF FILE****/
```

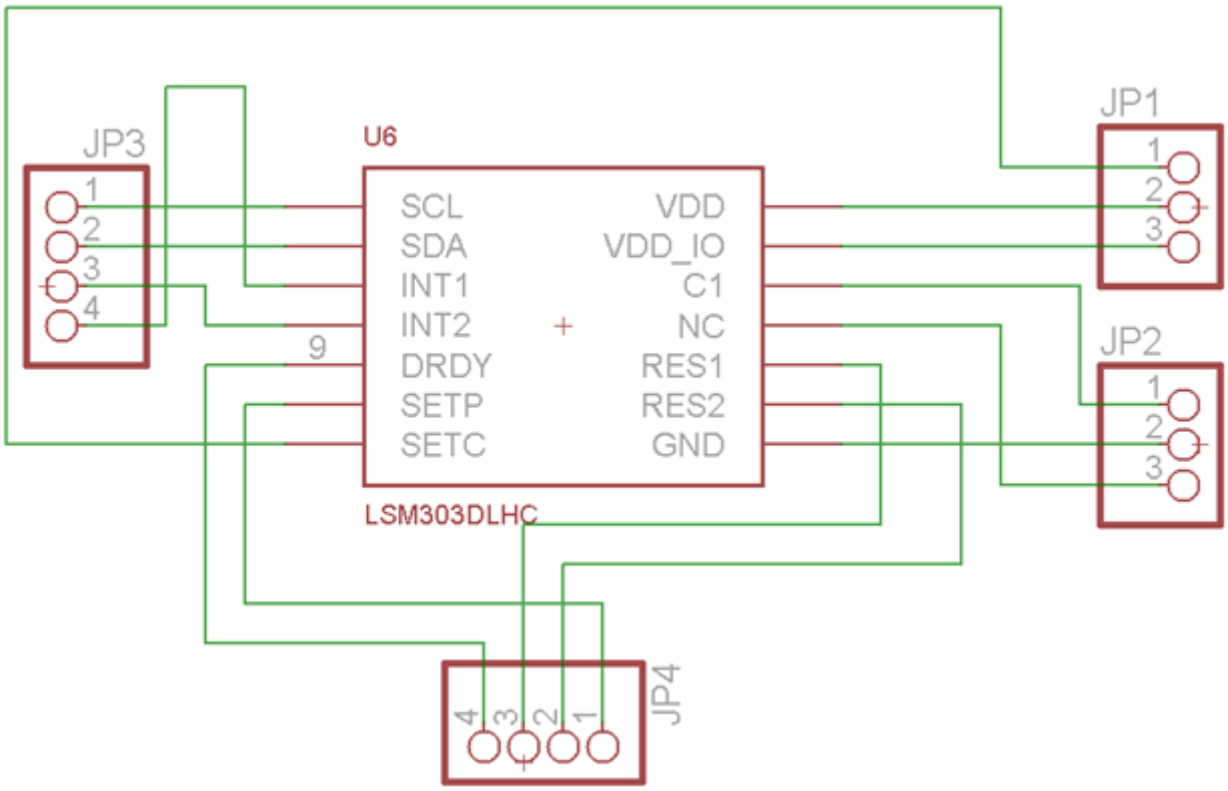
3.4.5 Magnetometr

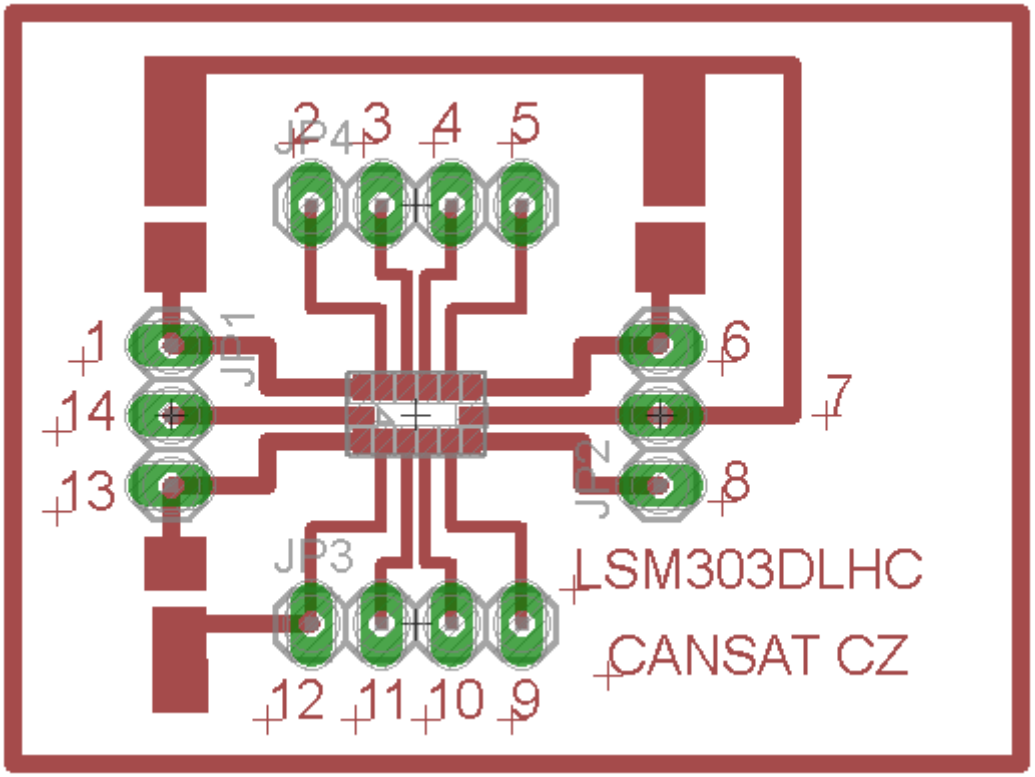
LSM303DLHC 3D magnetometr a 3D akcelerátor. Je v provedení LGA14

Figure 4. LSM303DLHC electrical connection



Po poslání START se na I2C sběrnici pošle 00110011 (33h) nebo 00110010 (32h) (první případ je s posledním bitem 1 tj R/W=1 neboli read, druhý případ je s 0 na konci tj R/W=0 neboli write).
Pozn. Existuje i 3D magnetometr a 3D akcelerometr LSM303DLM v provedení LGA28. Pro ten jsem našel aplikaci LSM303 Breakout Board na stránce http://www.sparkfun.com/products/9810 což je destička s tímto obvodem. Na uvedené stránce jsou dále ke stažení PCB Eagle soubory a dále jsou k dispozici zdrojáky v C pro Arduino v LSM303_example.zip
Pro naše provedení obvodu v LGA14 jsem v EAGLE udělal destičku:¨





Ty tři dvojice plošek jsou na kondenzátory.

Programová obsluha pro STM32F103 – soubory **LSM303DLHC.c** a **LSM303DLHC.h** :

```
/******************** ********************
* File Name          : lsm303dlh.h
* Description        : Descriptor Header for lsm303dlh file
*                     STM32F103
*
*******************************************************************************
****/

/* Define to prevent recursive inclusion -------------------------------
---*/
#ifndef __LSM303DLHC_H
#define __LSM303DLHC_H

/* Includes ------------------------------------------------------------------
---*/
#include "stm32f10x.h"

/* Exported types ------------------------------------------------------------
---*/
/* Exported constants --------------------------------------------------------
---*/

#define BIT(x) ( 1<<(x) )

#define VER                             0
#define DEV                             1
#define ZOFF                            2
#define ZON                             4
#define START                           5
#define STOP                            6
#define R                               7
#define W                               8
#define DEBUG                           9
// FIFO CONFIGURATIONS //
#define FIFO_BYPASS                     10
#define FIFO_STREAM                     11
#define FIFO_MODE                   12
#define FIFO_STOP_TRIG                  13
#define FIFO_READ                   14
#define FIFO_RUN                    15
#define MR                              16
#define MW                              17
#define LIST                            18
#define SINGLE                          19
#define ECHOON                          20
#define ECHOOFF                         21
#define LISTDEV                         22
#define DBRESET                         23


#define OUT_X_L                                         0x28
#define OUT_X_H                                         0x29
#define OUT_Y_L                                         0x2A
#define OUT_Y_H                                         0x2B
#define OUT_Z_L                                         0x2C
```

```
#define OUT_Z_H                                     0x2D

#define DATAREADY_BIT   STATUS_REG_ZYXDA

#define FIFO_CONTROL                    0x2E
#define FIFO_CONTROL_TRIG               BIT(5)
#define FIFO_CONTROL_WTMSAMP            (BIT(4) | BIT(3) | BIT(2) | BIT(1) |
BIT(0))

#define DEF_FIFO_CTRL_BYPASS            0x00
#define DEF_FIFO_CTRL_FIFO_MODE         BIT(6)
#define DEF_FIFO_CTRL_STREAM            BIT(7)
#define DEF_FIFO_CTRL_STOPTRIG          (BIT(7) | BIT(6))

#define FIFO_SOURCE                     0x2F
#define FIFO_SOURCE_SAMPLES             (BIT(4) | BIT(3) | BIT(2) | BIT(1) |
BIT(0))
#define FIFO_SOURCE_EMPTY               BIT(5)
#define FIFO_SOURCE_OVRN                BIT(6)
#define FIFO_SOURCE_WTM                 BIT(7)


#define WHO_AM_I                          0x0F // device identification
register - default value

//          Control Register 1
#define CTRL_REG1             0x20
#define CTRL_REG1_DEFAULT               (BIT(2)|BIT(1)|BIT(0))


//          Control Register 2
#define CTRL_REG2             0x21
#define CTRL_REG2_DEFAULT               0


//          Control Register 3
#define CTRL_REG3             0x22
#define CTRL_REG3_DEFAULT               0


//          Control Register 4
#define CTRL_REG4             0x23
#define CTRL_REG4_DEFAULT               0


//          Control Register 5
#define CTRL_REG5             0x24

#define CTRL_REG5_DEFAULT               0

//          STATUS REGISTER
#define STATUS_REG                  0x27

#define STATUS_REG_ZYXOR            7   // 1    :     new data set has over
written the previous one
                                  // 0    :     no overrun has occurred
(default)
#define STATUS_REG_ZOR              6   // 0    :     no overrun has
occurred (default)
```

```
                                    // 1    :     new Z-axis data has over
written the previous one
#define STATUS_REG_YOR              5   // 0    :     no overrun has
occurred (default)
                                    // 1    :     new Y-axis data has over
written the previous one
#define STATUS_REG_XOR              4   // 0    :     no overrun has
occurred (default)
                                    // 1    :     new X-axis data has over
written the previous one
#define STATUS_REG_ZYXDA            3   // 0    :     a new set of data is
not yet avvious one
                                          // 1     :     a new set of
data is available
#define STATUS_REG_ZDA              2   // 0    :     a new data for the Z-
Axis is not availvious one
                                          // 1     :     a new data for
the Z-Axis is available
#define STATUS_REG_YDA              1   // 0    :     a new data for the Y-
Axis is not avail
                                          // 1     :     a new data for
the Y-Axis is available
#define STATUS_REG_XDA              0   // 0    :     a new data for the X-
Axis is not avail


//          INT_SRC
#define INT_SRC                                   0x31
#define INT_SRC_IA                                BIT(6)

//          INT1_SRC
#define INT1_SRC                        0x31
#define INT1_SRC_IA                     BIT(6)

//          INT2_SRC

#define INT2_SRC                        0x35
#define INT2_SRC_IA                     BIT(6)

#define CLICK_SRC                       0x39
#define CLICK_SRC_IA                    BIT(6)

#define CTRL_REG5                       0x24
#define CTRL_REG5_FIFO_EN               BIT(6)

#define MAG_ADDRESS                     0x3C
#define ACC_ADDRESS                     0x32


#define MAG_STATUS_REG                  0x09
#define MAG_DATAREADY_BIT               0

#define MAG_MR_REG_M                    0x02

#define MAG_CFG_REGA                    0x00

#define OUT_MX_L                                0x04
#define OUT_MX_H                                0x03
#define OUT_MY_L                                0x08
#define OUT_MY_H                                0x07
```

```
#define OUT_MZ_L                                0x06
#define OUT_MZ_H                                0x05

#define MAG_IS_FASTEST                  0x00
#define ACC_IS_FASTEST                  0x01

#define MAGNETOMETER                            0x00
#define ACCELEROMETER                           0x01

/* Exported macro ------------------------------------------------------
---*/

/* Exported functions --------------------------------------------------
---*/
void lsm303dlhc_AppTick(void);

#endif /* __LSM303DLHC_H */

/****************** lsm303dlh.h *****END OF FILE****/
```

```
/******************* ******************
* File Name          : lsm303dlhc.c
* Description        : This file provides a set of functions needed to
manage the
*                      lsm303dlhc adapter board.
*                      STM32F103
*
********************************************************************************
****/

/* Includes ------------------------------------------------------------------
---*/
#include "lsm303dlhc.h"
#include "led.h"
#include "command_interpreter.h"
#include "string.h"
#include "usb_lib.h"
#include "usb_desc.h"
#include "usb_pwr.h"
#include "ctype.h"
#include "utility.h"
#include "interruptHandler.h"
#include "i2c_mems.h"
#include "hw_config.h"
#include "stdio.h"
#include "fw_selector.h"

/* Private typedef -----------------------------------------------------------
---*/
/* Private define ------------------------------------------------------------
---*/
#define SENSORS_NUMBER 2
/* Private macro -------------------------------------------------------------
---*/
#define ValBit(VAR,Place)         (VAR & (1<<Place))
/* Private variables ---------------------------------------------------------
---*/
```

```
static uint8_t usbBuff[MAX_COMMAND_LENGTH];
static uint8_t state = STOP;
static uint8_t stateOld = STOP;

static uint8_t magDataBuffer[6];
static uint8_t accDataBuffer[6];

static uint16_t magODR[] = {7,15,30,75,150,300,750,2200};
static uint16_t accODR[] = {0,10,100,250,500,1000,2000,4000,16000,12500};
static uint8_t fastest = 10; //no sensor is the fastest
static uint8_t sendFlag = 0;

static uint16_t actualSpeed[SENSORS_NUMBER] = {0};

static uint8_t interruptOne = 0;
static uint8_t interruptTwo = 0;

static uint8_t zoffFlag = 0;
static uint8_t echoOnFlag = 0;

extern uint8_t dbSelected;

// *** FIFO VARIABLES *** //
#define FIFO_DEPTH 32

static char FIFO_Src = 0, Old_FIFO_Src = 0;
static char FIFO_Ovr = 0;
static char FIFO_Trig = 0;
static char FIFO_interrupt = 0;
static char FIFO_count = 0;
static char FIFO_case = STOP;
static unsigned char state_machine = 0;
static unsigned char Power_Mode;
// *** END FIFO VARIABLES *** //

/* Private function prototypes -----------------------------------------
---*/
void lsm303dlhc_AppTick(void);
static uint8_t Decod();
static void StateMachine(void);
static void Version(void);
static void Device(void);
static void Zoff(void);
static void Zon(void);
static void Start(void);
static void Stop(void);
static void ReadReg(void);
static void WriteReg(void);
static void MagReadReg(void);
static void MagWriteReg(void);
static void Debug(void);
static void InterruptReceived(uint8_t interrupt);
static void DbReset(void);
static uint8_t GetFastest(uint16_t* buff, uint8_t elements);

//fifo functions
static void FifoBypass(void);
static void FifoStream(void);
static void FifoMode(void);
static void FifoRun(void);
```

```
static void FifoStopTrig(void);
static void FifoRead(void);
static void SendFiFoSrc(void);
static void SendFiFoDataVector(void);
/* Private functions ---------------------------------------------------
---*/


/*******************************************************************************
*****
* Function Name  : lsm303dlhc_AppTick
* Description    : lsm303dlhc Application Tick Function
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
void lsm303dlhc_AppTick(void) {
  EKSTM32_LEDOn(LED3);
  if(newCommandAvailable(usbBuff)) {

    state = Decod();
  }

  StateMachine();

}

/*******************************************************************************
*****
* Function Name   : Decod
* Description     : decode the incoming USB data
**                      :  decodification of RTX vector
**                      :  *start/r:            :starts sending data
**                      :  *stop/r:             :stops sending data
**                      :  *ver/r:              :sends firmware version
number
**                      :  *dev/r:            :sends device name
**                      :  *wxxdd/r:          :writes in register at
address xx:(0xadr) --> the value data dd:(0xdata)
**                      :  *rxx/r:              :reads register at address
xx:(0xadr)
**                      :  *zon/r:                :activates tristate mode
**              :  *zoff/r:           :activates connections with devices
**
**   Realizes the command interpreter (for general purpose commands).
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static uint8_t Decod(){

  if(strcmp((char*)&usbBuff[1],"VER")==0) {
    stateOld = state;
    state = VER;
  }
  else if(strcmp((char*)&usbBuff[1],"DEV")==0) {
    stateOld = state;
    state = DEV;
```

```
  }
  else if(strcmp((char*)&usbBuff[1],"ZOFF")==0) {
    stateOld = state;
    state = ZOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"ZON")==0) {
    stateOld = state;
    state = ZON;
  }
  else if(strcmp((char*)&usbBuff[1],"START")==0) {
    stateOld = state;
    state = START;
  }
  else if(strcmp((char*)&usbBuff[1],"STOP")==0) {
    stateOld = state;
    state = STOP;
  }
  else if( usbBuff[1] == 'R' ) {
    stateOld = state;
    state = R;
  }
  else if( usbBuff[1] == 'W' ) {
    stateOld = state;
    state = W;
  }
  else if(strncmp((char*)&usbBuff[1],"MR", 2)==0) {
    stateOld = state;
    state = MR;
  }
  else if(strncmp((char*)&usbBuff[1],"MW", 2)==0) {
    stateOld = state;
    state = MW;
  }
  else if(strcmp((char*)&usbBuff[1],"DEBUG")==0) {
    state = DEBUG;
  }
  else if(strcmp((char*)&usbBuff[1],"FIFOSTR")==0) {
    state = FIFO_BYPASS;
    FIFO_case = FIFO_STREAM;
  }
  else if(strcmp((char*)&usbBuff[1],"FIFOMDE")==0) {
    state = FIFO_BYPASS;
    FIFO_case = FIFO_MODE;
  }
  else if(strcmp((char*)&usbBuff[1],"FIFOTRG")==0) {
    state = FIFO_BYPASS;
    FIFO_case = FIFO_STOP_TRIG;
  }
  else if(strcmp((char*)&usbBuff[1],"FIFORST")==0) {
    state = FIFO_BYPASS;
    FIFO_case = STOP;
  }
  else if(strcmp((char*)&usbBuff[1],"LIST")==0) {
    stateOld = state;
    state = LIST;
  }
  else if(strcmp((char*)&usbBuff[1],"SINGLE")==0) {
    stateOld = state;
    state = SINGLE;
  }
```

```
  else if(strcmp((char*)&usbBuff[1],"ECHOON")==0) {
    stateOld = state;
    state = ECHOON;
  }
  else if(strcmp((char*)&usbBuff[1],"ECHOOFF")==0) {
    stateOld = state;
    state = ECHOOFF;
  }
  else if(strcmp((char*)&usbBuff[1],"LISTDEV")==0) {
    stateOld = state;
    state = LISTDEV;
  }
  else if(strcmp((char*)&usbBuff[1],"DBRESET")==0) {
    stateOld = state;
    state = DBRESET;
  }
 return state;
}

/*************************************************************************
*****
* Function Name  : StateMachine
* Description    : StateMachiine is called from the aAppTick main loop
*                  It calls functions that implement commands selected
*                  It implements state machine for commands interpreter.
*                  If Set of available commands is extended with device
specific commands
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void StateMachine(void) {

  switch(state) {
  case VER:
    Version();
    state = stateOld;
    break;
  case DEV:
    Device();
    state = stateOld;
    break;
  case ZOFF:
    Zoff();
    state = STOP;
    stateOld = STOP;
    break;
  case ZON:
    Zon();
    state = STOP;
    stateOld = STOP;
    break;
  case START:
    if(zoffFlag) {
      Start();
    }
    break;
  case STOP:
    if(zoffFlag) {
```

```
      Stop();
    }
    break;
  case R:
    if(zoffFlag) {
      ReadReg();
      state = stateOld;
    }
    break;
  case W:
    if(zoffFlag) {
      WriteReg();
      state = stateOld;
    }
    break;
  case MR:
    if(zoffFlag) {
      MagReadReg();
      state = stateOld;
    }
    break;
  case MW:
    if(zoffFlag) {
      MagWriteReg();
      state = stateOld;
    }
    break;
  case DEBUG:
    if(zoffFlag) {
      Debug();
    }
    break;
  case FIFO_BYPASS:
    if(zoffFlag) {
      FifoBypass();
    }
    break;
  case FIFO_STREAM:
    if(zoffFlag) {
      FifoStream();
    }
    break;
  case FIFO_MODE:
    if(zoffFlag) {
      FifoMode();
    }
    break;
  case FIFO_RUN:
    if(zoffFlag) {
      FifoRun();
    }
    break;
  case FIFO_STOP_TRIG:
    if(zoffFlag) {
      FifoStopTrig();
    }
    break;
  case FIFO_READ:
    if(zoffFlag) {
      FifoRead();
```

```
    }
    break;
  case LIST:
    //It Prints the list of firmwares implemented
    PrintListOfFirmware();
    state = stateOld;
    break;
  case SINGLE:
    if(zoffFlag) {
      Debug();
      state = stateOld;
    }
    break;
  case ECHOON:
      echoOnFlag = 1;
      state = stateOld;
    break;
  case ECHOOFF:
      echoOnFlag = 0;
      state = stateOld;
    break;
  case LISTDEV:
    //It Prints the list of device supported
    PrintListOfDevice();
    state = stateOld;
    break;
  case DBRESET:
    //unselect the db
    DbReset();
    state = STOP;
    break;
  }
}


/*****************************************************************************
*****
* Function Name  : Version
* Description    : Function to send Firmware version to host by USB
* Input          : None
* Output         : None
* Return         : None
*****************************************************************************
****/
static void Version(void) {

  //eMotion firmware version
  EmotionVer();

}

/*****************************************************************************
*****
* Function Name  : Device
* Description    : Function to send the Device Product Code
* Input          : None
* Output         : None
* Return         : None
*****************************************************************************
****/
```

```
static void Device(void) {
  uint8_t buffer[] = "LSM303DLHC \r\n";

  MEMS_Print(buffer, sizeof(buffer));
}



/*******************************************************************************
*****
* Function Name  : Zoff
* Description    : It initializes the peripherals according to the device
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zoff(void) {
  uint8_t data;

  RegisterInterrupt(InterruptReceived);

  //Initializes I2C Bus
  I2C_MEMS_Init();

  memset(magDataBuffer,0,6);
  memset(accDataBuffer,0,6);

  //Check which sensor is the fastest
  I2C_BufferRead(&data, MAG_ADDRESS, MAG_CFG_REGA, 1);
  data = (data & 0x1c) >> 2;
  actualSpeed[MAGNETOMETER] = magODR[data];

  I2C_BufferRead(&data, ACC_ADDRESS, CTRL_REG1, 1);
  data = (data & 0xf0) >> 4;
  actualSpeed[ACCELEROMETER] = accODR[data];

  fastest = GetFastest(actualSpeed, SENSORS_NUMBER);

  zoffFlag = 1;

  EKSTM32_LEDOn(LED2);

}

/*******************************************************************************
*****
* Function Name  : Zon
* Description    : It DeInitializes the peripherals according to the device
*                  Now the device is disconnected from the board
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Zon(void) {
  I2C_MEMS_DeInit();
  GPIOsDeInit();
  UnRegisterInterrupt();
  zoffFlag = 0;
```

```
  EKSTM32_LEDOff(LED2);
}


/*******************************************************************************
*****
* Function Name  : Start
* Description    : Function to Send the standard output data to USB
(associated to '*start' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by acceleration, interrupt and
*                  switch status.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Start(void) {

  uint8_t dataReady = 0;
  uint8_t buffer[19];
  uint8_t data = 0;

  if( fastest == MAG_IS_FASTEST) {
      I2C_BufferRead(&data, MAG_ADDRESS, MAG_STATUS_REG, 1);

    if(ValBit(data, MAG_DATAREADY_BIT)) {
      sendFlag = 1;
    }

  }
  else if(fastest == ACC_IS_FASTEST) {
      I2C_BufferRead(&dataReady, ACC_ADDRESS, STATUS_REG, 1);
      if(ValBit(dataReady, DATAREADY_BIT)) {
        sendFlag = 1;
      }
  }

  if(sendFlag) {
      sendFlag = 0;
    EKSTM32_LEDToggle(LED1);

    buffer[0]   = 's';
    buffer[1]   = 't';

      //Accelerometer multi read
      I2C_BufferRead(accDataBuffer, ACC_ADDRESS, OUT_X_L, 6);  //reads acc


    //dummy read
    I2C_BufferRead(&data, MAG_ADDRESS, MAG_MR_REG_M, 1);

    //magnetometer multi read
    I2C_BufferRead(magDataBuffer, MAG_ADDRESS, OUT_MX_H, 6);  //reads Mag


      buffer[2]   = accDataBuffer[1]; //xh
    buffer[3]   = accDataBuffer[0]; //xl
    buffer[4]   = accDataBuffer[3]; //yh
    buffer[5]   = accDataBuffer[2]; //yl
```

```
    buffer[6]   = accDataBuffer[5]; //zh
    buffer[7]   = accDataBuffer[4]; //zl

    buffer[8]   = magDataBuffer[0];
    buffer[9]   = magDataBuffer[1];
    buffer[12]  = magDataBuffer[2];
    buffer[13]  = magDataBuffer[3];
    buffer[10]  = magDataBuffer[4];
    buffer[11]  = magDataBuffer[5];

    //interrupt 1
    if(interruptOne == 1) {
      //interruptOne = 0;
      buffer[14]   = 0x40;
    }
    else {
      buffer[14]   = 0x00;
    }

    //interrupt 2
    if(interruptTwo == 1) {
      //interruptTwo = 0;
      buffer[15]   = 0x40;
    }
    else {
      buffer[15]   = 0x00;
    }

    buffer[16] = 0;

    if(SW1_BUTTON_PIN)
      buffer[16] |= 0x01;

    if(SW2_BUTTON_PIN)
      buffer[16] |= 0x02;

    buffer[17]  = '\r';
    buffer[18]  = '\n';

      MEMS_Print(buffer, sizeof(buffer));

  }
}

/*************************************************************************
*****
* Function Name  : Stop
* Description    : It stops the state machine
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void Stop(void){
  state = STOP;
}

/*************************************************************************
*****
* Function Name  : WriteReg
```

```
* Description    : write a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void WriteReg() {
  uint8_t data = 0;
  uint8_t address = 0;

  if( isxdigit(usbBuff[2]) && isxdigit(usbBuff[3]) && isxdigit(usbBuff[4])
&& isxdigit(usbBuff[5]) ) {
    address = StringToChar(&usbBuff[2]);
    data = StringToChar(&usbBuff[4]);
    I2C_ByteWrite(&data, ACC_ADDRESS, address);

    if(echoOnFlag) {
        uint8_t buffer[9];
        uint8_t response[2];

        address = StringToChar(&usbBuff[2]);

        I2C_BufferRead(&data, ACC_ADDRESS, address, 1);
        CharToString(data, response);

        buffer[0]  = 'R';
        buffer[1]  = usbBuff[2];
        buffer[2]  = usbBuff[3];
        buffer[3]  = 'h';
        buffer[4]  = response[0];
        buffer[5]  = response[1];
        buffer[6]  = 'h';
        buffer[7]  = '\r';
        buffer[8]  = '\n';

            MEMS_Print(buffer, sizeof(buffer));
    }

      //if the ODR is changed, recalculate the fastest
    if(address == CTRL_REG1) {
      data = (data & 0xf0) >> 4;

        actualSpeed[ACCELEROMETER] = accODR[data];

        fastest = GetFastest(actualSpeed, SENSORS_NUMBER);

    }
  }
}

/*******************************************************************************
*****
* Function Name  : ReadReg
* Description    : read a value in a specific sensor register
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void ReadReg(void) {
```

```
  uint8_t data = 0;
  uint8_t address = 0;
  uint8_t response[2];
  uint8 t buffer[9];

  address = StringToChar(&usbBuff[2]);
  I2C_BufferRead(&data, ACC_ADDRESS, address, 1);
  CharToString(data, response);

  buffer[0]  = 'R';
  buffer[1]  = usbBuff[2];
  buffer[2]  = usbBuff[3];
  buffer[3]  = 'h';
  buffer[4]  = response[0];
  buffer[5]  = response[1];
  buffer[6]  = 'h';
  buffer[7]  = '\r';
  buffer[8]  = '\n';

  MEMS_Print(buffer, sizeof(buffer));

}

/*******************************************************************************
*****
* Function Name  : MagReadReg
* Description    : read a value in a specific magnetometer register
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void MagReadReg(void) {
  uint8_t data = 0;
  uint8_t address = 0;
  uint8_t response[2];
  uint8_t buffer[10];

  address = StringToChar(&usbBuff[3]);
  I2C_BufferRead(&data, MAG_ADDRESS, address, 1);

  CharToString(data, response);

  buffer[0]  = 'M';
  buffer[1]  = 'R';
  buffer[2]  = usbBuff[3];
  buffer[3]  = usbBuff[4];
  buffer[4]  = 'h';
  buffer[5]  = response[0];
  buffer[6]  = response[1];
  buffer[7]  = 'h';
  buffer[8]  = '\r';
  buffer[9]  = '\n';

  MEMS_Print(buffer, sizeof(buffer));

}

/*******************************************************************************
*****
```

```
* Function Name  : MagWriteReg
* Description    : write a value in a specific magnetometer register
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void MagWriteReg(void) {
  uint8_t data = 0;
  uint8_t address = 0;

  if( isxdigit(usbBuff[3]) && isxdigit(usbBuff[4]) && isxdigit(usbBuff[5])
&& isxdigit(usbBuff[6]) ) {
    address = StringToChar(&usbBuff[3]);
    data = StringToChar(&usbBuff[5]);
    I2C_ByteWrite(&data, MAG_ADDRESS, address);
  }

  if(echoOnFlag) {
      uint8_t buffer[10];
      uint8_t response[2];

      address = StringToChar(&usbBuff[3]);

      I2C_BufferRead(&data, MAG_ADDRESS, address, 1);
      CharToString(data, response);

      buffer[0]  = 'M';
      buffer[1]  = 'R';
      buffer[2]  = usbBuff[3];
      buffer[3]  = usbBuff[4];
      buffer[4]  = 'h';
      buffer[5]  = response[0];
      buffer[6]  = response[1];
      buffer[7]  = 'h';
      buffer[8]  = '\r';
      buffer[9]  = '\n';

      MEMS_Print(buffer, sizeof(buffer));
  }

  //if the ODR is changed, recalculate the fastest
  if(address == MAG_CFG_REGA) {
      data = (data & 0x1c) >> 2;

      actualSpeed[MAGNETOMETER] = magODR[data];

      fastest = GetFastest(actualSpeed, SENSORS_NUMBER);

  }

}



/*******************************************************************************
*****
* Function Name  : Debug
* Description    : Function to Send the debug output data to USB
(associated to '*debug' command).
```

```
*                  The system send continuosly through the USB interface
*                  a readable string containg acquired data
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void Debug(void) {
  uint8_t buffer[56];
  uint8_t dataReady = 0;
  uint8_t data;
  int16_t data16 = 0;


  if( fastest == MAG_IS_FASTEST) {
    I2C_BufferRead(&data, MAG_ADDRESS, MAG_STATUS_REG, 1);

    if(ValBit(data, MAG_DATAREADY_BIT)) {
      sendFlag = 1;
    }
  }
  else if(fastest == ACC_IS_FASTEST) {
      I2C_BufferRead(&dataReady, ACC_ADDRESS, STATUS_REG, 1);
      if(ValBit(dataReady, DATAREADY_BIT)) {
        sendFlag = 1;
      }
  }


  if(sendFlag) {
    sendFlag = 0;

    EKSTM32_LEDToggle(LED1);

      I2C_BufferRead(accDataBuffer, ACC_ADDRESS, OUT_X_L, 6);

      //x acceleration data
      data16 = accDataBuffer[1] << 8;
    data16 |= accDataBuffer[0];
    data16 /= 16;

    //put accelerometer x value into the usb buffer
    sprintf((char*)buffer,"aX=%+5d ",data16);

    //y acceleration data
      data16 = accDataBuffer[3] << 8;
    data16 |= accDataBuffer[2];
    data16 /= 16;

    //put accelerometer y value into the usb buffer
    sprintf((char*)&buffer[9],"aY=%+5d ",data16);

    //z acceleration data
      data16 = accDataBuffer[5] << 8;
    data16 |= accDataBuffer[4];
    data16 /= 16;

    //put accelerometer z value into the usb buffer
    sprintf((char*)&buffer[19],"aZ=%+5d ",data16);
```

```
    //dummy read
    I2C_BufferRead(&data, MAG_ADDRESS, MAG_MR_REG_M, 1);

    //multi read magnetometer
    I2C_BufferRead(magDataBuffer, MAG_ADDRESS, OUT_MX_H, 6);

    //put magnetic x value into the usb buffer
    data16 = magDataBuffer[0]*256 + magDataBuffer[1];
    sprintf((char*)&buffer[28],"mX=%+5d ",data16);

    //put magnetic y value into the usb buffer
    data16 = magDataBuffer[4]*256 + magDataBuffer[5];
    sprintf((char*)&buffer[37],"mY=%+5d ",data16);

    //put magnetic z value into the usb buffer
    data16 = magDataBuffer[2]*256 + magDataBuffer[3];
    sprintf((char*)&buffer[46],"mZ=%+5d\r\n",data16);

    MEMS_Print(buffer, sizeof(buffer));
  }

}

/*******************************************************************************
*****
* Function Name  : InterruptReceived
* Description    : Interrupt One and Two Handler
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void InterruptReceived(uint8_t interrupt) {
  uint8_t data;

  if(interrupt == INTERRUPTONE) { //int1
    //read pin state
    data = GPIO_ReadInputDataBit(INT1_PORT, INT1_PIN);
    if(data == 0x01)
    interruptOne = 1;
    else
      interruptOne = 0;
  }

  else if(interrupt == INTERRUPTTWO) { //int2
    //read pin state
    data = GPIO_ReadInputDataBit(INT2_PORT, INT2_PIN);
    if(data == 0x01)
    interruptTwo = 1;
    else
      interruptTwo = 0;
  }

}


/*******************************************************************************
*****
* Function Name  : FifoBypass
* Description    :
```

```
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void FifoBypass(void) {
  uint8_t data = 0;

  //Read Trigger bit
  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
  FIFO_Trig = (data & FIFO_CONTROL_TRIG);

  // Clear Interrupt if LIR enabled
  if (!FIFO_Trig) {
    do {
      I2C_BufferRead(&data, ACC_ADDRESS, INT1_SRC, 1);
    } while(data & INT1_SRC_IA);
  }
  else {
    do {
      I2C_BufferRead(&data, ACC_ADDRESS, INT2_SRC, 1);
    } while(data & INT2_SRC_IA);

  }

  // Reset FIFO_STOP_Trigger State Machine
  state_machine = 0;

  // BYPASS Mode
  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
  data = (data & FIFO_CONTROL_WTMSAMP) | DEF_FIFO_CTRL_BYPASS;
  I2C_ByteWrite(&data, ACC_ADDRESS, FIFO_CONTROL);

  // Disable FIFO
  I2C_BufferRead(&data, ACC_ADDRESS, CTRL_REG5, 1);
  data = data &(~CTRL_REG5_FIFO_EN);
  I2C_ByteWrite(&data, ACC_ADDRESS, CTRL_REG5);

  // Send FIFO Source
  SendFiFoSrc();

  // Return to previous FIFO CASE
  state = FIFO_case;
}

/*******************************************************************************
*****
* Function Name  : FifoStream
* Description    :
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void FifoStream(void) {
  uint8_t data = 0;

  I2C_BufferRead(&data, ACC_ADDRESS, CTRL_REG5, 1);
  data = data | CTRL_REG5_FIFO_EN;
  I2C_ByteWrite(&data, ACC_ADDRESS, CTRL_REG5);
```

```
  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
  data = (data & FIFO_CONTROL_WTMSAMP) | DEF_FIFO_CTRL_STREAM;
  I2C ByteWrite(&data, ACC ADDRESS, FIFO CONTROL);

  // Send FIFO Source
  SendFiFoSrc();

  // In FIFO MODE, case FIFO STREAM is executed only one time!!
  state = FIFO_RUN;

}

/*******************************************************************************
*****
* Function Name  : FifoMode
* Description    :
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void FifoMode(void) {
  uint8_t data = 0;

  I2C_BufferRead(&data, ACC_ADDRESS, CTRL_REG5, 1);
  data = (data | CTRL_REG5_FIFO_EN);
  I2C_ByteWrite(&data, ACC_ADDRESS, CTRL_REG5);

  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
  data = (data  & FIFO_CONTROL_WTMSAMP) | DEF_FIFO_CTRL_FIFO_MODE;
  I2C_ByteWrite(&data, ACC_ADDRESS, FIFO_CONTROL);

  // Send FIFO Source
  SendFiFoSrc();

  state = FIFO_RUN;

}

/*******************************************************************************
*****
* Function Name  : FifoRun
* Description    :
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void FifoRun(void) {
  uint8_t data = 0;

  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_SOURCE, 1);
  FIFO_Src = data;

  FIFO_Ovr = FIFO_Src & FIFO_SOURCE_OVRN;

  // If new sample in FIFO
  if (FIFO_Src != Old_FIFO_Src) {
    // Send FIFO Source
```

```
    SendFiFoSrc();
      Old_FIFO_Src = FIFO_Src;
  }

  // if FIFO is full, read FIFO content
  if (FIFO_Ovr) {
    // Set next FIFO case
    FIFO_case = FIFO_RUN;
    state = FIFO_READ;

    // reset samples counter
    FIFO_count = 0;
  }

}

/**************************************************************************
*****
* Function Name  : FifoStopTrig
* Description    :
* Input          : None
* Output         : None
* Return         : None
**************************************************************************
****/
static void FifoStopTrig(void) {
  uint8_t data = 0;

  switch (state_machine) {
    case 0:
      // Clear Interrupt if LIR enabled
      I2C_BufferRead(&data, ACC_ADDRESS, INT1_SRC, 1);

      I2C_BufferRead(&data, ACC_ADDRESS, INT2_SRC, 1);

      // Interrupt spend 1/2 samples to go low!!!
      I2C_BufferRead(&data, ACC_ADDRESS, CTRL_REG5, 1);
      data = (data  | CTRL_REG5_FIFO_EN);
      I2C_ByteWrite(&data, ACC_ADDRESS, CTRL_REG5);

      I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
      data = (data  & FIFO_CONTROL_WTMSAMP) | DEF_FIFO_CTRL_STOPTRIG;
      I2C_ByteWrite(&data, ACC_ADDRESS, FIFO_CONTROL);

      // Send FIFO Source
      SendFiFoSrc();

      //Read Trigger bit
      I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
      FIFO_Trig = data & FIFO_CONTROL_TRIG;

      state_machine = 1;
      break;

  case 1:
    I2C_BufferRead(&data, ACC_ADDRESS, FIFO_SOURCE, 1);
    FIFO_Src = data;

    FIFO_Ovr = FIFO_Src & FIFO_SOURCE_OVRN;
```

```
    // If FIFO Overrun and Interrupt
    if (FIFO_Ovr) {
      // Interrupt selection
      if (!FIFO_Trig){
        I2C_BufferRead(&data, ACC_ADDRESS, INT1_SRC, 1);
        FIFO_interrupt = data & INT1_SRC_IA;
      }

      else {
        I2C_BufferRead(&data, ACC_ADDRESS, INT2_SRC, 1);
        FIFO_interrupt = data & INT2_SRC_IA;
      }

      if (FIFO_interrupt) {
        state_machine = 2;
      }
    }

    I2C_BufferRead(&data, ACC_ADDRESS, FIFO_SOURCE, 1);
    FIFO_Src = data;

    // If new sample in FIFO
    if (FIFO_Src != Old_FIFO_Src) {
      // Send FIFO Source
      SendFiFoSrc();
        Old_FIFO_Src = FIFO_Src;

    }

  break;

  case 2:
    // FIFO needs 1 ODR after TRIGGER
    // ODR 5KHz is selected in order to have fast ODR generation.
    // After 1 ODR FIFO is stopped.
    I2C_BufferRead(&data, ACC_ADDRESS, CTRL_REG1, 1);
    Power_Mode = data;

    data = 0x97;
    I2C_ByteWrite(&data, ACC_ADDRESS, CTRL_REG1);

    wait(1000);
    data = Power_Mode;
    I2C_ByteWrite(&data, ACC_ADDRESS, CTRL_REG1);

    // Set next FIFO case
    FIFO_case = FIFO_STOP_TRIG;
    state = FIFO_READ;

    // reset samples counter
    FIFO_count = 0;
    state_machine = 3;
  break;

  case 3:
    state = STOP;
  break;

  default: break;
```

```
  }

}

/*************************************************************************
*****
* Function Name  : FifoRead
* Description    :
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void FifoRead(void) {
  //FIFO READ : The system send 32 accelerations samples
  //through the USB interface
  //(FIFO content)

  if (FIFO_count < FIFO_DEPTH) {
    SendFiFoDataVector();
      FIFO_count++;
  }
  else
    state = FIFO_case; // Return to previous FIFO CASE

  wait(1000);
}

/*************************************************************************
*****
* Function Name  : SendFiFoSrc
* Description    : Function to Send the standard output data to USB
(associated to '*FIFO' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by, interrupt,
*                  FIFO_CONTROL and FIFO_SOURCE
* Input          : None
* Output         : None
* Return         : None
*************************************************************************
****/
static void SendFiFoSrc(void) {
  uint8_t FIFO_Vector[20];
  uint8_t data = 0;

  FIFO_Vector[0] = 's';
  FIFO_Vector[1] = 't';

  FIFO_Vector[2] = 0;
  FIFO_Vector[3] = 0;
  FIFO_Vector[4] = 0;
  FIFO_Vector[5] = 0;
  FIFO_Vector[6] = 0;
  FIFO_Vector[7] = 0;

  FIFO_Vector[8] = 0;
  FIFO_Vector[9] = 0;
  FIFO_Vector[10] = 0;
  FIFO_Vector[11] = 0;
  FIFO_Vector[12] = 0;
```

```
  FIFO_Vector[13] = 0;

  if(interruptOne == 1) {
      //interruptOne = 0;
      FIFO_Vector[14]   = 0x40;
  }
  else {
      FIFO_Vector[14]   = 0x00;
  }

  if(interruptTwo == 1) {
      //interruptTwo = 0;
      FIFO_Vector[15]   = 0x40;
  }
  else {
      FIFO_Vector[15]   = 0x00;
  }

  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
  FIFO_Vector[16] = data;

  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_SOURCE, 1);
  FIFO_Vector[17] = data;

  FIFO_Vector[18] = '\r';
  FIFO_Vector[19] = '\n';

  MEMS_Print(FIFO_Vector, sizeof(FIFO_Vector));

  Delay(1);

}

/*******************************************************************************
*****
* Function Name  : SendFiFoDataVector
* Description    : Function to Send the standard output data to USB
(associated to '*FIFO' command).
*                  The system send continuosly through the USB interface
*                  a char vector composed by, FIFO Data, interrupt,
*                  FIFO_CONTROL and FIFO_SOURCE
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
static void SendFiFoDataVector(void) {
  uint8_t FIFO_Data_Vector[14];
  uint8_t data = 0;

  FIFO_Data_Vector[0]   =     's';
  FIFO_Data_Vector[1]   =     't';

  I2C_BufferRead(accDataBuffer, ACC_ADDRESS, OUT_X_L, 6);

  FIFO_Data_Vector[3]   = accDataBuffer[0];     //xl
  FIFO_Data_Vector[2]   = accDataBuffer[1];     //xh
  FIFO_Data_Vector[5]   = accDataBuffer[2];     //yl
  FIFO_Data_Vector[4]   = accDataBuffer[3];     //yh
  FIFO_Data_Vector[7]   = accDataBuffer[4];     //zl
```

```
  FIFO_Data_Vector[6]   = accDataBuffer[5];     //zh

  if(interruptOne == 1) {
      //interruptOne = 0;
      FIFO_Data_Vector[8]   = 0x40;
  }
  else {
      FIFO_Data_Vector[8]   = 0x00;
  }

  if(interruptTwo == 1) {
      //interruptTwo = 0;
      FIFO_Data_Vector[9]   = 0x40;
  }
  else {
      FIFO_Data_Vector[9]   = 0x00;
  }

  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_CONTROL, 1);
  FIFO_Data_Vector[10]  = data;

  I2C_BufferRead(&data, ACC_ADDRESS, FIFO_SOURCE, 1);
  FIFO_Data_Vector[11]  = data;

  FIFO_Data_Vector[12]  =     '\r';
  FIFO_Data_Vector[13]  =     '\n';

  MEMS_Print(FIFO_Data_Vector, sizeof(FIFO_Data_Vector));

  Delay(1);
}

/*******************************************************************************
*****
* Function Name  : DbReset
* Description    : It reset the firmware selectet.
*                : A new firmware can be now selected
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************
****/
void DbReset(void) {
  dbSelected = 0;
  Zon();
  EKSTM32_LEDOff(LED1);
  EKSTM32_LEDOff(LED2);
  EKSTM32_LEDOff(LED3);
}

/*******************************************************************************
*****
* Function Name  : GetFastest(uint16_t* buff, uint8_t elements)
* Description    : It gets the fastest odr
* Input          : bufferNone
* Output         : None
* Return         : None
*******************************************************************************
****/
uint8_t GetFastest(uint16_t* buff, uint8_t elements) {
```

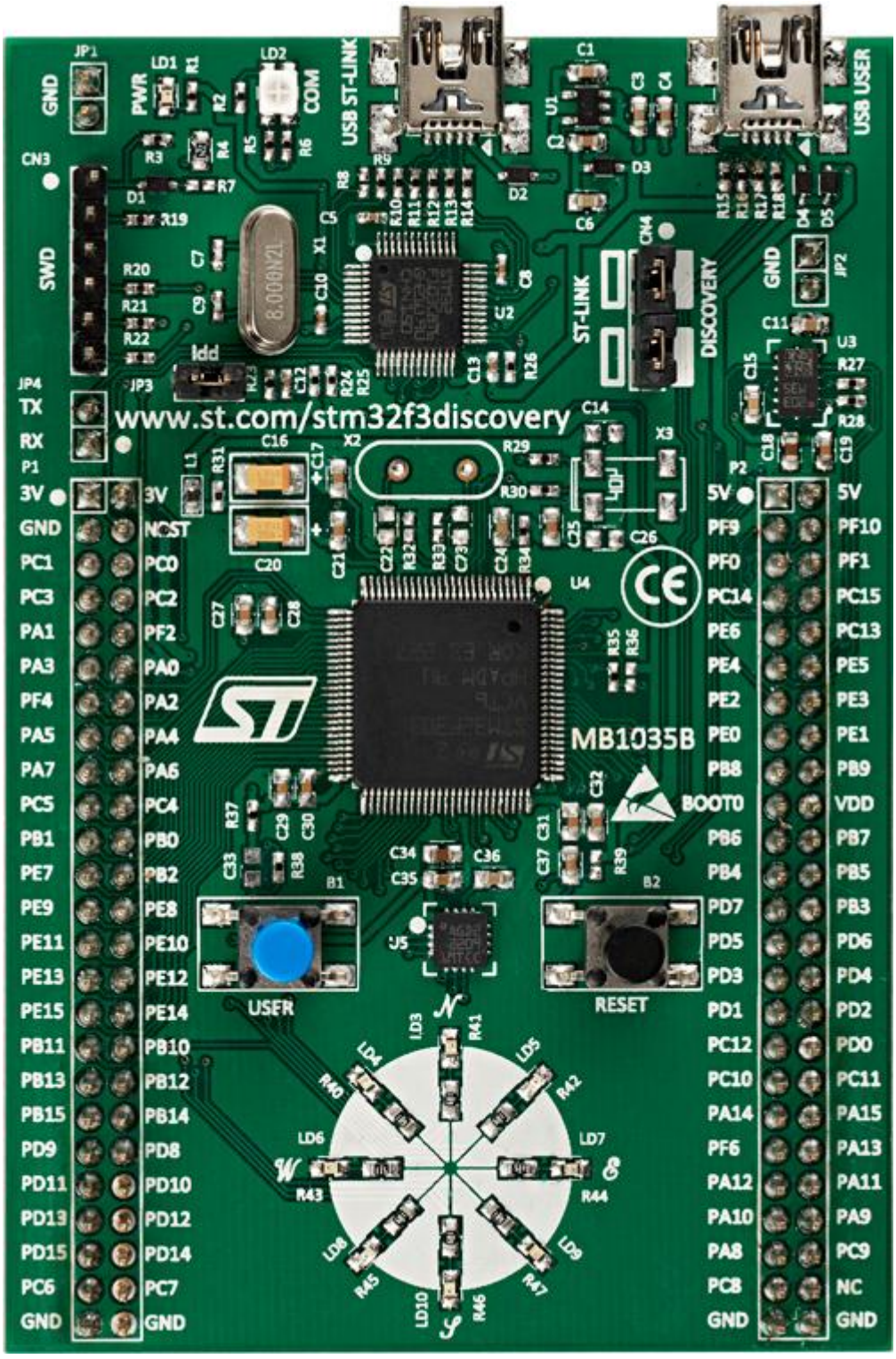
```
  uint8_t ret = 0;
  uint8_t i = 0;

  ret = 0;
  for(i = 0; i < elements; i++) {
      if(buff[ret] < buff[i])
        ret = i;
  }

  return ret;
}

/****************** lsm303dlhc.c ****END OF FILE****/
```

3.5 Startkit STM32F4Discovery

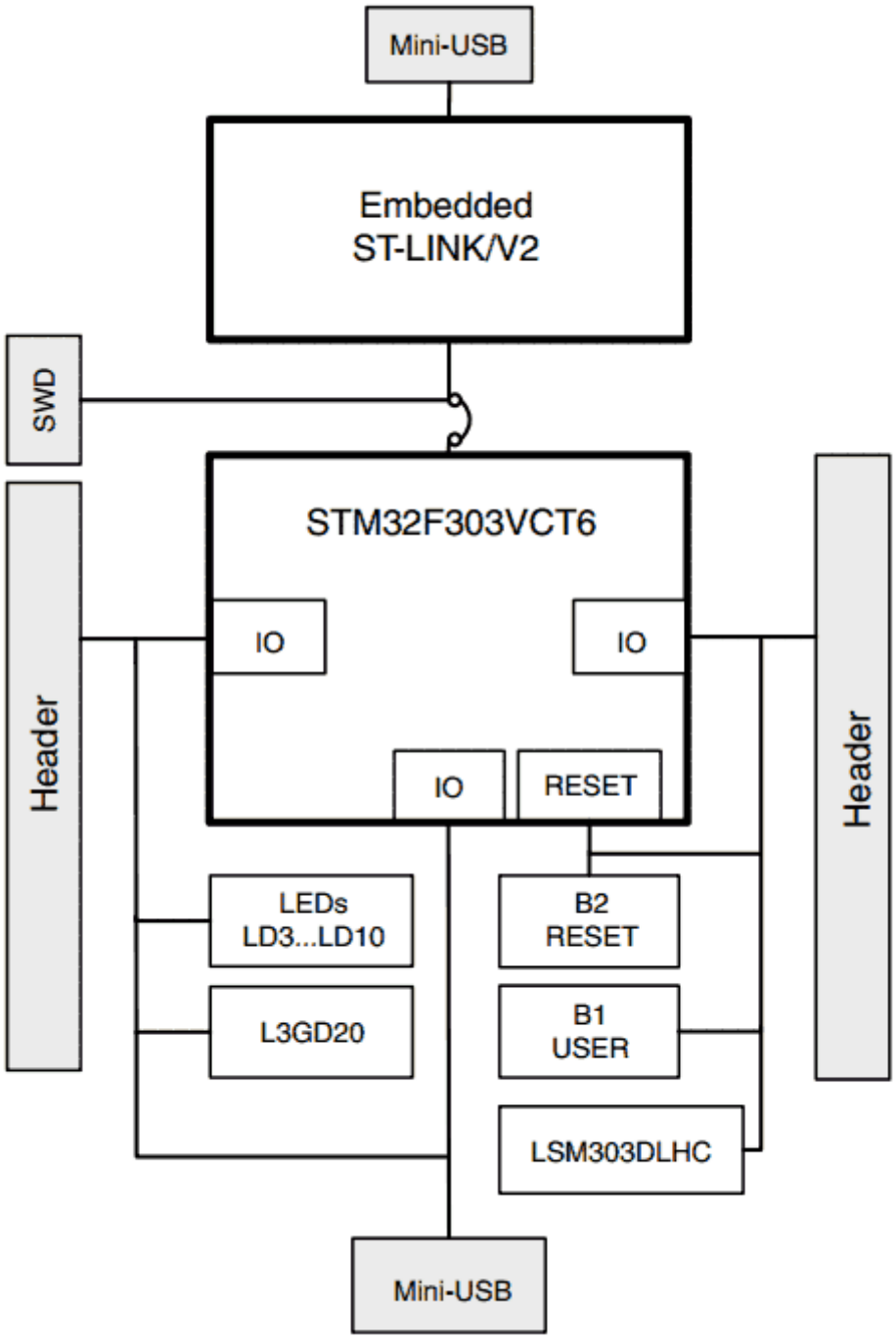
Obsahuje mikrokontrolér *STM32F407VGT6* s 32-bit ARM Cortex-M4F jádrem, 1 MB Flash, 192 KB RAMv pouzdře LQFP100. On-board ST-LINK/V2 programátor . Napájení přes USB nebo externí zdroj.
LIS302DL, ST MEMS sensor pohybu, 3-osý akcelerometr s digitálním výstupem
Audio MEMS sensor MP45DT02
CS43L22, audio DAC s integrovaným zesilovačen třídy D
8 LED, 2 tlačítka

3.6 Startkit STM32F3Discovery

Jde s startkit s výkonným ARM Cortex M4 mcu *STM32F303VCT6*, a doplněný o magnetometr a gyroskop, accelerometer, a e-kompas ST MEMs, USB , LEDky a tlačítka.



Blokové schéma



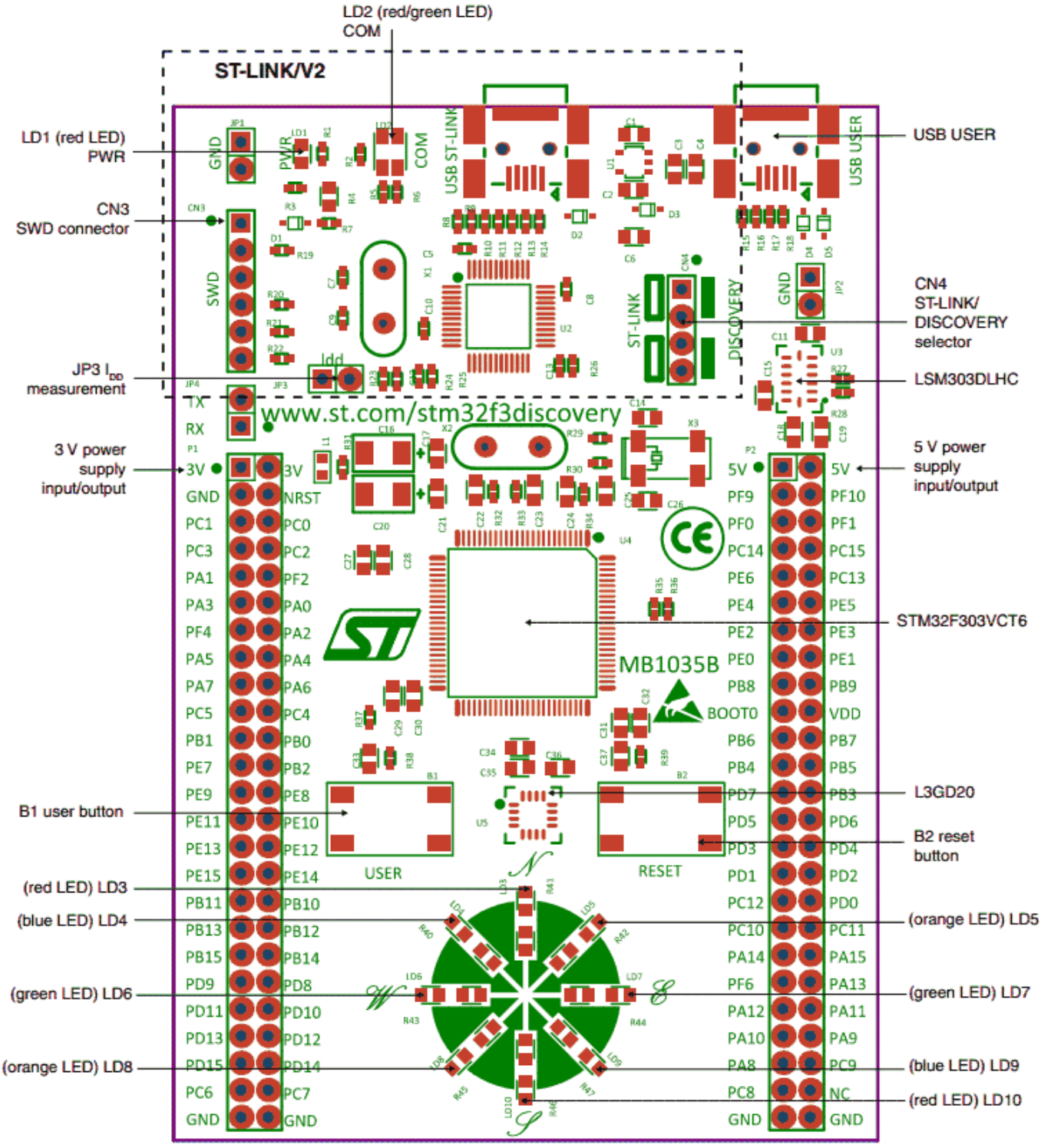
Vlevo dole vidíte rámeček s L3GD20, což je MEMS 3-osý gyroskop. A také na blokovém schématu naleznete vpravo dole LSM303DLHC, což je 3-osý akcelerometr a 3-osý magnetometr v jednom pouzdru.

Cílovým mcu je *STM32F303VCT6* ARM Cortex M4 v pouzdru LQFP100, obsahujícím 256 kB Flash a 48 kB SRAM. Maximální tak jádra je 72 MHz a pri běhu programu z Flash získáte výkon 62 DMIPS. Čip STM32F303 však obsahuje 8 kB z paměti SRAM přímo namapovaných na instrukční sběrnici a program umístěný v této části SRAM se vykonává bez čekacích cyklů - což vede k výkonu 94 DMIPS. Dále mcu obsahuje HW jednotku výpočtů v plovoucí desetinné čárce.

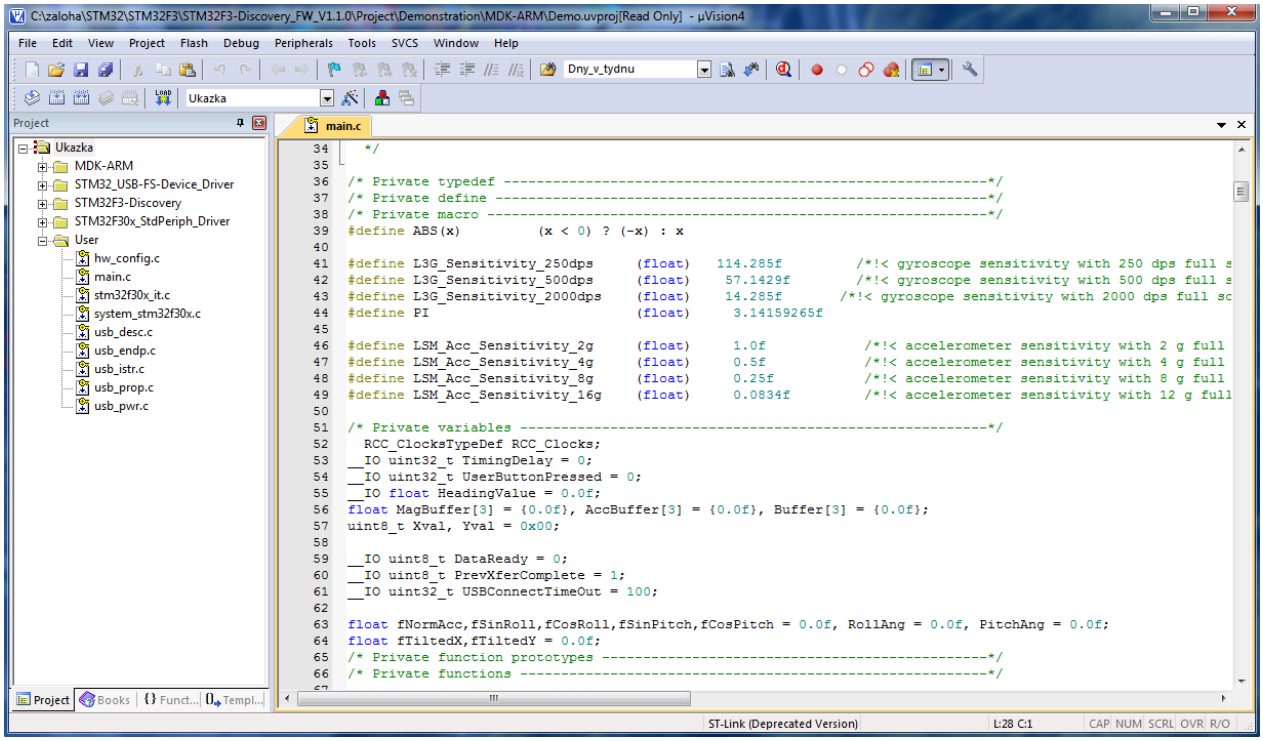
Na kitu jsou dva USB konektory - programovací a uživatelský. Programovací je propojen s ST-Link/v2 programátorem a debuggerem, který tvoří součást kitu (stejně jako u předchozích Discovery kitů). ST-

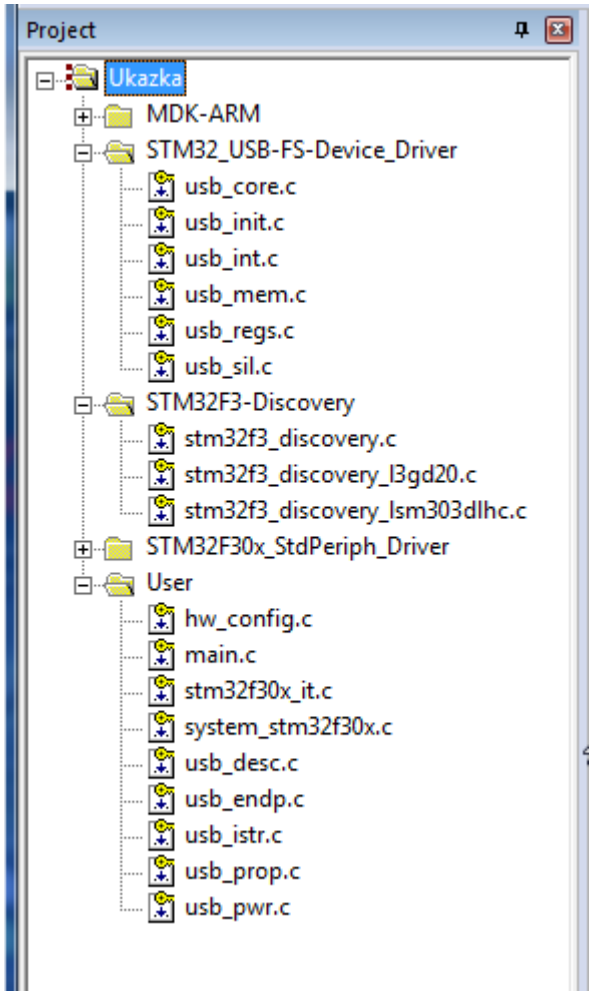
Link lze po rozpojení propojek použít k programování externích mcu na vaší vlastní desce. Uživatelský USB Mini-B konektor) je Vám k dispozici.

Kde na kitu se co nachází



Ještě ukázka programu s čidly MEMS:






```
/**
******************************************************************************
***
  * @file    main.c
  * @version V1.1.0
  * @date    20-September-2012
  * @brief   Main program body

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


/* Includes ------------------------------------------------------------------
---*/
#include "main.h"

/** @addtogroup STM32F3-Discovery
  * @{
  */
```

```
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
#define ABS(x)         (x < 0) ? (-x) : x

#define L3G_Sensitivity_250dps     (float)   114.285f         /*!< gyroscope sensitivity with 250 dps full scale [LSB/dps] */
#define L3G_Sensitivity_500dps     (float)    57.1429f        /*!< gyroscope sensitivity with 500 dps full scale [LSB/dps] */
#define L3G_Sensitivity_2000dps    (float)    14.285f      /*!< gyroscope sensitivity with 2000 dps full scale [LSB/dps] */
#define PI                         (float)     3.14159265f

#define LSM_Acc_Sensitivity_2g     (float)     1.0f            /*!< accelerometer sensitivity with 2 g full scale [LSB/mg] */
#define LSM_Acc_Sensitivity_4g     (float)     0.5f            /*!< accelerometer sensitivity with 4 g full scale [LSB/mg] */
#define LSM_Acc_Sensitivity_8g     (float)     0.25f           /*!< accelerometer sensitivity with 8 g full scale [LSB/mg] */
#define LSM_Acc_Sensitivity_16g    (float)     0.0834f         /*!< accelerometer sensitivity with 12 g full scale [LSB/mg] */

/* Private variables ---------------------------------------------------------*/
  RCC_ClocksTypeDef RCC_Clocks;
__IO uint32_t TimingDelay = 0;
__IO uint32_t UserButtonPressed = 0;
__IO float HeadingValue = 0.0f;
float MagBuffer[3] = {0.0f}, AccBuffer[3] = {0.0f}, Buffer[3] = {0.0f};
uint8_t Xval, Yval = 0x00;

__IO uint8_t DataReady = 0;
__IO uint8_t PrevXferComplete = 1;
__IO uint32_t USBConnectTimeOut = 100;

float fNormAcc,fSinRoll,fCosRoll,fSinPitch,fCosPitch = 0.0f, RollAng = 0.0f, PitchAng = 0.0f;
float fTiltedX,fTiltedY = 0.0f;
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */
int main(void)
{
  uint8_t i = 0;
  /* SysTick end of count event each 10ms */
  RCC_GetClocksFreq(&RCC_Clocks);
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 100);

  /* Initialize LEDs and User Button available on STM32F3-Discovery board
*/
```

```
  STM_EVAL_LEDInit(LED3);
  STM_EVAL_LEDInit(LED4);
  STM_EVAL_LEDInit(LED5);
  STM_EVAL_LEDInit(LED6);
  STM_EVAL_LEDInit(LED7);
  STM_EVAL_LEDInit(LED8);
  STM_EVAL_LEDInit(LED9);
  STM_EVAL_LEDInit(LED10);

  STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);

  /* Configure the USB */
  Demo_USB();

  /* Reset UserButton_Pressed variable */
  UserButtonPressed = 0x00;

  /* Infinite loop */
  while (1)
  {
    /* LEDs Off */
    STM_EVAL_LEDOff(LED3);
    STM_EVAL_LEDOff(LED6);
    STM_EVAL_LEDOff(LED7);
    STM_EVAL_LEDOff(LED4);
    STM_EVAL_LEDOff(LED10);
    STM_EVAL_LEDOff(LED8);
    STM_EVAL_LEDOff(LED9);
    STM_EVAL_LEDOff(LED5);

    /* Waiting User Button is pressed */
    while (UserButtonPressed == 0x00)
    {
      /* Toggle LD3 */
      STM_EVAL_LEDToggle(LED3);
      /* Insert 50 ms delay */
      Delay(5);
      /* Toggle LD5 */
      STM_EVAL_LEDToggle(LED5);
      /* Insert 50 ms delay */
      Delay(5);
      /* Toggle LD7 */
      STM_EVAL_LEDToggle(LED7);
      /* Insert 50 ms delay */
      Delay(5);
      /* Toggle LD9 */
      STM_EVAL_LEDToggle(LED9);
      /* Insert 50 ms delay */
      Delay(5);
      /* Toggle LD10 */
      STM_EVAL_LEDToggle(LED10);
      /* Insert 50 ms delay */
      Delay(5);
      /* Toggle LD8 */
      STM_EVAL_LEDToggle(LED8);
      /* Insert 50 ms delay */
      Delay(5);
      /* Toggle LD6 */
      STM_EVAL_LEDToggle(LED6);
      /* Insert 50 ms delay */
```

```
      Delay(5);
      /* Toggle LD4 */
      STM_EVAL_LEDToggle(LED4);
      /* Insert 50 ms delay */
      Delay(5);
    }

    DataReady = 0x00;

    /* All LEDs Off */
    STM_EVAL_LEDOff(LED3);
    STM_EVAL_LEDOff(LED6);
    STM_EVAL_LEDOff(LED7);
    STM_EVAL_LEDOff(LED4);
    STM_EVAL_LEDOff(LED10);
    STM_EVAL_LEDOff(LED8);
    STM_EVAL_LEDOff(LED9);
    STM_EVAL_LEDOff(LED5);

    /* Demo Gyroscope */
    Demo_GyroConfig();

    /* Waiting User Button is pressed */
    while (UserButtonPressed == 0x01)
    {
      /* Wait for data ready */
      while(DataReady != 0x05)
      {}
      DataReady = 0x00;

      /* LEDs Off */
      STM_EVAL_LEDOff(LED3);
      STM_EVAL_LEDOff(LED6);
      STM_EVAL_LEDOff(LED7);
      STM_EVAL_LEDOff(LED4);
      STM_EVAL_LEDOff(LED10);
      STM_EVAL_LEDOff(LED8);
      STM_EVAL_LEDOff(LED9);
      STM_EVAL_LEDOff(LED5);

      /* Read Gyro Angular data */
      Demo_GyroReadAngRate(Buffer);

      /* Update autoreload and capture compare registers value*/
      Xval = ABS((int8_t)(Buffer[0]));
      Yval = ABS((int8_t)(Buffer[1]));

      if ( Xval>Yval)
      {
        if ((int8_t)Buffer[0] > 5.0f)
        {
          /* LD10 On */
          STM_EVAL_LEDOn(LED10);
        }
        if ((int8_t)Buffer[0] < -5.0f)
        {
          /* LD3 On */
          STM_EVAL_LEDOn(LED3);
        }
      }
```

```
      else
      {
        if ((int8_t)Buffer[1] < -5.0f)
        {
          /* LD6 on */
          STM_EVAL_LEDOn(LED6);
        }
        if ((int8_t)Buffer[1] > 5.0f)
        {
          /* LD7 On */
          STM_EVAL_LEDOn(LED7);
        }
      }
    }

    DataReady = 0x00;

    /* LEDs Off */
    STM_EVAL_LEDOff(LED4);
    STM_EVAL_LEDOff(LED3);
    STM_EVAL_LEDOff(LED6);
    STM_EVAL_LEDOff(LED7);
    STM_EVAL_LEDOff(LED10);
    STM_EVAL_LEDOff(LED8);
    STM_EVAL_LEDOff(LED9);
    STM_EVAL_LEDOff(LED5);

    /* Demo Compass */
    Demo_CompassConfig();

    /* Waiting User Button is pressed */
    while (UserButtonPressed == 0x02)
    {
      /* Wait for data ready */
      while(DataReady !=0x05)
      {}
      DataReady = 0x00;

      /* Read Compass data */
      Demo_CompassReadMag(MagBuffer);
      Demo_CompassReadAcc(AccBuffer);

      for(i=0;i<3;i++)
        AccBuffer[i] /= 100.0f;

      fNormAcc =
sqrt((AccBuffer[0]*AccBuffer[0])+(AccBuffer[1]*AccBuffer[1])+(AccBuffer[2]*
AccBuffer[2]));

      fSinRoll = -AccBuffer[1]/fNormAcc;
      fCosRoll = sqrt(1.0-(fSinRoll * fSinRoll));
      fSinPitch = AccBuffer[0]/fNormAcc;
      fCosPitch = sqrt(1.0-(fSinPitch * fSinPitch));
     if ( fSinRoll >0)
     {
       if (fCosRoll>0)
       {
         RollAng = acos(fCosRoll)*180/PI;
       }
       else
```

```
        {
          RollAng = acos(fCosRoll)*180/PI + 180;
        }
      }
      else
      {
        if (fCosRoll>0)
        {
          RollAng = acos(fCosRoll)*180/PI + 360;
        }
        else
        {
          RollAng = acos(fCosRoll)*180/PI + 180;
        }
      }

       if ( fSinPitch >0)
      {
        if (fCosPitch>0)
        {
             PitchAng = acos(fCosPitch)*180/PI;
        }
        else
        {
           PitchAng = acos(fCosPitch)*180/PI + 180;
        }
      }
      else
      {
        if (fCosPitch>0)
        {
             PitchAng = acos(fCosPitch)*180/PI + 360;
        }
        else
        {
           PitchAng = acos(fCosPitch)*180/PI + 180;
        }
      }

       if (RollAng >=360)
       {
         RollAng = RollAng - 360;
       }

       if (PitchAng >=360)
       {
         PitchAng = PitchAng - 360;
       }

       fTiltedX = MagBuffer[0]*fCosPitch+MagBuffer[2]*fSinPitch;
       fTiltedY = MagBuffer[0]*fSinRoll*fSinPitch+MagBuffer[1]*fCosRoll-
MagBuffer[1]*fSinRoll*fCosPitch;

       HeadingValue = (float)
((atan2f((float)fTiltedY,(float)fTiltedX))*180)/PI;

       if (HeadingValue < 0)
       {
         HeadingValue = HeadingValue + 360;
       }
```

```
      if ((RollAng <= 40.0f) && (PitchAng <= 40.0f))
      {
        if (((HeadingValue < 25.0f)&&(HeadingValue >=
0.0f))||((HeadingValue >=340.0f)&&(HeadingValue <= 360.0f)))
        {
          STM_EVAL_LEDOn(LED10);
          STM_EVAL_LEDOff(LED3);
          STM_EVAL_LEDOff(LED6);
          STM_EVAL_LEDOff(LED7);
          STM_EVAL_LEDOff(LED4);
          STM_EVAL_LEDOff(LED8);
          STM_EVAL_LEDOff(LED9);
          STM_EVAL_LEDOff(LED5);
        }
        else  if ((HeadingValue <70.0f)&&(HeadingValue >= 25.0f))
        {
          STM_EVAL_LEDOn(LED9);
          STM_EVAL_LEDOff(LED6);
          STM_EVAL_LEDOff(LED10);
          STM_EVAL_LEDOff(LED3);
          STM_EVAL_LEDOff(LED8);
          STM_EVAL_LEDOff(LED5);
          STM_EVAL_LEDOff(LED4);
          STM_EVAL_LEDOff(LED7);
        }
        else  if ((HeadingValue < 115.0f)&&(HeadingValue >= 70.0f))
        {
          STM_EVAL_LEDOn(LED7);
          STM_EVAL_LEDOff(LED3);
          STM_EVAL_LEDOff(LED4);
          STM_EVAL_LEDOff(LED9);
          STM_EVAL_LEDOff(LED10);
          STM_EVAL_LEDOff(LED8);
          STM_EVAL_LEDOff(LED6);
          STM_EVAL_LEDOff(LED5);
        }
        else  if ((HeadingValue <160.0f)&&(HeadingValue >= 115.0f))
        {
          STM_EVAL_LEDOn(LED5);
          STM_EVAL_LEDOff(LED6);
          STM_EVAL_LEDOff(LED10);
          STM_EVAL_LEDOff(LED8);
          STM_EVAL_LEDOff(LED9);
          STM_EVAL_LEDOff(LED7);
          STM_EVAL_LEDOff(LED4);
          STM_EVAL_LEDOff(LED3);
        }
        else  if ((HeadingValue <205.0f)&&(HeadingValue >= 160.0f))
        {
          STM_EVAL_LEDOn(LED3);
          STM_EVAL_LEDOff(LED6);
          STM_EVAL_LEDOff(LED4);
          STM_EVAL_LEDOff(LED8);
          STM_EVAL_LEDOff(LED9);
          STM_EVAL_LEDOff(LED5);
          STM_EVAL_LEDOff(LED10);
          STM_EVAL_LEDOff(LED7);
        }
        else  if ((HeadingValue <250.0f)&&(HeadingValue >= 205.0f))
```

```
        {
          STM_EVAL_LEDOn(LED4);
          STM_EVAL_LEDOff(LED6);
          STM_EVAL_LEDOff(LED10);
          STM_EVAL_LEDOff(LED8);
          STM_EVAL_LEDOff(LED9);
          STM_EVAL_LEDOff(LED5);
          STM_EVAL_LEDOff(LED3);
          STM_EVAL_LEDOff(LED7);
        }
        else  if ((HeadingValue < 295.0f)&&(HeadingValue >= 250.0f))
        {
          STM_EVAL_LEDOn(LED6);
          STM_EVAL_LEDOff(LED9);
          STM_EVAL_LEDOff(LED10);
          STM_EVAL_LEDOff(LED8);
          STM_EVAL_LEDOff(LED3);
          STM_EVAL_LEDOff(LED5);
          STM_EVAL_LEDOff(LED4);
          STM_EVAL_LEDOff(LED7);
        }
        else  if ((HeadingValue < 340.0f)&&(HeadingValue >= 295.0f))
        {
          STM_EVAL_LEDOn(LED8);
          STM_EVAL_LEDOff(LED6);
          STM_EVAL_LEDOff(LED10);
          STM_EVAL_LEDOff(LED7);
          STM_EVAL_LEDOff(LED9);
          STM_EVAL_LEDOff(LED3);
          STM_EVAL_LEDOff(LED4);
          STM_EVAL_LEDOff(LED5);
        }
      }
      else
      {
        /* Toggle All LEDs */
        STM_EVAL_LEDToggle(LED7);
        STM_EVAL_LEDToggle(LED6);
        STM_EVAL_LEDToggle(LED10);
        STM_EVAL_LEDToggle(LED8);
        STM_EVAL_LEDToggle(LED9);
        STM_EVAL_LEDToggle(LED3);
        STM_EVAL_LEDToggle(LED4);
        STM_EVAL_LEDToggle(LED5);
        /* Delay 50ms */
        Delay(5);
      }
    }
  }
}


/**
  * @brief  Configure the USB.
  * @param  None
  * @retval None
  */
void Demo_USB (void)
{
  Set_System();
```

```
  Set_USBClock();
  USB_Interrupts_Config();

  USB_Init();

  while ((bDeviceState != CONFIGURED)&&(USBConnectTimeOut != 0))
  {}
}

/**
  * @brief  Configure the Mems to gyroscope application.
  * @param  None
  * @retval None
  */
void Demo_GyroConfig(void)
{
  L3GD20_InitTypeDef L3GD20_InitStructure;
  L3GD20_FilterConfigTypeDef L3GD20_FilterStructure;

  /* Configure Mems L3GD20 */
  L3GD20_InitStructure.Power_Mode = L3GD20_MODE_ACTIVE;
  L3GD20_InitStructure.Output_DataRate = L3GD20_OUTPUT_DATARATE_1;
  L3GD20_InitStructure.Axes_Enable = L3GD20_AXES_ENABLE;
  L3GD20_InitStructure.Band_Width = L3GD20_BANDWIDTH_4;
  L3GD20_InitStructure.BlockData_Update = L3GD20_BlockDataUpdate_Continous;
  L3GD20_InitStructure.Endianness = L3GD20_BLE_LSB;
  L3GD20_InitStructure.Full_Scale = L3GD20_FULLSCALE_500;
  L3GD20_Init(&L3GD20_InitStructure);

  L3GD20_FilterStructure.HighPassFilter_Mode_Selection
=L3GD20_HPM_NORMAL_MODE_RES;
  L3GD20_FilterStructure.HighPassFilter_CutOff_Frequency = L3GD20_HPFCF_0;
  L3GD20_FilterConfig(&L3GD20_FilterStructure) ;

  L3GD20_FilterCmd(L3GD20_HIGHPASSFILTER_ENABLE);
}

/**
  * @brief  Calculate the angular Data rate Gyroscope.
  * @param  pfData : Data out pointer
  * @retval None
  */
void Demo_GyroReadAngRate (float* pfData)
{
  uint8_t tmpbuffer[6] ={0};
  int16_t RawData[3] = {0};
  uint8_t tmpreg = 0;
  float sensitivity = 0;
  int i =0;

  L3GD20_Read(&tmpreg,L3GD20_CTRL_REG4_ADDR,1);

  L3GD20_Read(tmpbuffer,L3GD20_OUT_X_L_ADDR,6);

  /* check in the control register 4 the data alignment (Big Endian or
Little Endian)*/
  if(!(tmpreg & 0x40))
  {
    for(i=0; i<3; i++)
    {
```

```
      RawData[i]=(int16_t)(((uint16_t)tmpbuffer[2*i+1] << 8) +
tmpbuffer[2*i]);
    }
  }
  else
  {
    for(i=0; i<3; i++)
    {
      RawData[i]=(int16_t)(((uint16_t)tmpbuffer[2*i] << 8) +
tmpbuffer[2*i+1]);
    }
  }

  /* Switch the sensitivity value set in the CRTL4 */
  switch(tmpreg & 0x30)
  {
  case 0x00:
    sensitivity=L3G_Sensitivity_250dps;
    break;

  case 0x10:
    sensitivity=L3G_Sensitivity_500dps;
    break;

  case 0x20:
    sensitivity=L3G_Sensitivity_2000dps;
    break;
  }
  /* divide by sensitivity */
  for(i=0; i<3; i++)
  {
    pfData[i]=(float)RawData[i]/sensitivity;
  }
}

/**
  * @brief  Configure the Mems to compass application.
  * @param  None
  * @retval None
  */
void Demo_CompassConfig(void)
{
  LSM303DLHCMag_InitTypeDef LSM303DLHC_InitStructure;
  LSM303DLHCAcc_InitTypeDef LSM303DLHCAcc_InitStructure;
  LSM303DLHCAcc_FilterConfigTypeDef LSM303DLHCFilter_InitStructure;

  /* Configure MEMS magnetometer main parameters: temp, working mode, full
Scale and Data rate */
  LSM303DLHC_InitStructure.Temperature_Sensor =
LSM303DLHC_TEMPSENSOR_DISABLE;
  LSM303DLHC_InitStructure.MagOutput_DataRate =LSM303DLHC_ODR_30_HZ ;
  LSM303DLHC_InitStructure.MagFull_Scale = LSM303DLHC_FS_8_1_GA;
  LSM303DLHC_InitStructure.Working_Mode = LSM303DLHC_CONTINUOS_CONVERSION;
  LSM303DLHC_MagInit(&LSM303DLHC_InitStructure);

   /* Fill the accelerometer structure */
  LSM303DLHCAcc_InitStructure.Power_Mode = LSM303DLHC_NORMAL_MODE;
  LSM303DLHCAcc_InitStructure.AccOutput_DataRate = LSM303DLHC_ODR_50_HZ;
  LSM303DLHCAcc_InitStructure.Axes_Enable= LSM303DLHC_AXES_ENABLE;
  LSM303DLHCAcc_InitStructure.AccFull_Scale = LSM303DLHC_FULLSCALE_2G;
```

```
  LSM303DLHCAcc_InitStructure.BlockData_Update =
LSM303DLHC_BlockUpdate_Continous;
  LSM303DLHCAcc_InitStructure.Endianness=LSM303DLHC_BLE_LSB;
  LSM303DLHCAcc InitStructure.High Resolution=LSM303DLHC HR ENABLE;
  /* Configure the accelerometer main parameters */
  LSM303DLHC_AccInit(&LSM303DLHCAcc_InitStructure);

  /* Fill the accelerometer LPF structure */
  LSM303DLHCFilter_InitStructure.HighPassFilter_Mode_Selection
=LSM303DLHC_HPM_NORMAL_MODE;
  LSM303DLHCFilter_InitStructure.HighPassFilter_CutOff_Frequency =
LSM303DLHC_HPFCF_16;
  LSM303DLHCFilter_InitStructure.HighPassFilter_AOI1 =
LSM303DLHC_HPF_AOI1_DISABLE;
  LSM303DLHCFilter_InitStructure.HighPassFilter_AOI2 =
LSM303DLHC_HPF_AOI2_DISABLE;

  /* Configure the accelerometer LPF main parameters */
  LSM303DLHC_AccFilterConfig(&LSM303DLHCFilter_InitStructure);
}

/**
* @brief Read LSM303DLHC output register, and calculate the acceleration
ACC=(1/SENSITIVITY)* (out_h*256+out_l)/16 (12 bit rappresentation)
* @param pnData: pointer to float buffer where to store data
* @retval None
*/
void Demo_CompassReadAcc(float* pfData)
{
  int16_t pnRawData[3];
  uint8_t ctrlx[2];
  uint8_t buffer[6], cDivider;
  uint8_t i = 0;
  float LSM_Acc_Sensitivity = LSM_Acc_Sensitivity_2g;

  /* Read the register content */
  LSM303DLHC_Read(ACC_I2C_ADDRESS, LSM303DLHC_CTRL_REG4_A, ctrlx,2);
  LSM303DLHC_Read(ACC_I2C_ADDRESS, LSM303DLHC_OUT_X_L_A, buffer, 6);

  if(ctrlx[1]&0x40)
    cDivider=64;
  else
    cDivider=16;

  /* check in the control register4 the data alignment*/
  if(!(ctrlx[0] & 0x40) || (ctrlx[1] & 0x40)) /* Little Endian Mode or FIFO
mode */
  {
    for(i=0; i<3; i++)
    {
      pnRawData[i]=((int16_t)((uint16_t)buffer[2*i+1] << 8) +
buffer[2*i])/cDivider;
    }
  }
  else /* Big Endian Mode */
  {
    for(i=0; i<3; i++)
      pnRawData[i]=((int16_t)((uint16_t)buffer[2*i] << 8) +
buffer[2*i+1])/cDivider;
  }
```

```
  /* Read the register content */
  LSM303DLHC_Read(ACC_I2C_ADDRESS, LSM303DLHC_CTRL_REG4_A, ctrlx,2);


  if(ctrlx[1]&0x40)
  {
    /* FIFO mode */
    LSM_Acc_Sensitivity = 0.25;
  }
  else
  {
    /* normal mode */
    /* switch the sensitivity value set in the CRTL4*/
    switch(ctrlx[0] & 0x30)
    {
    case LSM303DLHC_FULLSCALE_2G:
      LSM_Acc_Sensitivity = LSM_Acc_Sensitivity_2g;
      break;
    case LSM303DLHC_FULLSCALE_4G:
      LSM_Acc_Sensitivity = LSM_Acc_Sensitivity_4g;
      break;
    case LSM303DLHC_FULLSCALE_8G:
      LSM_Acc_Sensitivity = LSM_Acc_Sensitivity_8g;
      break;
    case LSM303DLHC_FULLSCALE_16G:
      LSM_Acc_Sensitivity = LSM_Acc_Sensitivity_16g;
      break;
    }
  }

  /* Obtain the mg value for the three axis */
  for(i=0; i<3; i++)
  {
    pfData[i]=(float)pnRawData[i]/LSM_Acc_Sensitivity;
  }

}

/**
  * @brief  calculate the magnetic field Magn.
* @param  pfData: pointer to the data out
  * @retval None
  */
void Demo_CompassReadMag (float* pfData)
{
  static uint8_t buffer[6] = {0};
  uint8_t CTRLB = 0;
  uint16_t Magn_Sensitivity_XY = 0, Magn_Sensitivity_Z = 0;
  uint8_t i =0;
  LSM303DLHC_Read(MAG_I2C_ADDRESS, LSM303DLHC_CRB_REG_M, &CTRLB, 1);

  LSM303DLHC_Read(MAG_I2C_ADDRESS, LSM303DLHC_OUT_X_H_M, buffer, 1);
  LSM303DLHC_Read(MAG_I2C_ADDRESS, LSM303DLHC_OUT_X_L_M, buffer+1, 1);
  LSM303DLHC_Read(MAG_I2C_ADDRESS, LSM303DLHC_OUT_Y_H_M, buffer+2, 1);
  LSM303DLHC_Read(MAG_I2C_ADDRESS, LSM303DLHC_OUT_Y_L_M, buffer+3, 1);
  LSM303DLHC_Read(MAG_I2C_ADDRESS, LSM303DLHC_OUT_Z_H_M, buffer+4, 1);
  LSM303DLHC_Read(MAG_I2C_ADDRESS, LSM303DLHC_OUT_Z_L_M, buffer+5, 1);
  /* Switch the sensitivity set in the CRTLB*/
  switch(CTRLB & 0xE0)
  {
```

```
  case LSM303DLHC_FS_1_3_GA:
    Magn_Sensitivity_XY = LSM303DLHC_M_SENSITIVITY_XY_1_3Ga;
    Magn_Sensitivity_Z = LSM303DLHC_M_SENSITIVITY_Z_1_3Ga;
    break;
  case LSM303DLHC_FS_1_9_GA:
    Magn_Sensitivity_XY = LSM303DLHC_M_SENSITIVITY_XY_1_9Ga;
    Magn_Sensitivity_Z = LSM303DLHC_M_SENSITIVITY_Z_1_9Ga;
    break;
  case LSM303DLHC_FS_2_5_GA:
    Magn_Sensitivity_XY = LSM303DLHC_M_SENSITIVITY_XY_2_5Ga;
    Magn_Sensitivity_Z = LSM303DLHC_M_SENSITIVITY_Z_2_5Ga;
    break;
  case LSM303DLHC_FS_4_0_GA:
    Magn_Sensitivity_XY = LSM303DLHC_M_SENSITIVITY_XY_4Ga;
    Magn_Sensitivity_Z = LSM303DLHC_M_SENSITIVITY_Z_4Ga;
    break;
  case LSM303DLHC_FS_4_7_GA:
    Magn_Sensitivity_XY = LSM303DLHC_M_SENSITIVITY_XY_4_7Ga;
    Magn_Sensitivity_Z = LSM303DLHC_M_SENSITIVITY_Z_4_7Ga;
    break;
  case LSM303DLHC_FS_5_6_GA:
    Magn_Sensitivity_XY = LSM303DLHC_M_SENSITIVITY_XY_5_6Ga;
    Magn_Sensitivity_Z = LSM303DLHC_M_SENSITIVITY_Z_5_6Ga;
    break;
  case LSM303DLHC_FS_8_1_GA:
    Magn_Sensitivity_XY = LSM303DLHC_M_SENSITIVITY_XY_8_1Ga;
    Magn_Sensitivity_Z = LSM303DLHC_M_SENSITIVITY_Z_8_1Ga;
    break;
  }

  for(i=0; i<2; i++)
  {
    pfData[i]=(float)((int16_t)(((uint16_t)buffer[2*i] << 8) +
buffer[2*i+1])*1000)/Magn_Sensitivity_XY;
  }
  pfData[2]=(float)((int16_t)(((uint16_t)buffer[4] << 8) +
buffer[5])*1000)/Magn_Sensitivity_Z;
}

/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(__IO uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);
}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
```

```
    TimingDelay--;
  }
}

/**
  * @brief  Basic management of the timeout situation.
  * @param  None.
  * @retval None.
  */
uint32_t LSM303DLHC_TIMEOUT_UserCallback(void)
{
  return 0;
}

/**
  * @brief  Basic management of the timeout situation.
  * @param  None.
  * @retval None.
  */
uint32_t L3GD20_TIMEOUT_UserCallback(void)
{
  return 0;
}

#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */

  /* Infinite loop */
  while (1)
  {
  }
}
#endif
```

Práce s výše uvedenýcmi kódy v jazyce C nemusí být pro studenty středních škol jednoduchá. Naštěstí je k dispozici implementace *micro .NET* pro startkit *STM32F4-Discovery*, což umožní programovat v C#. Proto si uvedeme postup, jak tuto implementaci provést.

K jejímu provedení využijeme podklady z http://netmf4stm32.codeplex.com/ kde najdeme .NET *MicroFramework* portovaný na startkity STM32 . Další zdroje jsou na http://netmf4stm32.codeplex.com/discussions/400293 a na http://netmf4stm32.codeplex.com/team/view.

Tedy potřebujeme

1) STM32F4 Discovery startkit

2) USB Micro a USB Mini kabel

3) STM32 ST-LINK Utility (k natažení *bootloaderu* do discovery startkitu. Dále jeho ovladač)
http://www.st.com/internet/com/SOFTWARE_RESOURCES/TOOL/DEVICE_PROGRAMMER/stm32_st-link_utility.zip

4) Stáhneme stm32f4discovery.zip a STM32_WinUSB_drivers_(for_evaluation_purposes_only).zip z http://netmf4stm32.codeplex.com/

5) Visual C# Express nebo Visual Studio 2010 (nikoli verzi 2012
) http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express

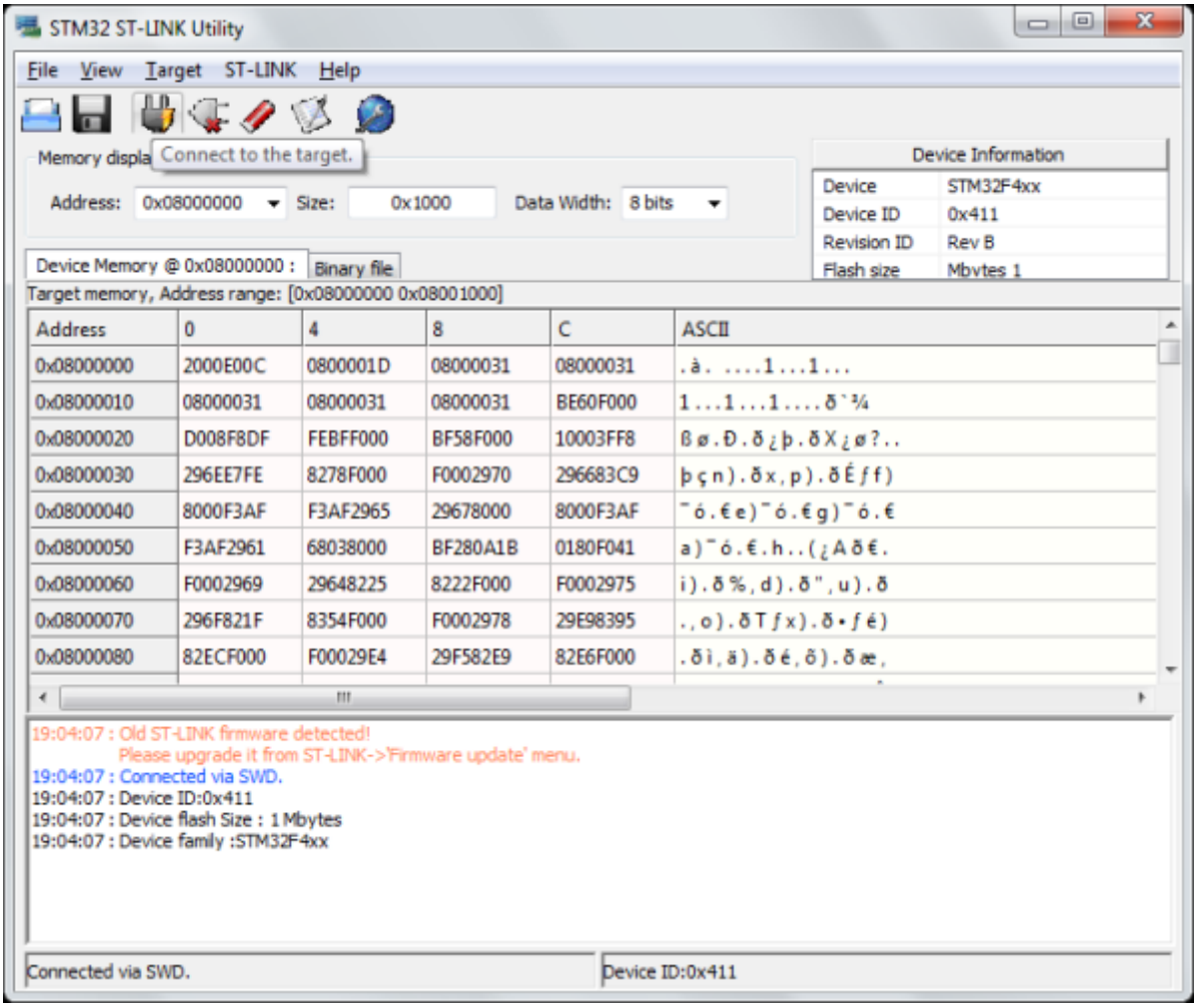
6) .NET MicroFramework SDK (verzi 4.2) http://netmf.codeplex.com/releases/view/91594

7) LEDku (na test hello blinky)

Nyní již připojíme USB Micro kabel.

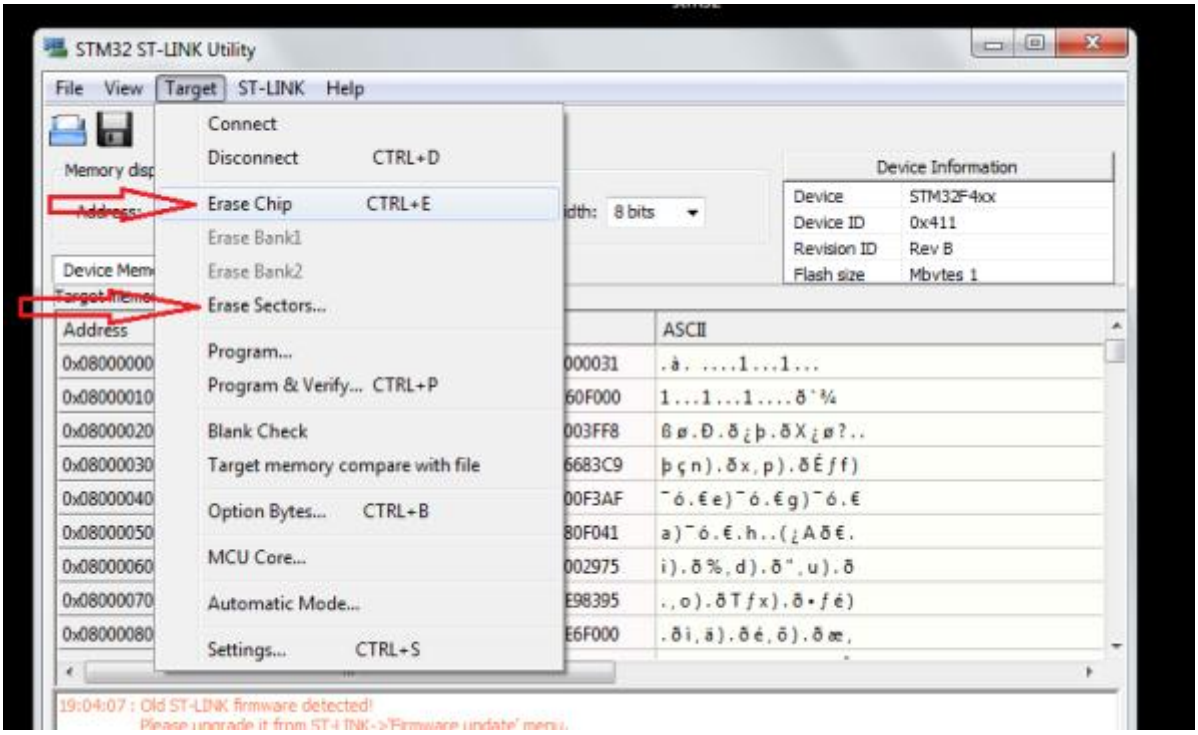
STM32F propojen s Micro USB

Pokud připojíme startkit, začne se vyhledávat ovladač. Pokud jsme již nainstalovali **STM32 STLink utility**, pak je ovladač již nainstalován. V opačném případě ho musíme doinstalovat. Spustíme **STM32 Utility** (obvykle je umístěn v C:\Program Files (x86)\STMicroelectronics\STM32 ST-LINK Utility\ST-LINK Utility\STM32 ST-LINK Utility.exe pro defaultní instalaci).

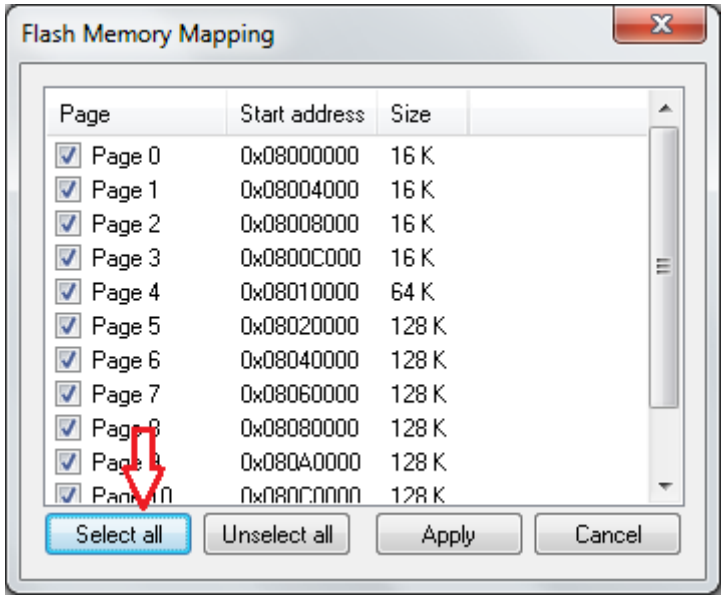


STLink utility se připojí k startkitu

Po vytvoření spojení touto utilitou, vymažeme vše v *STM32F4* podle následujících screenshotů (**delete** jak **Chip** tak **Sector**, nejprv jeden, pak druhý)



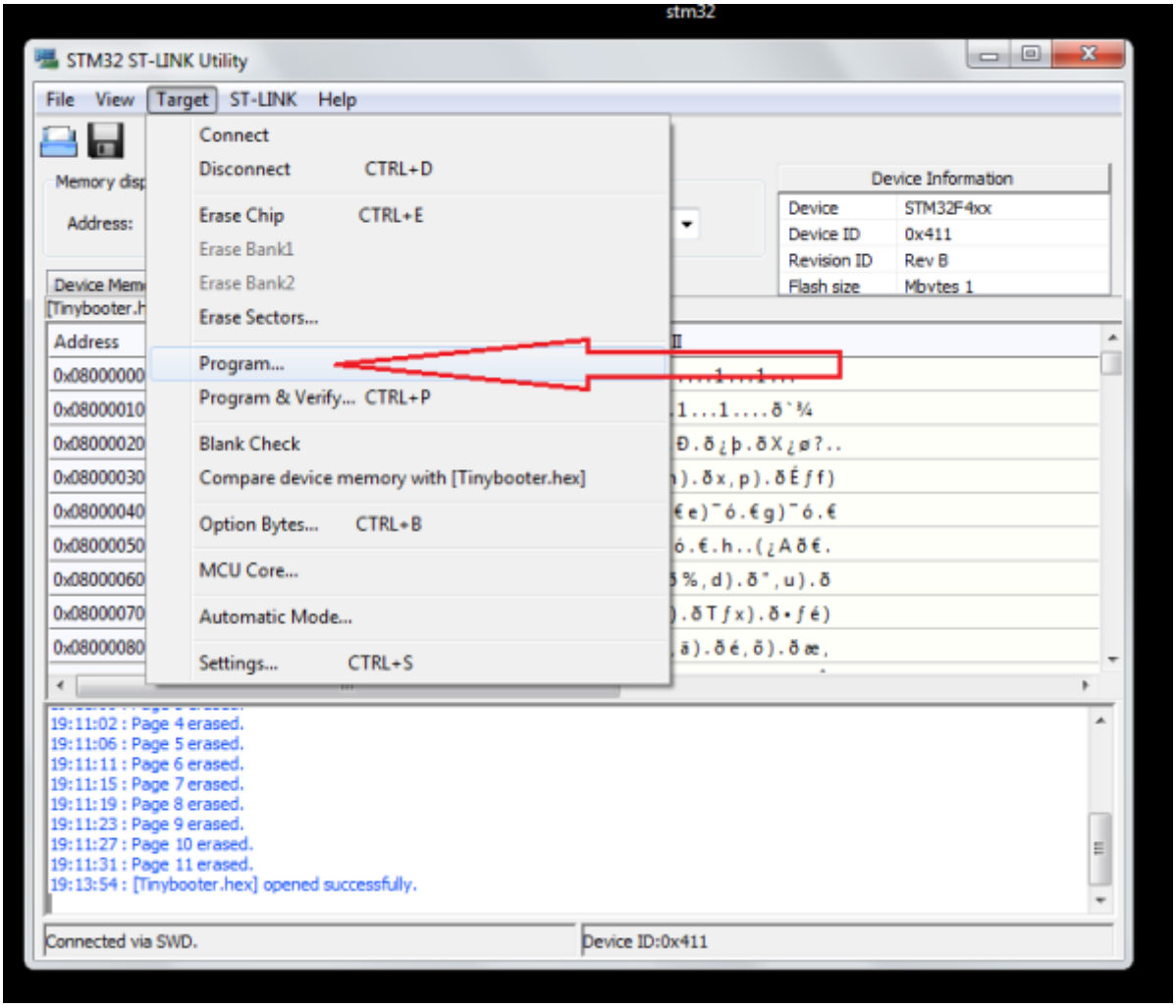
Klikneme na **Erase Chip**



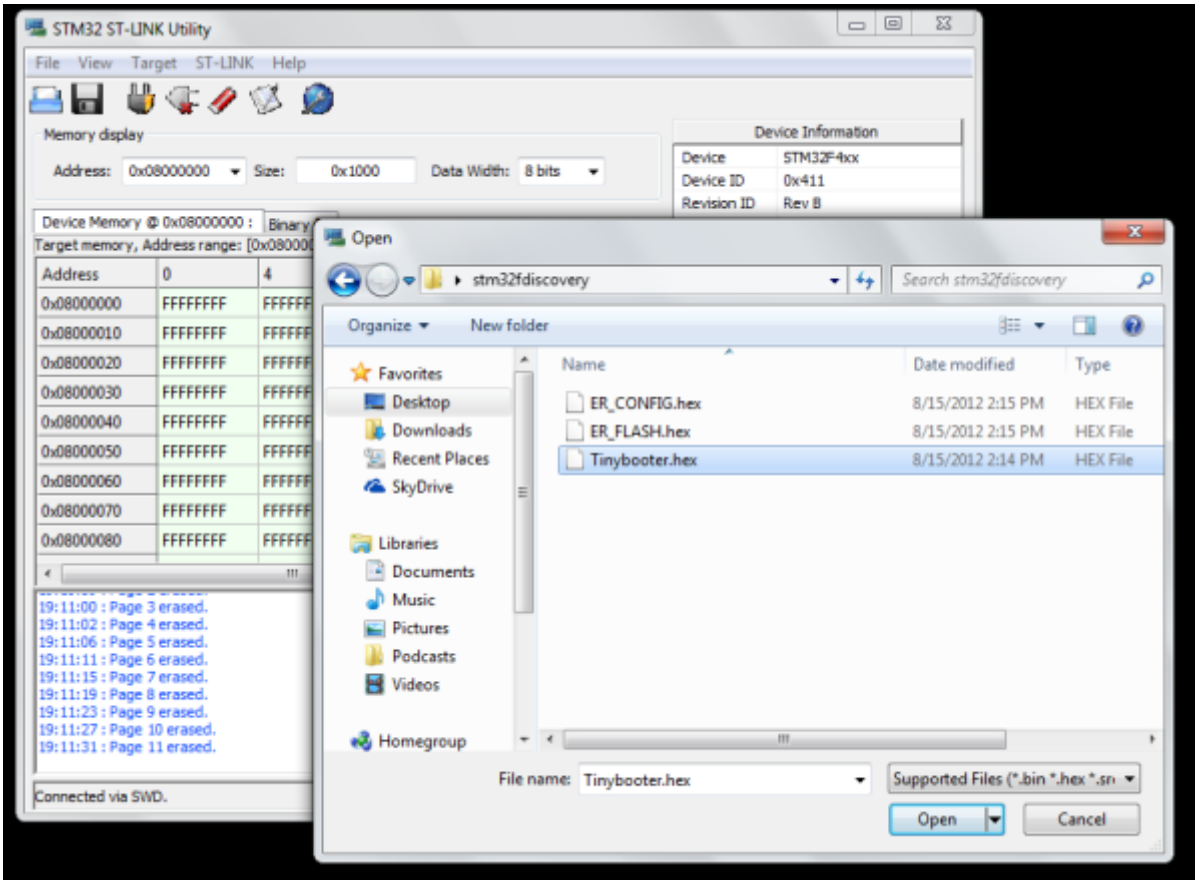
Dále klikneme na **Erase Sectors**

Nyní je obvod vymazán. Dále z stm32f4discovery.zip dostaneme 3 soubory: **Tinybooter.hex**, **ER_Flash.hex** a **ER_Config.hex**.

Použijeme **ST Link utility** k nahrání **Tinybooter.hex** do obvodu (Screenshot níže)

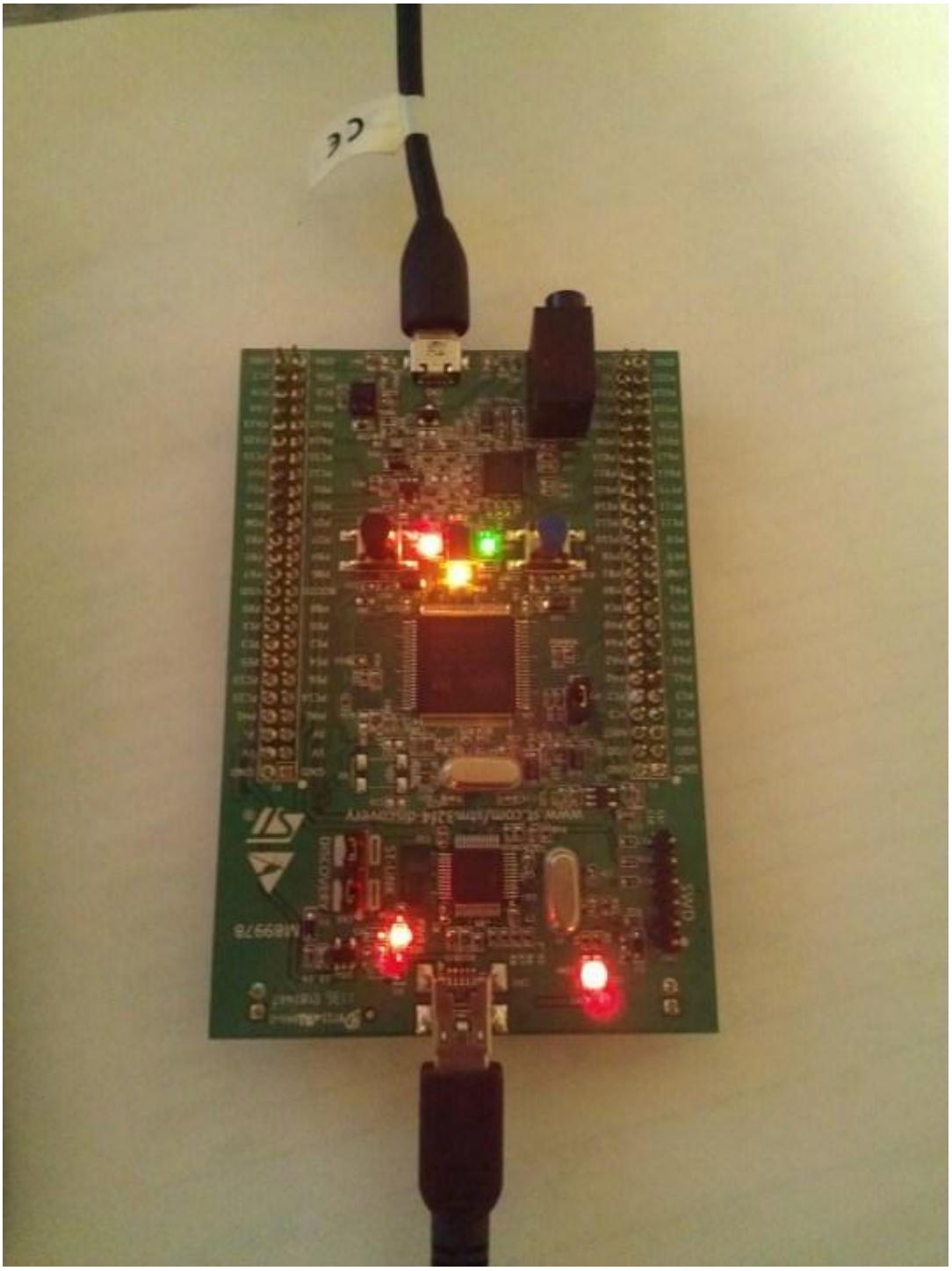


Vybereme **Program**



Dále zvolíme **Tinybooter.hex**

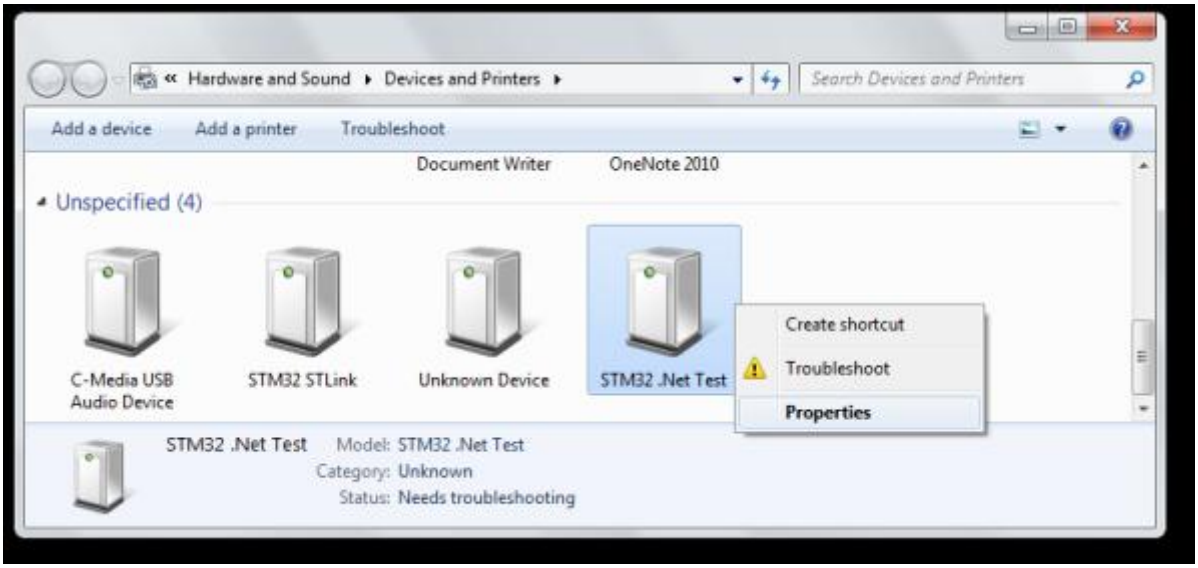
Poté co nahrajeme uvedené soubory, stiskneme tlačítko reset na startkitu (nebo vyjmeme usb kabel a znovu ho připojíme). Po nahrání **Tinybooter.hex**, bude k napájení startkitu sloužit již jen Mini USB. Po zresetování startkitu ho připojíme pomocí USB Micro USB . Např. Můžeme použít napájecí kabel k mobilnímu telefonu neboť většina mobilů používá právě Micro USB.



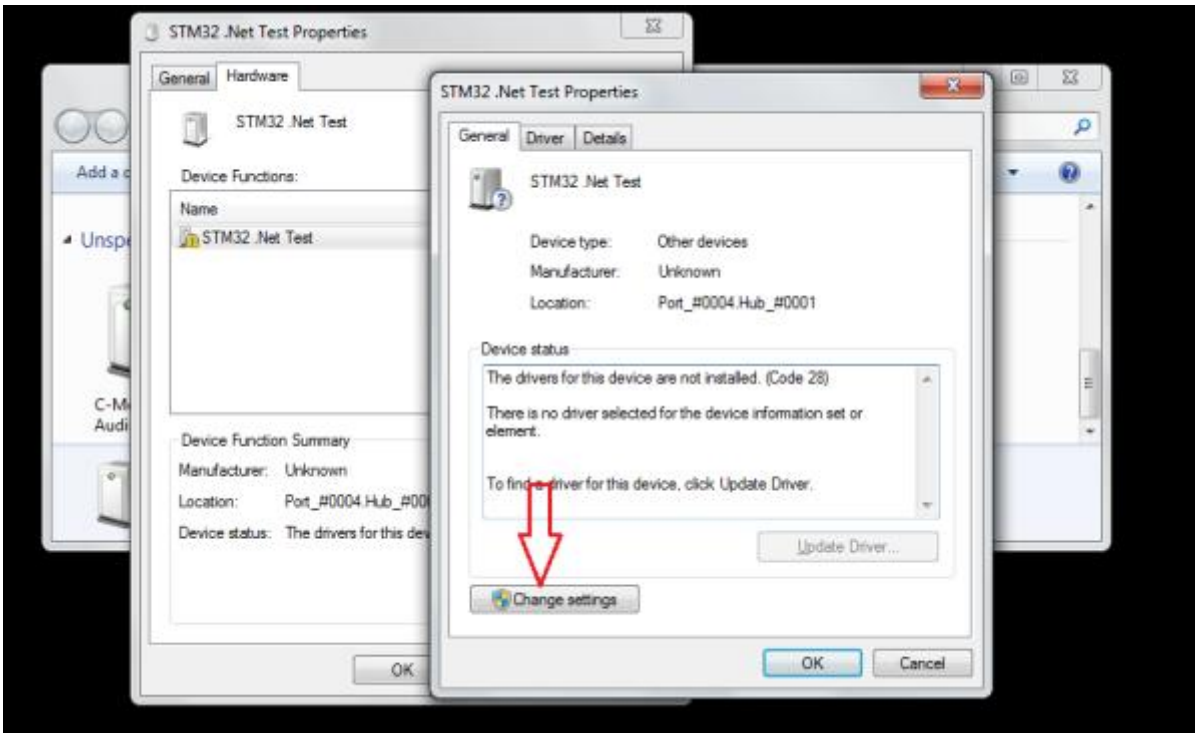
Micro and Mini USB Connected

Po připojení s Micro USB, windows budou vyhledávat ovladač, což se jim ale nepodaří. Takže ovladač musíme nainstalovat sami, přičemž ho získáme z souboru zip s jménem "STM32_WinUSB_drivers_(for_evaluation_purposes_only).zip". Ukazuje to následující screen shot.

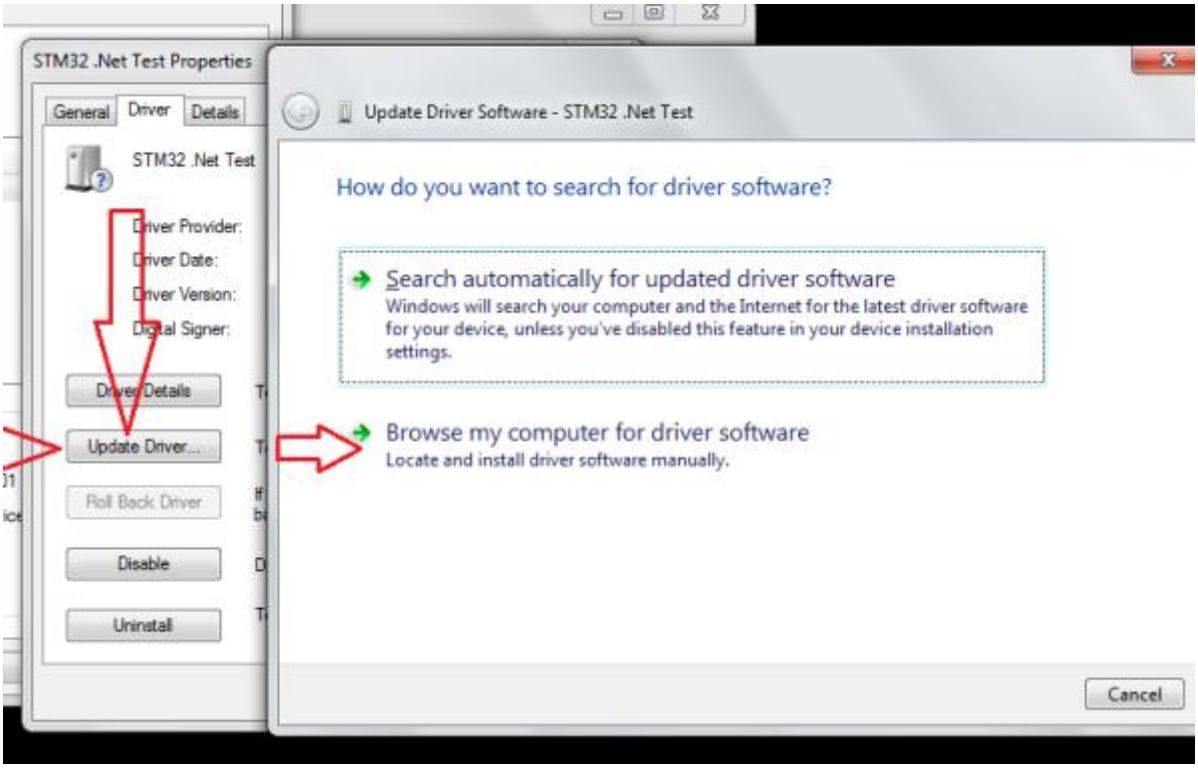
Přejdeme na **Zařízení a tiskárny** (Devices and Printers) z menu **Start** a pak



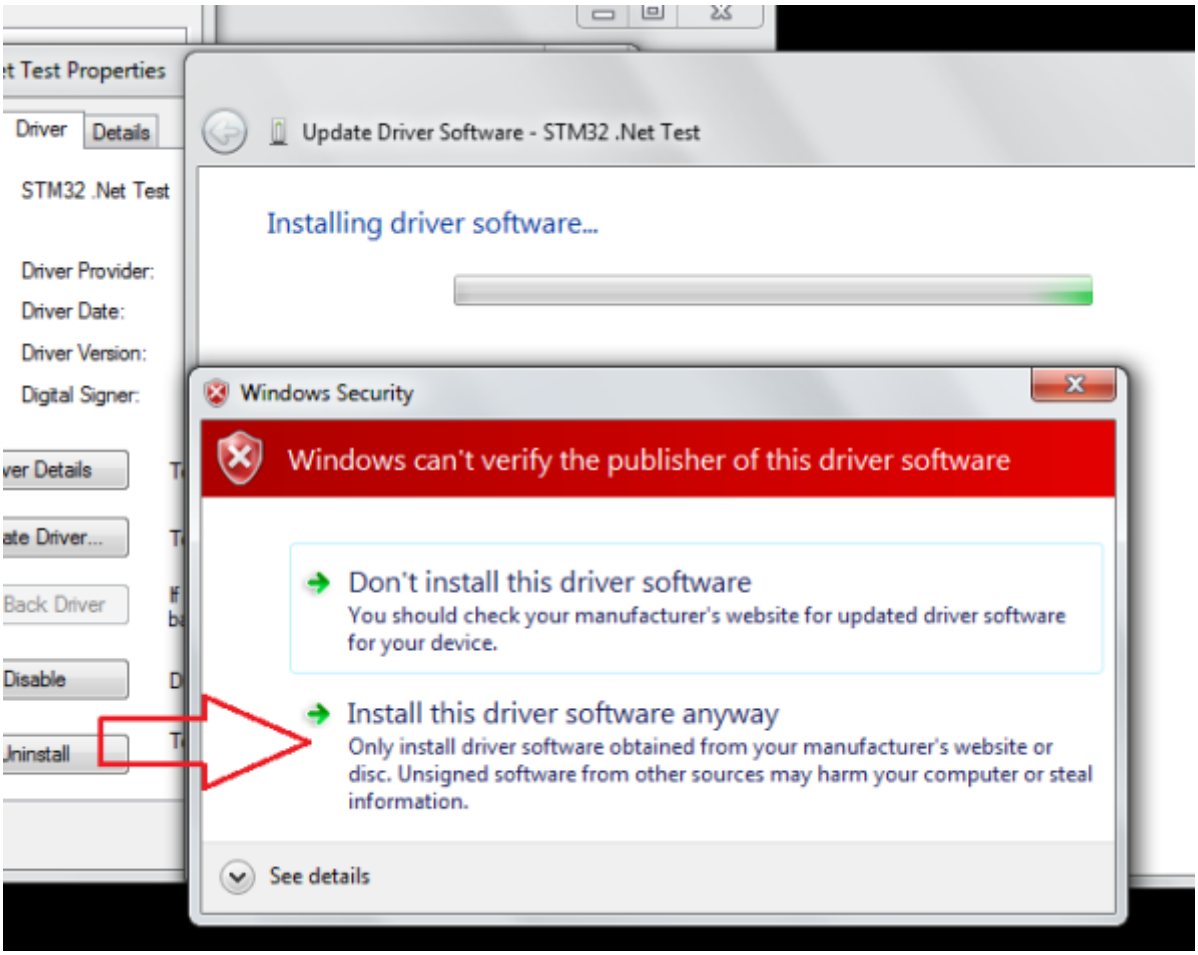
Pravým tlačítkem klikneme na **STM 32 .NET Test**



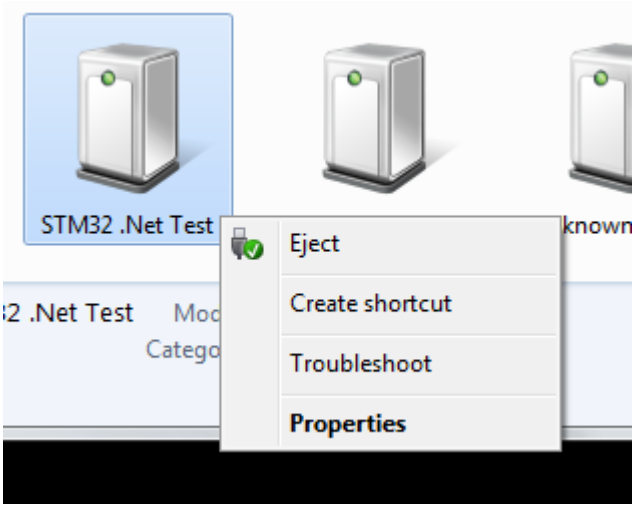
Dále vybíráme **Properties -> Change Settings**



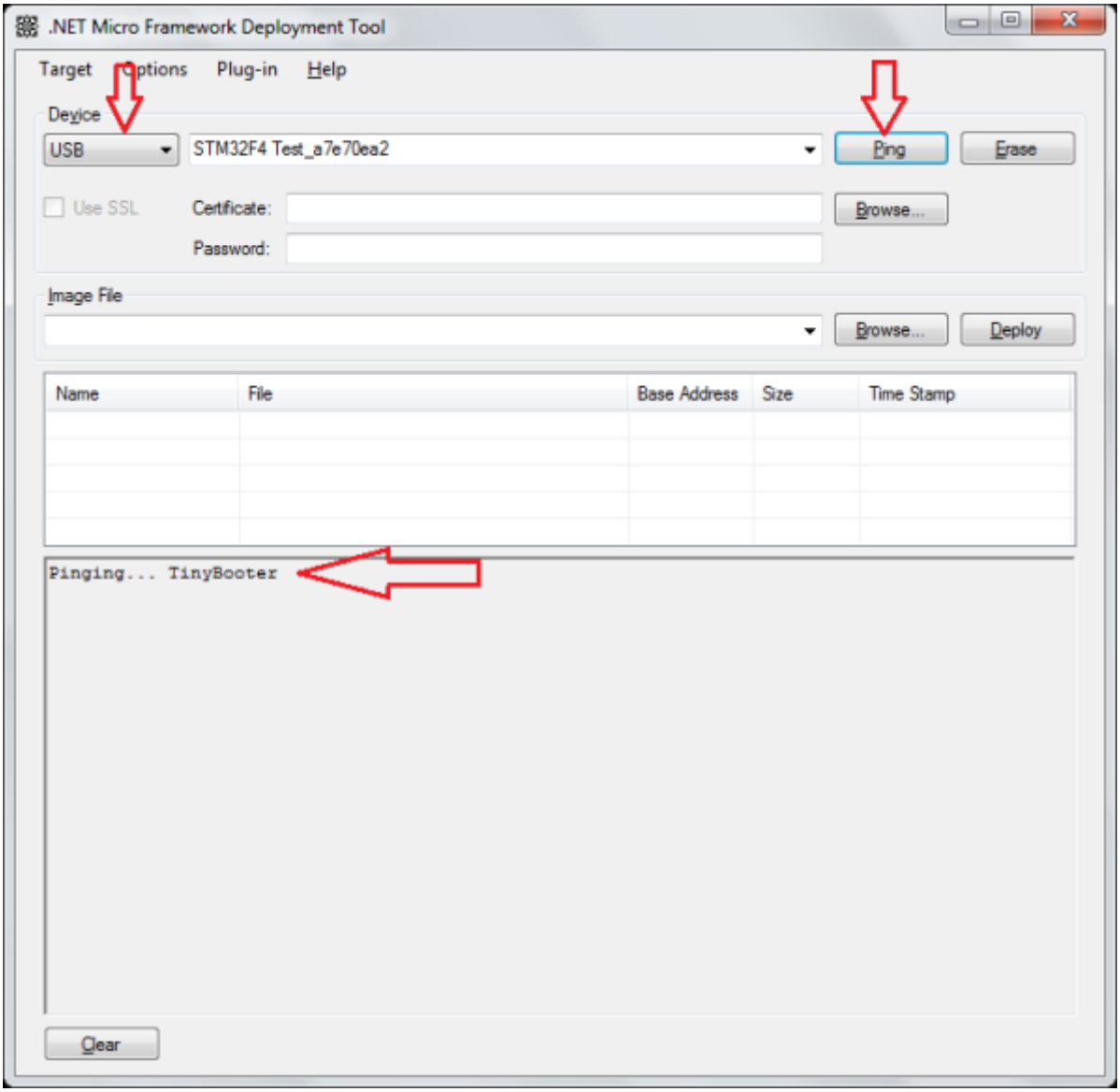
Vyhledáme nový ovladač (**Search new driver**)



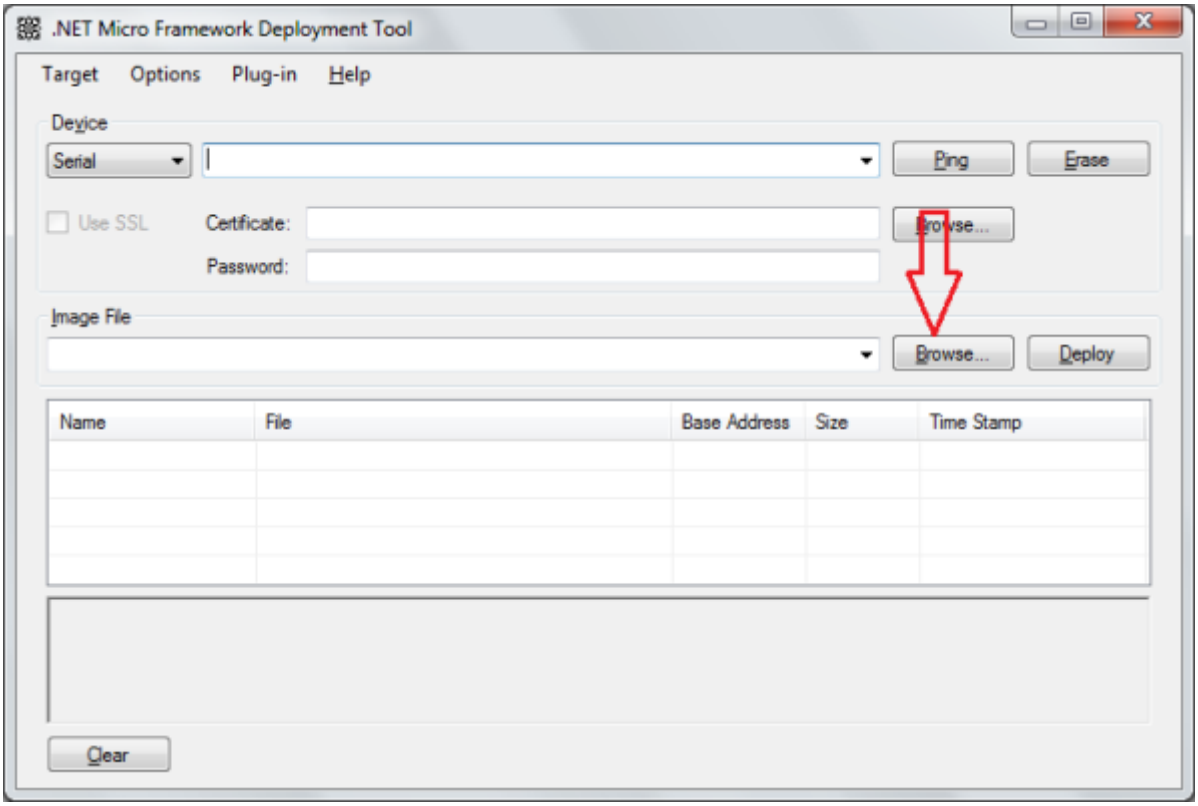
Ignorujeme upozornění (warning)



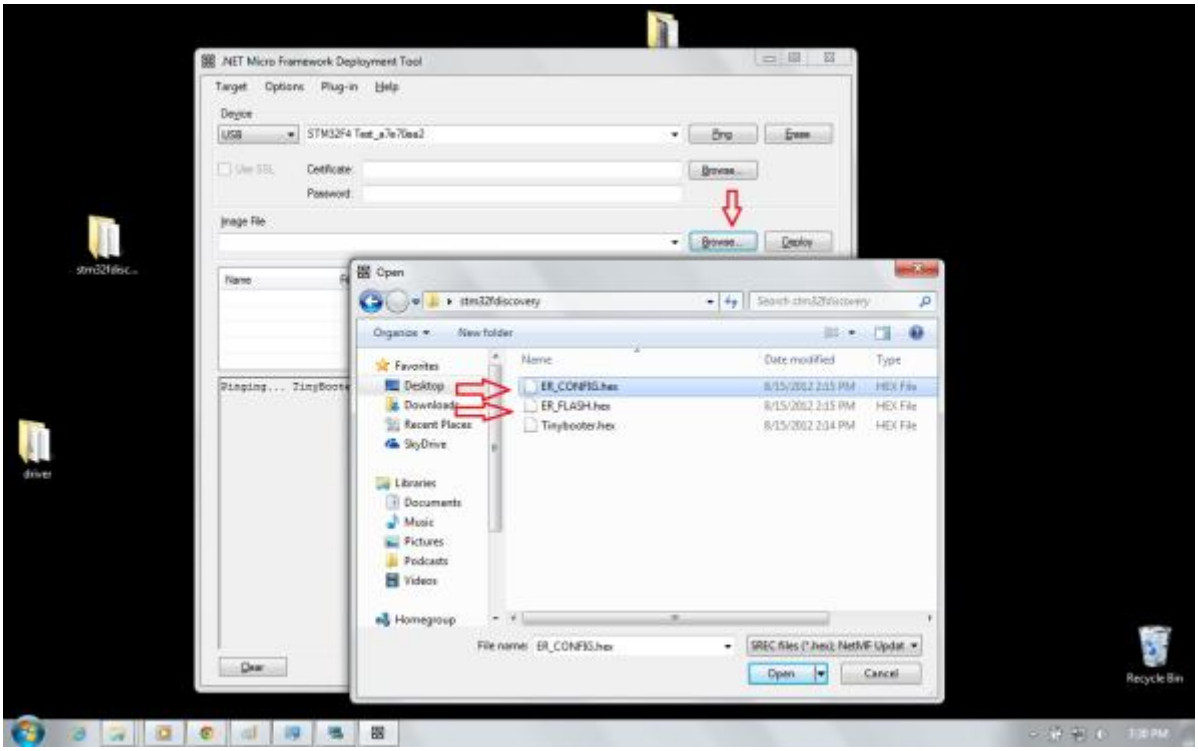
Nyní by již **MFDeploy** měl vidět náš embedded počítač. MFDeploy je MicroFramework Deployer . S použitím tohoto software nainstalujeme běhové prostředí CLR na němž poběží náš .net kód. Takže spustíme **MFDeploy.exe** (měli bychom ho nalézt v C:\Program Files (x86)\Microsoft .NET Micro Framework\v4.2\Tools\MFDeploy.exe) který by se tam měl dostat při naší instalaci SDK. Po spuštění MFDeploy bychom měli vidět náš systém:



Klikneme na **MFDeploy Ping**. Pokud bude v pořádku tak ještě nahrajeme další 2 soubory .hex (ER_Config.hex a ER_Flash.hex) získané z *stm32f4discovery.zip*. K nahrání použijeme *MFDeploy*:

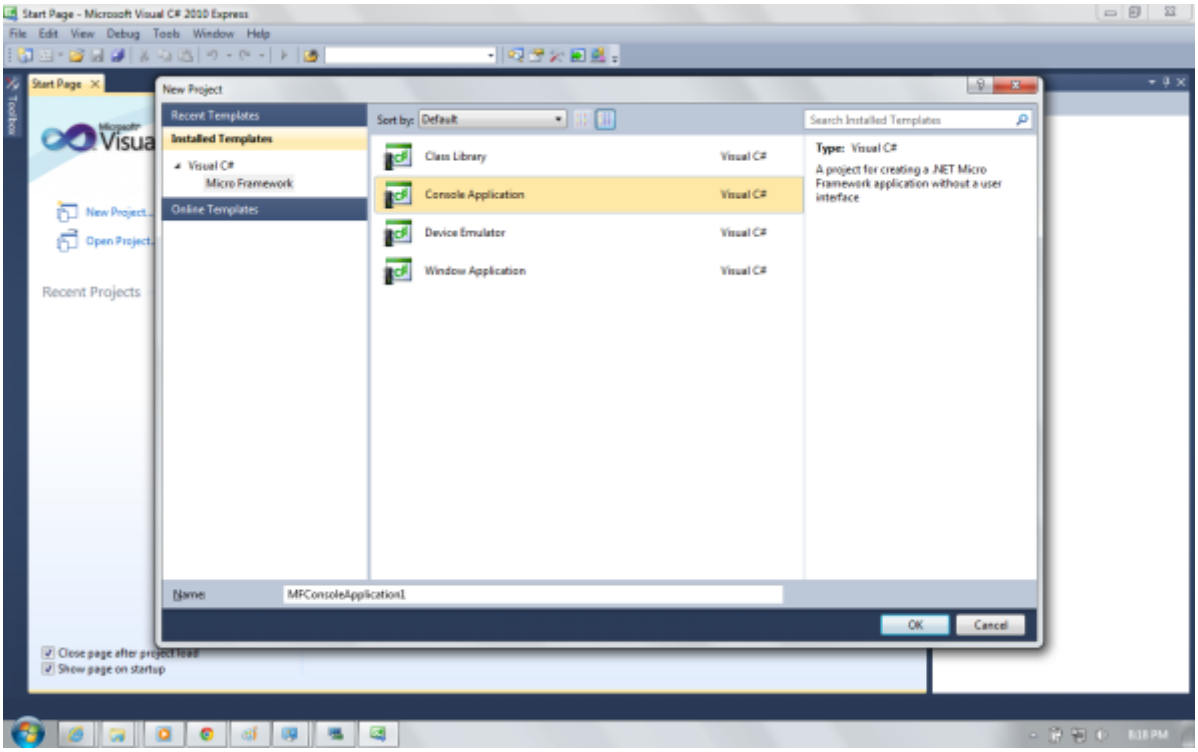


Klikneme na **Browse** abychom nahráli (download) dva další hex soubory



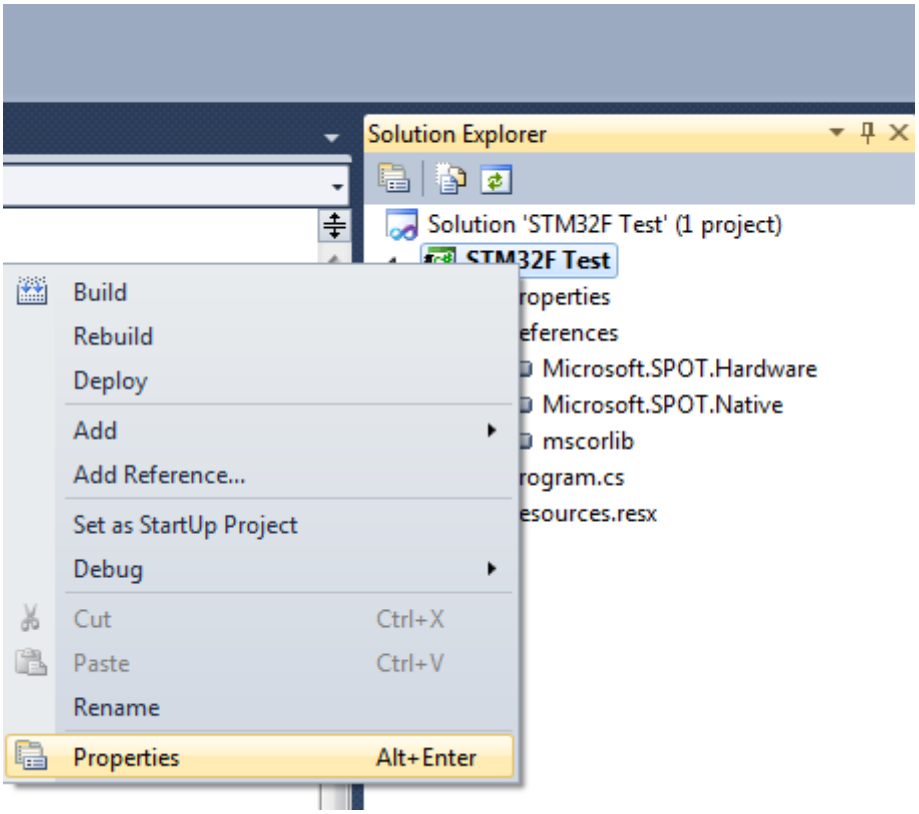
Ještě zresetujeme onboard počítač a tím již na něm můžeme provozovat *.NET Microframework board.*

Nyní ještě otestujeme náš palubní počítač. Otevřeme Visual C# Express/ Visual Studio. Předpokládám, že máme také nainstalovaný *Microframework SDK*. Vybereme **project type** podle obázku :

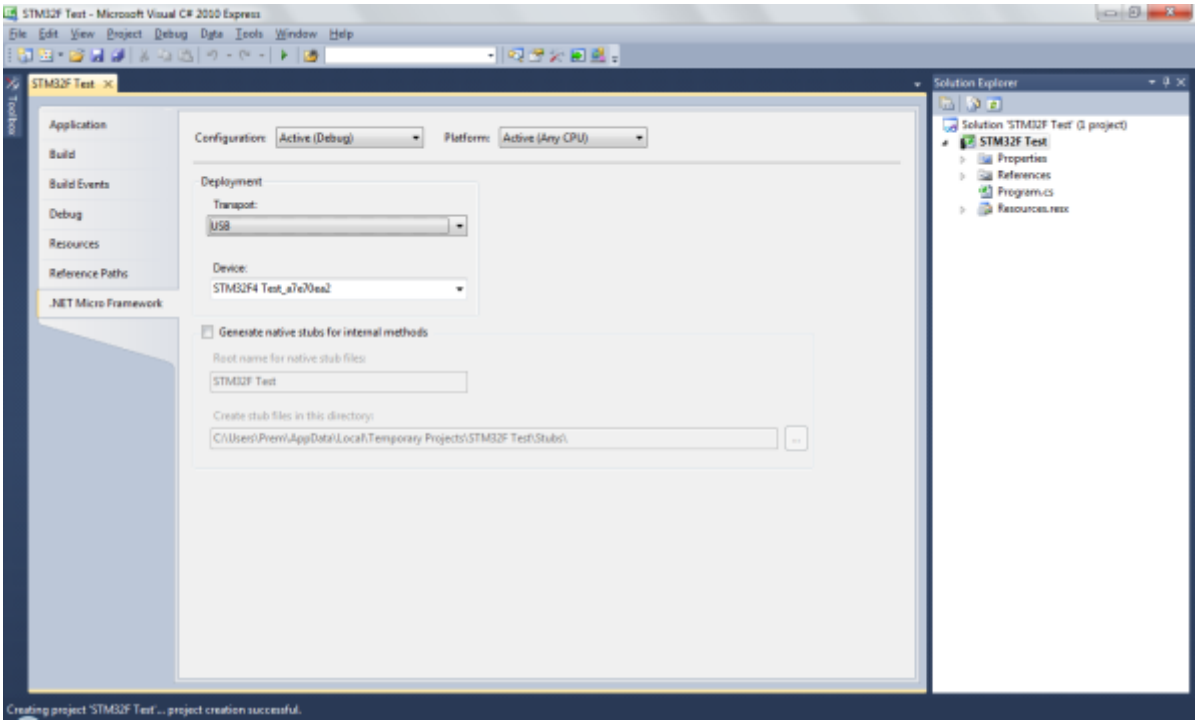


Zvolíme **MicroFramework Project**

Nyní nastavíme vlastnosti (properties) tak, aby e Visual Express/Studio mohlo program přenést do hardware. Proto změníme **project properties** následujícím postupem:

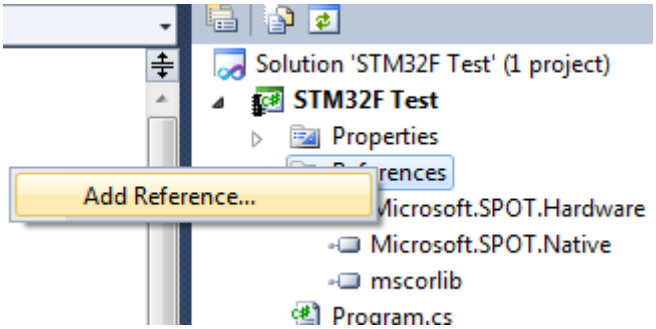


Vybereme položku **Properties**

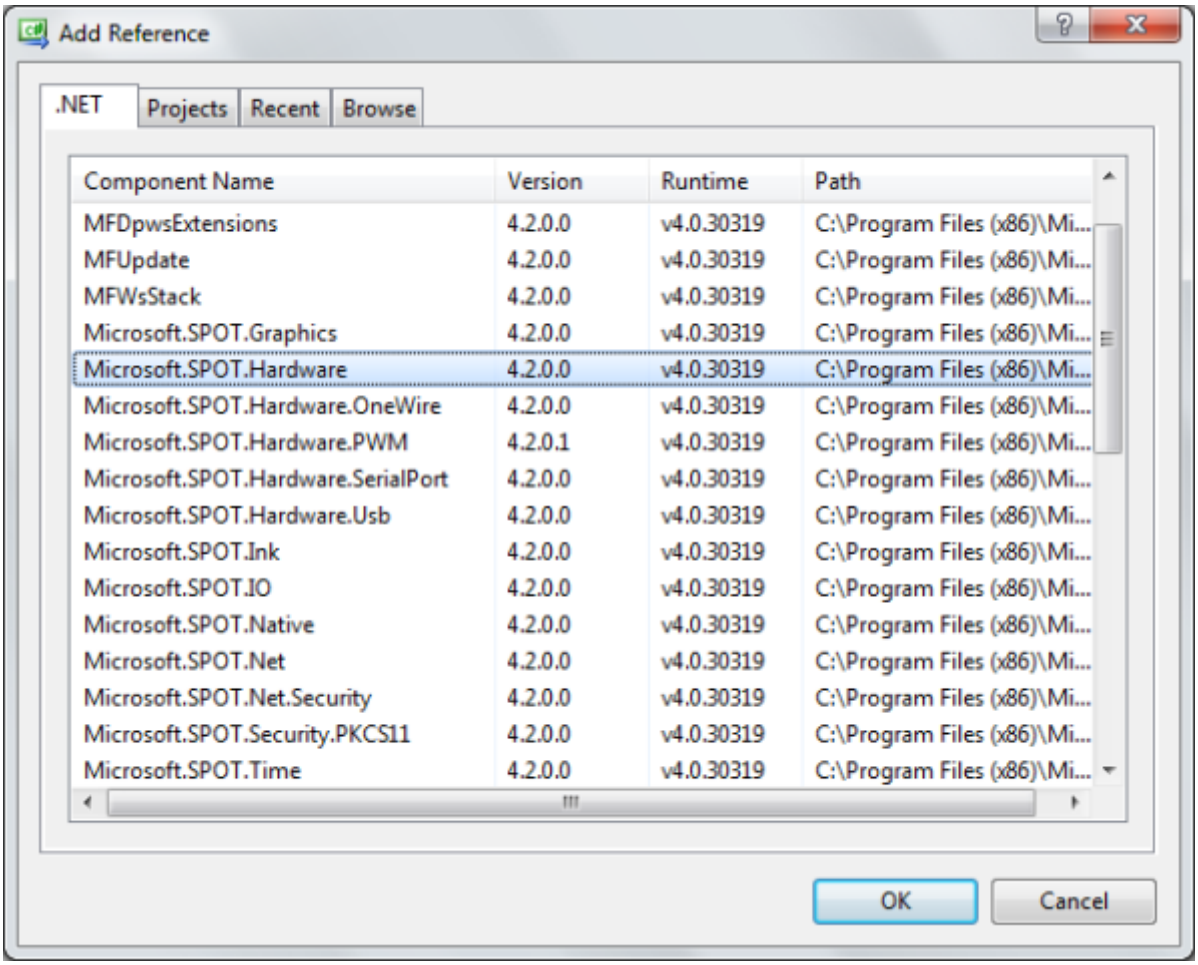


Vybereme **USB**

Nyní potřebujeme přidat třídu hardware abychom mohli spustit program blikání LED.



Vybereme **Add Reference**



Vybereme **Microsoft.SPOT.Hardware**

Dále si uvedeme kód testovacího programu

```
using System;
using System.Threading
using Microsoft.SPOT;
```

```
using Microsoft.SPOT.Hardware;

namespace STM32F_Test
{
    public class Program
    {
        public static void Main()
        {
            OutputPort led = new OutputPort(Cpu.Pin.GPIO_Pin1, false);
//PA1 on discovery board

            while (true)
            {
                led.Write(true);
                Thread.Sleep(500);
                led.Write(false);
                Thread.Sleep(500);
            }
        }

    }
}
```

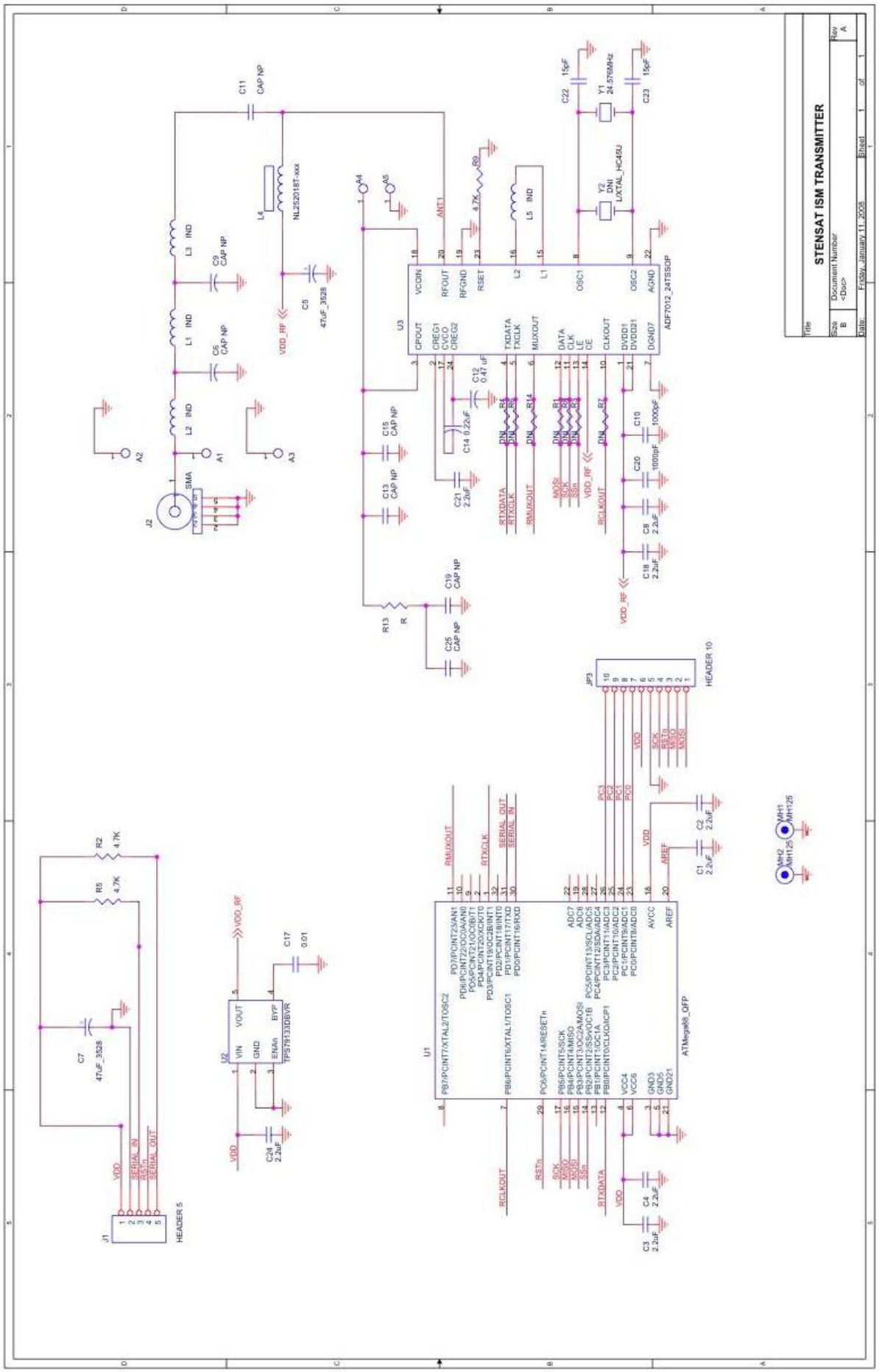
Abychom byli zcela kompatibilní, využijeme stejné namapování pinů jaké je uvedeno v souboru "C:\MicroFrameworkPK_v4_2\DeviceCode\Targets\Native\STM32\ManagedCode\Hardware\CPU.cs" který najdeme v právě nainstalovaném sw systému.

4. Komunikační systém

4.1 Vysílač

4.1.1 Vysílač STENSAT TX51

Zabývali jsme se jím v 1.díle skript, takže uvedeme jen jeho schema:


STENSAT ISM TRANSMITTER
Document Number
<Doc>
Friday, January 11, 2008
ADF7012_24TSSOP
ATMega88_QFP
TPS79133DBVR
HEADER 5
HEADER 10
SERIAL_IN
RSTn
SERIAL_OUT
RTXDATA
RTXCLK
RMUXOUT
RCLKOUT
SCK
MOSI
MISO
SSn
VDD
VDD_RF

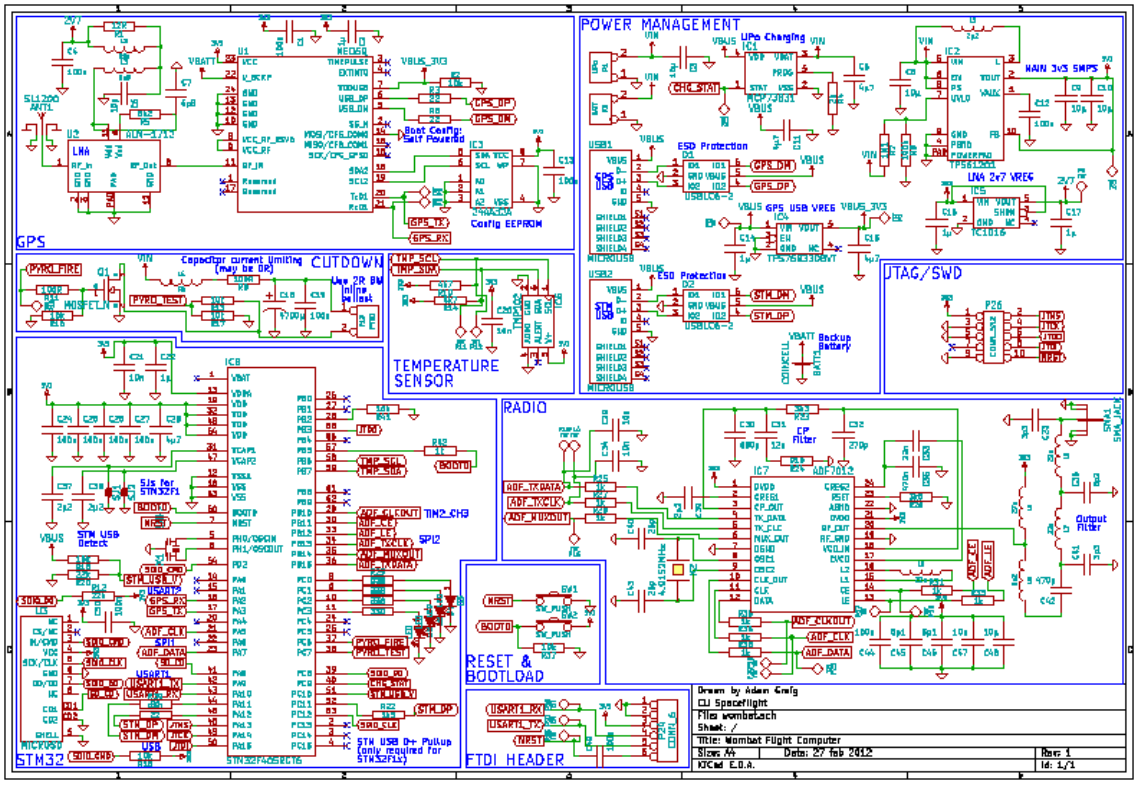
4.1.2 Projekt WOMBAT

Projekt WOMBAT je součástí studentské aktivity Cambridge University Spaceflight http://www.cusf.co.uk/ kdy studenti pre i post graduálního studia se snaží vyvinout laciné technologie sloužící k výzkumu a vynášení různých zařízení na sub-orditální dráhy balonem či raketou.

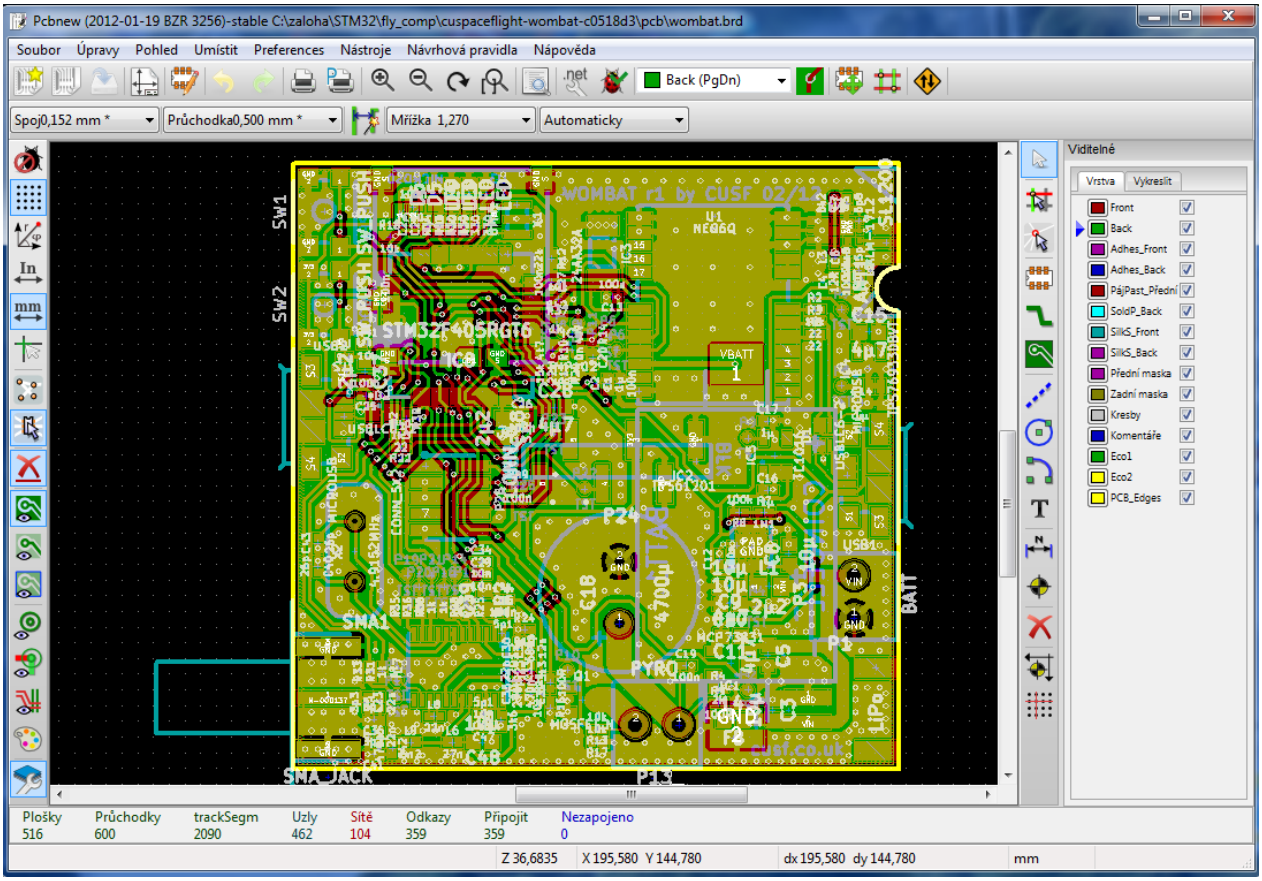
Wombat je "flight computer" obsahující
STM32F405RGT6 microcontroller na 168MHz s 192KB RAM, 1MB flash, DSP, FPU
uBlox NEO-6Q GPS s LNA a filtry
ADF7012 vysílač, programovatelný pro 70cm pásmo
SMPS napájecí zdroj z lipo baterie
Záložní onboard baterii, cutdown controller, micro SD card, USB

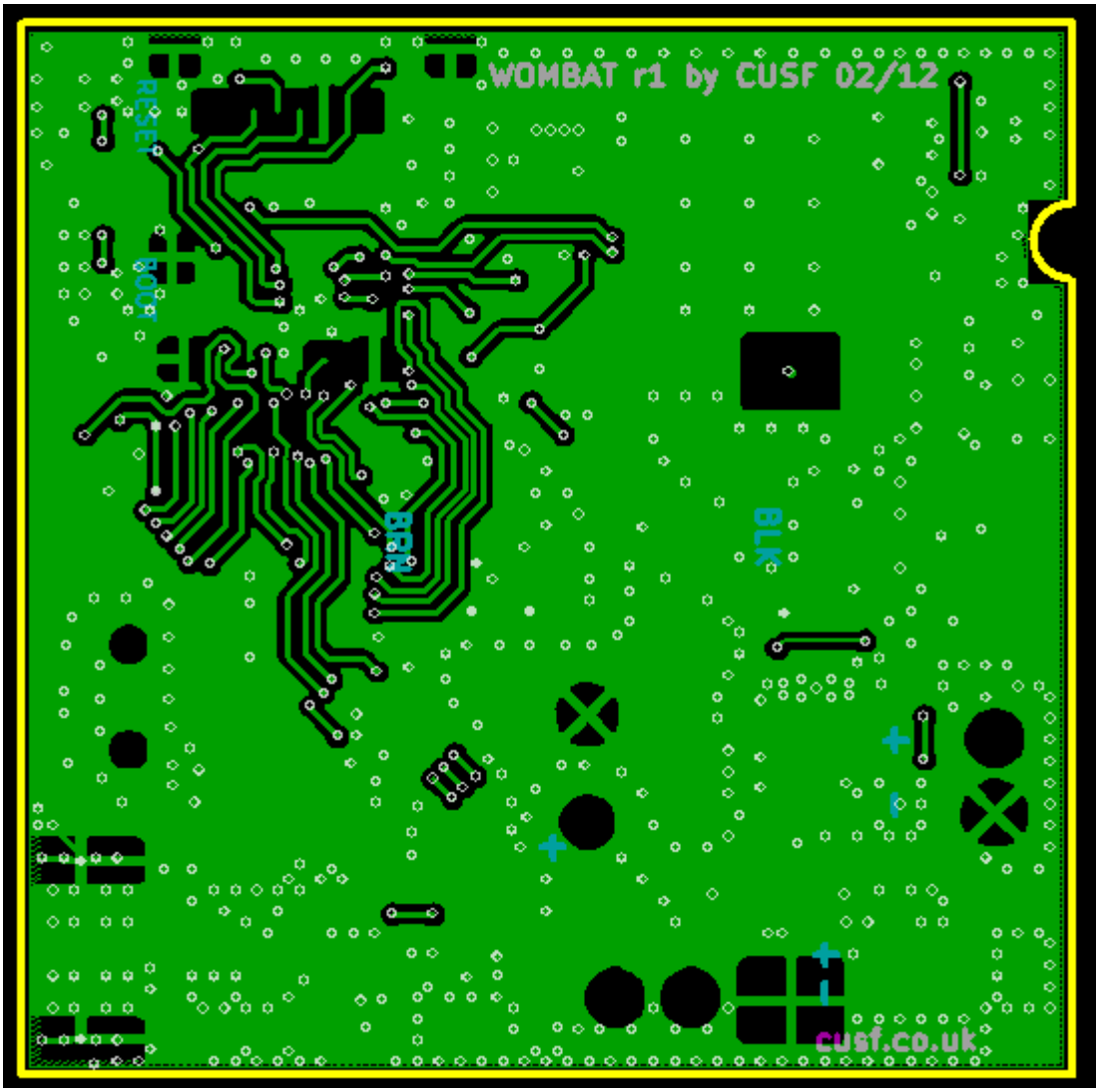
Flight Computer Wombat vzhledem ke svému konstrukčnímu provedení sice není přímo použitelný pro CANSATy (kvůli rozměrům PCB), ale můžeme se jím inspirovat při vlastním vývoji pro CANSAT. Pro vysílání dat v pásmu 70cm používá stejný integrovaný vysílač ADF7012, který je použit i v Cansat startkitu PrattHobbies. Místo MCU ATMega88 ovládajícího vysílač startkitu PrattHobbies použil konstruktér Wombatu Adam Greig mnohem výkonnější počítač s ARM Cortex STM32F405RGT6, tedy stejný , který jsem použil při vývoji palubního počítače i já, viz kap.2.1. Protože A. Greig uveřejnil i kompletní dokumentaci a to včetně zdrojových kódů na https://github.com/cuspaceflight/wombat může posloužit jako inspirace při vývoji CANSATu.

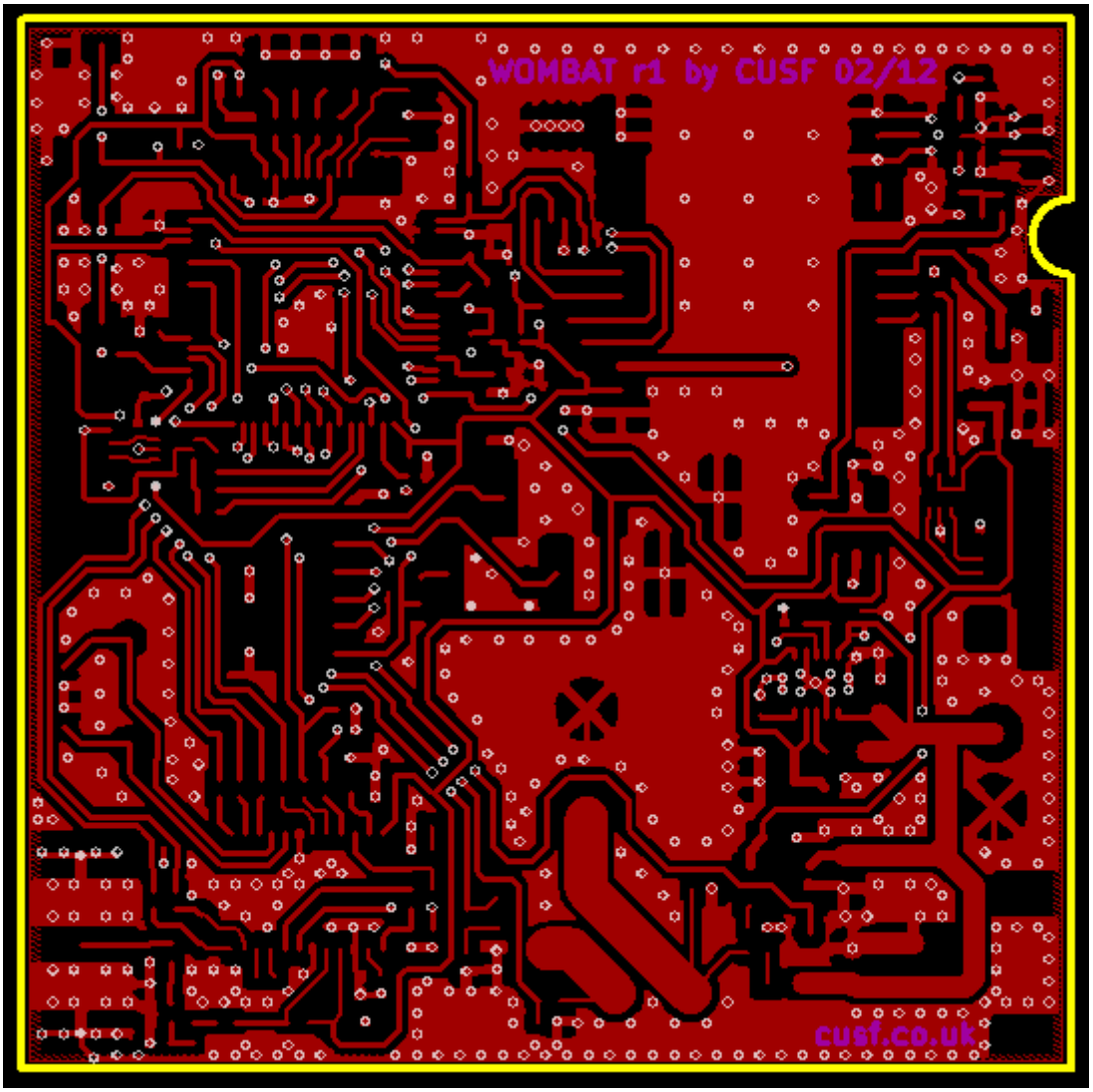
Hardware WOMBATu je zcela popsáno schématem a PCB. Schema:



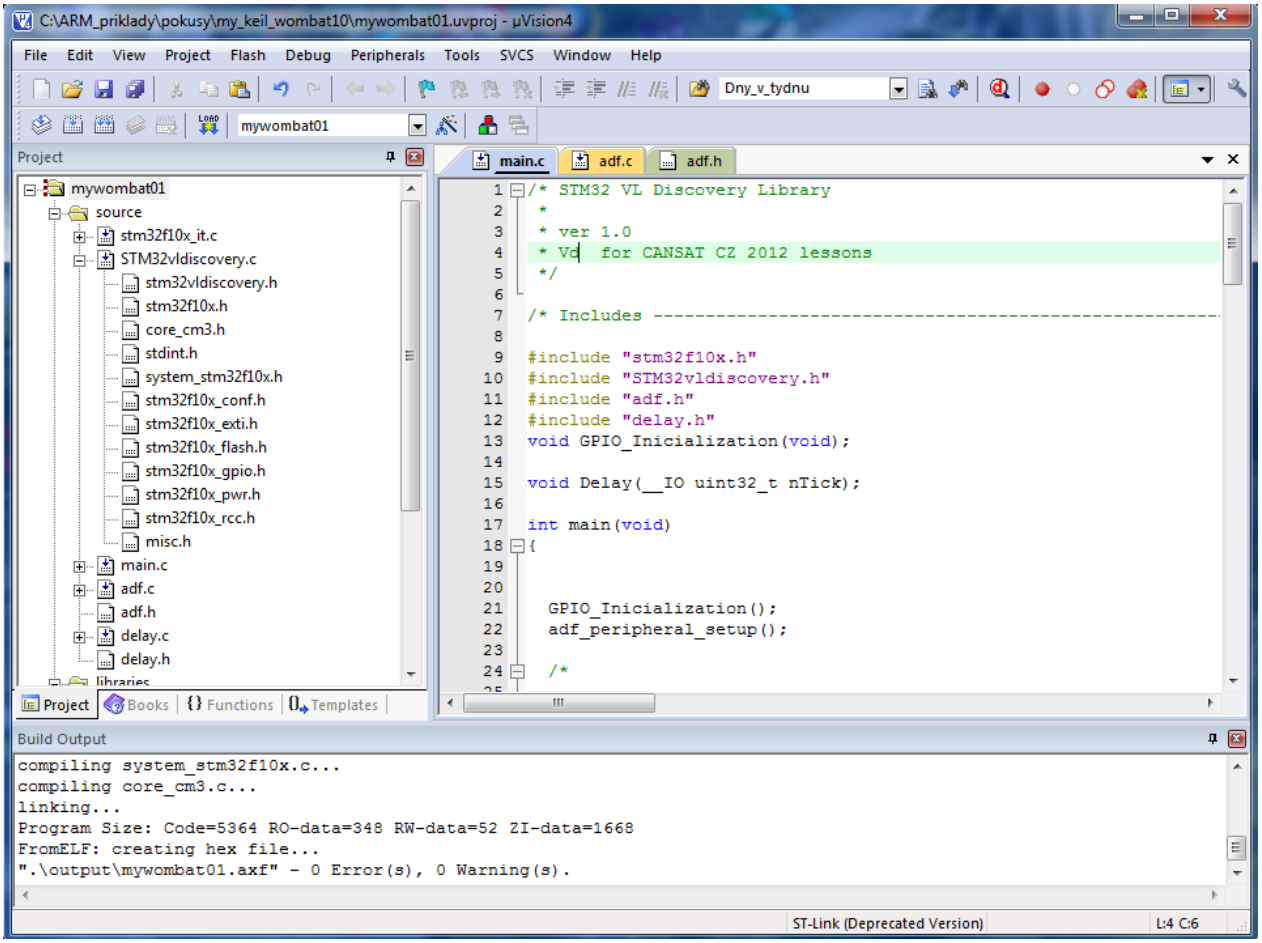
i PCB byly vytvořeny ve free nástroji KiCad http://kicad.wbs.cz/ a http://www.kicad-pcb.org

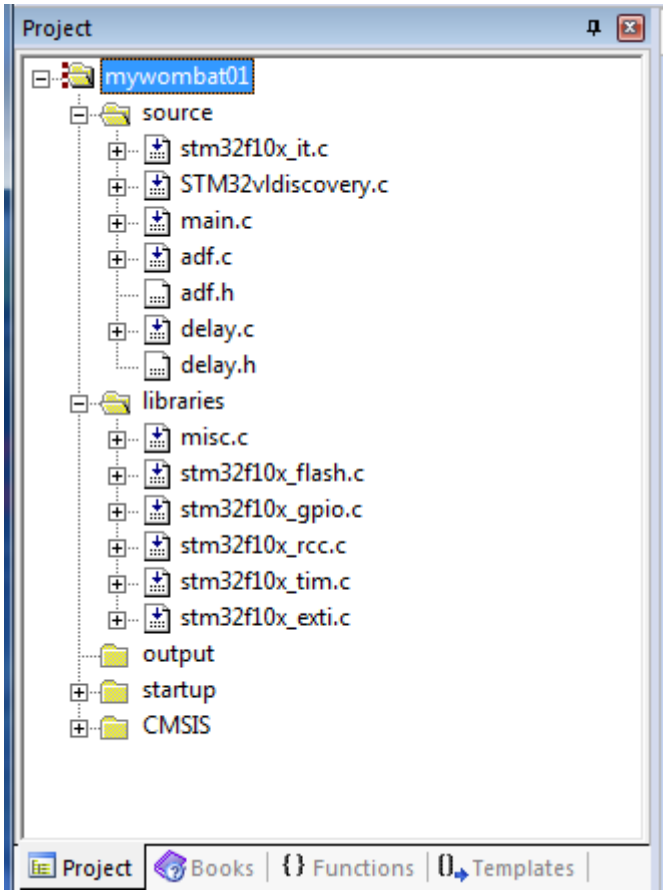

Pcbnew (2012-01-19 BZR 3256)-stable C:\zaloha\STM32\fly_comp\cuspaceflight-wombat-c0518d3\pcb\wombat.brd
Soubor Úpravy Pohled Umístit Preferences Nástroje Návrhová pravidla Nápověda
Back (PgDn)
Spoj0,152 mm *
Průchodka0,500 mm *
Mřížka 1,270
Automaticky
Viditelné
Vrstva
Vykreslit
Front
Back
Adhes_Front
Adhes_Back
PájPast_Přední
SoldP_Back
SilkS_Front
SilkS_Back
Přední maska
Zadní maska
Kresby
Komentáře
Eco1
Eco2
PCB_Edges
WOMBAT r1 by CUSF 02/12
STM32F405RGT6
SW1
SW2
SMA1
SMA_JACK
PYRO
BATT
Plošky 516
Průchodky 600
trackSegm 2090
Uzly 462
Sítě 104
Odkazy 359
Připojit 359
Nezapojeno 0
Z 36,6835
X 195,580 Y 144,780
dx 195,580 dy 144,780
mm


WOMBAT r1 by CUSF 02/12
RESET
BOOT
BOOT
BLK
cusf.co.uk



Rovněž pro vývoj software byl použit free compilátor jazyka C GCC a free knihovna libopencm3 pro STM32F1 F2 a F4. Přitom můžeme pracovat jak nad OS Linux, tak OS Windows. V případě Windows potřebujeme ještě MSYS. Pomocí stejných free nástrojů můžeme vyvíjet i sw pro Cansat, bohužel knihovna libopensm3 není tak rozsáhlá jako knihovny dodávané STMicroelectronics pro STM32F1 a STM32F4 a pokud budeme používat i čidla od STM (např. MEMSy) je bude používání knihoven od STM ještě výhodnější, mj. i proto, že pro STM32F a čidla získáme od STM i řadu vzorových kódů.
Ze zdrojového kódu WOMBATu doporučuji převzít kód pro řízení obvodu vysílače ADF7012. Vyznačuje se tím, že je dobře čitelný a proto použitelný i při výuce. Proto jsem ho přepsal do projektu nad uKeil Vision4. Práci v tomto prostředí jsem popsal již v 2.díle skript [1.2] a rovněž v předchozích kapitolách tohoto výukového materiálu.


C:\ARM_priklady\pokusy\my_keil_wombat10\mywombat01.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Dny_v_tydnu
mywombat01
Project
mywombat01
source
stm32f10x_it.c
STM32vldiscovery.c
stm32vldiscovery.h
stm32f10x.h
core_cm3.h
stdint.h
system_stm32f10x.h
stm32f10x_conf.h
stm32f10x_exti.h
stm32f10x_flash.h
stm32f10x_gpio.h
stm32f10x_pwr.h
stm32f10x_rcc.h
misc.h
main.c
adf.c
adf.h
delay.c
delay.h
libraries
Project Books Functions Templates
main.c adf.c adf.h
1 /* STM32 VL Discovery Library
2 *
3 * ver 1.0
4 * Vo for CANSAT CZ 2012 lessons
5 */
6
7 /* Includes --
8
9 #include "stm32f10x.h"
10 #include "STM32vldiscovery.h"
11 #include "adf.h"
12 #include "delay.h"
13 void GPIO_Inicialization(void);
14
15 void Delay(__IO uint32_t nTick);
16
17 int main(void)
18 {
19
20
21 GPIO_Inicialization();
22 adf_peripheral_setup();
23
24 /*
Build Output
compiling system_stm32f10x.c...
compiling core_cm3.c...
linking...
Program Size: Code=5364 RO-data=348 RW-data=52 ZI-data=1668
FromELF: creating hex file...
".\output\mywombat01.axf" - 0 Error(s), 0 Warning(s).
ST-Link (Deprecated Version)
L:4 C:6




```
//tx with ADF7012  by OK Vd
// adf.h
// ADF API
//=========================================================================
uint32_t pokus_CtiBit(void);
void adf_peripheral_setup(void);

void adf_turn_off(void);
void adf_turn_on(void);

void adf_reset_config(void);
/**/
void adf_write_config(void);

void adf_transmit_string(char* data, u32 baud);

u8 adf_locked(void);
u8 adf_reg_ready(void);


void adf_set_frequency_error_correction(u16 error);
void adf_set_r_divider(u8 r);
void adf_set_vco_adjust(u8 adjust);

/**/
void adf_set_frequency(u32 khz);
```

```
void adf_set_n(u8 n);
void adf_set_m(u16 m);
void adf set pa level(u8 level);
void adf_set_pll_enable(u8 enable);
void adf_set_pa_enable(u8 enable);
void adf_set_clkout_enable(u8 enable);
void adf_set_charge_pump_current(u8 current);
void adf_set_vco_disable(u8 disable);
void adf_set_muxout(u8 muxout);
void adf_set_vco_bias(u8 bias);
void adf_set_pa_bias(u8 bias);
/**/
void adf_setup(void);

void adf_power_on(void);
void adf_power_off(void);

u8 adf_lock(void);

u8 adf_get_n(void);
u16 adf_get_m(void);
u8 adf_get_vco_adjust(void);
u8 adf_get_vco_bias(void);


// The ADF7012 pin connections
=============================================
#define ADF_CLK          GPIO_Pin_5
#define ADF_CLK_PORT     GPIOA
#define ADF_DATA         GPIO_Pin_10
#define ADF_DATA_PORT    GPIOB
#define ADF_CLKOUT       GPIO_Pin_7
#define ADF_CLKOUT_PORT GPIOA
#define ADF_CE           GPIO_Pin_11
#define ADF_CE_PORT      GPIOB
#define ADF_LE           GPIO_Pin_12
#define ADF_LE_PORT      GPIOB
#define ADF_TXCLK        GPIO_Pin_13
#define ADF_TXCLK_PORT   GPIOB
#define ADF_MUXOUT       GPIO_Pin_14
#define ADF_MUXOUT_PORT GPIOB
#define ADF_TXDATA       GPIO_Pin_15
#define ADF_TXDATA_PORT GPIOB

// Timing constants
=============================================================
#define ADF_50_BAUD      (u32)(479441)
#define ADF_300_BAUD     (u32)(79915)
#define ADF_1200_BAUD    (u32)(19979)

// General Purpose
=============================================================
#define ADF_OFF 0
#define ADF_ON  1

// Register Constants
=========================================================
// Register 1 ----------------------------------------------------------------
---
```

```
#define ADF_PRESCALER_4_5 0
#define ADF_PRESCALER_8_9 1

// Register 2 ----------------------------------------------------------
---
#define ADF_MODULATION_FSK 0
#define ADF_MODULATION_GFSK 1
#define ADF_MODULATION_ASK 2
#define ADF_MODULATION_OOK 3

// Register 3 ----------------------------------------------------------
---
#define ADF_CP_CURRENT_0_3 0
#define ADF_CP_CURRENT_0_9 1
#define ADF_CP_CURRENT_1_5 2
#define ADF_CP_CURRENT_2_1 3
#define ADF_MUXOUT_LOGIC_LOW 0
#define ADF_MUXOUT_LOGIC_HIGH 1
#define ADF_MUXOUT_REG_READY 3
#define ADF_MUXOUT_DIGITAL_LOCK 4
#define ADF_MUXOUT_ANALOGUE_LOCK 5
#define ADF_MUXOUT_R_DIVIDER_2 6
#define ADF_MUXOUT_N_DIVIDER_2 7
#define ADF_MUXOUT_RF_R_DIVIDER 8
#define ADF_MUXOUT_DATA_RATE 9
#define ADF_MUXOUT_BATT_2_35 10
#define ADF_MUXOUT_BATT_2_75 11
#define ADF_MUXOUT_BATT_3 12
#define ADF_MUXOUT_BATT_3_25 13
#define ADF_MUXOUT_TEST_MODE 14
#define ADF_MUXOUT_SD_TEST_MODE 15
#define ADF_LD_PRECISION_3_CYCLES 0
#define ADF_LD_PRECISION_5_CYCLES 1
```

```
//adf.c

#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include <stdio.h>
#include "delay.h"
#include "adf.h"
//#include "led.h"

// Configuration storage structs
=============================================
struct {
    struct {
        u16 frequency_error_correction;
        u8 r_divider;
        u8 crystal_doubler;
        u8 crystal_oscillator_disable;
        u8 clock_out_divider;
        u8 vco_adjust;
        u8 output_divider;
```

```
    } r0;

    struct {
        u16 fractional_n;
        u8 integer_n;
        u8 prescaler;
    } r1;

    struct {
        u8 mod_control;
        u8 gook;
        u8 power_amplifier_level;
        u16 modulation_deviation;
        u8 gfsk_modulation_control;
        u8 index_counter;
    } r2;

    struct {
        u8 pll_enable;
        u8 pa_enable;
        u8 clkout_enable;
        u8 data_invert;
        u8 charge_pump_current;
        u8 bleed_up;
        u8 bleed_down;
        u8 vco_disable;
        u8 muxout;
        u8 ld_precision;
        u8 vco_bias;
        u8 pa_bias;
        u8 pll_test_mode;
        u8 sd_test_mode;
    } r3;
} adf_config;

// Prototypes for internal functions
=========================================
/*
void adf_set_pin_output(u32 prt, u16 pin);
void adf_set_pin_input(u32 prt, u16 pin);
*/
void adf_set_pin_output(GPIO_TypeDef * prt, u16 pin);
void adf_set_pin_input(GPIO_TypeDef *  prt, u16 pin);




void adf_reset_register_zero(void);
void adf_reset_register_one(void);
void adf_reset_register_two(void);
void adf_reset_register_three(void);


void adf_reset(void);
void adf_write_register(u32 reg);
void adf_write_register_zero(void);
void adf_write_register_one(void);
void adf_write_register_two(void);
void adf_write_register_three(void);
void adf_write_config(void);
```

```
void adf_transmit_bit(u8 bit);
void adf_transmit_byte(u8 byte, u32 baud);
u8 adf_read_muxout(void);


// STM32 peripheral functions
================================================


void adf_set_pin_output(GPIO_TypeDef * prt, u16 pin) {
    //gpio_mode_setup(prt, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, pin);

              GPIO_InitTypeDef GPIO_InitStructure;
              if (prt == GPIOA)
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
              if (prt == GPIOB)
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
              if (prt == GPIOC) RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC
, ENABLE);


              GPIO_InitStructure.GPIO_Pin = pin;
              GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
             // GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; //
alternate function!
              GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
              if (prt == GPIOA) GPIO_Init(GPIOA, &GPIO_InitStructure);
              if (prt == GPIOB) GPIO_Init(GPIOB, &GPIO_InitStructure);
              if (prt == GPIOC) GPIO_Init(GPIOC, &GPIO_InitStructure);

       /*
       GPIO_InitTypeDef GPIO_InitStructure;
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

  GPIO_InitStructure.GPIO_Pin =GPIO_Pin_11;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  GPIO_ResetBits(GPIOB, GPIO_Pin_11);

      */

}



void adf_set_pin_input(GPIO_TypeDef *  prt, u16 pin) {
    //gpio_mode_setup(prt, GPIO_MODE_INPUT, GPIO_PUPD_NONE, pin);
         GPIO_InitTypeDef GPIO_InitStructure;
              if (prt == GPIOA)
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
              if (prt == GPIOB)
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
              if (prt == GPIOC) RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC
, ENABLE);

         GPIO_InitStructure.GPIO_Pin = pin;
         GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    if (prt == GPIOA) GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
        if (prt == GPIOB) GPIO_Init(GPIOB, &GPIO_InitStructure);
        if (prt == GPIOC) GPIO_Init(GPIOC, &GPIO_InitStructure);

}


/**/
void adf_peripheral_setup() {


    adf_set_pin_output(ADF_CLK_PORT, ADF_CLK);
    adf_set_pin_output(ADF_DATA_PORT, ADF_DATA);

    adf_set_pin_output(ADF_CE_PORT, ADF_CE);
    adf_set_pin_output(ADF_LE_PORT, ADF_LE);
    adf_set_pin_output(ADF_TXCLK_PORT, ADF_TXCLK);
    adf_set_pin_output(ADF_TXDATA_PORT, ADF_TXDATA);
    adf_set_pin_input(ADF_MUXOUT_PORT, ADF_MUXOUT);
    adf_set_pin_input(ADF_CLKOUT_PORT, ADF_CLKOUT);

}


// Configuration functions
=================================================

// Config resetting functions ----------------------------------------------
/**/
void adf_reset_config(void) {
    adf_reset_register_zero();
    adf_reset_register_one();
    adf_reset_register_two();
    adf_reset_register_three();

    adf_reset();

    while(!adf_reg_ready());

    adf_write_config();
}


void adf_reset_register_zero(void) {
    adf_config.r0.frequency_error_correction = 0;
    adf_config.r0.r_divider = 1;
    adf_config.r0.crystal_doubler = ADF_OFF;
    adf_config.r0.crystal_oscillator_disable = ADF_OFF;
    //adf_config.r0.clock_out_divider = 0;
      adf_config.r0.clock_out_divider = 0xF;
    adf_config.r0.vco_adjust = 0;
    adf_config.r0.output_divider = 0;
}

void adf_reset_register_one(void) {
    adf_config.r1.fractional_n = 0;
    adf_config.r1.integer_n = 0;
    adf_config.r1.prescaler = ADF_PRESCALER_4_5;
}

void adf_reset_register_two(void) {
```

```
    adf_config.r2.mod_control = ADF_MODULATION_FSK;
    adf_config.r2.gook = ADF_OFF;
    adf_config.r2.power_amplifier_level = 0;
    adf_config.r2.modulation_deviation = 3;
    adf_config.r2.gfsk_modulation_control = 0;
    adf_config.r2.index_counter = 0;
}

void adf_reset_register_three(void) {
    adf_config.r3.pll_enable = ADF_OFF;
    adf_config.r3.pa_enable = ADF_OFF;
    adf_config.r3.clkout_enable = ADF_OFF;
    adf_config.r3.data_invert = ADF_OFF;
    adf_config.r3.charge_pump_current = ADF_CP_CURRENT_2_1;
    adf_config.r3.bleed_up = ADF_OFF;
    adf_config.r3.bleed_down = ADF_OFF;
    adf_config.r3.vco_disable = ADF_OFF;
    adf_config.r3.muxout = ADF_MUXOUT_REG_READY;
    adf_config.r3.ld_precision = ADF_LD_PRECISION_5_CYCLES;
    adf_config.r3.vco_bias = 6;
    adf_config.r3.pa_bias = 4;
    adf_config.r3.pll_test_mode = 0;
    adf_config.r3.sd_test_mode = 0;
}



void adf_reset(void) {
    GPIO_ResetBits(ADF_CE_PORT, ADF_CE);
    GPIO_SetBits(ADF_LE_PORT, ADF_LE);
    GPIO_SetBits(ADF_TXDATA_PORT, ADF_TXDATA);
    GPIO_SetBits(ADF_CLK_PORT, ADF_CLK);
    GPIO_SetBits(ADF_DATA_PORT, ADF_DATA);
    delay_ms(1);
    GPIO_SetBits(ADF_CE_PORT, ADF_CE);
    delay_ms(100);
}



void adf_turn_off(void) {
      GPIO_ResetBits(ADF_CE_PORT, ADF_CE);
}

void adf_turn_on(void) {
      //GPIO_ResetBits(GPIOB, GPIO_Pin_11);
  GPIO_SetBits(ADF_CE_PORT, ADF_CE);
}

uint32_t pokus_CtiBit(void) //pozn. vraci 0 nebo 1
{
  return GPIO_ReadInputDataBit(ADF_MUXOUT_PORT, ADF_MUXOUT);
}

// Configuration writing functions -------------------------------------

/**/
void adf_write_config(void) {
    adf_write_register_zero();
    adf_write_register_one();
```

```
    adf_write_register_two();
    adf_write_register_three();
}


void adf_write_register_zero(void) {
    u32 reg =
        (0) |
        ((adf_config.r0.frequency_error_correction  & 0x7FF) <<  2) |
        ((adf_config.r0.r_divider                   & 0xF  ) << 13) |
        ((adf_config.r0.crystal_doubler             & 0x1  ) << 17) |
        ((adf_config.r0.crystal_oscillator_disable  & 0x1  ) << 18) |
        ((adf_config.r0.clock_out_divider           & 0xF  ) << 19) |
        ((adf_config.r0.vco_adjust                  & 0x3  ) << 23) |
        ((adf_config.r0.output_divider              & 0x3  ) << 25);
    adf_write_register(reg);
}

void adf_write_register_one(void) {
    u32 reg =
        (1) |
        ((adf_config.r1.fractional_n                & 0xFFF) <<  2) |
        ((adf_config.r1.integer_n                   & 0xFF ) << 14) |
        ((adf_config.r1.prescaler                   & 0x1  ) << 22);
    adf_write_register(reg);
}

void adf_write_register_two(void) {
    u32 reg =
        (2) |
        ((adf_config.r2.mod_control                 & 0x3  ) <<  2) |
        ((adf_config.r2.gook                        & 0x1  ) <<  4) |
        ((adf_config.r2.power_amplifier_level       & 0x3F ) <<  5) |
        ((adf_config.r2.modulation_deviation        & 0x1FF) << 11) |
        ((adf_config.r2.gfsk_modulation_control     & 0x7  ) << 20) |
        ((adf_config.r2.index_counter               & 0x3  ) << 23);
    adf_write_register(reg);
}

void adf_write_register_three(void) {
    u32 reg =
        (3) |
        ((adf_config.r3.pll_enable                  & 0x1  ) <<  2) |
        ((adf_config.r3.pa_enable                   & 0x1  ) <<  3) |
        ((adf_config.r3.clkout_enable               & 0x1  ) <<  4) |
        ((adf_config.r3.data_invert                 & 0x1  ) <<  5) |
        ((adf_config.r3.charge_pump_current         & 0x3  ) <<  6) |
        ((adf_config.r3.bleed_up                    & 0x1  ) <<  8) |
        ((adf_config.r3.bleed_down                  & 0x1  ) <<  9) |
        ((adf_config.r3.vco_disable                 & 0x1  ) << 10) |
        ((adf_config.r3.muxout                      & 0xF  ) << 11) |
        ((adf_config.r3.ld_precision                & 0x1  ) << 15) |
        ((adf_config.r3.vco_bias                    & 0xF  ) << 16) |
        ((adf_config.r3.pa_bias                     & 0x7  ) << 20) |
        ((adf_config.r3.pll_test_mode               & 0x1F ) << 23) |
        ((adf_config.r3.sd_test_mode                & 0xF  ) << 28);
    adf_write_register(reg);
}
```

```
void adf_write_register(u32 data) {
    s8 i;

delay_ns(20);
    GPIO_ResetBits(ADF_CLK_PORT, ADF_CLK);
    delay_ns(20);

   GPIO_ResetBits(ADF_LE_PORT, ADF_LE);
    delay_ns(100);

    for(i=31; i>=0; i--) {
        if((data & (1<<i))>>i)
            GPIO_SetBits(ADF_DATA_PORT, ADF_DATA);
        else
            GPIO_ResetBits(ADF_DATA_PORT, ADF_DATA);
        delay_ns(20);

        GPIO_SetBits(ADF_CLK_PORT, ADF_CLK);
        delay_ns(100);

        GPIO_ResetBits(ADF_CLK_PORT, ADF_CLK);
        delay_ns(100);

    }

    delay_ns(20);

    GPIO_SetBits(ADF_LE_PORT, ADF_LE);
}



// Configuration setting functions ---------------------------------------


void adf_set_frequency_error_correction(u16 error) {
    adf_config.r0.frequency_error_correction = error;
}

void adf_set_r_divider(u8 r) {
    adf_config.r0.r_divider = r;
}

void adf_set_vco_adjust(u8 adjust) {
    adf_config.r0.vco_adjust = adjust;
}


/* nutno upravit */
void adf_set_frequency(u32 khz) {
    u32 f_pfd = 24576;
        u32 m;
    u8 n = (khz*10) / f_pfd;
   //  m = (u64)((u64)khz * 10000) / f_pfd;
    m -= n * 1000;
    m *= 4096;
```

```
    m /= 1000;
    adf_set_n(n);
    adf_set_m((u16)m);
         m = 2300; //f= 433,900 MHz
         n = 176;
}



void adf_set_n(u8 n) {
    adf_config.r1.integer_n = n;
}

void adf_set_m(u16 m) {
    adf_config.r1.fractional_n = m;
}

void adf_set_pa_level(u8 level) {
    adf_config.r2.power_amplifier_level = level;
}

void adf_set_pll_enable(u8 enable) {
    adf_config.r3.pll_enable = enable;
}

void adf_set_pa_enable(u8 enable) {
    adf_config.r3.pa_enable = enable;
}

void adf_set_clkout_enable(u8 enable) {
    adf_config.r3.clkout_enable = enable;
}

void adf_set_charge_pump_current(u8 current) {
    adf_config.r3.charge_pump_current = current;
}

void adf_set_vco_disable(u8 disable) {
    adf_config.r3.vco_disable = disable;
}

void adf_set_muxout(u8 muxout) {
    adf_config.r3.muxout = muxout;
}

void adf_set_vco_bias(u8 bias) {
    adf_config.r3.vco_bias = bias;
}

void adf_set_pa_bias(u8 bias) {
    adf_config.r3.pa_bias = bias;
}

u8 adf_get_n() {
    return adf_config.r1.integer_n;
}

u16 adf_get_m() {
    return adf_config.r1.fractional_n;
}
```

```
u8 adf_get_vco_adjust(void) {
    return adf_config.r0.vco_adjust;
}

u8 adf_get_vco_bias(void) {
    return adf_config.r3.vco_bias;
}


/*
toto je mustr
 adf_set_pin_output(ADF_TXDATA_PORT, ADF_TXDATA);
    adf_set_pin_input(ADF_MUXOUT_PORT, ADF_MUXOUT);
GPIO_ResetBits(ADF_CE_PORT, ADF_CE);
GPIO_SetBits(ADF_CE_PORT, ADF_CE);
*/

/**/

void adf_setup(void) {
    adf_reset_config();
    adf_set_r_divider(2);
    adf_set_frequency(434000);
    adf_set_muxout(ADF_MUXOUT_DIGITAL_LOCK);
    adf_set_pll_enable(ADF_ON);
    adf_write_config();
}



u8 adf_lock(void) {
    u8 adj = 0, bias = 5;
    while(!adf_locked()) {
        adf_set_vco_adjust(adj);
        adf_set_vco_bias(bias);
        adf_write_config();
        delay_ms(50);
        if(++bias == 14) {
            bias = 1;
            if(++adj == 4) {
                return 0;
            }
        }
    }
    return 1;
}

void adf_power_on(void) {
    adf_set_pa_enable(ADF_ON);
    adf_set_pa_level(60);
    adf_write_config();
}

void adf_power_off(void) {
    adf_set_pa_enable(ADF_OFF);
    adf_write_config();
}
```

```
// Data transmission functions
================================================
/**/
void adf_transmit_string(char* data, u32 baud) {
    u32 i;
    for(i=0; data[i]; i++)
        adf_transmit_byte(data[i], baud);
}



void adf_transmit_byte(u8 byte, u32 baud) {
    s8 i;
////////////////////////////////////////////////////////////
    //led_quickflash(LED4);

    // Start bit
    adf_transmit_bit(0);
    delay(baud);

    // Data byte
    for(i=0; i<8; i++) {
        if(byte & 1<<i)
            adf_transmit_bit(1);
        else
            adf_transmit_bit(0);
        delay(baud);
    }

    // Stop bits
    adf_transmit_bit(1);
    delay(baud);
    delay(baud);
}



void adf_transmit_bit(u8 bit) {
    if(bit) {
        GPIO_SetBits(ADF_TXDATA_PORT, ADF_TXDATA);
    } else {
        GPIO_ResetBits(ADF_TXDATA_PORT, ADF_TXDATA);
    }
}



// MUXOUT reading functions
================================================



u8 adf_read_muxout(void) {
    return (u8)(GPIO_ReadInputDataBit(ADF_MUXOUT_PORT, ADF_MUXOUT) > 0);
}


u8 adf_locked(void) {
    return adf_read_muxout();
```

```
}

u8 adf_reg_ready(void) {
    return adf read muxout();
}
```

```
/* STM32 VL Discovery Library
 *
 * ver 1.0
 * Vd  for CANSAT CZ 2012 lessons
 */

/* Includes ------------------------------------------------------------------
---*/

#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include "adf.h"
#include "delay.h"
void GPIO_Inicialization(void);

void Delay(__IO uint32_t nTick);

int main(void)
{


  GPIO_Inicialization();
  adf_peripheral_setup();

      /*

  STM32vldiscovery_LEDOff(LED4);     // LED4 - modra dioda
  STM32vldiscovery_LEDOff(LED3); // LED3 - zelena dioda pridal jsem
      Delay(0xAFFFFF);
      STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda
      STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
  Delay(0xAFFFFF);
  STM32vldiscovery_LEDOff(LED3); // LED3 - zelena    dioda
      STM32vldiscovery_LEDOff(LED4); // LED4 - modra dioda
      //Delay(0xAFFFFF);

*/
       adf_turn_on();
       adf_setup();
      delay_ns(20);
   if(!adf_lock())
      {

        //for(;;)
           // led_flash_outer()
             STM32vldiscovery_LEDOff(LED3);
                 }

                         //else
```

```
                STM32vldiscovery_LEDOn(LED3);
     Delay(0x00FFFF);
                adf_turn_on();
          adf setup();
     adf_power_on();
     //adf_transmit_string("\n\n*** WOMBAT INITIALISING ***\n\n",
ADF_300_BAUD);

  while (1)
  {

/*

   if(!adf_lock())
             {

        //for(;;)
           // led_flash_outer()
             STM32vldiscovery_LEDOff(LED3);
             STM32vldiscovery_LEDOn(LED4);
                 }
                        else
                             {
                 STM32vldiscovery_LEDOn(LED3);
      STM32vldiscovery_LEDOff(LED4);
                             }
     adf_power_on();
           Delay(0x00FFFF);
*/
    // adf_transmit_string("\n\n*** WOMBAT INITIALISING ***\n\n",
ADF_300_BAUD);
            //    Delay(0xAFFFFF);
            //adf_power_off();

    if(pokus_CtiBit() == 0)//kdyz se stiskne tlacitko, tak se provede
nasledujici:
    {
    adf_turn_on();
        adf_setup();
            delay_ns(20);
        adf_power_on();
        delay_ns(200);
     }
             else
            {
   // adf_turn_on();
        adf_setup();
            STM32vldiscovery_LEDOn(LED3);
            Delay(0xFFFFFF);
            STM32vldiscovery_LEDOff(LED3);
            Delay(0xFFFFFF);
            delay_ns(20);
      //  adf_power_on();
    delay_ns(200);


     }
      /*
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
```

```
*/

  }
}


void GPIO_Inicialization(void)
{

  STM32vldiscovery_LEDInit(LED3);
  STM32vldiscovery_LEDInit(LED4);
  STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
  STM32vldiscovery_LEDOff(LED3);
  STM32vldiscovery_LEDOff(LED4);
}


void Delay(__IO uint32_t nTick)
{
  for(; nTick != 0; nTick--);
}
```

4.1.3 AFSK AX25 vysílač ovládaný z BeRTOSu (Slovenský projekt)

Na http://mcu.cz/comment-n3003.html jsem se poprvé dozvěděl o tomto projektu. Autorem projektu je slovenský radioamatér Michal Demin - OM5AMX. Audio Frequency Shift Keying (AFSK) je obvyklý způsob datového vysílání rychlostí 1200 baud, který je využíván i ve vysílači PrattHobbies pro CANSATy, přičemž pro datový přenos používá AX25 paket radio. K tomuto provozu se často využívá nějaký TNC. A právě konstrukci TNC založenou na STM32VL Discovery kitu Michal Demin publikoval http://michaldemin.wordpress.com/2012/02/27/cheap-afsk-tnc/ . Zdrojový kód publikoval na https://github.com/robots/APRS Inspirativní jsou především soubory AX25.c , AX25.h , AFSK.c a AFSK.h. Můžeme je použít i v případě tvorby vlastního sw pro Cansat využívající modulaci AFSK a protokol AX25. Nejprve tedy AFSK:

```
/**
 * hlavickovy soubor afsk.h - je z BeRTOS
 * $WIZ$ module_name = "afsk"
 * $WIZ$ module_configuration = "bertos/cfg/cfg_afsk.h"
 * $WIZ$ module_depends = "timer", "kfile"
 * $WIZ$ module_hw = "bertos/hw/hw_afsk.h"
 */
#ifndef NET_AFSK_H
#define NET_AFSK_H
#include "cfg/cfg_afsk.h"
#include "hw/hw_afsk.h"
#include <cfg/compiler.h>
#include <io/kfile.h>
#include <struct/fifobuf.h>

/**
 * ADC sample rate.
 * The demodulator filters are designed to work at this frequency.
 * If you need to change this remember to update afsk_adc_isr().
 */
#define SAMPLERATE 9600
```

```
/**
 * Bitrate of the received/transmitted data.
 * The demodulator filters and decoderes are designed to work at this
frequency.
 * If you need to change this remember to update afsk_adc_isr().
 */
#define BITRATE    1200
#define SAMPLEPERBIT (SAMPLERATE / BITRATE)

/**
 * HDLC (High-Level Data Link Control) context.
 * Maybe to be moved in a separate HDLC module one day.
 */
typedef struct Hdlc
{
      uint8_t demod_bits; ///< Bitstream from the demodulator.
      uint8_t bit_idx;    ///< Current received bit.
      uint8_t currchar;   ///< Current received character.
      bool rxstart;       ///< True if an HDLC_FLAG char has been found in
the bitstream.
} Hdlc;

#define FIR_MAX_TAPS 16
typedef struct FIR
{
      int8_t taps;
      int8_t coef[FIR_MAX_TAPS];
      int16_t mem[FIR_MAX_TAPS];
} FIR;

/**
 * RX FIFO buffer full error.
 */
#define AFSK_RXFIFO_OVERRUN BV(0)

/**
 * AFSK1200 modem context.
 */
typedef struct Afsk
{
      /** Base "class" */
      KFile fd;

      /** ADC channel to be used by the demodulator */
      int adc_ch;

      /** DAC channel to be used by the modulator */
      int dac_ch;

      /** Current sample of bit for output data. */
      uint8_t sample_count;

      /** Current character to be modulated */
      uint8_t curr_out;

      /** Mask of current modulated bit */
      uint8_t tx_bit;

      /** True if bit stuff is allowed, false otherwise */
```

```
      bool bit_stuff;

      /** Counter for bit stuffing */
      uint8 t stuff cnt;
      /**
       * DDS phase accumulator for generating modulated data.
       */
      uint16_t phase_acc;

      /** Current phase increment for current modulated bit */
      uint16_t phase_inc;

      /** Delay line used to delay samples by (SAMPLEPERBIT / 2) */
      FIFOBuffer delay_fifo;

      /**
       * Buffer for delay FIFO.
       * The 1 is added because the FIFO macros need
       * 1 byte more to handle a buffer (SAMPLEPERBIT / 2) bytes long.
       */
      int8_t delay_buf[SAMPLEPERBIT / 2 + 1];

      /** FIFO for received data */
      FIFOBuffer rx_fifo;

      /** FIFO rx buffer */
      uint8_t rx_buf[CONFIG_AFSK_RX_BUFLEN];

      /** FIFO for transmitted data */
      FIFOBuffer tx_fifo;

      /** FIFO tx buffer */
      uint8_t tx_buf[CONFIG_AFSK_TX_BUFLEN];

      /** IIR filter X cells, used to filter sampled data by the
demodulator */
      int16_t iir_x[2];

      /** IIR filter Y cells, used to filter sampled data by the
demodulator */
      int16_t iir_y[2];

      /**
       * Bits sampled by the demodulator are here.
       * Since ADC samplerate is higher than the bitrate, the bits here are
       * SAMPLEPERBIT times the bitrate.
       */
      uint8_t sampled_bits;

      bool cd;
      uint8_t cd_state;

      /**
       * Current phase, needed to know when the bitstream at ADC speed
       * should be sampled.
       */
      int8_t curr_phase;

      /** Bits found by the demodulator at the correct bitrate speed. */
      uint8_t found_bits;
```

```
      /** True while modem sends data */
      volatile bool sending;

      /**
       * AFSK modem status.
       * If 0 all is ok, otherwise errors are present.
       */
      volatile int status;

      /** Hdlc context */
      Hdlc hdlc;

      /**
       * Preamble length.
       * When the AFSK modem wants to send data, before sending the actual
data,
       * shifts out preamble_len HDLC_FLAG characters.
       * This helps to synchronize the demodulator filters on the receiver
side.
       */
      uint16_t preamble_len;

      /**
       * Trailer length.
       * After sending the actual data, the AFSK shifts out
       * trailer_len HDLC_FLAG characters.
       * This helps to synchronize the demodulator filters on the receiver
side.
       */
      uint16_t trailer_len;
} Afsk;

#define KFT_AFSK MAKE_ID('A', 'F', 'S', 'K')

INLINE Afsk *AFSK_CAST(KFile *fd)
{
  ASSERT(fd->_type == KFT_AFSK);
  return (Afsk *)fd;
}


void afsk_adc_isr(Afsk *af, int8_t sample);
uint8_t afsk_dac_isr(Afsk *af);
void afsk_init(Afsk *af, int adc_ch, int dac_ch);

/**
 * \name Afsk filter types.
 * $WIZ$ afsk_filter_list = "AFSK_BUTTERWORTH", "AFSK_CHEBYSHEV"
 * \{
 */
#define AFSK_BUTTERWORTH  0
#define AFSK_CHEBYSHEV    1
#define AFSK_FIR          2
/* \} */

int afsk_testSetup(void);
int afsk_testRun(void);
int afsk_testTearDown(void);
```

```
#endif /* NET_AFSK_H */
```

Definice funkcí AFSK, všimněme si tabulky pro generování sinusového průběhu výstupního signálu z sw vytvořeného TNC.

```
/**
 * soubor afsk.c
 */

#include "afsk.h"
#include <net/ax25.h>
#include "cfg/cfg_afsk.h"
#include "hw/hw_afsk.h"
#include <drv/timer.h>
#include <cfg/module.h>
#define LOG_LEVEL   AFSK_LOG_LEVEL
#define LOG_FORMAT  AFSK_LOG_FORMAT
#include <cfg/log.h>
#include <cpu/power.h>
#include <cpu/pgm.h>
#include <struct/fifobuf.h>
#include <string.h> /* memset */
#define PHASE_BIT    8
#define PHASE_INC    1
#define PHASE_MAX    (SAMPLEPERBIT * PHASE_BIT)
#define PHASE_THRES  (PHASE_MAX / 2) // - PHASE_BIT / 2)

// Modulator constants
#define MARK_FREQ  1200
#define MARK_INC   (uint16_t)(DIV_ROUND(SIN_LEN * (uint32_t)MARK_FREQ,
CONFIG_AFSK_DAC_SAMPLERATE))

#define SPACE_FREQ 2200
#define SPACE_INC  (uint16_t)(DIV_ROUND(SIN_LEN * (uint32_t)SPACE_FREQ,
CONFIG_AFSK_DAC_SAMPLERATE))

//Ensure sample rate is a multiple of bit rate
STATIC_ASSERT(!(CONFIG_AFSK_DAC_SAMPLERATE % BITRATE));

#define DAC_SAMPLEPERBIT (CONFIG_AFSK_DAC_SAMPLERATE / BITRATE)

/**
 * Sine table for the first quarter of wave.
 * The rest of the wave is computed from this first quarter.
 * This table is used to generate the modulated data.
 */
static const uint8_t PROGMEM sin_table[] =
{
      128, 129, 131, 132, 134, 135, 137, 138, 140, 142, 143, 145, 146, 148,
149, 151,
      152, 154, 155, 157, 158, 160, 162, 163, 165, 166, 167, 169, 170, 172,
173, 175,
      176, 178, 179, 181, 182, 183, 185, 186, 188, 189, 190, 192, 193, 194,
196, 197,
      198, 200, 201, 202, 203, 205, 206, 207, 208, 210, 211, 212, 213, 214,
215, 217,
```

```
      218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,
232, 233,
      234, 234, 235, 236, 237, 238, 238, 239, 240, 241, 241, 242, 243, 243,
244, 245,
      245, 246, 246, 247, 248, 248, 249, 249, 250, 250, 250, 251, 251, 252,
252, 252,
      253, 253, 253, 253, 254, 254, 254, 254, 254, 255, 255, 255, 255, 255,
255, 255,
};

#define SIN_LEN 512 ///< Full wave length
STATIC_ASSERT(sizeof(sin_table) == SIN_LEN / 4);

#if (CONFIG_AFSK_FILTER == AFSK_FIR)
enum fir_filters
{
      FIR_1200_BP=0,
      FIR_2200_BP=1,
      FIR_1200_LP=2
};

static FIR fir_table[] =
{
      [FIR_1200_BP] = {
            .taps = 11,
            .coef = {
                  -12, -16, -15, 0, 20, 29, 20, 0, -15, -16, -12
            },
            .mem = {
                  0,
            },
      },
      [FIR_2200_BP] = {
            .taps = 11,
            .coef = {
                  11, 15, -8, -26, 4, 30, 4, -26, -8, 15, 11
            },
            .mem = {
                  0,
            },
      },
      [FIR_1200_LP] = {
            .taps = 8,
            .coef = {
                  -9, 3, 26, 47, 47, 26, 3, -9
            },
            .mem = {
                  0,
            },
      },
};
#endif

/**
 * Given the index, this function computes the correct sine sample
 * based only on the first quarter of wave.
 */
INLINE uint8_t sin_sample(uint16_t idx)
{
      ASSERT(idx < SIN_LEN);
```

```
	uint16_t new_idx = idx % (SIN_LEN / 2);
	new_idx = (new_idx >= (SIN_LEN / 4)) ? (SIN_LEN / 2 - new_idx - 1) :
new_idx;

	uint8_t data = pgm_read8(&sin_table[new_idx]);

	return (idx >= (SIN_LEN / 2)) ? (255 - data) : data;
}

#if (CONFIG_AFSK_FILTER == AFSK_FIR)
static int8_t fir_filter(int8_t s, enum fir_filters f)
{
	int8_t Q = fir_table[f].taps - 1;
	int8_t *B = fir_table[f].coef;
	int16_t *Bmem = fir_table[f].mem;

	int8_t i;
	int16_t y;

	Bmem[0] = s;
	y = 0;

	for (i = Q; i >= 0; i--)
	{
		y += Bmem[i] * B[i];
		Bmem[i + 1] = Bmem[i];
	}

	return (int8_t) (y / 128);
}
#endif

#define BIT_DIFFER(bitline1, bitline2) (((bitline1) ^ (bitline2)) & 0x01)
#define EDGE_FOUND(bitline)            BIT_DIFFER((bitline), (bitline) >>
1)

/**
 * High-Level Data Link Control parsing function.
 * Parse bitstream in order to find characters.
 *
 * \param hdlc HDLC context.
 * \param bit  current bit to be parsed.
 * \param fifo FIFO buffer used to push characters.
 *
 * \return true if all is ok, false if the fifo is full.
 */
static bool hdlc_parse(Hdlc *hdlc, bool bit, FIFOBuffer *fifo)
{
	bool ret = true;

	hdlc->demod_bits <<= 1;
	hdlc->demod_bits |= bit ? 1 : 0;

	/* HDLC Flag */
	if (hdlc->demod_bits == HDLC_FLAG)
	{
		if (!fifo_isfull(fifo))
		{
			fifo_push(fifo, HDLC_FLAG);
			hdlc->rxstart = true;
```

```
            }
            else
            {
                  ret = false;
                  hdlc->rxstart = false;
            }

            hdlc->currchar = 0;
            hdlc->bit_idx = 0;
            return ret;
      }

      /* Reset */
      if ((hdlc->demod_bits & HDLC_RESET) == HDLC_RESET)
      {
            hdlc->rxstart = false;
            return ret;
      }

      if (!hdlc->rxstart)
            return ret;

      /* Stuffed bit */
      if ((hdlc->demod_bits & 0x3f) == 0x3e)
            return ret;

      if (hdlc->demod_bits & 0x01)
            hdlc->currchar |= 0x80;

      if (++hdlc->bit_idx >= 8)
      {
            if ((hdlc->currchar == HDLC_FLAG
                  || hdlc->currchar == HDLC_RESET
                  || hdlc->currchar == AX25_ESC))
            {
                  if (!fifo_isfull(fifo))
                        fifo_push(fifo, AX25_ESC);
                  else
                  {
                        hdlc->rxstart = false;
                        ret = false;
                  }
            }

            if (!fifo_isfull(fifo))
                  fifo_push(fifo, hdlc->currchar);
            else
            {
                  hdlc->rxstart = false;
                  ret = false;
            }

            hdlc->currchar = 0;
            hdlc->bit_idx = 0;
      }
      else
            hdlc->currchar >>= 1;

      return ret;
}
```

```
/**
 * ADC ISR callback.
 * This function has to be called by the ADC ISR when a sample of the
configured
 * channel is available.
 * \param af Afsk context to operate on.
 * \param curr_sample current sample from the ADC.
 */
void afsk_adc_isr(Afsk *af, int8_t curr_sample)
{
      AFSK_STROBE_ON();

      /*
       * Frequency discriminator and LP IIR filter.
       * This filter is designed to work
       * at the given sample rate and bit rate.
       */
      STATIC_ASSERT(SAMPLERATE == 9600);
      STATIC_ASSERT(BITRATE == 1200);

#if (CONFIG_AFSK_FILTER != AFSK_FIR)

      /*
       * Frequency discrimination is achieved by simply multiplying
       * the sample with a delayed sample of (samples per bit) / 2.
       * Then the signal is lowpass filtered with a first order,
       * 600 Hz filter. The filter implementation is selectable
       * through the CONFIG_AFSK_FILTER config variable.
       */
      af->iir_x[0] = af->iir_x[1];

      #if (CONFIG_AFSK_FILTER == AFSK_BUTTERWORTH)
            af->iir_x[1] = ((int8_t)fifo_pop(&af->delay_fifo) *
curr_sample) >> 2;
            //af->iir_x[1] = ((int8_t)fifo_pop(&af->delay_fifo) *
curr_sample) / 6.027339492;
      #elif (CONFIG_AFSK_FILTER == AFSK_CHEBYSHEV)
            af->iir_x[1] = ((int8_t)fifo_pop(&af->delay_fifo) *
curr_sample) >> 2;
            //af->iir_x[1] = ((int8_t)fifo_pop(&af->delay_fifo) *
curr_sample) / 3.558147322;
      #else
            #error Filter type not found!
      #endif

      af->iir_y[0] = af->iir_y[1];

      #if CONFIG_AFSK_FILTER == AFSK_BUTTERWORTH
            /*
             * This strange sum + shift is an optimization for af->iir_y[0]
* 0.668.
             * iir * 0.668 ~= (iir * 21) / 32 =
             * = (iir * 16) / 32 + (iir * 4) / 32 + iir / 32 =
             * = iir / 2 + iir / 8 + iir / 32 =
             * = iir >> 1 + iir >> 3 + iir >> 5
             */
            af->iir_y[1] = af->iir_x[0] + af->iir_x[1] + (af->iir_y[0] >>
1) + (af->iir_y[0] >> 3) + (af->iir_y[0] >> 5);
```

```
            //af->iir_y[1] = af->iir_x[0] + af->iir_x[1] + af->iir_y[0] *
0.6681786379;
      #elif CONFIG_AFSK_FILTER == AFSK_CHEBYSHEV
            /*
             * This should be (af->iir_y[0] * 0.438) but
             * (af->iir_y[0] >> 1) is a faster approximation :-)
             */
            af->iir_y[1] = af->iir_x[0] + af->iir_x[1] + (af->iir_y[0] >>
1);
            //af->iir_y[1] = af->iir_x[0] + af->iir_x[1] + af->iir_y[0] *
0.4379097269;
      #endif

      /* Save this sampled bit in a delay line */
      af->sampled_bits <<= 1;
      af->sampled_bits |= (af->iir_y[1] > 0) ? 1 : 0;

      if (ABS(af->iir_y[1]) - 20 > 0) {
            af->cd_state++;
            if (af->cd_state > 30) {
                  af->cd_state = 30;
                  af->cd = true;
            }
      } else {
            if (af->cd_state > 0) {
                  af->cd_state --;

                  if (af->cd_state == 0) {
                        af->cd = false;
                  }
            }
      }

      /* Store current ADC sample in the af->delay_fifo */
      fifo_push(&af->delay_fifo, curr_sample);

#elif (CONFIG_AFSK_FILTER == AFSK_FIR)

#define DCD_LEVEL 5

      af->iir_y[0] = ABS(fir_filter(curr_sample, FIR_1200_BP));
      af->iir_y[1] = ABS(fir_filter(curr_sample, FIR_2200_BP));

      af->sampled_bits <<= 1;
      af->sampled_bits |= fir_filter(af->iir_y[1] - af->iir_y[0],
FIR_1200_LP) > 0;

      if (af->iir_y[1] > DCD_LEVEL || af->iir_y[0] > DCD_LEVEL) {
            af->cd_state++;
            if (af->cd_state > 30) {
                  af->cd_state = 30;
                  af->cd = true;
            }
      } else {
            if (af->cd_state > 0) {
                  af->cd_state --;

                  if (af->cd_state == 0) {
                        af->cd = false;
                  }
```

```
            }
      }

#endif

//kprintf("%+03d %+03d %+03d %d\n", curr_sample, af->iir_x[1], af-
>iir_y[1], (af->cd)?1:0);


      /* If there is an edge, adjust phase sampling */
      if (EDGE_FOUND(af->sampled_bits))
      {
            if (af->curr_phase < PHASE_THRES)
                  af->curr_phase += PHASE_INC;
            else
                  af->curr_phase -= PHASE_INC;
      }
      af->curr_phase += PHASE_BIT;

      /* sample the bit */
      if (af->curr_phase >= PHASE_MAX)
      {
            af->curr_phase %= PHASE_MAX;

            /* Shift 1 position in the shift register of the found bits */
            af->found_bits <<= 1;

            /*
             * Determine bit value by reading the last 3 sampled bits.
             * If the number of ones is two or greater, the bit value is a
1,
             * otherwise is a 0.
             * This algorithm presumes that there are 8 samples per bit.
             */
            STATIC_ASSERT(SAMPLEPERBIT == 8);
            uint8_t bits = af->sampled_bits & 0x07;
            if (bits == 0x07 // 111, 3 bits set to 1
             || bits == 0x06 // 110, 2 bits
             || bits == 0x05 // 101, 2 bits
             || bits == 0x03 // 011, 2 bits
            )
                  af->found_bits |= 1;

            /*
             * NRZI coding: if 2 consecutive bits have the same value
             * a 1 is received, otherwise it's a 0.
             */
            if (!hdlc_parse(&af->hdlc, !EDGE_FOUND(af->found_bits), &af-
>rx_fifo))
                  af->status |= AFSK_RXFIFO_OVERRUN;
      }

      AFSK_STROBE_OFF();
}

static void afsk_txStart(Afsk *af)
{
      if (!af->sending)
      {
            af->phase_inc = MARK_INC;
```

```
            af->phase_acc = 0;
            af->stuff_cnt = 0;
            af->sending = true;
            af->preamble_len = DIV_ROUND(CONFIG_AFSK_PREAMBLE_LEN *
BITRATE, 8000);
            AFSK_DAC_IRQ_START(af->dac_ch);
      }
      ATOMIC(af->trailer_len  = DIV_ROUND(CONFIG_AFSK_TRAILER_LEN  *
BITRATE, 8000));
}

#define BIT_STUFF_LEN 5

#define SWITCH_TONE(inc)  (((inc) == MARK_INC) ? SPACE_INC : MARK_INC)

/**
 * DAC ISR callback.
 * This function has to be called by the DAC ISR when a sample of the
configured
 * channel has been converted out.
 *
 * \param af Afsk context to operate on.
 *
 * \return The next DAC output sample.
 */
uint8_t afsk_dac_isr(Afsk *af)
{
      AFSK_STROBE_ON();

      /* Check if we are at a start of a sample cycle */
      if (af->sample_count == 0)
      {
            if (af->tx_bit == 0)
            {
                  /* We have just finished transimitting a char, get a new
one. */
                  if (fifo_isempty(&af->tx_fifo) && af->trailer_len == 0)
                  {
                        AFSK_DAC_IRQ_STOP(af->dac_ch);
                        af->sending = false;
                        AFSK_STROBE_OFF();
                        return 0;
                  }
                  else
                  {
                        /*
                         * If we have just finished sending an unstuffed
byte,
                         * reset bitstuff counter.
                         */
                        if (!af->bit_stuff)
                              af->stuff_cnt = 0;

                        af->bit_stuff = true;

                        /*
                         * Handle preamble and trailer
                         */
                        if (af->preamble_len == 0)
                        {
```

```
                              if (fifo_isempty(&af->tx_fifo))
                              {
                                    af->trailer_len--;
                                    af->curr out = HDLC FLAG;
                              }
                              else
                                    af->curr_out = fifo_pop(&af->tx_fifo);
                        }
                        else
                        {
                              af->preamble_len--;
                              af->curr_out = HDLC_FLAG;
                        }

                        /* Handle char escape */
                        if (af->curr_out == AX25_ESC)
                        {
                              if (fifo_isempty(&af->tx_fifo))
                              {
                                    AFSK_DAC_IRQ_STOP(af->dac_ch);
                                    af->sending = false;
                                    AFSK_STROBE_OFF();
                                    return 0;
                              }
                              else
                                    af->curr_out = fifo_pop(&af->tx_fifo);
                        }
                        else if (af->curr_out == HDLC_FLAG || af->curr_out
== HDLC_RESET)
                              /* If these chars are not escaped disable bit
stuffing */
                              af->bit_stuff = false;
                  }
                  /* Start with LSB mask */
                  af->tx_bit = 0x01;
            }

            /* check for bit stuffing */
            if (af->bit_stuff && af->stuff_cnt >= BIT_STUFF_LEN)
            {
                  /* If there are more than 5 ones in a row insert a 0 */
                  af->stuff_cnt = 0;
                  /* switch tone */
                  af->phase_inc = SWITCH_TONE(af->phase_inc);
            }
            else
            {
                  /*
                   * NRZI: if we want to transmit a 1 the modulated
frequency will stay
                   * unchanged; with a 0, there will be a change in the
tone.
                   */
                  if (af->curr_out & af->tx_bit)
                  {
                        /*
                         * Transmit a 1:
                         * - Stay on the previous tone
                         * - Increase bit stuff counter
                         */
```

```
                        af->stuff_cnt++;
                }
                else
                {
                        /*
                         * Transmit a 0:
                         * - Reset bit stuff counter
                         * - Switch tone
                         */
                        af->stuff_cnt = 0;
                        af->phase_inc = SWITCH_TONE(af->phase_inc);
                }

                /* Go to the next bit */
                af->tx_bit <<= 1;
            }
            af->sample_count = DAC_SAMPLEPERBIT;
        }

        /* Get new sample and put it out on the DAC */
        af->phase_acc += af->phase_inc;
        af->phase_acc %= SIN_LEN;

        af->sample_count--;
        AFSK_STROBE_OFF();
        return sin_sample(af->phase_acc);
}


static size_t afsk_read(KFile *fd, void *_buf, size_t size)
{
        Afsk *af = AFSK_CAST(fd);
        uint8_t *buf = (uint8_t *)_buf;

        #if CONFIG_AFSK_RXTIMEOUT == 0
        while (size-- && !fifo_isempty_locked(&af->rx_fifo))
        #else
        while (size--)
        #endif
        {
            #if CONFIG_AFSK_RXTIMEOUT != -1
            ticks_t start = timer_clock();
            #endif

            while (fifo_isempty_locked(&af->rx_fifo))
            {
                cpu_relax();
                #if CONFIG_AFSK_RXTIMEOUT != -1
                if (timer_clock() - start >
ms_to_ticks(CONFIG_AFSK_RXTIMEOUT))
                    return buf - (uint8_t *)_buf;
                #endif
            }

            *buf++ = fifo_pop_locked(&af->rx_fifo);
        }

        return buf - (uint8_t *)_buf;
}
```

```
static size_t afsk_write(KFile *fd, const void *_buf, size_t size)
{
	Afsk *af = AFSK_CAST(fd);
	const uint8 t *buf = (const uint8 t *) buf;

	while (size--)
	{
		while (fifo_isfull_locked(&af->tx_fifo))
			cpu_relax();

		fifo_push_locked(&af->tx_fifo, *buf++);
		afsk_txStart(af);
	}

	return buf - (const uint8_t *)_buf;
}

static int afsk_flush(KFile *fd)
{
	Afsk *af = AFSK_CAST(fd);
	while (af->sending)
		cpu_relax();
	return 0;
}

static int afsk_error(KFile *fd)
{
	Afsk *af = AFSK_CAST(fd);
	int err;

	ATOMIC(err = af->status);
	return err;
}

static void afsk_clearerr(KFile *fd)
{
	Afsk *af = AFSK_CAST(fd);
	ATOMIC(af->status = 0);
}


/**
 * Initialize an AFSK1200 modem.
 * \param af Afsk context to operate on.
 * \param adc_ch  ADC channel used by the demodulator.
 * \param dac_ch  DAC channel used by the modulator.
 */
void afsk_init(Afsk *af, int adc_ch, int dac_ch)
{
	#if CONFIG_AFSK_RXTIMEOUT != -1
	MOD_CHECK(timer);
	#endif
	memset(af, 0, sizeof(*af));
	af->adc_ch = adc_ch;
	af->dac_ch = dac_ch;

	fifo_init(&af->delay_fifo, (uint8_t *)af->delay_buf, sizeof(af-
>delay_buf));
	fifo_init(&af->rx_fifo, af->rx_buf, sizeof(af->rx_buf));
```

```
	/* Fill sample FIFO with 0 */
	for (int i = 0; i < SAMPLEPERBIT / 2; i++)
		fifo_push(&af->delay_fifo, 0);

	fifo_init(&af->tx_fifo, af->tx_buf, sizeof(af->tx_buf));

	AFSK_ADC_INIT(adc_ch, af);
	AFSK_DAC_INIT(dac_ch, af);
	AFSK_STROBE_INIT();
	LOG_INFO("MARK_INC %d, SPACE_INC %d\n", MARK_INC, SPACE_INC);

	DB(af->fd._type = KFT_AFSK);
	af->fd.write = afsk_write;
	af->fd.read = afsk_read;
	af->fd.flush = afsk_flush;
	af->fd.error = afsk_error;
	af->fd.clearerr = afsk_clearerr;
	af->phase_inc = MARK_INC;
}
```

Pro vytvoření paketů AX25 slouží:

```
/**
 * soubor ax25.h z BeRTOS
 *
 * $WIZ$ module_name = "ax25"
 * $WIZ$ module_configuration = "bertos/cfg/cfg_ax25.h"
 * $WIZ$ module_depends = "kfile", "crc-ccitt"
 */

#ifndef NET_AX25_H
#define NET_AX25_H
#include "cfg/cfg_ax25.h"
#include <cfg/compiler.h>
#include <io/kfile.h>

/**
 * Maximum size of a AX25 frame.
 */
#define AX25_MIN_FRAME_LEN 18

/**
 * CRC computation on correct AX25 packets should
 * give this result (don't ask why).
 */
#define AX25_CRC_CORRECT 0xF0B8

struct AX25Msg; // fwd declaration

/**
 * Type for AX25 messages callback.
 */
typedef void (*ax25_callback_t)(struct AX25Msg *msg);
```

```
/**
 * AX25 Protocol context.
 */
typedef struct AX25Ctx
{
      uint8_t buf[CONFIG_AX25_FRAME_BUF_LEN]; ///< buffer for received
chars
      KFile *ch;         ///< KFile used to access the physical medium
      size_t frm_len;    ///< received frame length.
      uint16_t crc_in;   ///< CRC for current received frame
      uint16_t crc_out; ///< CRC of current sent frame
      ax25_callback_t hook; ///< Hook function to be called when a message
is received
      bool raw;
      bool sync;    ///< True if we have received a HDLC flag.
      bool escape; ///< True when we have to escape the following char.
      uint8_t dcd_state;
      bool dcd;
} AX25Ctx;


/**
 * AX25 Call sign.
 */
typedef struct AX25Call
{
      char call[6]; ///< Call string, max 6 character
      uint8_t ssid; ///< SSID (secondary station ID) for the call
} AX25Call;

/**
 * Create an AX25Call structure on the fly.
 * \param str callsign, can be 6 characters or shorter.
 * \param id  ssid associated with the callsign.
 */
#define AX25_CALL(str, id) {.call = (str), .ssid = (id) }

/**
 * Maximum number of Repeaters in a AX25 message.
 */
#define AX25_MAX_RPT 8

/*
 * Has to be lesser than 8 in order to fit in one byte
 * change AX25Msg.rpt_flags if you need more repeaters.
 */
STATIC_ASSERT(AX25_MAX_RPT <= 8);

/**
 * AX25 Message.
 * Used to handle AX25 sent/received messages.
 */
typedef struct AX25Msg
{
      AX25Call src;  ///< Source adress
      AX25Call dst;  ///< Destination address
      #if CONFIG_AX25_RPT_LST
      AX25Call rpt_lst[AX25_MAX_RPT]; ///< List of repeaters
      uint8_t rpt_cnt; ///< Number of repeaters in this message
```

```
      uint8_t rpt_flags; ///< Has-been-repeated flags for each repeater
(bit-mapped)
      #define AX25_REPEATED(msg, idx) ((msg)->rpt_flags & BV(idx))
      #endif
      uint16_t ctrl; ///< AX25 control field
      uint8_t pid;   ///< AX25 PID field
      const uint8_t *info; ///< Pointer to the info field (payload) of the
message
      size_t len;    ///< Payload length
} AX25Msg;


#define AX25_CTRL_UI      0x03
#define AX25_PID_NOLAYER3 0xF0

/**
 * \name HDLC flags.
 * These should be moved in
 * a separated HDLC related file one day...
 * \{
 */
#define HDLC_FLAG  0x7E
#define HDLC_RESET 0x7F
#define AX25_ESC   0x1B
/* \} */


/**
 * Declare an AX25 path.
 * \param dst the destination callsign for the path, \see AX25_CALL
 *        for a handy way to create a callsign on the fly.
 * \param src the source callsign for the path, \see AX25_CALL
 *        for a handy way to create a callsign on the fly.
 *
 * Additional optional callsigns can be specified at the end of this macro
 * in order to add repeater callsigns or specific unproto paths.
 *
 * This macro can be used to simply path array declaration.
 * Should be used in this way:
 * \code
 * AX25Call path[] = AX25_PATH(AX25_CALL("abcdef", 0), AX25_CALL("ghjklm",
0), AX25_CALL("wide1", 1), AX25_CALL("wide2", 2));
 * \endcode
 *
 * The declared path can then be passed to ax25_sendVia().
 */
#define AX25_PATH(dst, src, ...) { dst, src, ## __VA_ARGS__ }

void ax25_poll(AX25Ctx *ctx);
void ax25_sendVia(AX25Ctx *ctx, const AX25Call *path, size_t path_len,
const void *_buf, size_t len);
void ax25_sendRaw(AX25Ctx *ctx, const void *_buf, size_t len);
void ax25_putchar(AX25Ctx *ctx, uint8_t c);

/**
 * Send an AX25 frame on the channel.
 * \param ctx AX25 context to operate on.
 * \param dst the destination callsign for the frame, \see AX25_CALL
 *        for a handy way to create a callsign on the fly.
 * \param src the source callsign for the frame, \see AX25_CALL
```

```
 *        for a handy way to create a callsign on the fly.
 * \param buf payload buffer.
 * \param len length of the payload.
 *
 * \see ax25_sendVia() if you want to send a frame with a specific path.
 */
#define ax25_send(ctx, dst, src, buf, len) ax25_sendVia(ctx, ({static
AX25Call __path[]={dst, src}; __path;}), 2, buf, len)
void ax25_init(AX25Ctx *ctx, KFile *channel, bool raw, ax25_callback_t
hook);

void ax25_print(KFile *ch, const AX25Msg *msg);
int ax25_testSetup(void);
int ax25_testTearDown(void);
int ax25_testRun(void);

#endif /* NET_AX25_H */
```

A definice funkci AX25

```
/**
 * soubor ax25.c z BeRTOS
 *
 */

#include "ax25.h"
#include "cfg/cfg_ax25.h"
#include <algo/crc_ccitt.h>
#define LOG_LEVEL  AX25_LOG_LEVEL
#define LOG_FORMAT AX25_LOG_FORMAT
#include <cfg/log.h>
#include <string.h> //memset, memcmp
#include <ctype.h>  //isalnum, toupper

#if CONFIG_AX25_RPT_LST
      #define AX25_SET_REPEATED(msg, idx, val) \
            do \
            { \
                  if (val) \
                        (msg)->rpt_flags |= BV(idx) ; \
                  else \
                        (msg)->rpt_flags &= ~BV(idx) ; \
            } while(0)
#endif

#define DECODE_CALL(buf, addr) \
      for (unsigned i = 0; i < sizeof((addr)); i++) \
      { \
            char c = (*(buf)++ >> 1); \
            (addr)[i] = (c == ' ') ? '\x0' : c; \
      }

static void ax25_decode(AX25Ctx *ctx)
{
      AX25Msg msg;
      uint8_t *buf = ctx->buf;
```

```
	DECODE_CALL(buf, msg.dst.call);
	msg.dst.ssid = (*buf++ >> 1) & 0x0F;

	DECODE_CALL(buf, msg.src.call);
	msg.src.ssid = (*buf >> 1) & 0x0F;

	LOG_INFO("SRC[%.6s-%d], DST[%.6s-%d]\n", msg.src.call, msg.src.ssid,
msg.dst.call, msg.dst.ssid);

	/* Repeater addresses */
	#if CONFIG_AX25_RPT_LST
		for (msg.rpt_cnt = 0; !(*buf++ & 0x01) && (msg.rpt_cnt <
countof(msg.rpt_lst)); msg.rpt_cnt++)
		{
			DECODE_CALL(buf, msg.rpt_lst[msg.rpt_cnt].call);
			msg.rpt_lst[msg.rpt_cnt].ssid = (*buf >> 1) & 0x0F;
			AX25_SET_REPEATED(&msg, msg.rpt_cnt, (*buf & 0x80));

			LOG_INFO("RPT%d[%.6s-%d]%c\n", msg.rpt_cnt,
				msg.rpt_lst[msg.rpt_cnt].call,
				msg.rpt_lst[msg.rpt_cnt].ssid,
				(AX25_REPEATED(&msg, msg.rpt_cnt) ? '*' : ' '));
		}
	#else
		while (!(*buf++ & 0x01))
		{
			char rpt[6];
			uint8_t ssid;
			DECODE_CALL(buf, rpt);
			ssid = (*buf >> 1) & 0x0F;
			LOG_INFO("RPT[%.6s-%d]\n", rpt, ssid);
		}
	#endif

	msg.ctrl = *buf++;
	if (msg.ctrl != AX25_CTRL_UI)
	{
		LOG_WARN("Only UI frames are handled, got [%02X]\n", msg.ctrl);
		return;
	}

	msg.pid = *buf++;
	if (msg.pid != AX25_PID_NOLAYER3)
	{
		LOG_WARN("Only frames without layer3 protocol are handled, got
[%02X]\n", msg.pid);
		return;
	}

	msg.len = ctx->frm_len - 2 - (buf - ctx->buf);
	msg.info = buf;
	LOG_INFO("DATA: %.*s\n", msg.len, msg.info);

	if (ctx->hook)
		ctx->hook(&msg);
}

/**
 * Check if there are any AX25 messages to be processed.
```

```
 * This function read available characters from the medium and search for
 * any AX25 messages.
 * If a message is found it is decoded and the linked callback executed.
 * This function may be blocking if there are no available chars and the
KFile
 * used in \a ctx to access the medium is configured in blocking mode.
 *
 * \param ctx AX25 context to operate on.
 */
void ax25_poll(AX25Ctx *ctx)
{
      int c;

      while ((c = kfile_getc(ctx->ch)) != EOF)
      {
            if (!ctx->escape && c == HDLC_FLAG)
            {
                  if (ctx->frm_len >= AX25_MIN_FRAME_LEN)
                  {
                        if (ctx->crc_in == AX25_CRC_CORRECT)
                        {
                              LOG_INFO("Frame found!\n");
                              if (ctx->raw) {
                                    if (ctx->hook) {
                                          //TODO: make MSG union and pass to
hook
                                          ctx->hook(NULL);
                                    }
                              } else {
                                    ax25_decode(ctx);
                              }
                        }
                        else
                        {
                              LOG_INFO("CRC error, computed [%04X]\n", ctx-
>crc_in);
                        }
                  }
                  ctx->sync = true;
                  ctx->crc_in = CRC_CCITT_INIT_VAL;
                  ctx->frm_len = 0;

                  ctx->dcd_state = 0;
                  ctx->dcd = false;
                  continue;
            }

            if (!ctx->escape && c == HDLC_RESET)
            {
                  LOG_INFO("HDLC reset\n");
                  ctx->sync = false;
                  ctx->dcd = false;
                  continue;
            }

            if (!ctx->escape && c == AX25_ESC)
            {
                  ctx->escape = true;
                  continue;
            }
```

```
            if (ctx->sync)
            {
                  if (ctx->frm len < CONFIG AX25 FRAME BUF LEN)
                  {
                        ctx->buf[ctx->frm_len++] = c;
                        ctx->crc_in = updcrc_ccitt(c, ctx->crc_in);

                        if (ctx->dcd_state == 1 && c == AX25_PID_NOLAYER3)
{
                              ctx->dcd_state ++;
                              ctx->dcd = true;
                        }

                        if (ctx->dcd_state == 0 && c == AX25_CTRL_UI) {
                              ctx->dcd_state ++;
                        }
                  }
                  else
                  {
                        LOG_INFO("Buffer overrun");
                        ctx->sync = false;
                        ctx->dcd = false;
                  }
            }
            ctx->escape = false;
      }

      if (kfile_error(ctx->ch))
      {
            LOG_ERR("Channel error [%04x]\n", kfile_error(ctx->ch));
            kfile_clearerr(ctx->ch);
            ctx->dcd = false;
      }
}

void ax25_putchar(AX25Ctx *ctx, uint8_t c)
{
      if (c == HDLC_FLAG || c == HDLC_RESET
            || c == AX25_ESC)
            kfile_putc(AX25_ESC, ctx->ch);
      ctx->crc_out = updcrc_ccitt(c, ctx->crc_out);
      kfile_putc(c, ctx->ch);
}

static void ax25_sendCall(AX25Ctx *ctx, const AX25Call *addr, bool last)
{
      unsigned len = MIN(sizeof(addr->call), strlen(addr->call));

      for (unsigned i = 0; i < len; i++)
      {
            uint8_t c = addr->call[i];
            ASSERT(isalnum(c) || c == ' ');
            c = toupper(c);
            ax25_putchar(ctx, c << 1);
      }

      /* Fill with spaces the rest of the CALL if it's shorter */
      if (len < sizeof(addr->call))
            for (unsigned i = 0; i < sizeof(addr->call) - len; i++)
```

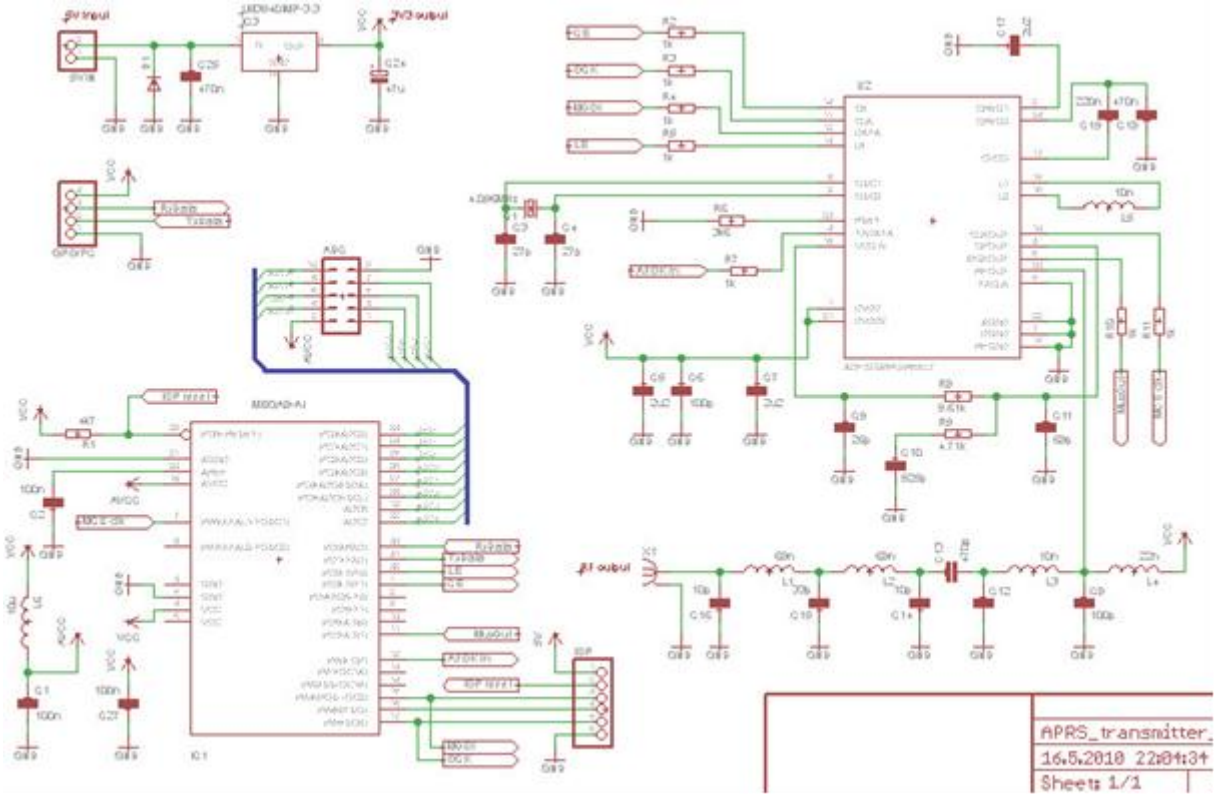
```
                ax25_putchar(ctx, ' ' << 1);

        /* The bit7 "has-been-repeated" flag is not implemented here */
        /* Bits6:5 should be set to 1 for all SSIDs (0x60) */
        /* The bit0 of last call SSID should be set to 1 */
        uint8_t ssid = 0x60 | (addr->ssid << 1) | (last ? 0x01 : 0);
        ax25_putchar(ctx, ssid);
}

/**
 * Send an AX25 frame on the channel through a specific path.
 * \param ctx AX25 context to operate on.
 * \param path An array of callsigns used as path, \see AX25_PATH for
 *        an handy way to create a path.
 * \param path_len callsigns path lenght.
 * \param _buf payload buffer.
 * \param len length of the payload.
 */
void ax25_sendVia(AX25Ctx *ctx, const AX25Call *path, size_t path_len,
const void *_buf, size_t len)
{
        const uint8_t *buf = (const uint8_t *)_buf;
        ASSERT(path);
        ASSERT(path_len >= 2);

        ctx->crc_out = CRC_CCITT_INIT_VAL;
        kfile_putc(HDLC_FLAG, ctx->ch);


        /* Send call */
        for (size_t i = 0; i < path_len; i++)
                ax25_sendCall(ctx, &path[i], (i == path_len - 1));

        ax25_putchar(ctx, AX25_CTRL_UI);
        ax25_putchar(ctx, AX25_PID_NOLAYER3);

        while (len--)
                ax25_putchar(ctx, *buf++);

        /*
         * According to AX25 protocol,
         * CRC is sent in reverse order!
         */
        uint8_t crcl = (ctx->crc_out & 0xff) ^ 0xff;
        uint8_t crch = (ctx->crc_out >> 8) ^ 0xff;
        ax25_putchar(ctx, crcl);
        ax25_putchar(ctx, crch);

        ASSERT(ctx->crc_out == AX25_CRC_CORRECT);

        kfile_putc(HDLC_FLAG, ctx->ch);
}

void ax25_sendRaw(AX25Ctx *ctx, const void *_buf, size_t len)
{
        const uint8_t *buf = (const uint8_t *)_buf;

        ctx->crc_out = CRC_CCITT_INIT_VAL;
        kfile_putc(HDLC_FLAG, ctx->ch);
```

```
	while (len--)
		ax25_putchar(ctx, *buf++);

	/*
	 * According to AX25 protocol,
	 * CRC is sent in reverse order!
	 */
	uint8_t crcl = (ctx->crc_out & 0xff) ^ 0xff;
	uint8_t crch = (ctx->crc_out >> 8) ^ 0xff;
	ax25_putchar(ctx, crcl);
	ax25_putchar(ctx, crch);

	ASSERT(ctx->crc_out == AX25_CRC_CORRECT);

	kfile_putc(HDLC_FLAG, ctx->ch);
}

static void print_call(KFile *ch, const AX25Call *call)
{
	kfile_printf(ch, "%.6s", call->call);
	if (call->ssid)
		kfile_printf(ch, "-%d", call->ssid);
}

/**
 * Print a AX25 message in TNC-2 packet monitor format.
 * \param ch a kfile channel where the message will be printed.
 * \param msg the message to be printed.
 */
void ax25_print(KFile *ch, const AX25Msg *msg)
{
	print_call(ch, &msg->src);
	kfile_putc('>', ch);
	print_call(ch, &msg->dst);

	#if CONFIG_AX25_RPT_LST
	for (int i = 0; i < msg->rpt_cnt; i++)
	{
		kfile_putc(',', ch);
		print_call(ch, &msg->rpt_lst[i]);
		/* Print a '*' if packet has already been transmitted
		 * by this repeater */
		if (AX25_REPEATED(msg, i))
			kfile_putc('*', ch);
	}
	#endif

	kfile_printf(ch, ":%.*s\n", msg->len, msg->info);
}


/**
 * Init the AX25 protocol decoder.
 *
 * \param ctx AX25 context to init.
 * \param channel Used to gain access to the physical medium
 * \param hook Callback function called when a message is received
 */
void ax25_init(AX25Ctx *ctx, KFile *channel, bool raw, ax25_callback_t
hook)
```

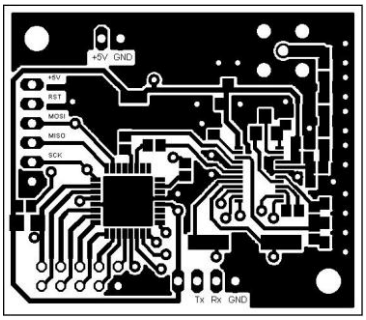
```
{
	ASSERT(ctx);
	ASSERT(channel);

	memset(ctx, 0, sizeof(*ctx));
	ctx->ch = channel;
	ctx->hook = hook;
	ctx->raw = raw;
	ctx->crc_in = ctx->crc_out = CRC_CCITT_INIT_VAL;
}
```

4.1.4 ADF7012 v APRS na 144MHz pásmu

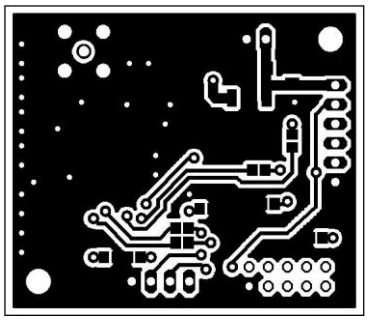
V diplomové práci Sabol Martin: „FM VYSÍLAČ APRS TELEMETRICKÝCH DAT V PÁSMU 144MHZ“, VUT Brno 2010 , www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=26883 rovněž použil ve vysílači obvod ADF7012 řízený ATMega88 stejně jako je tomu u vysílače PrattHobbies. K dispozici jsou i podklady pro výrobu PCB, stejně jako zdrojové kódy k firmware ATMegg88 i software pro nastavení vysílače z PC. Schema tohoto vysílače



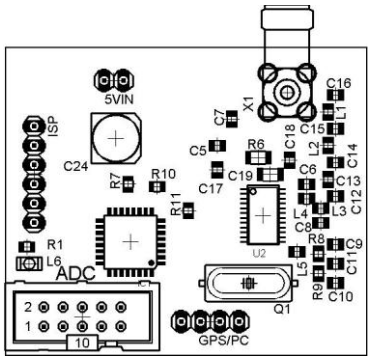
Konstrukční provedení a obrazce PCB:



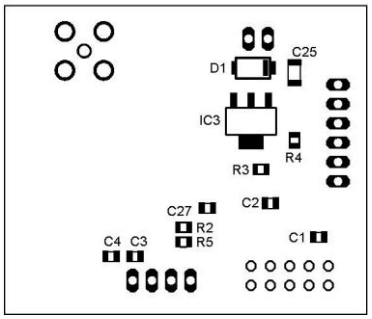
Pohled ze strany „TOP“



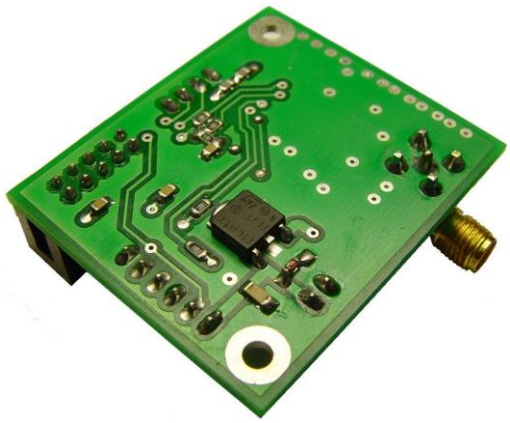
Pohled ze strany „Bottom“



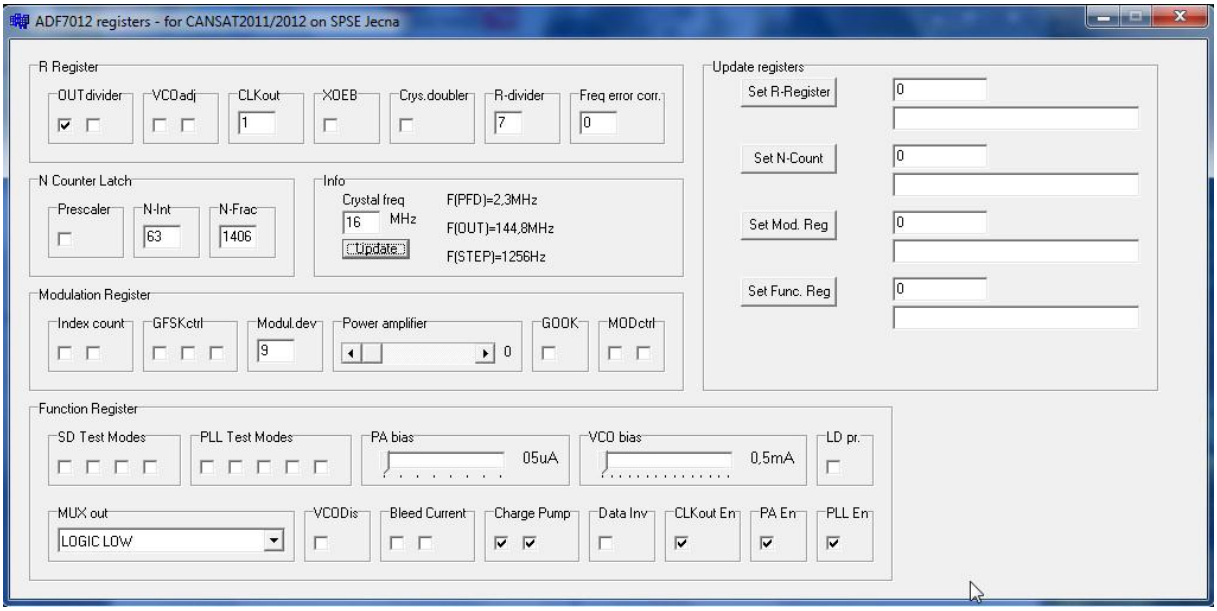
Osazení ze strany „TOP“



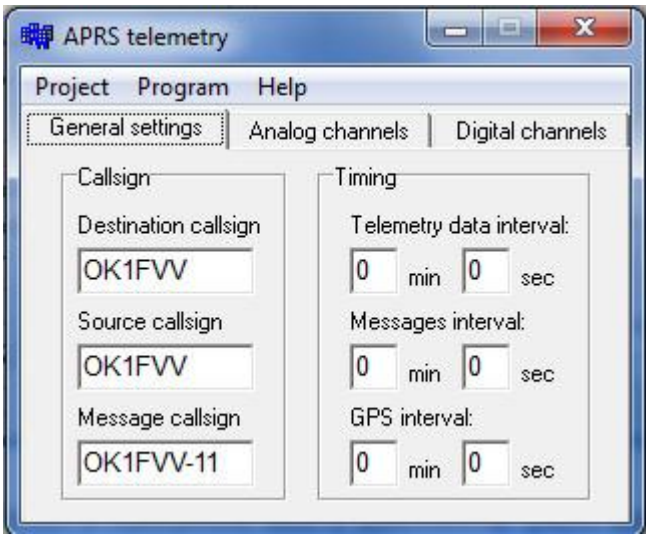
Osazení ze strany „Bottom“

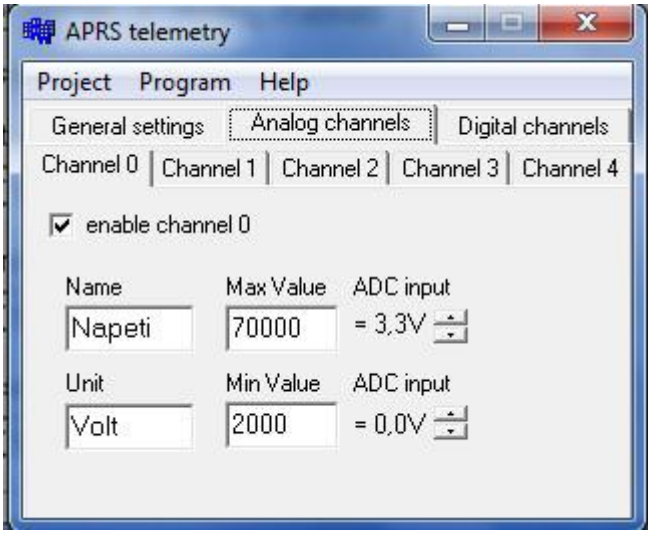


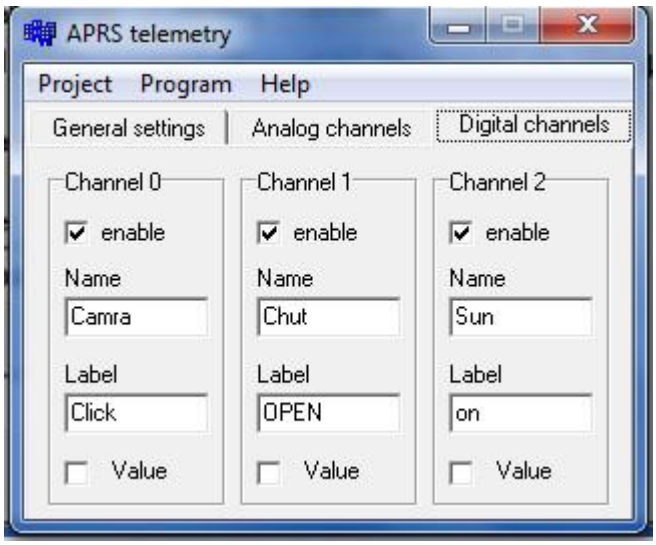
Pro představu ještě uvedu uživatelké rozhraní programů na PC komunikující s tímto vysílačem a umožňujícím jeho nastavení. To je výhoda oproti vysílači PrattHobbies. Jeden slouží pro inicializaci ADF7012



Další sw slouží k nastavení ARPS telemetrie (ADC kanálů apod.)



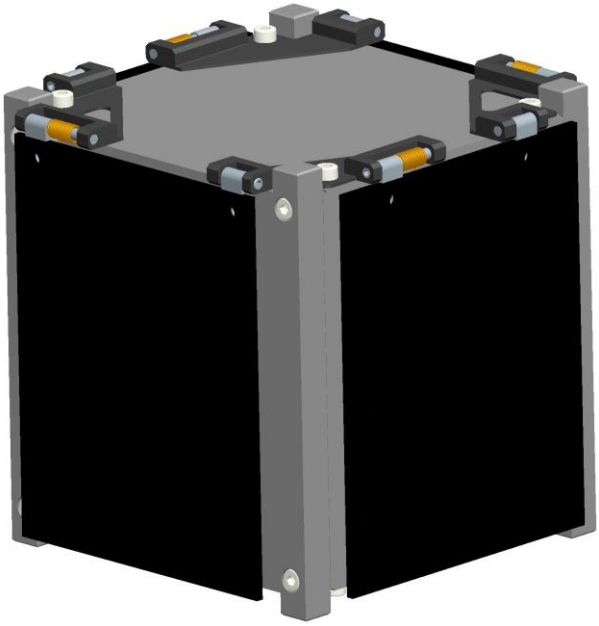




Oba tyto programy komunikují s ATMega88 vysílače prostřednictvím sériového kanálu rychlostí 9600baud, 8 datových bitu, jeden stop bit a bez paritního bitu. Přitom předpokládá ze strany PC použití USB kanálu a dále převodníku USB na sériový kanál. Tento převodník je založen na obvodu FT232RL. Oba programy kontrolují i přítomnost převodníku z USB na UART s FT232RL

4.1.5 Projekt PilsenCUBE

Na stránkách http://pilsencube.zcu.cz/ najdeme popis tohoto projektu. Projektu si klade za cíl s finanční podporou Grantové agentury České Republiky postavit na Fakultě elektrotechnické v Plzni pikosatelit třídy CubeSat s využitím moderních poznatků vědy v oblasti komunikačních a napájecích systémů. V kolektivu mladých pracovníků fakulty a vybraných studentů provádějí základní výzkum obvodového řešení stávajících úspěšných i neúspěšných pikosatelitů CubeSat s ohledem na rozbor jejich spolehlivosti a efektivnosti napájecích a komunikačních subsystémů.



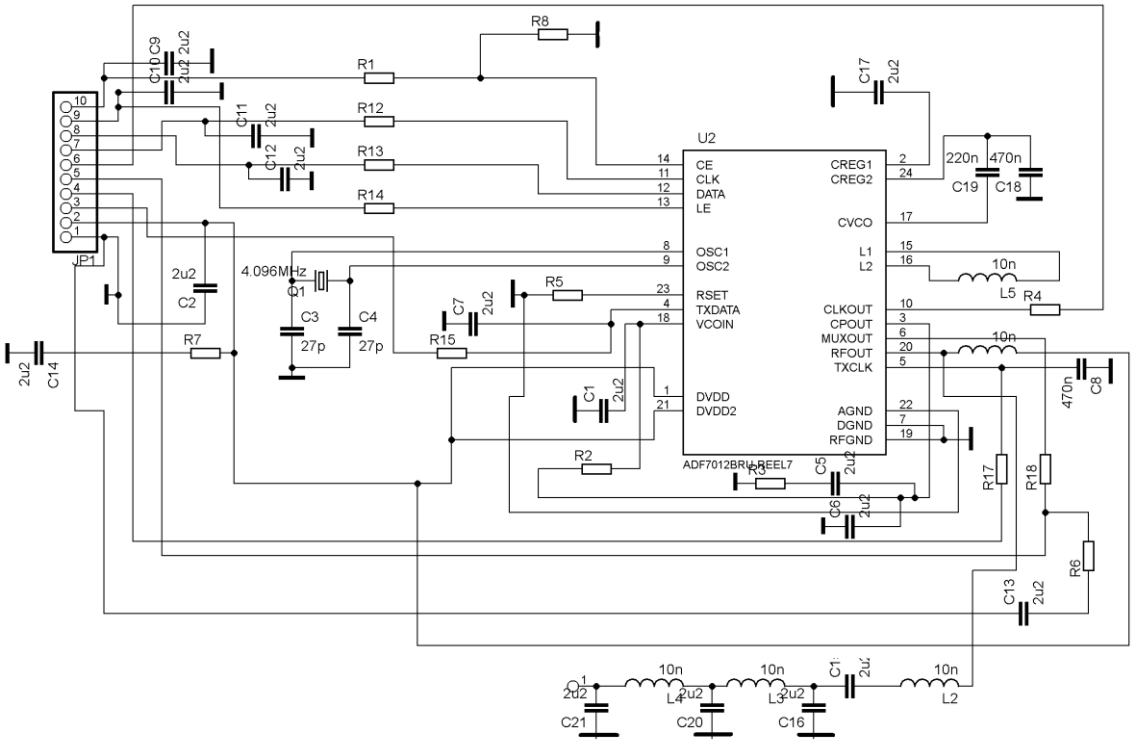
Univerzitní pikosatelit PilsenCUBE.

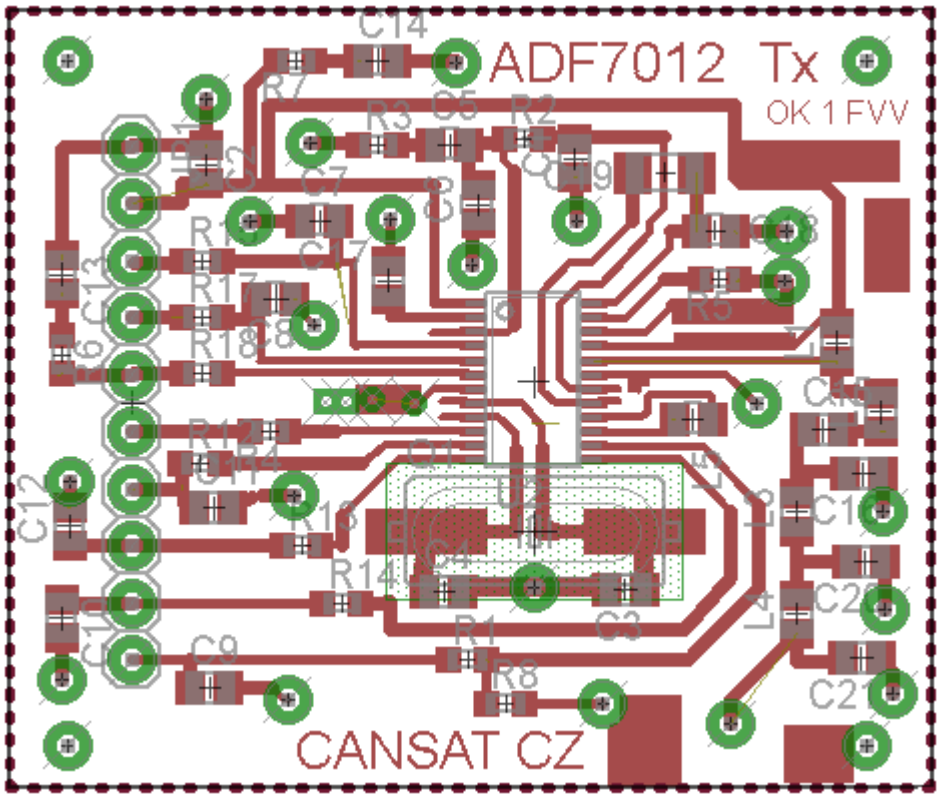
Radiomaják pikosatelitu PilsenCUBE
Radiomaják vysílá periodicky se sníženým výkonem identifikační označení pikosatelitu a případně i základní stavová slova o funkčnosti jednotlivých subsystémů pomocí OOK modulace a Morseova kódu. Radiomaják našeho pikosatelitu bude pracovat konvenčně v pásmu **435 MHz**, aby byl zachytitelný radioamatéry po celém světě nebo sítí unifikovaných spolupracujících pozemních stanic (www.genso.org). Je realizován pomocí monolitického obvodu **ADF7012** a procesoru ATMega16. V základním režimu bude přebírat a vysílat základní telemetrii od řídícího počítače pikosatelitu, má však implementován i autonomní režim pro případ selhání hlavního řídícího počítače.

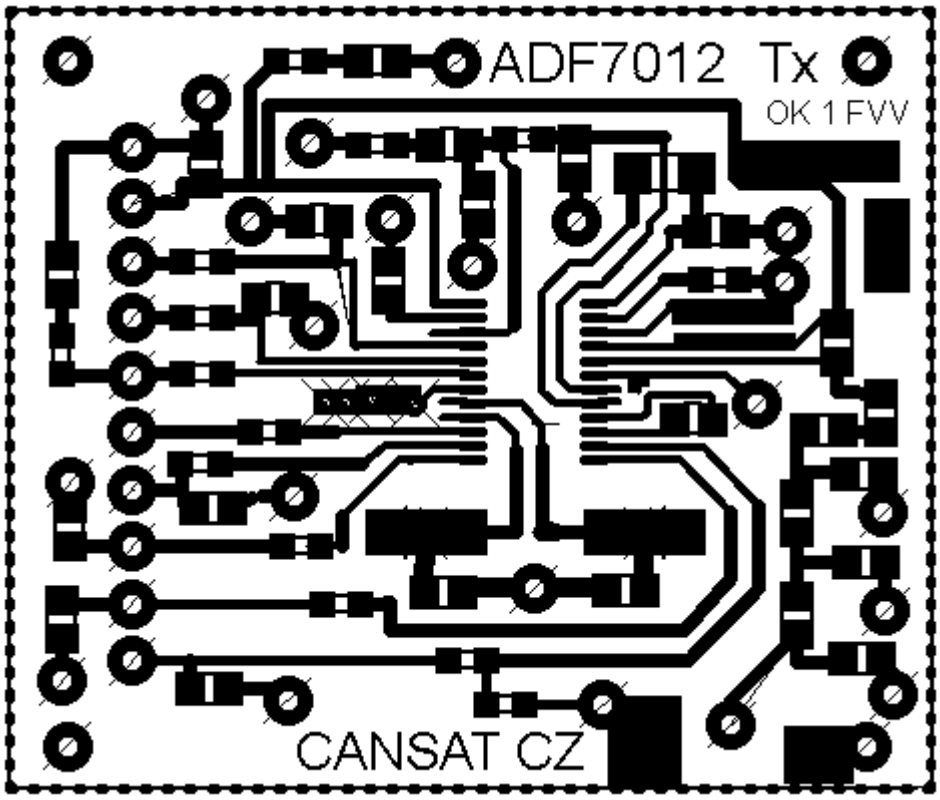
Při řešení projektu je tedy pro radiomaják použit stejný obvod (ADF7012) pracující ve stejném pásmu (70 cm) jako vysílač startkitu PrattHobbies používaném v CanSATech. Bohužel podrobnější informace zatím nemám.

4.1.6 Vlastní konstrukce vysílače s ADF7012

Konstrukce vysílače vychází z doporučeného zapojení z datasheetu obvodu ADF7012 firmy AnalogDevice. Ostatně stejně je tomu i u vysílačů uváděných v předchozích kapitolách. Proto vývoj vysílače spočívá především v návrhu PCB. Jako vývojový nástroj jsem použil Eagle.


U2
CE
CLK
DATA
LE
OSC1
OSC2
RSET
TXDATA
VCOIN
DVDD
DVDD2
CREG1
CREG2
CVCO
L1
L2
CLKOUT
CPOUT
MUXOUT
RFOUT
TXCLK
AGND
DGND
RFGND
ADF7012BRU-REEL7
JP1
4.096MHz
27p
27p
220n
470n
10n
2u2


ADF7012 Tx
OK 1 FVV
CANSAT CZ

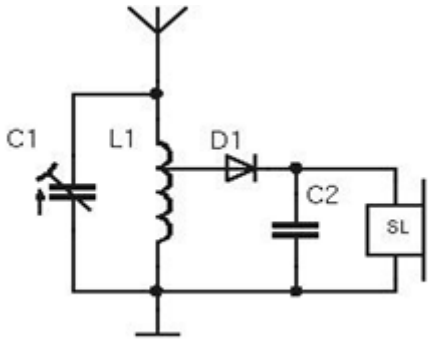


4.2 Přijímač

4.2.1 Klasické přijímače

Oblast bezdrátového přenosu prošla v průběhu několika uplynulých desetiletí bouřlivým rozvojem a rozšířila se v masové míře mezi koncové uživatele, zejména díkysoučasnému zdokonalení výpočetní techniky. Využití bezdrátového přenosu číslicovýchsignálů dnes sahá od poměrně pomalého přenosu dat mezi čili a řídící jednotkou, přes mobilní telefonní spojení, pozemní rozhlasové a televizní vysílání, až po širokopásmová spojení využívaná pro videokonference a stahování multimediálního obsahu z internetu.Masové nasazení bezdrátových technologií ovšem vyžaduje maximální spektrální účin-nost použitých modulací. Tyto modulace jsou náročné na obvody zpracování signálu,proto se hojně využívá číslicových obvodů a poznatků o zpracování číslicových signálů.

Pro snadnější pochopení základních principů digitálních modulací a digitálních přijímačů je dobré znát princip fungování základních analogových přijímačů. Nejjedno-dušším přijímačem pro AM je laděný LC obvod s obálkovým detektorem zvaný krys-talka, který byl používán v počátcích radiotechniky. Její zajímavou vlastností je, že k příjmu nepotřebuje žádný přídavný napájecí zdroj, vystačí si totiž s energií přijímanouanténou, její nevýhodou je malá hlasitost reprodukce, malá citlivost a malá selektivita. Přidáním zesilovače ke krystalce vznikne přímozesilující přijímač, který ke své

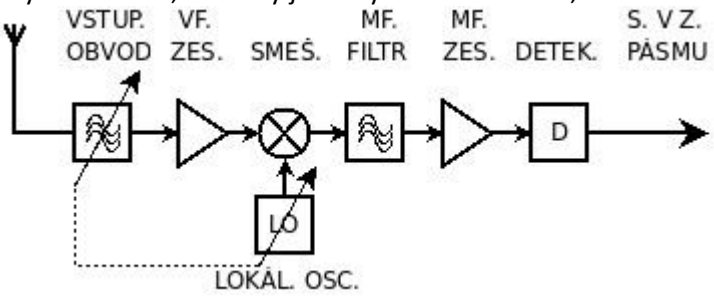


Krystalka

činnosti již potřebuje externí zdroj, ale je schopen hlasitější reprodukce. Zavedenímkladné zpětné vazby vznikne audion, který má díky kladné zpětné vazbě lepší citlivost a selektivitu, ale je náchylný k samovolnému rozkmitání hrozícímu při nevhodném nasta-vení zpětné vazby, nebo po přeladění na jiný kmitočet, neboť obvody zpětné vazby jsoufrekvenčně závislé. Vlastní oscilace audionu mohou být vyzářeny anténou a způsobit vysokofrekvenční rušení v okolí přijímače. Z tohoto důvodu se v dnešní době audiony nepoužívají, což neplatí o superreakčním přijímači, který koncepčně z audionu vychází. Superreakční přijímač opět obsahuje vstupní laděný obvod a kladnou zpětnou vazbu (kZV), která však svoji hodnotu periodicky mění v rytmu pomocného kmitočtu (desítkykHz, musí splnit vzorkovací teorém pro přenášené pásmo), úroveň kZV se neustále po-hybuje na hranici nasazení oscilací, což dovoluje tomuto typu přijímače dosáhnout dob-ré citlivosti na úkor selektivity a vyzařování tzv. superreakčního šumu. V dnešní doběse superreakční přijímač používá v nenáročných aplikacích pro příjem na pevně nasta-veném kmitočtu, podmínkou je maximální potlačení nežádoucího vyzařování. Základní nedostatek (špatnou přeladitelnost) přímozesilujících přijímačů odstranilsuperheterodyn, který vznikne zařazením frekvenčího konvertoru (dále směšovač) předpřímozesilující přijímač, naladěný na pevný tzv. mezifrekvenční kmitočet (mf.), vlastnípřeladění přijímaného kmitočtu se dosáhne změnou pomocného kmitočtu, generovaného lokálním oscilátorem (LO - local oscilator), který se spolu s přijímanýmsignálem přivádí na vstup směšovače. Na výstupu reálného směšovače se objeví velkémnožství směšovacích produktů, idealizovaný směšovač má na výstupu pouze čtyři základní složky:

1. fv (frekvence vstupního signálu)
2. fo (frekvence lokálního oscilátoru)
3. fo + fv (součtová složka)
4. |fo - fs| (rozdílová složka), obvykle bývá fo > fs, potom vztah přejde na fo - fs

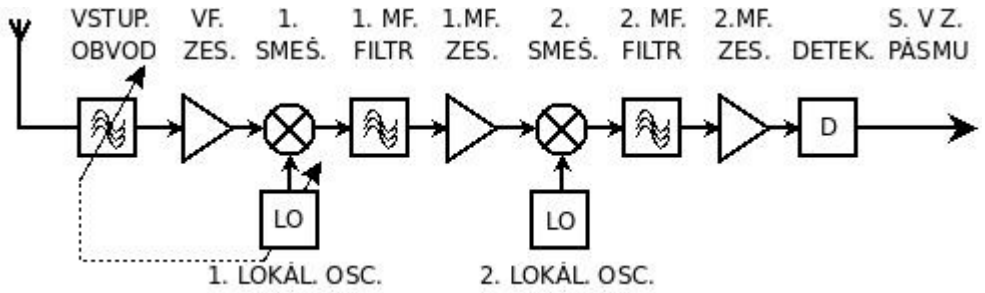
Mezifrekvenční zesilovač plní funkci přímozesilujícího přijímače, je obvyklenaladěn na rozdílový kmitočet směšovače. Mezifrekvenční zesilovač zajišťuje výsledný tvar a šířku přenášeného pásma, zásadně ovlivňuje kvalitu celého přijímače. Protože se mf při přeladění nemění, je šířka přenášeného pásma a zesílení nezávislé na přijímanémkmitočtu, další výhodou kmitočtové konverze je možnost zpracování velmi vysokýchkmitočtů, které by jinak bylo velmi obtížné, nebo dokonce nemožné.



Mezi nevýhody superheterodynu patří obtížné nastavení souběhu mezi vstupnímobvodem a lokálním oscilátorem. Také záleží na kmitočtové stabilitě lokálníhooscilátoru, tento problém řeší moderní koncepce oscilátorů řízených smyčkou fázovéhozávěsu (PLL), ale při návrhu PLL se musí dbát na minimalizaci fázového šumu. Dalšínevýhodou je možnost rušení na mezifrekvenčním kmitočtu, které lze eliminovatkvalitním odstíněním přijímače a využitím normalizovaných mezifrekvenčníchkmitočtů, na kterých se nevysílá. Nejvážnější problém představují tzv. zrcadlové kmitočty. Zrcadlový kmitočet fz, je kmitočet signálu vstupujícího anténou do příjímače spolus přijímaným signálem fv, uprostřed mezi fz a fs leží kmitočet lokálního oscilátoru fo, pro vzdálenost mezi přijímaným signálem a zrcadlovým kmitočtem platí

$$fZ - fS = 2fm \quad (1)$$

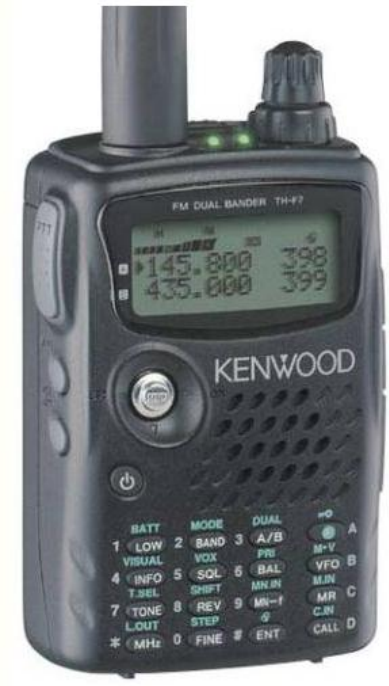
Za předpokladu, že platí fs < fo < fz, bude fo - fs =fm a zároveň fz - fo=fm , tedydva různé signály po konverzi spadnou do mezifrekvence, což může zhoršit kvalitu, nebo přímo znehodnotit přijímanou informaci. Zrcadlové kmitočty lze potlačit několika způsoby, základem je filtrace ve vstupních obvodech ještě před samotným směšováním, ale vstupní obvod musí být přeladitelný v celém rozsahu přijímaných kmitočtů, protonemůže být příliš selektivní. Na obrázku 2 je blokové schéma superheterodynu. Obsahuje vstupní laděný obvodpro potlačení rušení mimo přijímané pásmo a zrcadlových kmitočtů, směšovač, lokálníoscilátor, mf. zesilovač, demodulátor, nf zesilovač a AVC (automatické vyrovnánícitlivosti). Další možností jak zvýšit potlačení zrcadlových kmitočtů je zvýšenímezifrekvence, potom bude vzdálenost fv signálu a zrcadlového kmitočtu fz větší, tím
se zvetší i potlačení fz ve vstupním filtru, na druhou stranu ale dojde k zvětšení šířkypásma, které se projeví zhoršením selektivity. Řešením může být přijímač s dvojím [2] a [3] nebo i vícenásobným směšováním. Superhet s dvojím směšováním



Na obrázku je superhet s dvojím směšováním, za anténou je laděný VF zesilovač, následuje 1. směšovač s přeladitelným oscilátorem OSC1, 1. mf je relativně na vysoké frekvenci s ohledem na dobré potlačení zrcadlových kmitočtů, oscilátor druhéhosměšovače je naladěný na konstantní kmitočet takový, aby rozdílový kmitočet ležel v pásmu propustnosti 2. mf zesilovače. Ten pracuje s poměrně nízkým kmitočtem, sohledem na dobrou selektivitu. Konstrukce superhetů s dvojím směšováním je velmi náročná a drahá, proto se tyto přijímače používají především u kvalitníchkomunikačních superhetů, nebo měřících přístrojů například u spektrálních analyzátorů. Zajímavé řešení potlačení zrcadlových kmitočtů představuje homodyn, kterývyužívá opačného přístupu, tedy místo zvyšování mezifrekvence ji snižuje až nazákladní pásmo, mezifrekvenční kmitočet je potom roven nule a lokální oscilátor je naladěn na stejnou frekvenci, jakou má přijímaný signál, tím se podle (1) fz=fs a zrcadlový jev se neuplatní. Mezifrekvenční filtr potom tvoří dolní propust. Relativní jednoduchost přijímače je vyvážena problémy se zpracováním stejnosměrné složky, velkou citlivostí na fázový šum oscilátoru a pro správnou detekci signálu AM je nutné fázově synchronizovat lokální oscilátor obnovenou nosnou [2] . Konstrukce komunikačního přijímače

klasickou technologií však obtížná a pro soutěže CANSAT je snadnější zakoupit takový přijímač. Ukázkou může být:

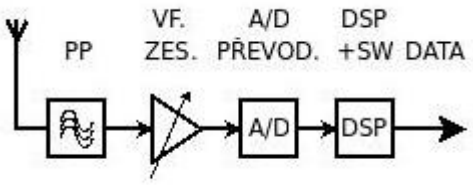
Kenwood TH-F7E



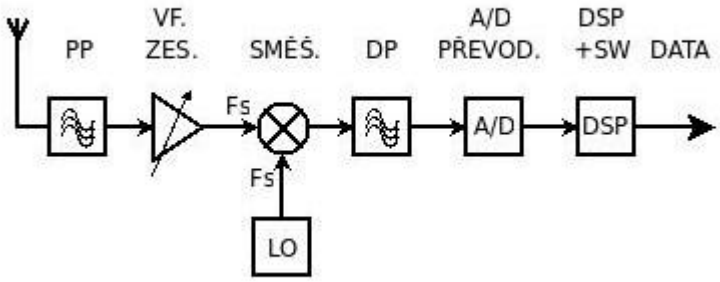
4.2.2 Softwarově definované přijímače

4.2.2.1 Teorie SDR

Přestože se s analogovými přijímači založenými na superheterodynu lze setkat prakticky na každém kroku, vývoj se ubírá směrem postupného nahrazení analogových částí digitálními, důvody tohoto trendu jsou především snaha po dosažení stále lepších parametrů radiového přenosu, jak digitálního (dosažení velkých datových rychlostí vyžaduje komplikované, analogově těžce realizovatelné modulace), tak i analogového, kde je snahou vytvořit přijímač s vlastnostmi odvozenými od přesného hodinového krystalu, pomocí softwaru pro digitální zpracování signálu tak, aby při výrobě odpadlo složité a nákladné slaďování analogových obvodů (vstupních obvodů, mf filtrů, souběh...), následně aby během provozu zůstaly parametry konstantní (u analogových obvodů mají velký vliv stárnutí, změna teploty, otřesy). V případě potřeby je možné změnit vlastnosti výměnou softwaru. Radiový přijímač, vysílač, nebo obecně radiový systém, jehož základní části (směšovače, filtry, zesilovače, modulátory/demodulátory) jsou určeny softwarovým vybavením, se označuje jako softwarově definované rádio nebo zkratkou SDR (Software Defined Radio). Cílem postupného nahrazování analogových obvodů digitálními je konstrukce přijímače, u kterého by hned za anténou následovala pásmová propust a rychlý A/D převodník, ostatní části by potom obstaral rychlý obvod DSP nebo FPGA. Tato koncepce se nazývá Ideální SDR přijímač.



Signál přijatý anténou postupuje do bloku pásmové propusti, zároveň plnící funkci impedančního přizpůsobení, kde je kmitočtově omezen. Následuje nízkošumový VF zesilovač s řízeným ziskem, na jehož výstupu má signál úroveň vhodnou pro rychlý A/D převodník. Po digitalizaci je signál zpracován blokem číslicového zpracování signálu (DSP), který obsahuje příslušný software. Z bloku DSP již vystupují data v požadovaném formátu. Tato koncepce dovoluje měnit parametry přijímače v maximální míře a je vhodná pro širokopásmové systémy s rozprostřeným kmitočtovým spektrem, je však obtížně realizovatelná u přijímačů pracujících na vysokých kmitočtech, protože klade vysoké nároky na rychlost A/D převodníku a následujících číslicových stupňů. Pokud má přijímaný signál kmitočet vyšší než řádově několik set MHz, nebo z finančních důvodů není možné použít rychlé A/D převodníky, je možným východiskem použití přijímačů kombinujících klasické analogové obvody s digitálními. Nejjednodušší variantu smíšeného přijímače představuje homodyn, jeho blokové schéma je na obrázku



Za anténou je pásmová propust, která odstraní nežádoucí signál a rušení leží mimo přijímáné pásmo. Dalším stupněm je nízkošumový VF zesilovač, který zesílí přijímaný signál na úroveň vhodnou pro směšovač. Směšovač představuje analogová násobička, která přijímaný signál konvertuje přímo do základního pásma, následuje dolní propust, která zároveň plní funkci mezifrekvenčního a antialiasingového filtru. Signál v základ-ním pásmu je přiveden na vstup A/D převodníku, za kterým následuje blok digitálního zpracování signálu, za ním už jsou data v požadovaném formátu. Stejně jako v případě analogové obdoby homodynu i zde je nutné pro bezchybnou demodulaci obnovit nos-nou a synchronizovat jí lokální oscilátor tak, aby byla přesně zachována i fáze. Možným východiskem je použít kvadraturní detektor . I při použití kvadraturního detektoru má koncepce homodynu mnoho nevýhodných vlastností, největší problém představuje velký zisk v základním pásmu a nutnost téměř stejnosměrné vazby pro modulace obsahující nízké kmitočty, dalším problémem je vyzařování lokálního oscilátoru na přijímané frekvenci a také velká citlivost na šum 1/f.

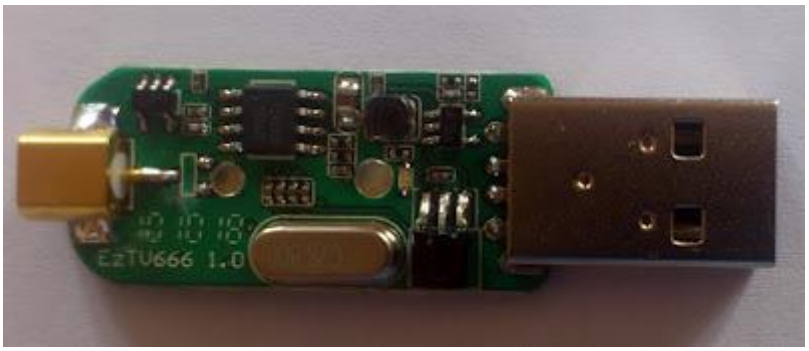
4.2.2.2 Praktická realizace levného SDR

Vývojář ovladačů V4L/DVB pro linuxové jádro **Antti Palosaari** objevil možnost získat z čipu Realtek RTL2832U přímo IQ data do počítače [4.2] . Tento čip je primárně určen pro DAB/DAB+/FM a je použit v hodně typech DVB-T USB dongle. Seznam podporovaných typů je:

VID	PID	tuner	device name
0x0bda	0x2832	all of them	Generic RTL2832U (e.g. hama nano)
0x0bda	0x2838	E4000	ezcap USB 2.0 DVB-T/DAB/FM dongle
0x0ccd	0x00a9	FC0012	Terratec Cinergy T Stick Black (rev 1)
0x0ccd	0x00b3	FC0013	Terratec NOXON DAB/DAB+ USB dongle (rev 1)
0x0ccd	0x00d3	E4000	Terratec Cinergy T Stick RC (Rev.3)
0x0ccd	0x00e0	E4000	Terratec NOXON DAB/DAB+ USB dongle (rev 2)
0x185b	0x0620	E4000	Compro Videomate U620F
0x185b	0x0650	E4000	Compro Videomate U650F
0x1f4d	0xb803	FC0012	GTek T803
0x1f4d	0xc803	FC0012	Lifeview LV5TDeluxe
0x1b80	0xd3a4	FC0013	Twintech UT-40
0x1d19	0x1101	FC2580	Dexatek DK DVB-T Dongle (Logilink VG0002A)
0x1d19	0x1102	?	Dexatek DK DVB-T Dongle (MSI DigiVox? mini II V3.0)
0x1d19	0x1103	FC2580	Dexatek Technology Ltd. DK 5217 DVB-T Dongle
0x0458	0x707f	?	Genius TVGo DVB-T03 USB dongle (Ver. B)
0x1b80	0xd393	FC0012	GIGABYTE GT-U7300
0x1b80	0xd394	?	DIKOM USB-DVBT HD
0x1b80	0xd395	FC0012	Peak 102569AGPK
0x1b80	0xd39d	FC0012	SVEON STV20 DVB-T USB & FM

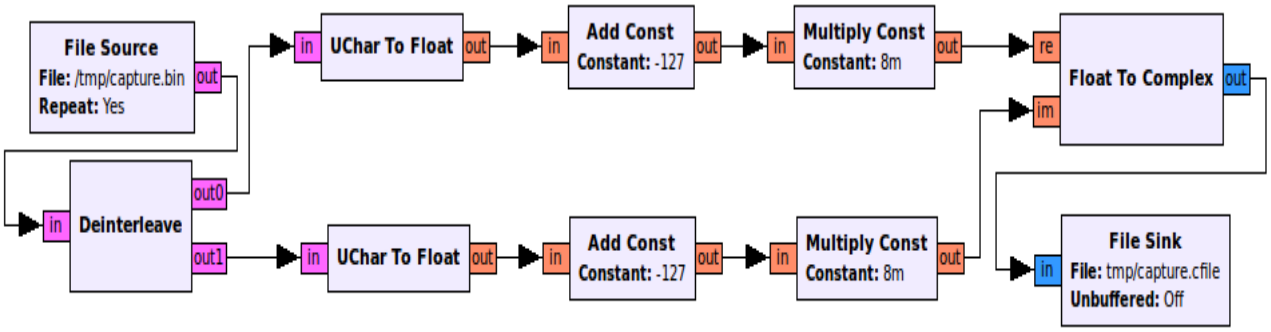
Pomocí vhodných ovladačů a software vznikla možnost využít tohoto dongle jako SDR přijímače. Kmitočtový rozsah závisí na použitém tuneru. Tuner E4000 funguje od 64 MHz do 1700 MHz (stejný tuner je použit i ve FunCube dongle). Tuner FC0013 funguje od 22MHz do 800MHz (takže je použitelný i na 28 a 50 MHz). Na internetu se dají pořídit od 10USD do 50USD. Velice vhodný je např. Sencor SDB 522RT.

Obsahuje tuner E4000. Tuner E4000 funguje od 64 MHz do 1700 MHz. K dalším parametrům: dynamický rozsah je cca 48 dB a šířka pásma od 900kHz po 2.8 MHz. AD převodník je jen 8 bitový oproti 16 bitovému ve Funcube dongle. Zato převodník je velmi rychlý a máme k dispozici daleko větší úsek pásma.



Pořídil jsem ho za 437 Kč v Alza.cz (objednací kód GK761v2)

Pozn.:
Na Linuxu je více aplikací s výše uvedenými TV tunery. Využívá se přitom GNU Radio:



Je využit např. Pro příjem FM rozhlasu a AM leteckého provozu
https://www.cgran.org/browser/projects/multimode/trunk
Trunked Radio systému TETRA http://tetra.osmocom.org/trac/
GSM https://svn.berlin.ccc.de/projects/airprobe/
Některé další známé aplikace a knihovny používající knihovnu **librtlsdr** nebo **gnuradio**

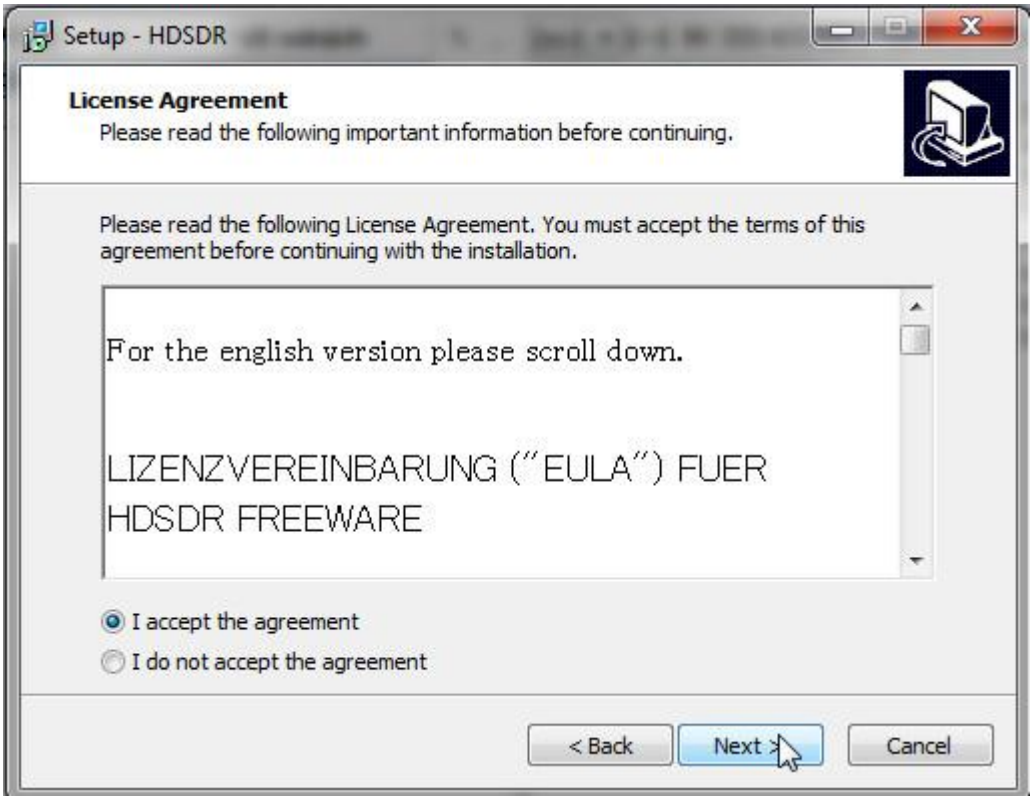
Name	Type	Author	URL
gr-pocsag	GRC Flowgraph	Marcus Leech	https://www.cgran.org/browser/projects/gr-pocsag/trunk
multimode	GRC	Marcus	https://www.cgran.org/browser/projects/multimode/

RX	Flowgraph	Leech	trunk
simple_fm_rvc	GRC Flowgraph	Marcus Leech	https://www.cgran.org/browser/projects/simple_fm_rcv/trunk
python-librtlsdr	Python Wrapper	David Basden	https://github.com/dbasden/python-librtlsdr
pyrtlsdr	Python Wrapper	Roger	https://github.com/roger-/pyrtlsdr
rtlsdr-waterfall	Python FFT GUI	Kyle Keen	https://github.com/keenerd/rtlsdr-waterfall
Wireless Temp. Sensor RX	Gnuradio App	Kevin Mehall	https://github.com/kevinmehall/rtlsdr-433m-sensor
QtRadio	SDR GUI	Andrea Montefusco et al.	http://napan.ca/ghpsdr3/index.php/RTL-SDR
gqrx (fork)	SDR GUI	Alexandru Csete	https://github.com/mathisschmieder/gqrx
rtl_fm **(NEW)**	SDR CLI	Kyle Keen	https://github.com/keenerd/rtl-sdr
SDR# **(NEW)**	SDR GUI	Youssef	http://sdrsharp.com/
gr-air-modes (fork) **(NEW)**	SDR CLI/server	Nick Foster	https://github.com/steve-m/gr-air-modes
tetra_demod_fft **(NEW)**	SDR GUI	osmocom team	osmosdr-tetra_demod_fft.py and the HOWTO
gqrx (original) **(NEW)**	SDR GUI	Alexandru Csete	https://github.com/csete/gqrx
airprobe **(NEW)**	GSM sniffer	osmocom team et al	http://git.gnumonks.org/cgi-bin/gitweb.cgi?p=airprobe.git

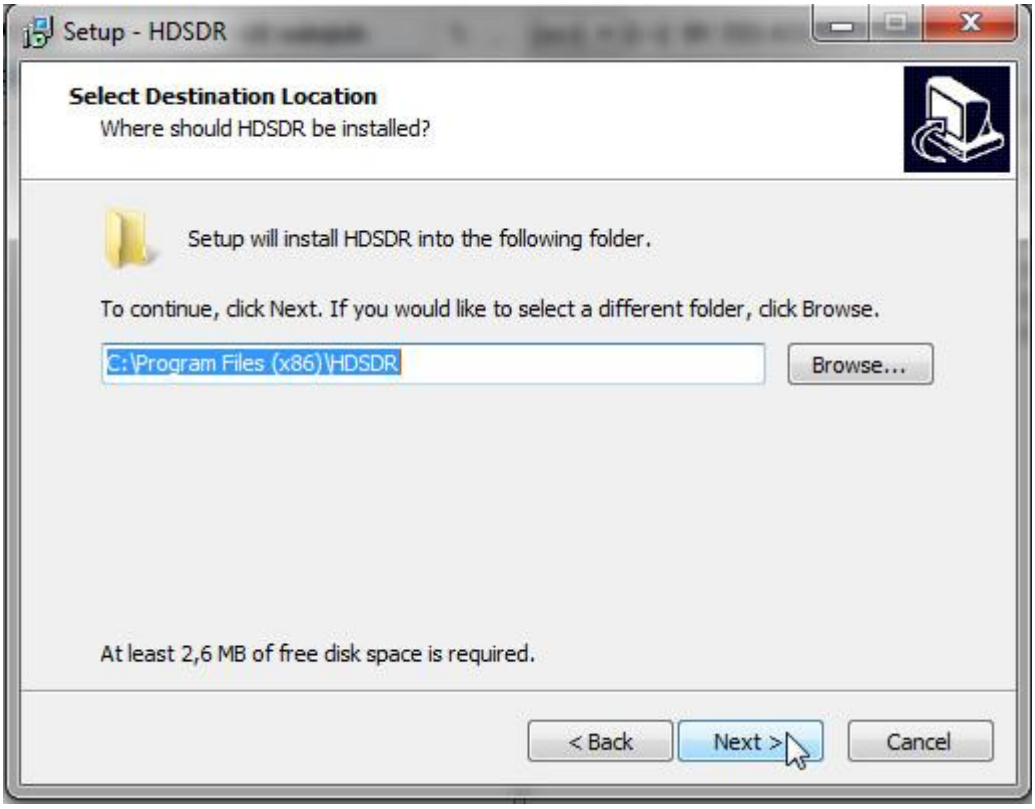
4.2.2.3 Instalace

Vrátíme s k windows a nejprve nainstalujeme HDSRV (HDSDR umí jen CW, SSB a úzkopásmovou FM, kterou právě potřebujeme). Jeho instalačky HSDR_install.exe stáhneme z http://www.hdsdr.de/ a program HSDR nainstalujeme. Spustíme instalačky

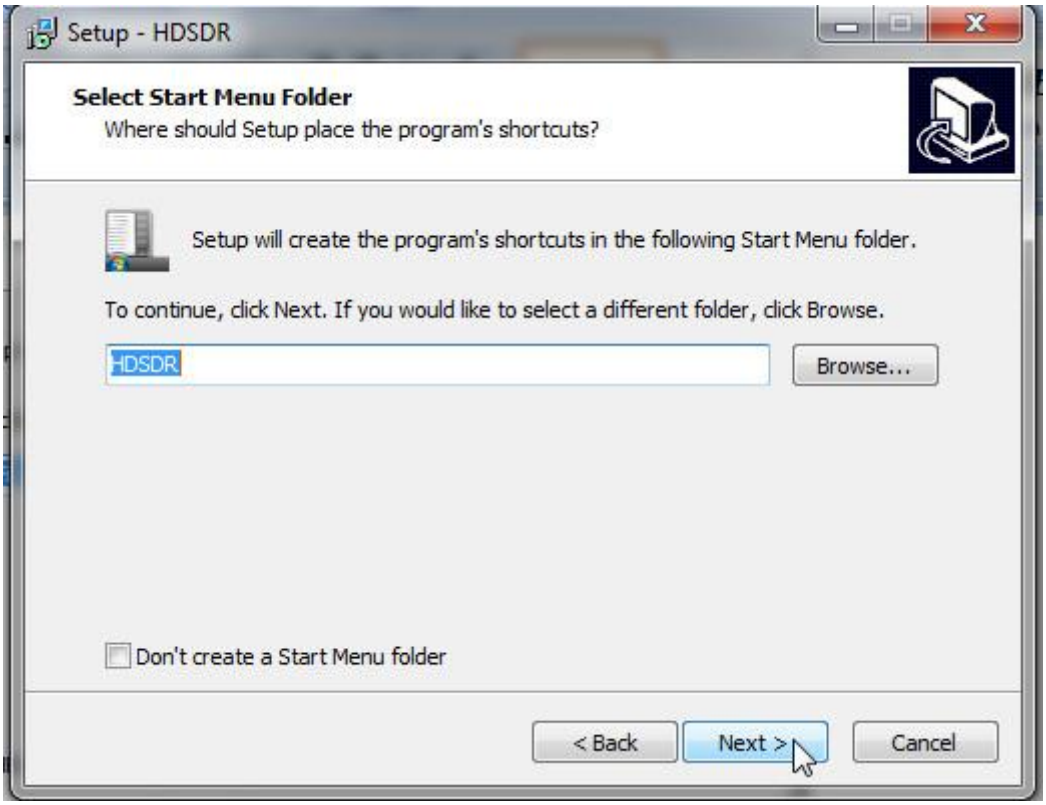
Klikneme na **Next** dostaneme



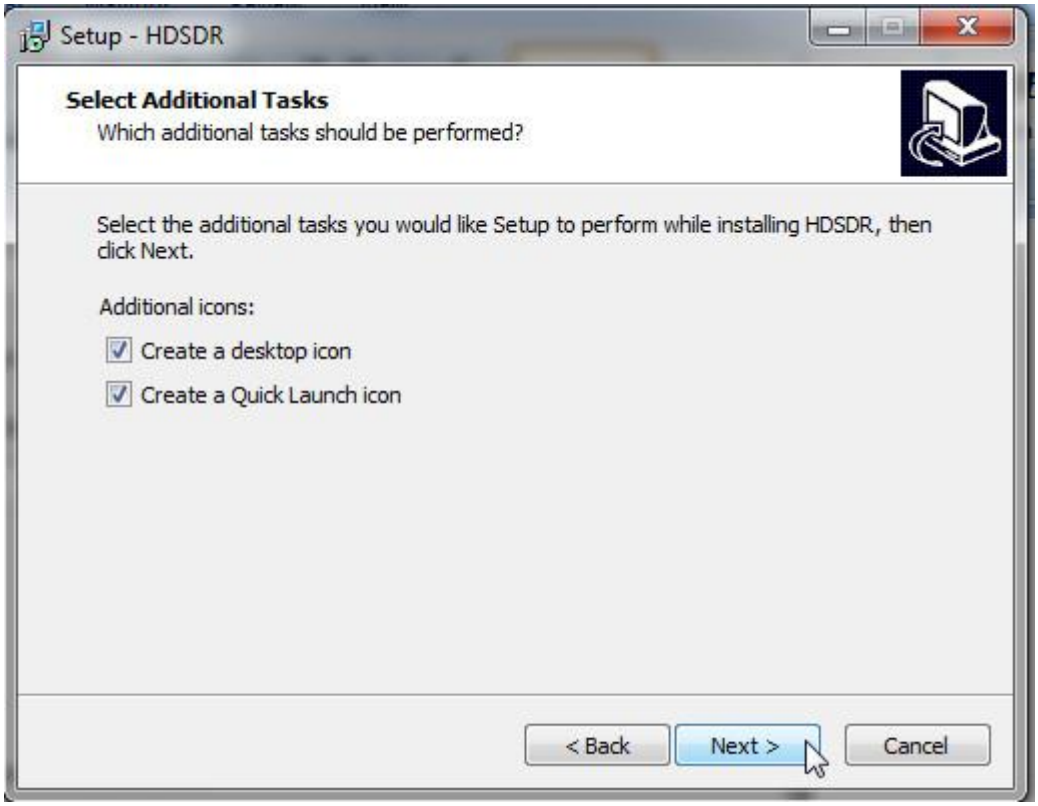
Akceptujeme licenční podmínky a klikneme na **Next**. Tak dostaneme



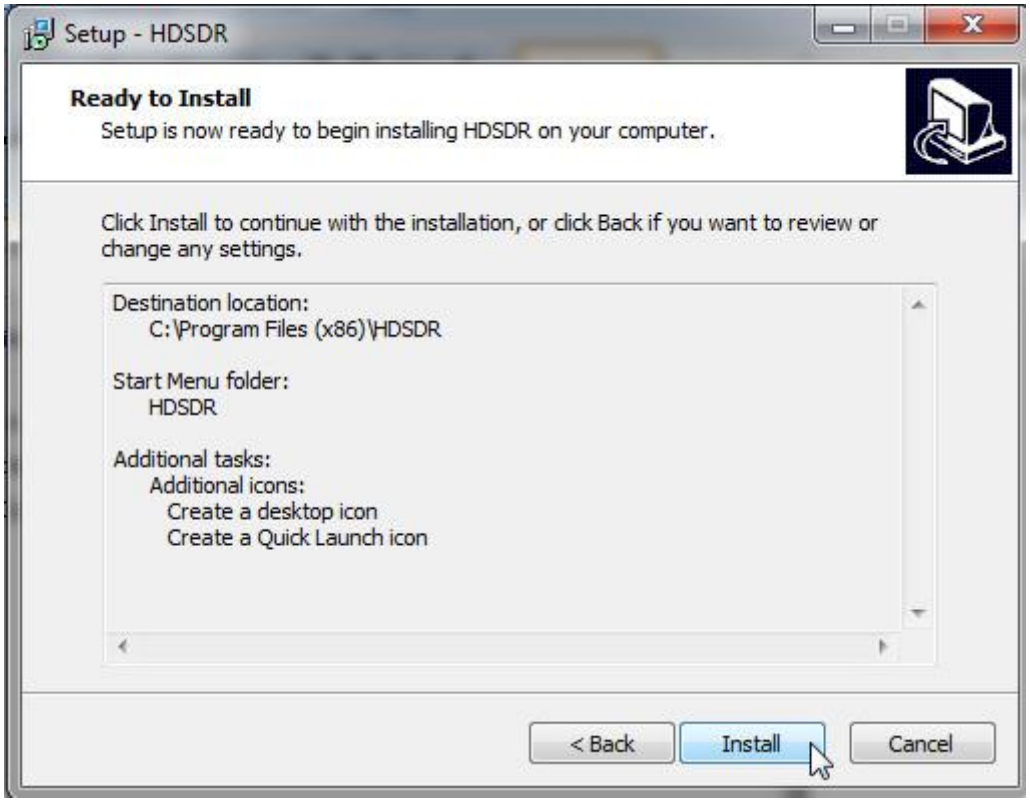
Opět budeme pokračovat kliknutím na **Next**



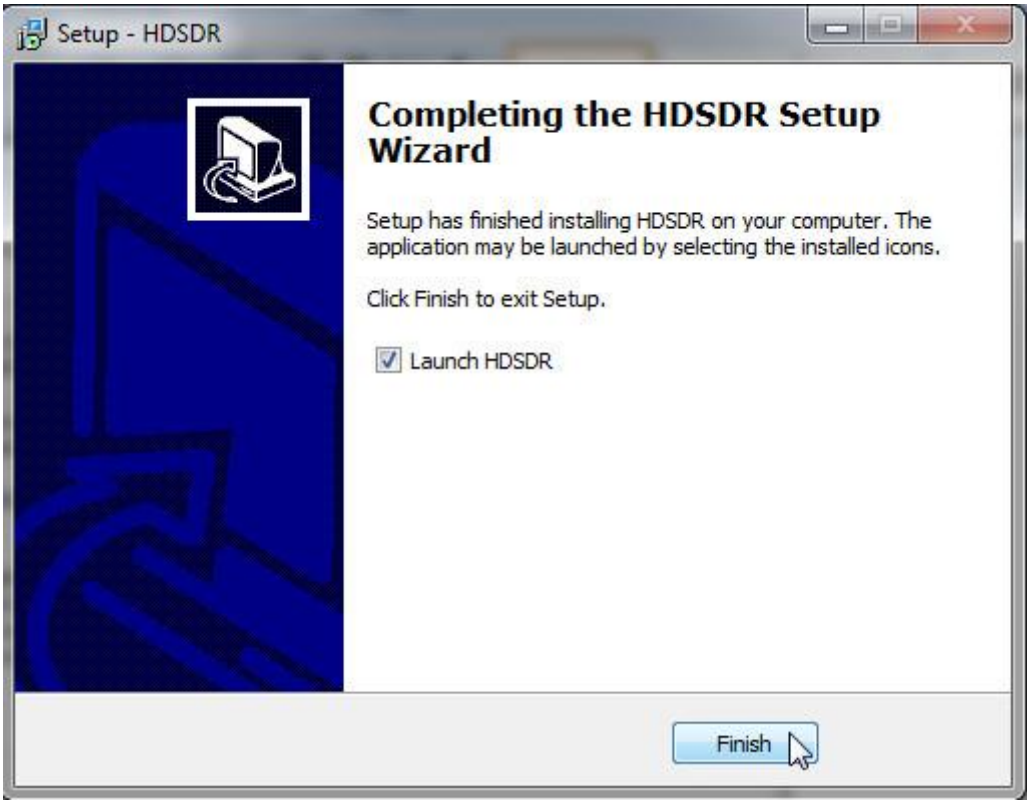
Další **Next**



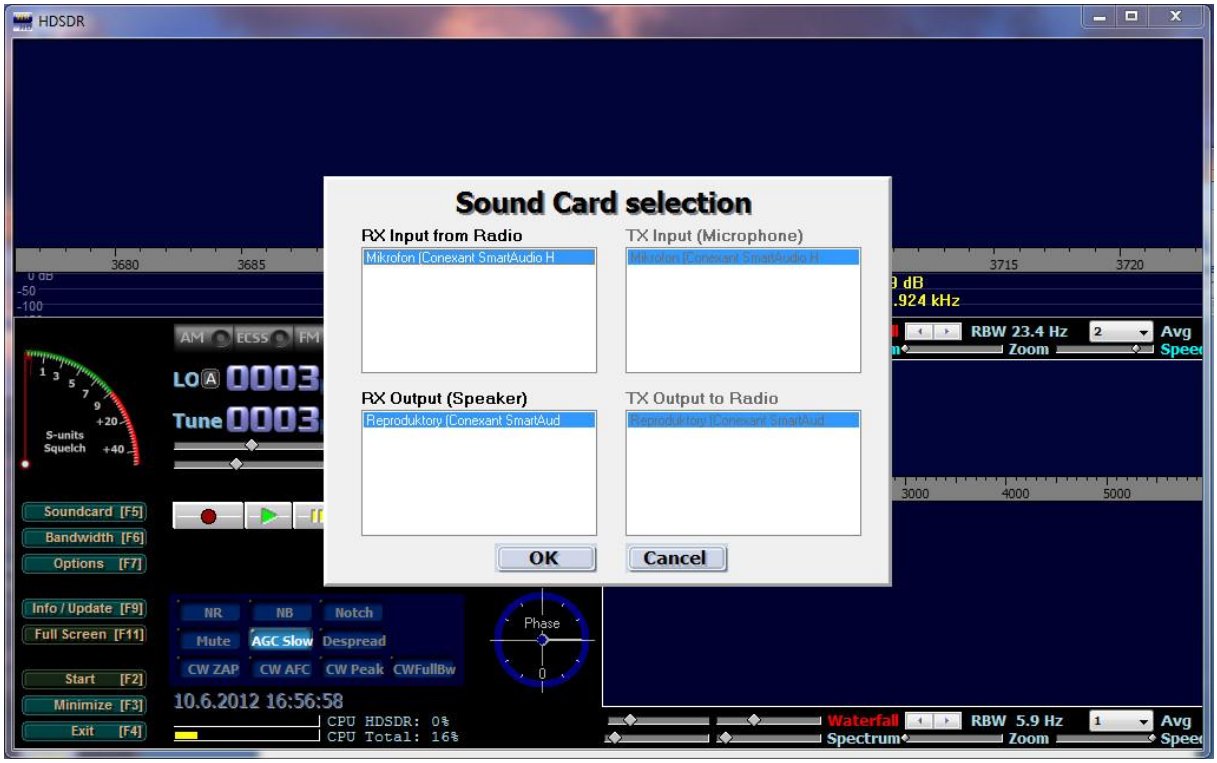
Opět klikneme na **Next**



Nyní již zahájíme instalaci kliknutím na tlačítko **Install**

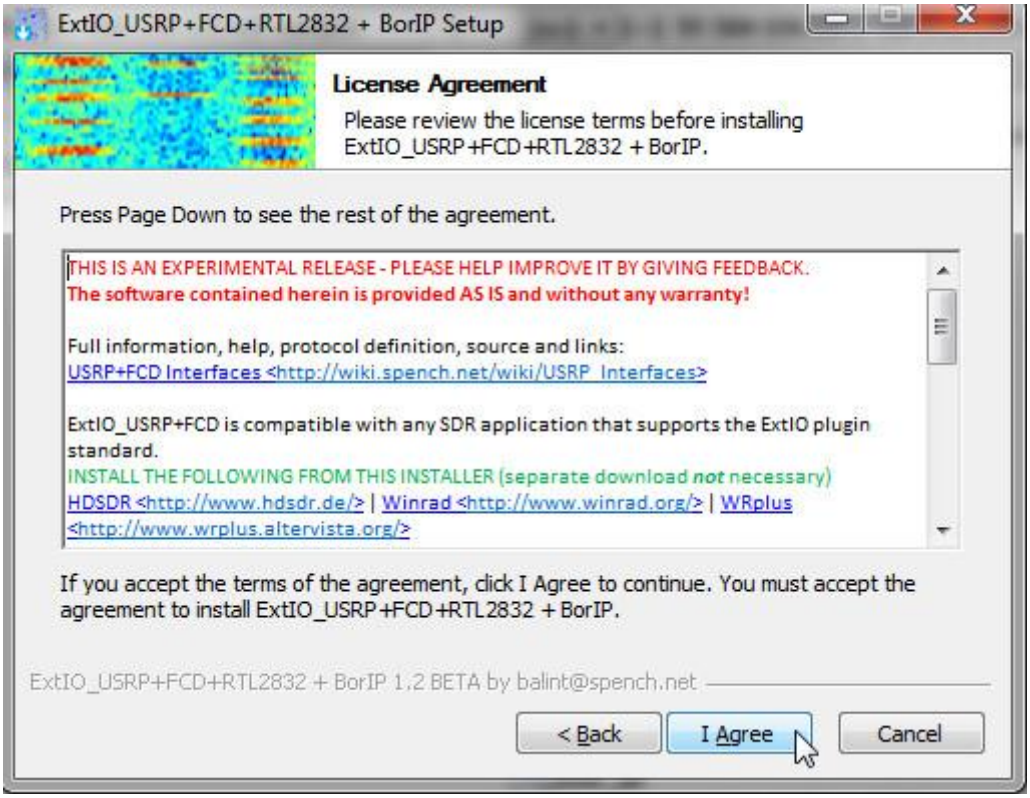


Ta je rychle dokončena. Nainstalovaný spuštěný program vypadá takto:

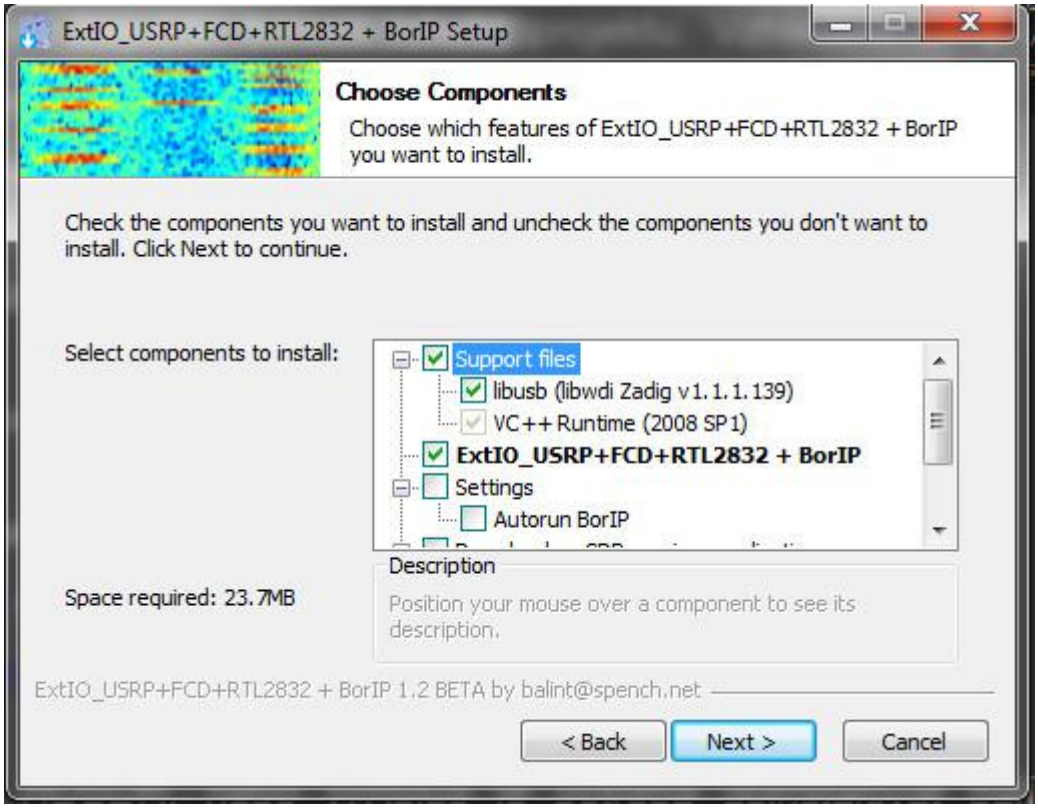


Klikneme na **Cancel** a program zatím uzavřeme. Dále z [4.3]
http://wiki.spench.net/wiki/USRP_Interfaces#Download stáhneme a nainstaluje USPR ovladače:
Instalačka se jmenuje **ExtIO_USRP+FCD+RTL2832 + BorIP-1.2 BETA10_Setup.exe** .
Ten obsahuje vlastní pluginu, která se stará o komunikaci s tunerem a také program Zadig s ovladači na USB. Spustíme instalaci

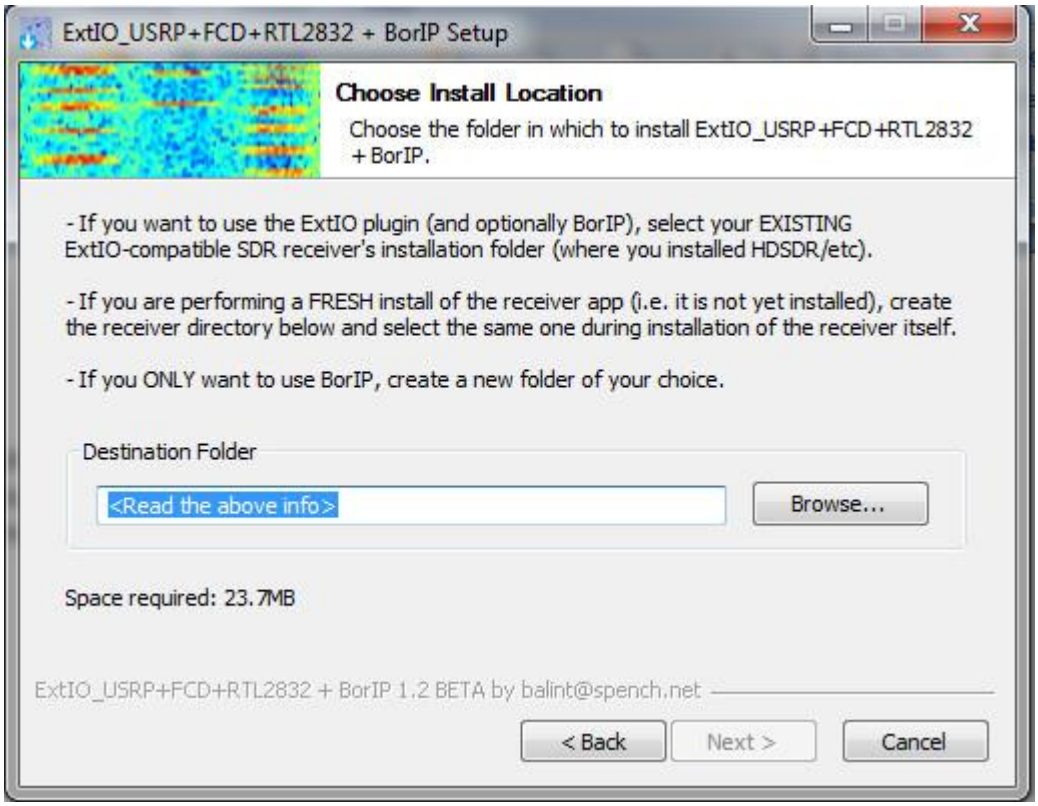
Klikneme na **Next** , máme tak



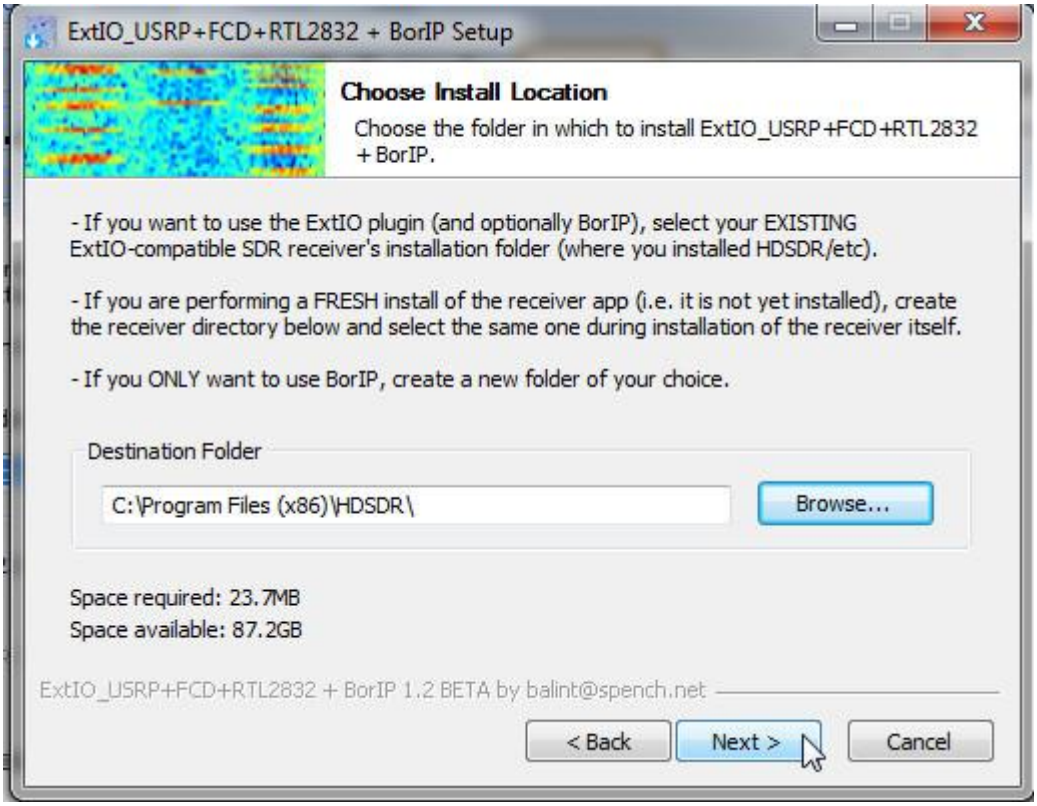
Souhlas s licenčními podmínkami potvrdíme kliknutím na **I Agree** a dostaneme



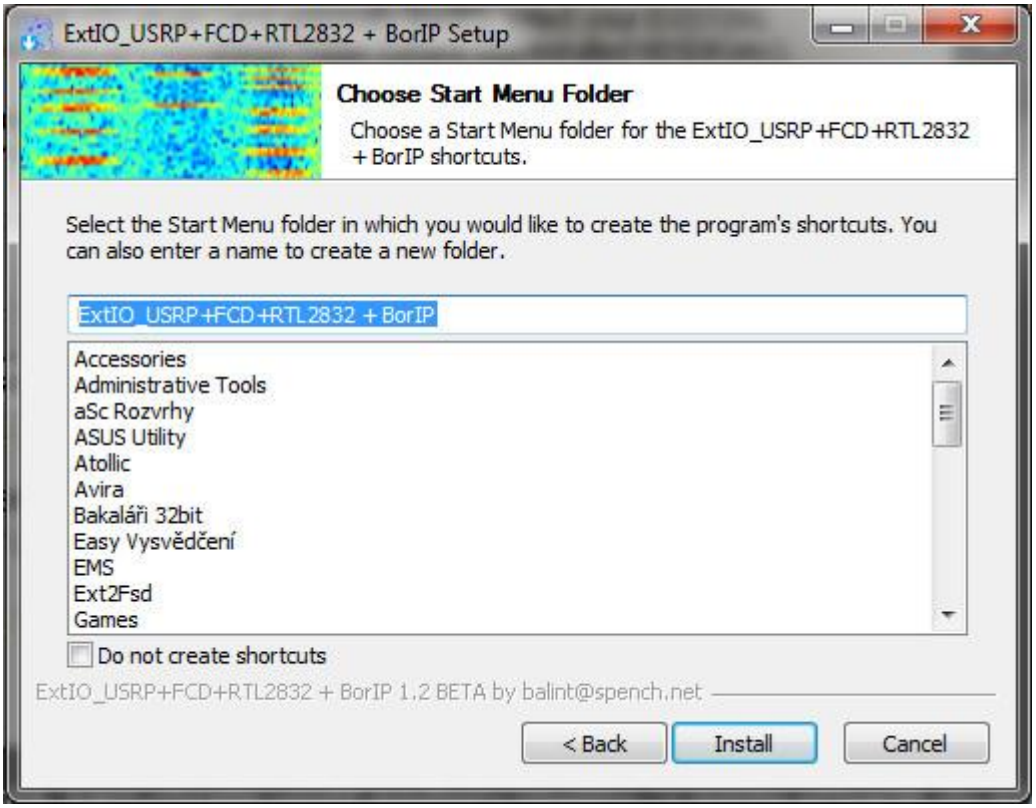
Nejprve zvolíme potřebné komponenty k instalaci (libusb a ExtIO_USRP). Ke zprovoznění je nutný balíček Microsoft Visual C++ Runtime (2008 SP1). Pokud ho v systému již nemáme, **zatrhneme** také "VC++ Runtime". Poté klikneme na **Next .** Dále jsme dotázáni na cestu k HDSDR



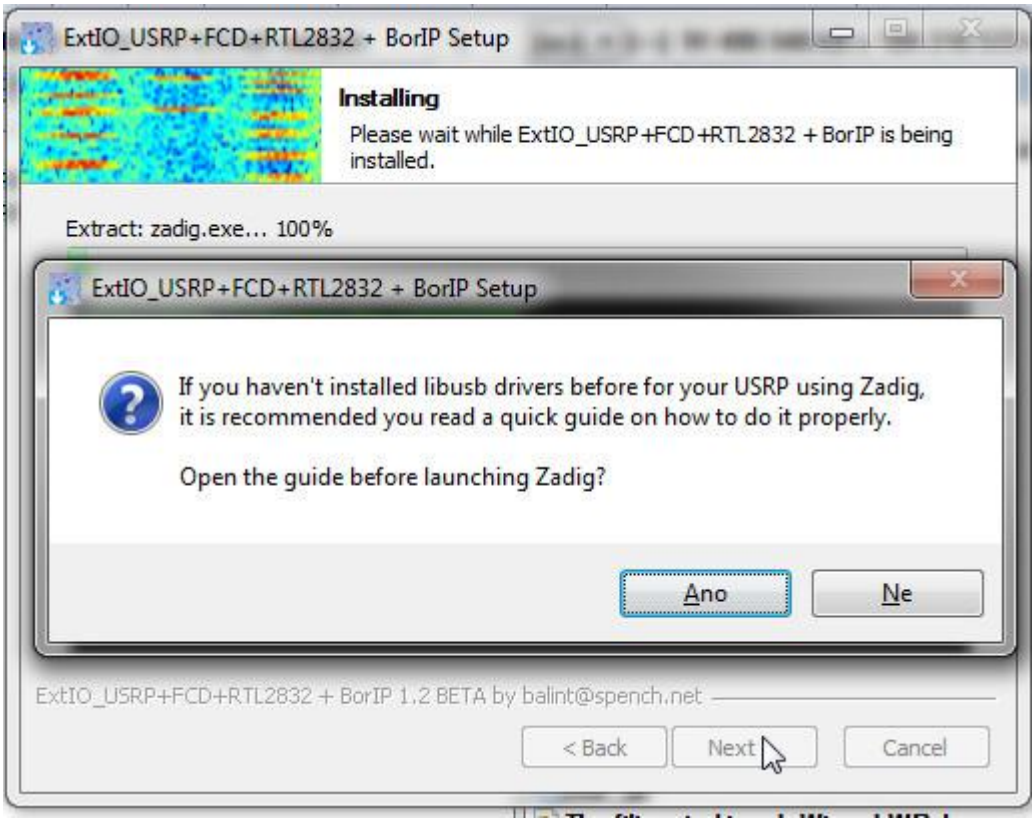
Klikneme na Browse a vybereme cestu



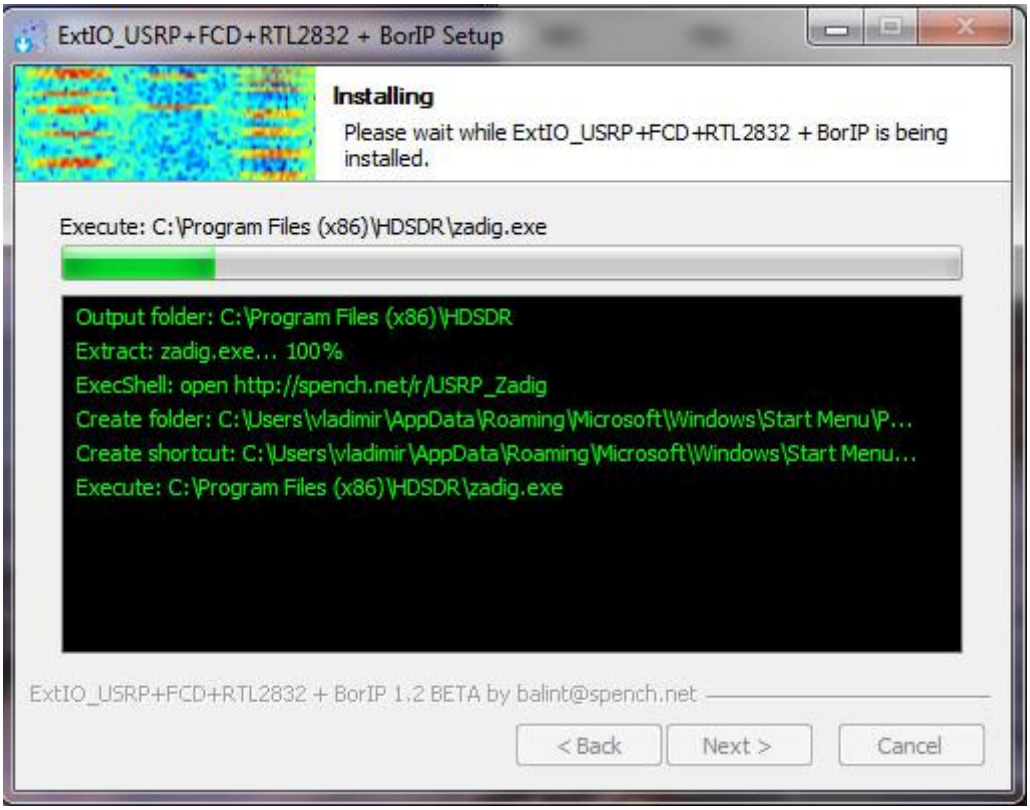
Potvrdíme tlačítkem **Next** a dostaneme



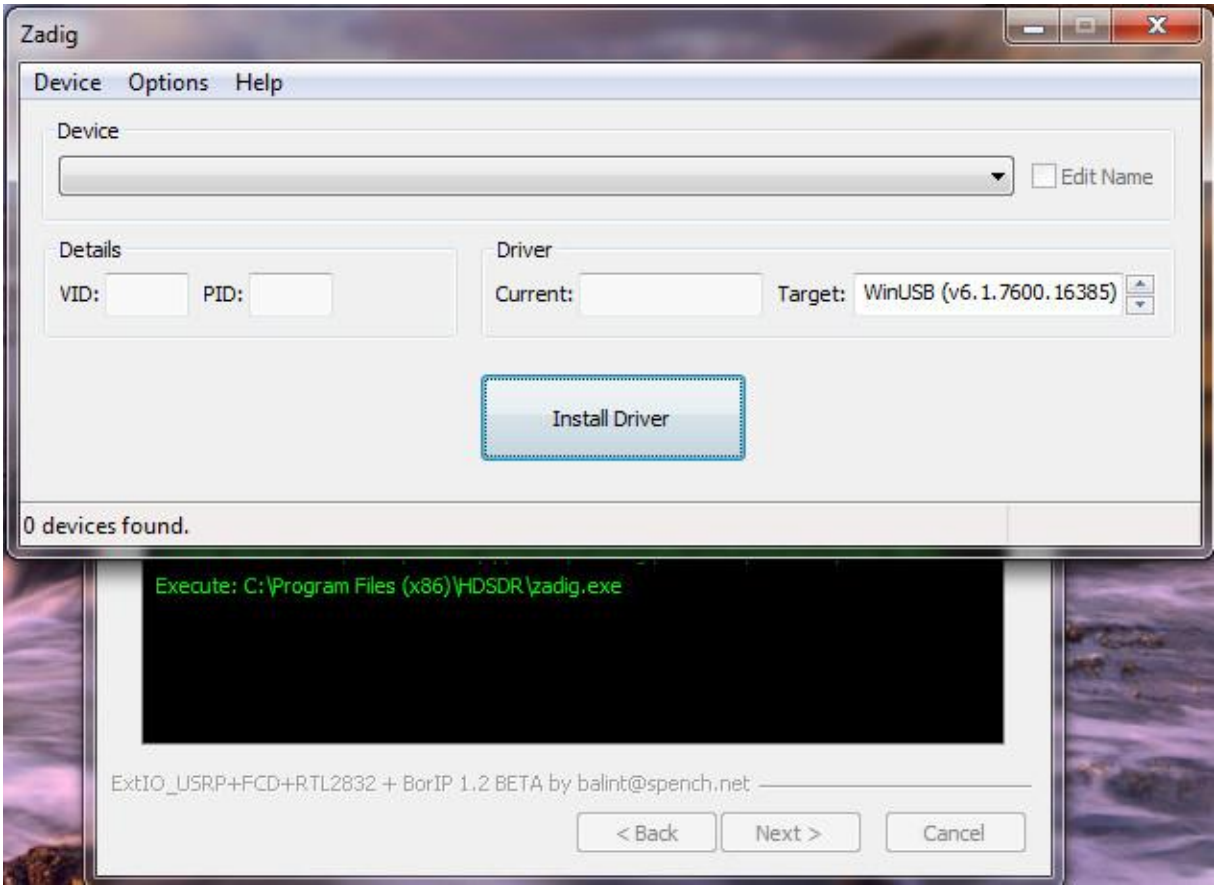
Spustíme instalaci



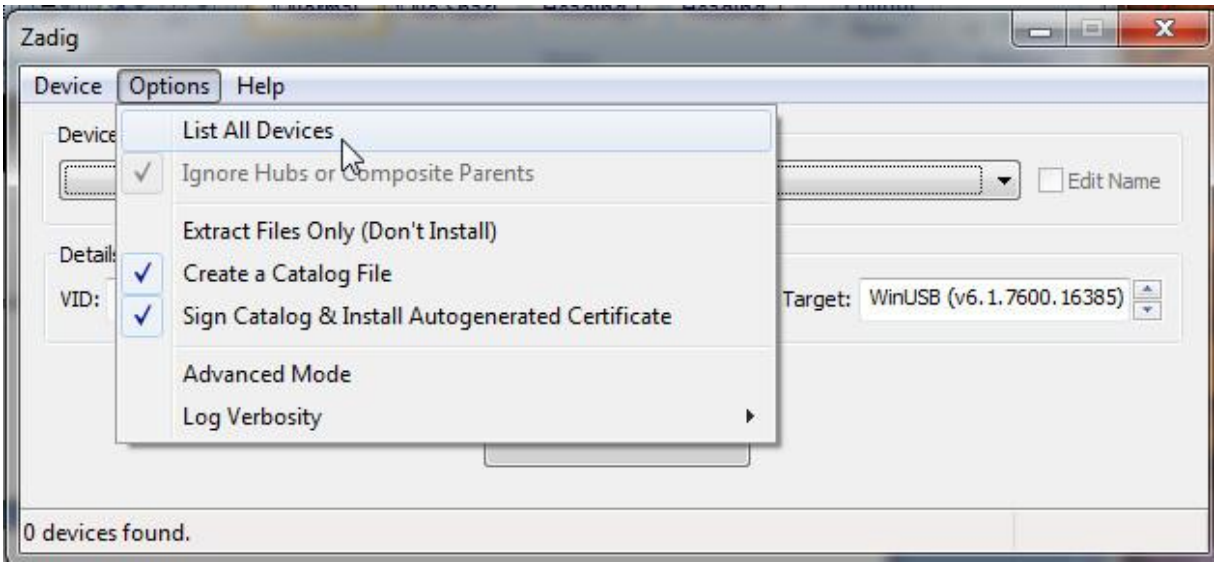
Klikneme na Ne. Dostaneme



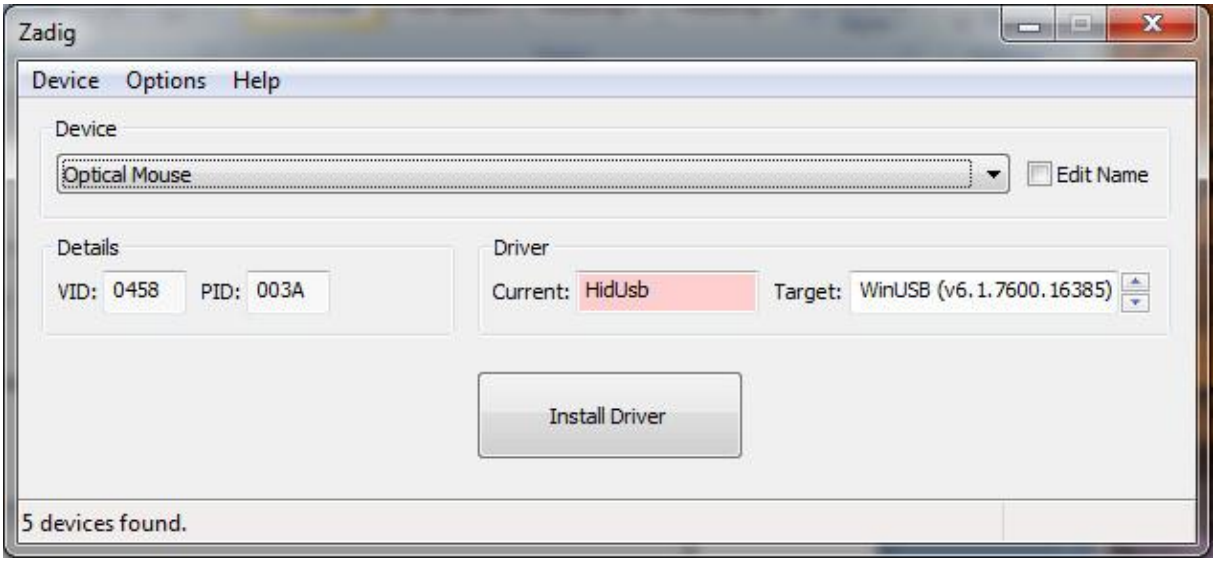
Načež se instalačka zeptá



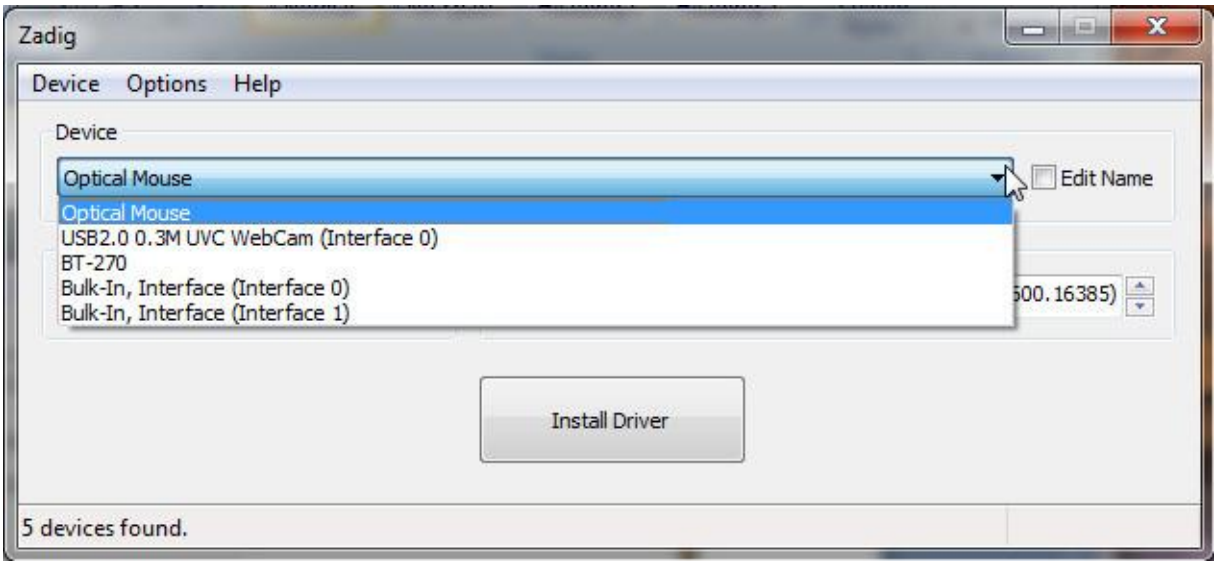
Nyní již musíme mít k našemu PC připojen SENSOR DVB-TTV tuner. Okno Zadig nyní vyplníme. Nejprve v menu **Options** zatrhneme **List All Devices**



A dostaneme např.

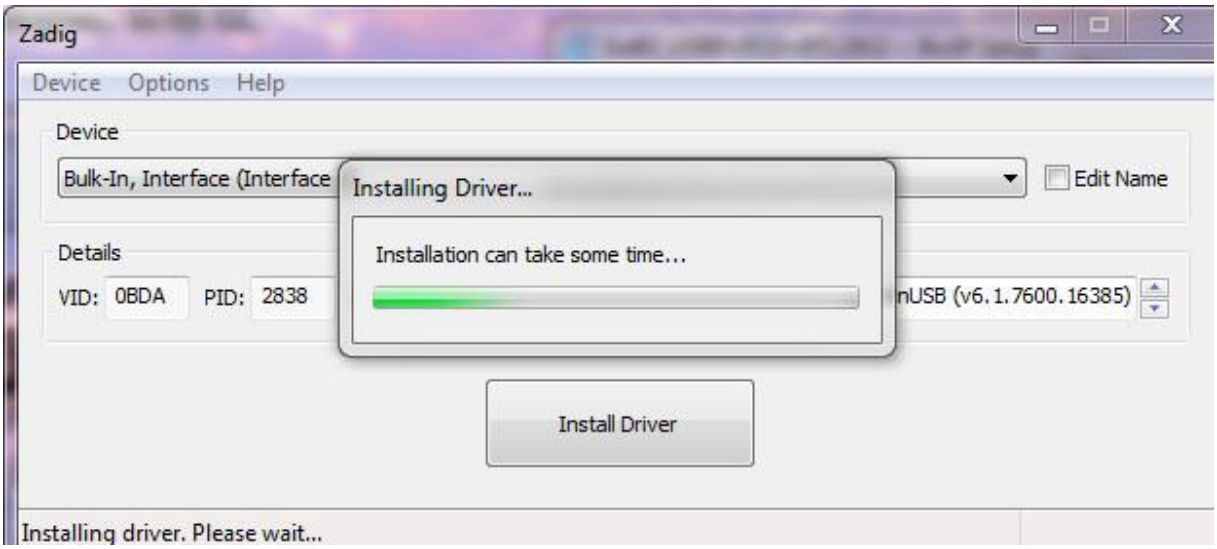


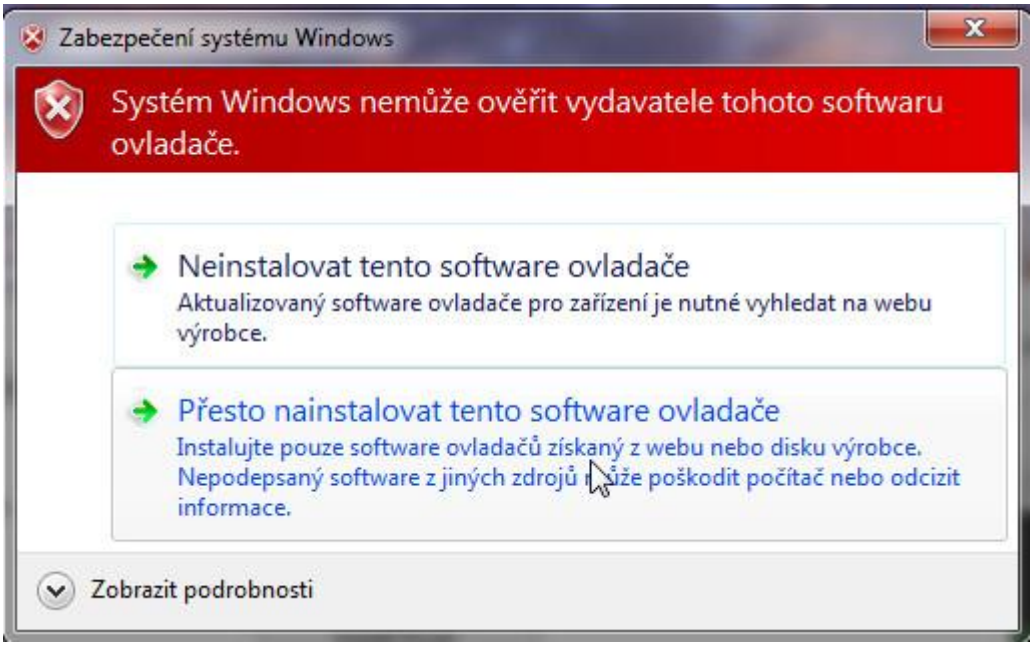
Nyní rozklikneme roletku Device

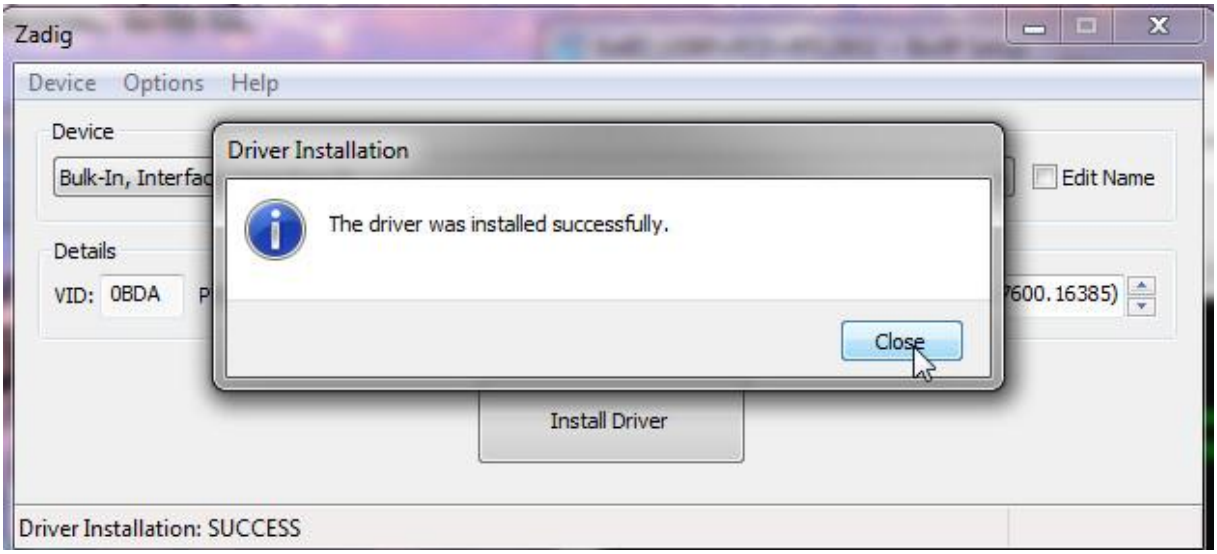


A vybereme tuner (název se může lišit)

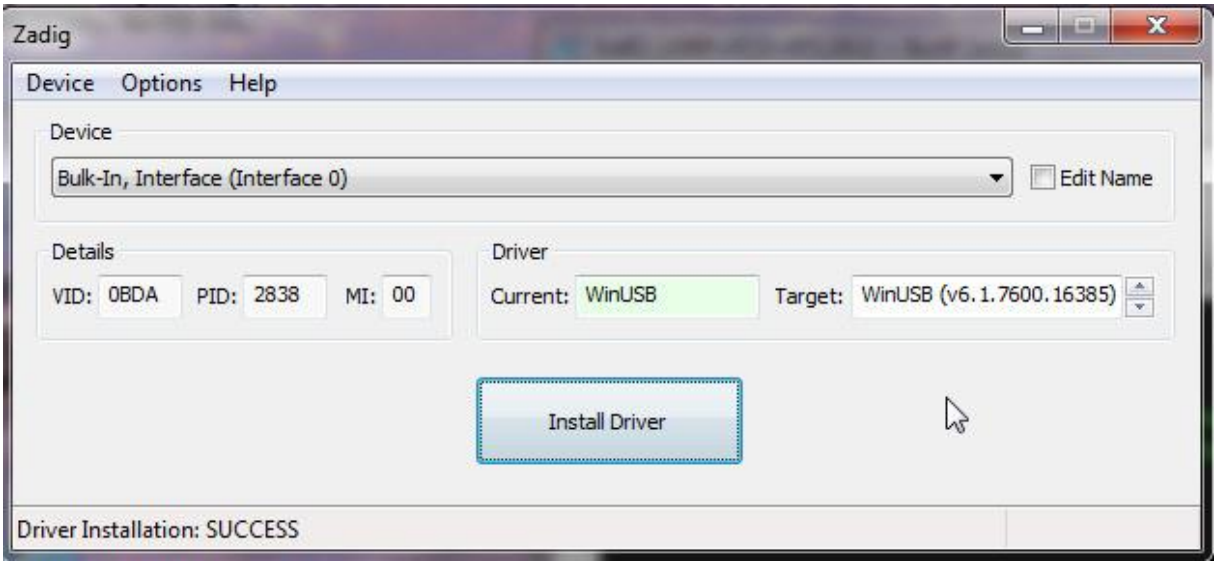
Spustíme instalaci tlačítkem **Install Driver**



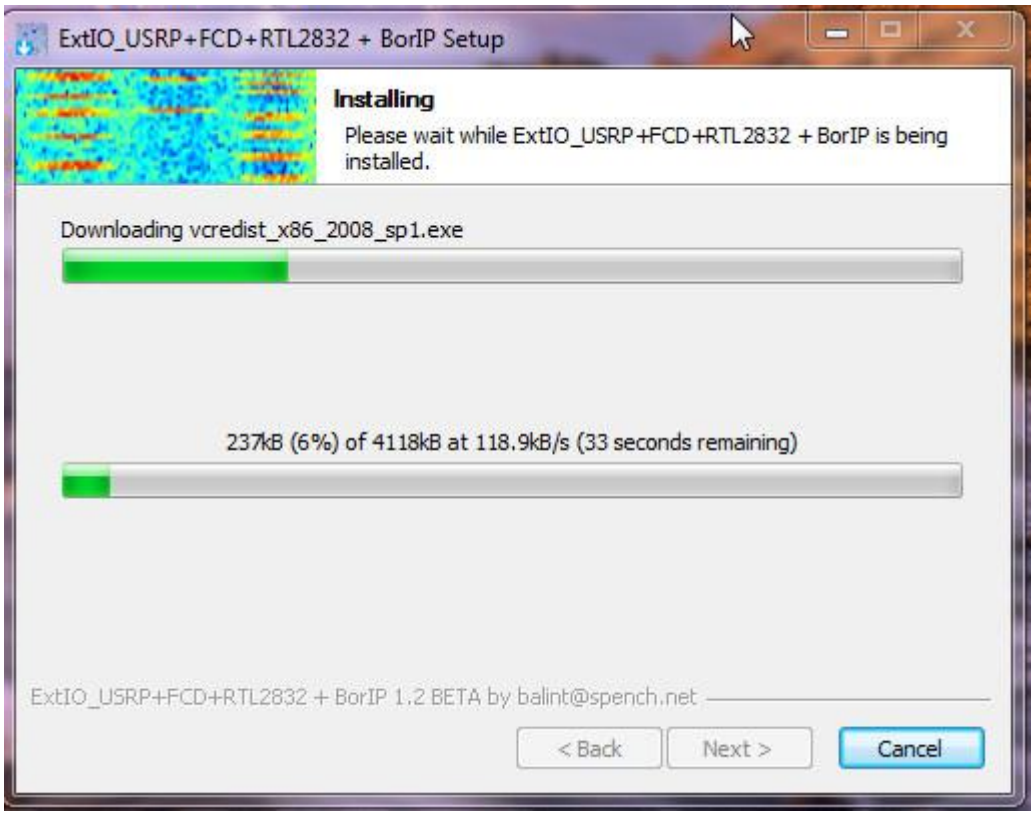


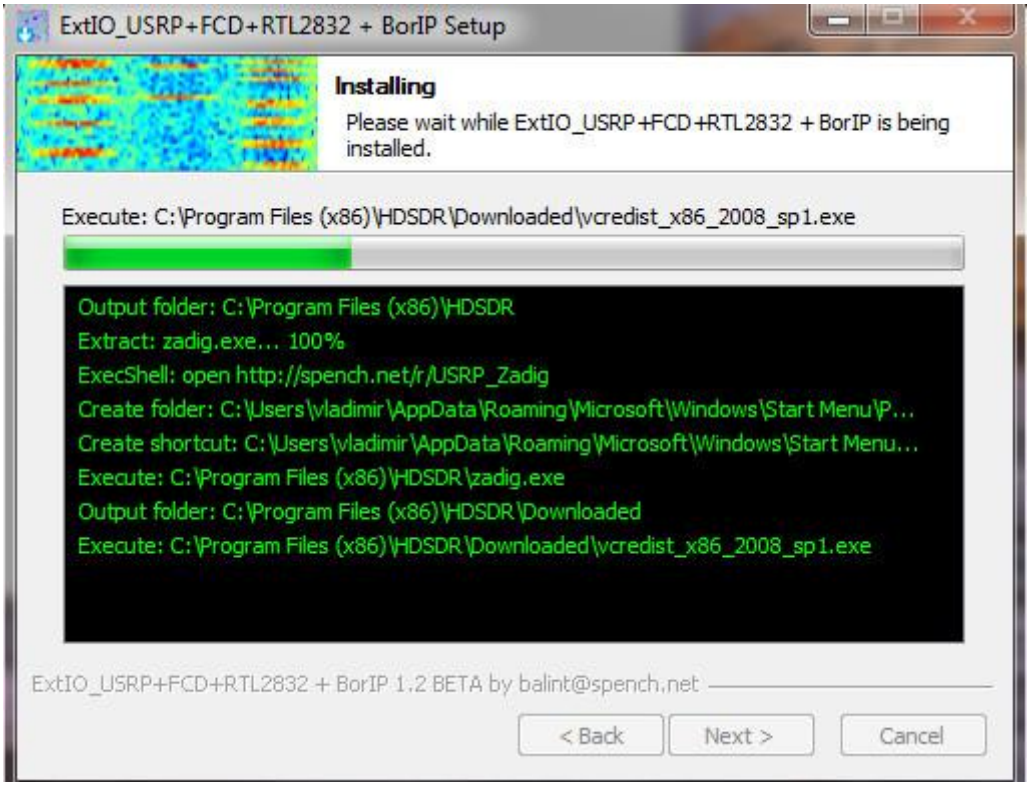


Klikneme na Close

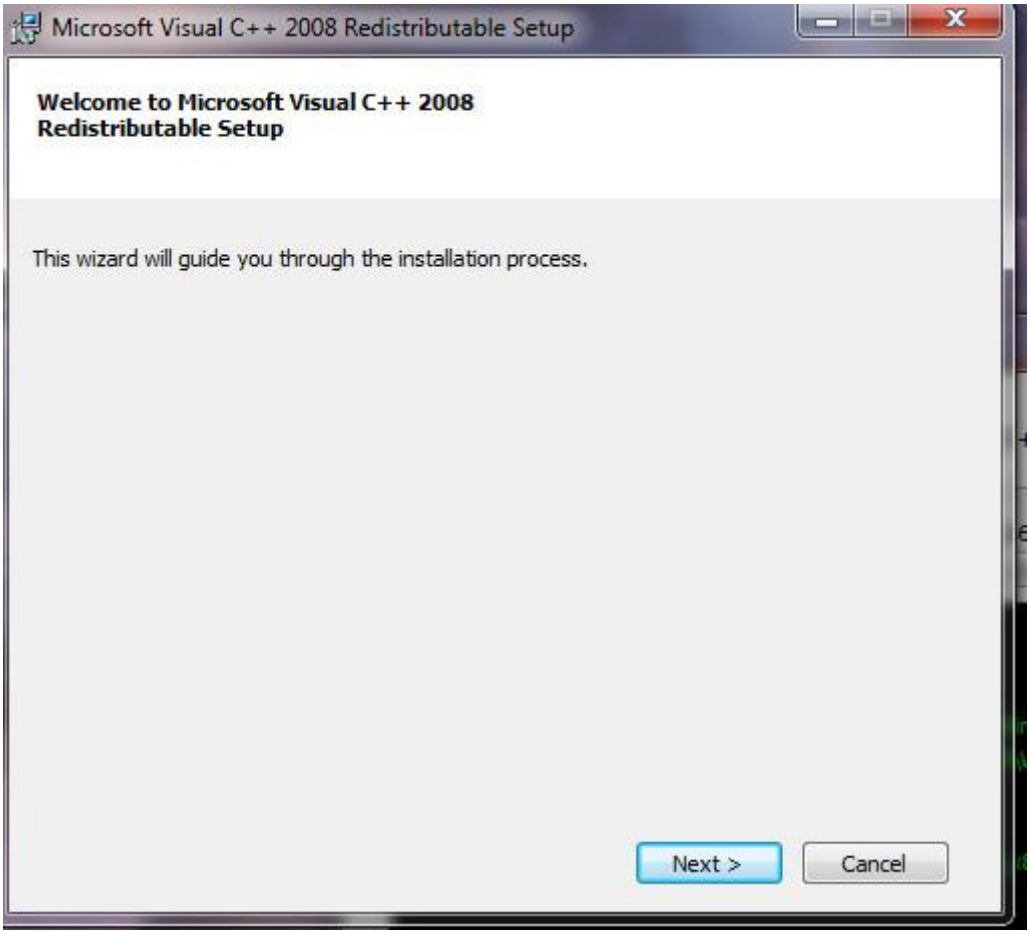


Okno instalace Zadigu nyní zavřeme. Poté se dokončuje instalace

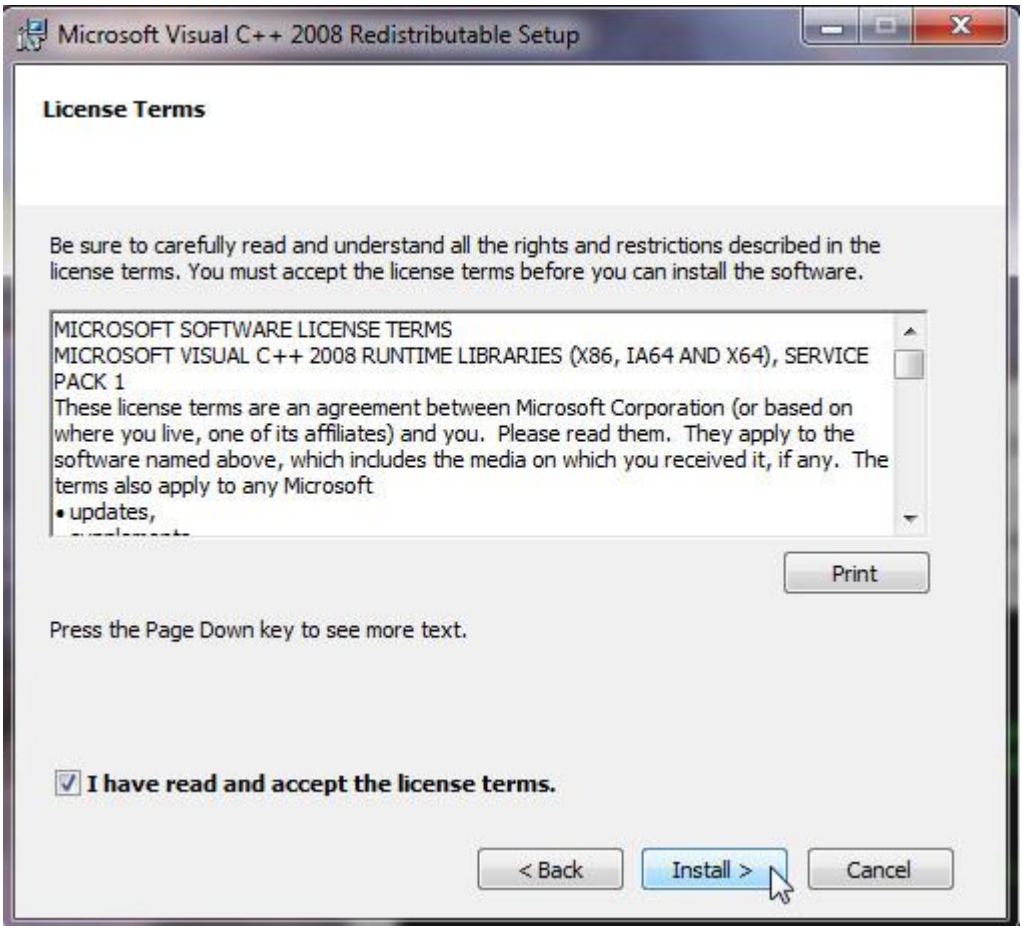



ExtIO_USRP+FCD+RTL2832 + BorIP Setup
Installing
Please wait while ExtIO_USRP+FCD+RTL2832 + BorIP is being installed.
Execute: C:\Program Files (x86)\HDSDR\Downloaded\vcredist_x86_2008_sp1.exe
Output folder: C:\Program Files (x86)\HDSDR
Extract: zadig.exe... 100%
ExecShell: open http://spench.net/r/USRP_Zadig
Create folder: C:\Users\vladimir\AppData\Roaming\Microsoft\Windows\Start Menu\P...
Create shortcut: C:\Users\vladimir\AppData\Roaming\Microsoft\Windows\Start Menu...
Execute: C:\Program Files (x86)\HDSDR\zadig.exe
Output folder: C:\Program Files (x86)\HDSDR\Downloaded
Execute: C:\Program Files (x86)\HDSDR\Downloaded\vcredist_x86_2008_sp1.exe
ExtIO_USRP+FCD+RTL2832 + BorIP 1.2 BETA by balint@spench.net
< Back
Next >
Cancel

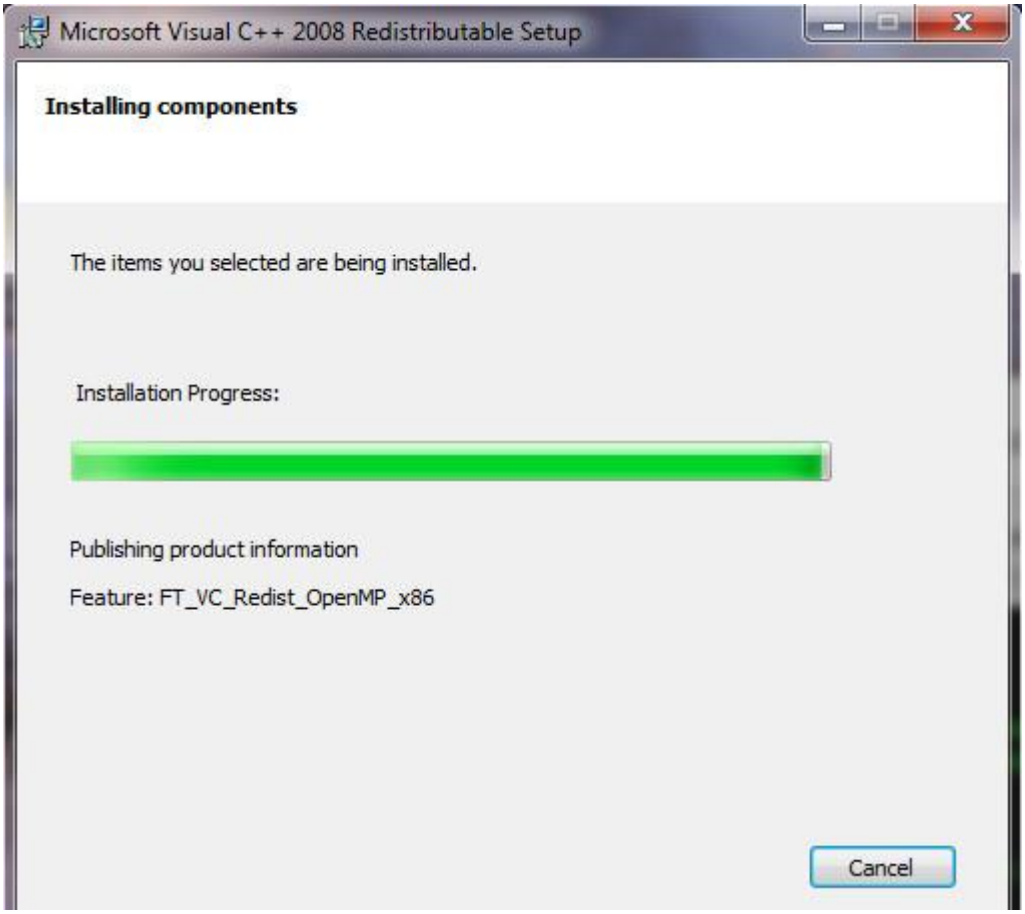
Musíme ale

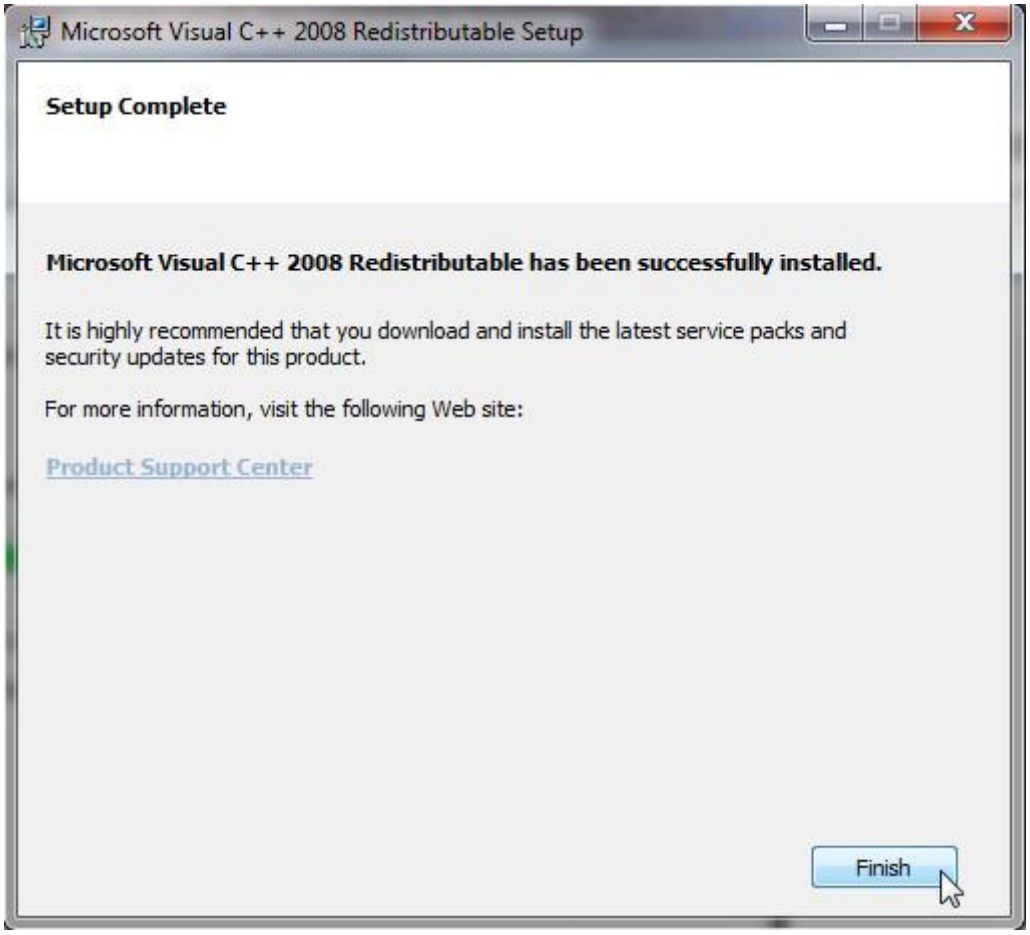

Microsoft Visual C++ 2008 Redistributable Setup
Welcome to Microsoft Visual C++ 2008 Redistributable Setup
This wizard will guide you through the installation process.
Next >
Cancel

Next

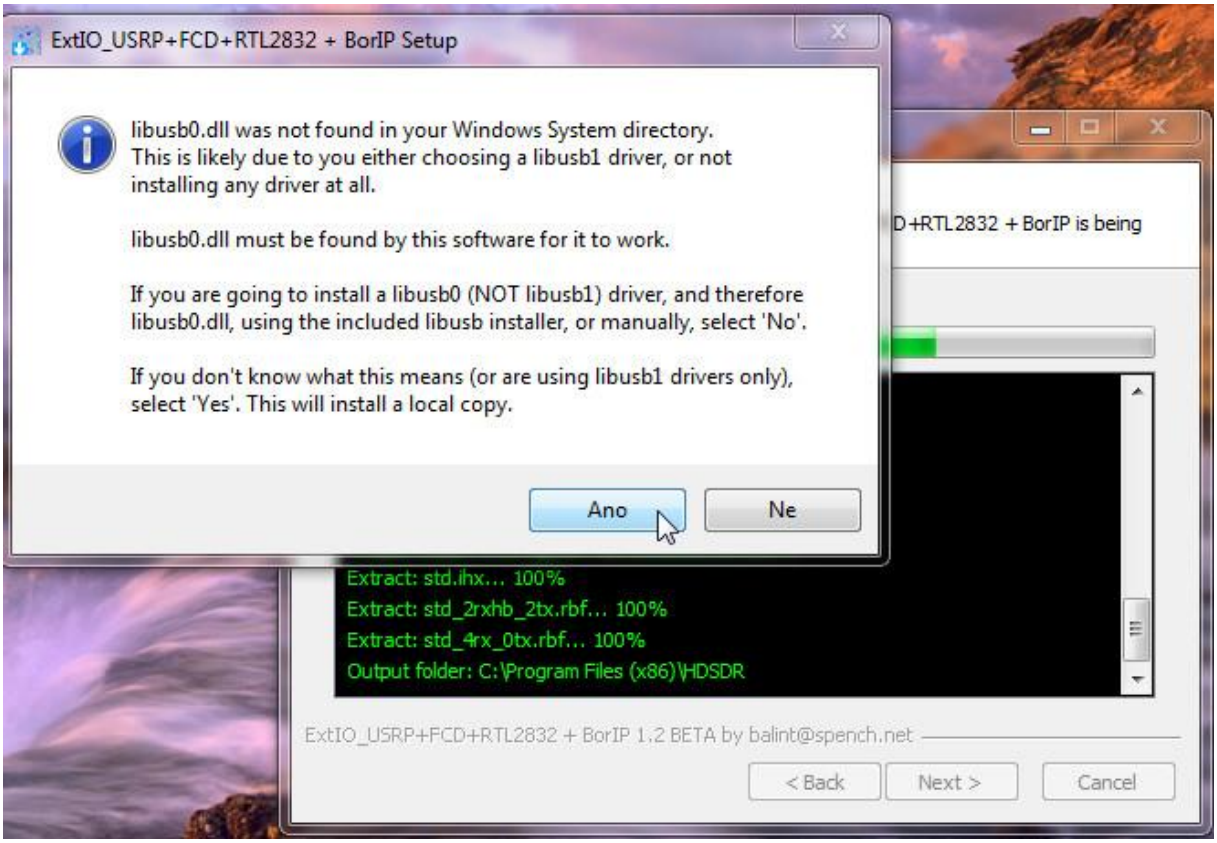


Install

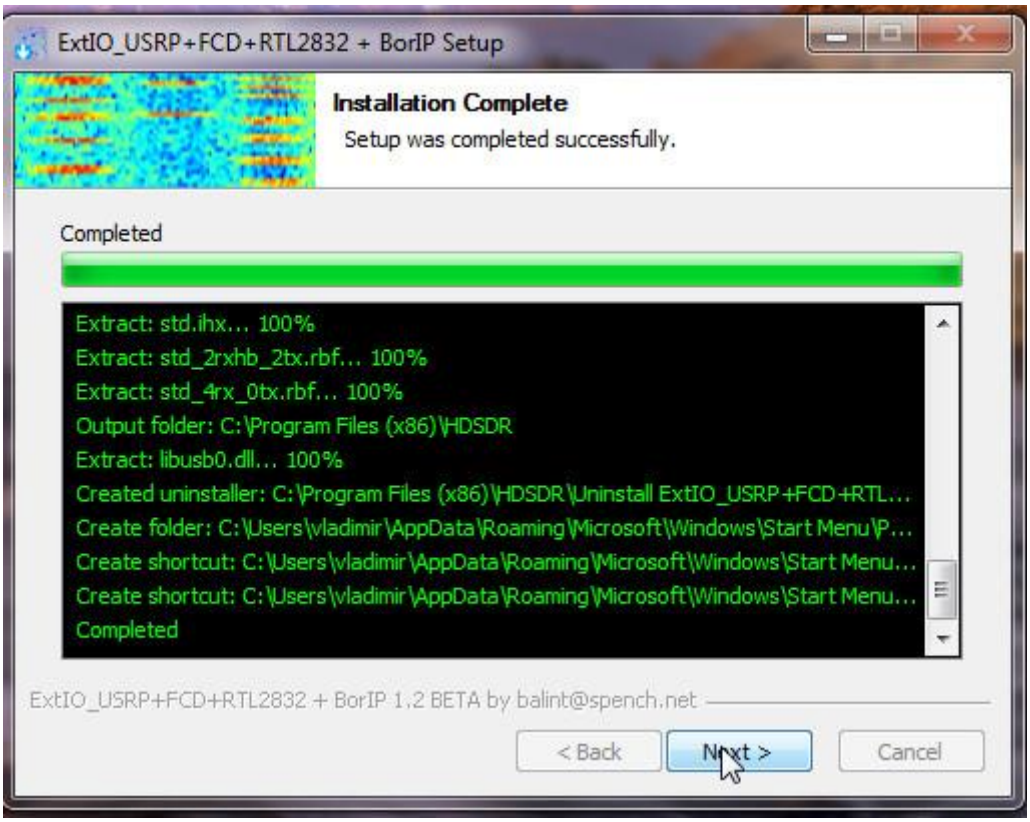

Microsoft Visual C++ 2008 Redistributable Setup
Installing components
The items you selected are being installed.
Installation Progress:
Publishing product information
Feature: FT_VC_Redist_OpenMP_x86
Cancel



Klikneme na Finish

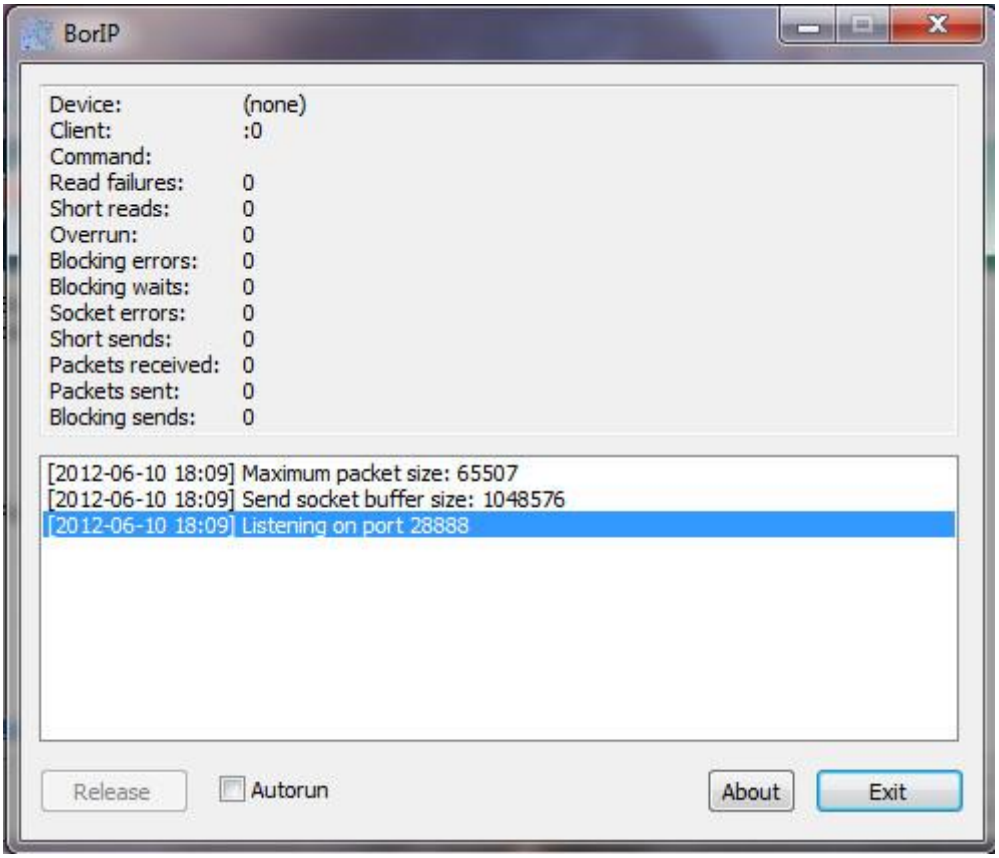

ExtIO_USRP+FCD+RTL2832 + BorIP Setup
libusb0.dll was not found in your Windows System directory.
This is likely due to you either choosing a libusb1 driver, or not installing any driver at all.
libusb0.dll must be found by this software for it to work.
If you are going to install a libusb0 (NOT libusb1) driver, and therefore libusb0.dll, using the included libusb installer, or manually, select 'No'.
If you don't know what this means (or are using libusb1 drivers only), select 'Yes'. This will install a local copy.
Ano
Ne
D+RTL2832 + BorIP is being
Extract: std.ihx... 100%
Extract: std_2rxhb_2tx.rbf... 100%
Extract: std_4rx_0tx.rbf... 100%
Output folder: C:\Program Files (x86)\HDSDR
ExtIO_USRP+FCD+RTL2832 + BorIP 1.2 BETA by balint@spench.net
< Back
Next >
Cancel

Ano

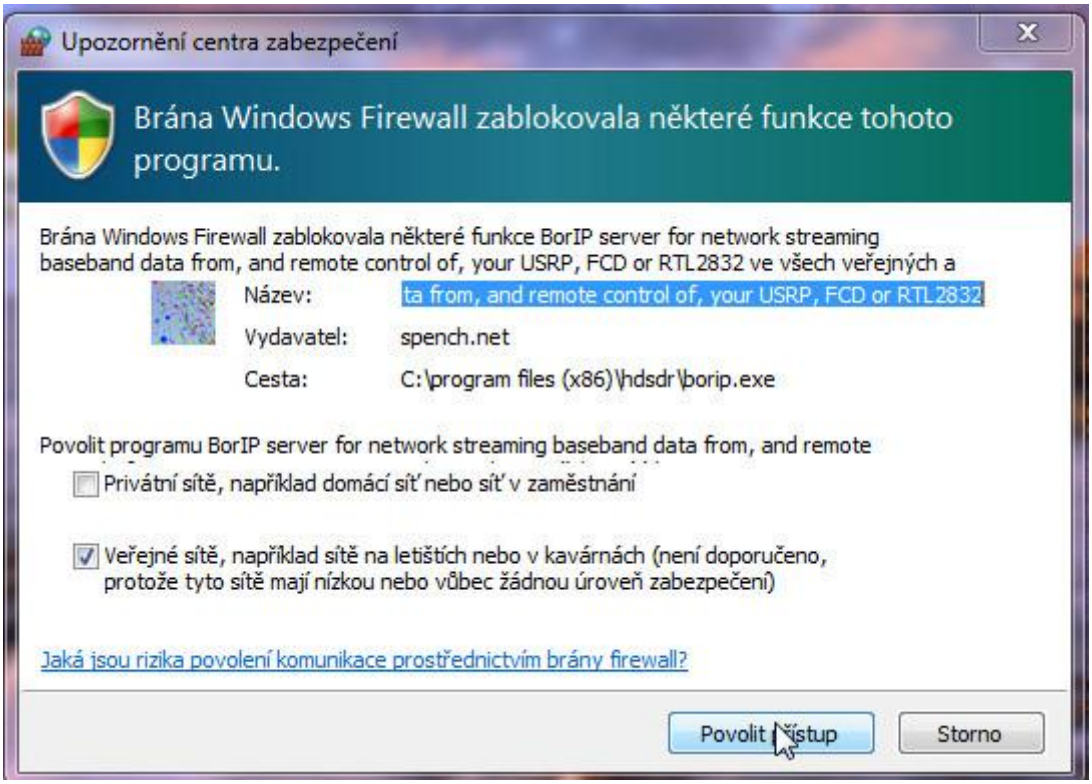

ExtIO_USRP+FCD+RTL2832 + BorIP Setup
Installation Complete
Setup was completed successfully.
Completed
Extract: std.ihx... 100%
Extract: std_2rxhb_2tx.rbf... 100%
Extract: std_4rx_0tx.rbf... 100%
Output folder: C:\Program Files (x86)\HDSDR
Extract: libusb0.dll... 100%
Created uninstaller: C:\Program Files (x86)\HDSDR\Uninstall ExtIO_USRP+FCD+RTL...
Create folder: C:\Users\vladimir\AppData\Roaming\Microsoft\Windows\Start Menu\P...
Create shortcut: C:\Users\vladimir\AppData\Roaming\Microsoft\Windows\Start Menu...
Create shortcut: C:\Users\vladimir\AppData\Roaming\Microsoft\Windows\Start Menu...
Completed
ExtIO_USRP+FCD+RTL2832 + BorIP 1.2 BETA by balint@spench.net
< Back
Next >
Cancel

Klikneme na Next

Klikneme na Finish.

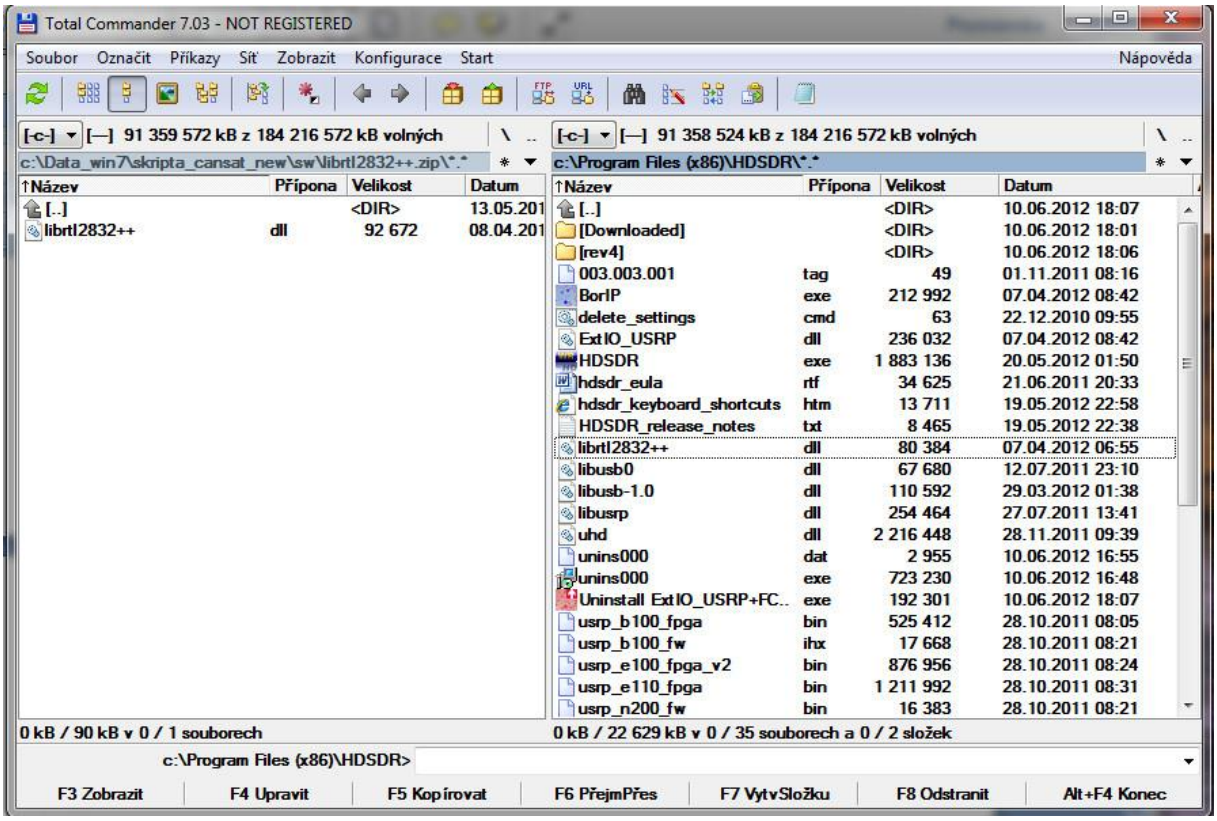


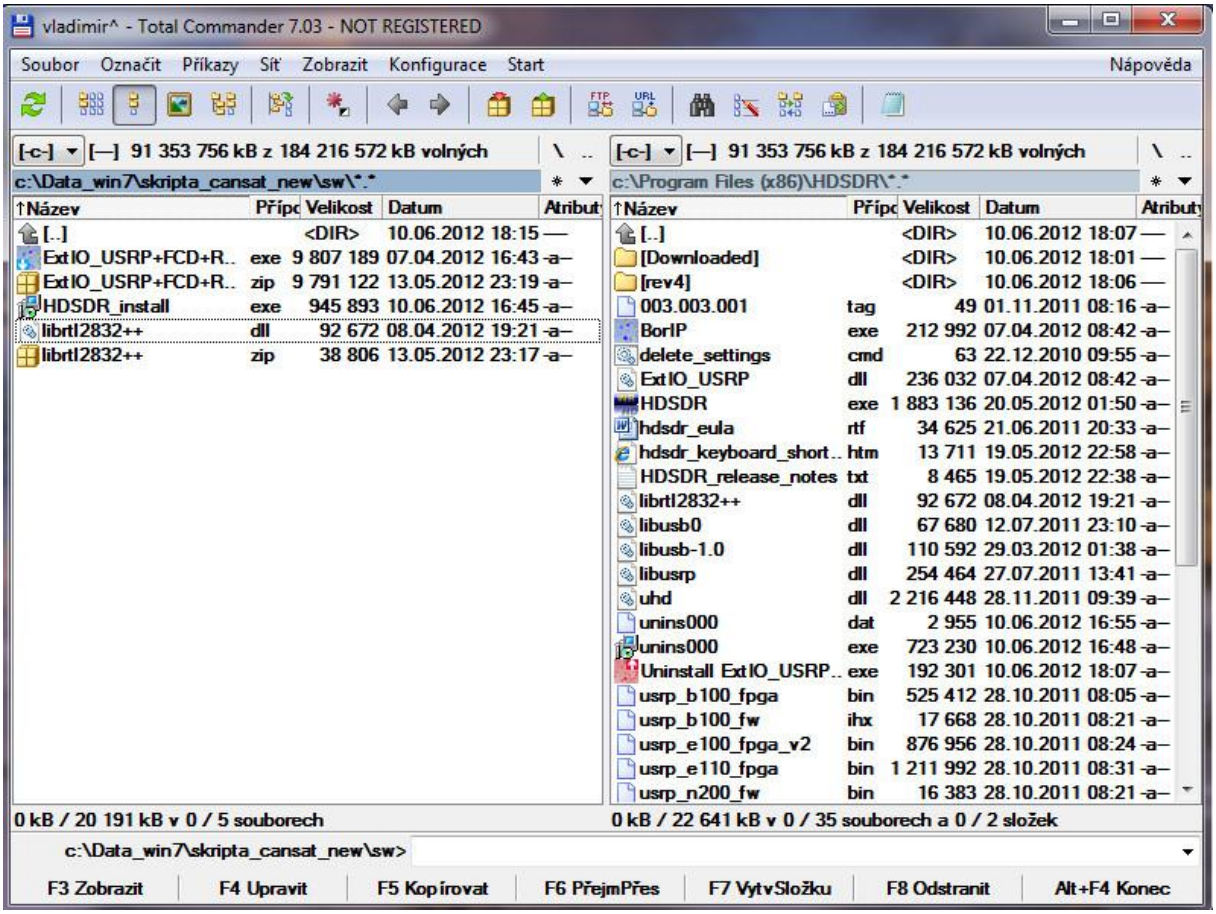
Exit



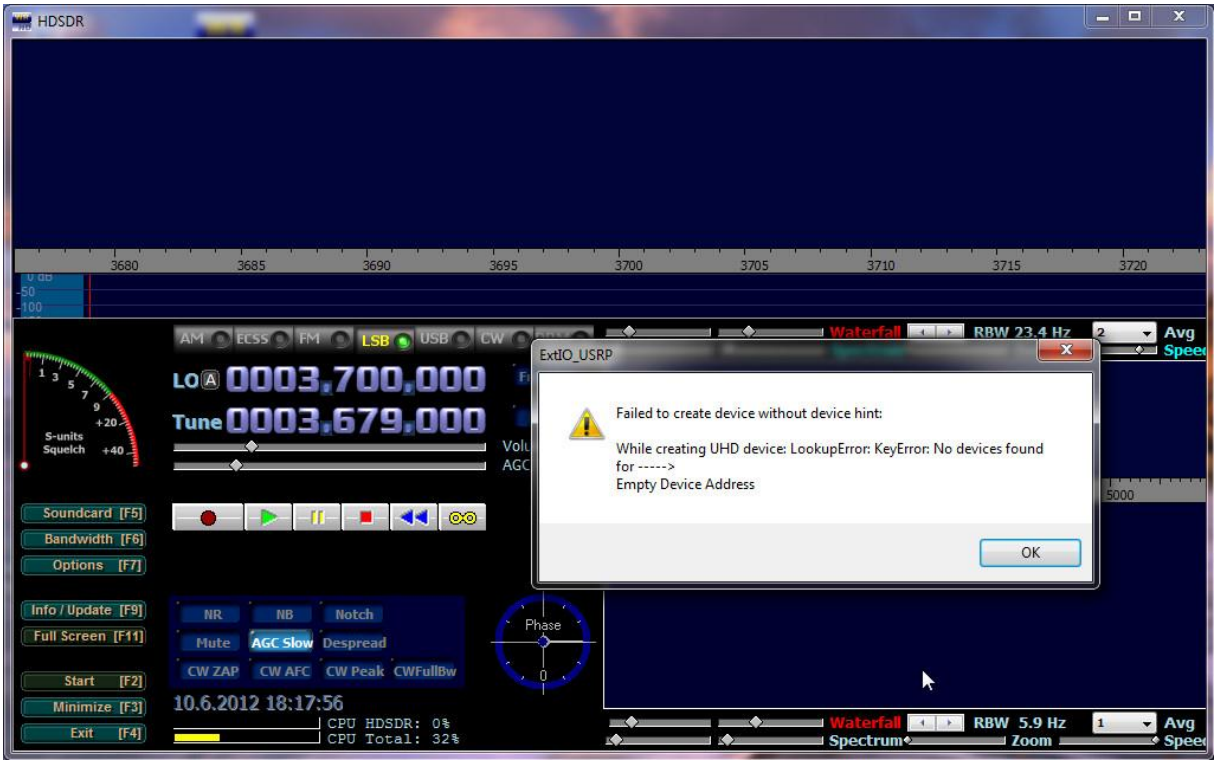
Povolit přístup

Po skončení instalace rozbalíme a nakopírujeme novější verzi knihovny librtl2832++ do složky programu

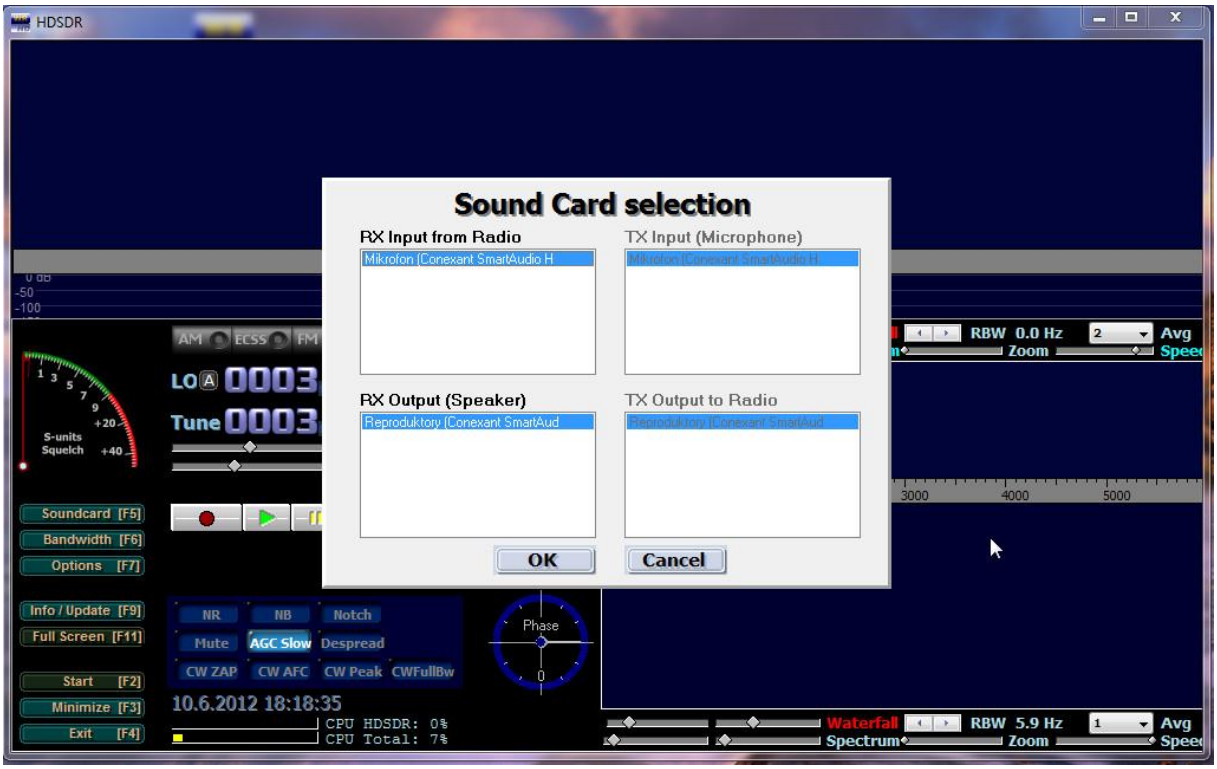




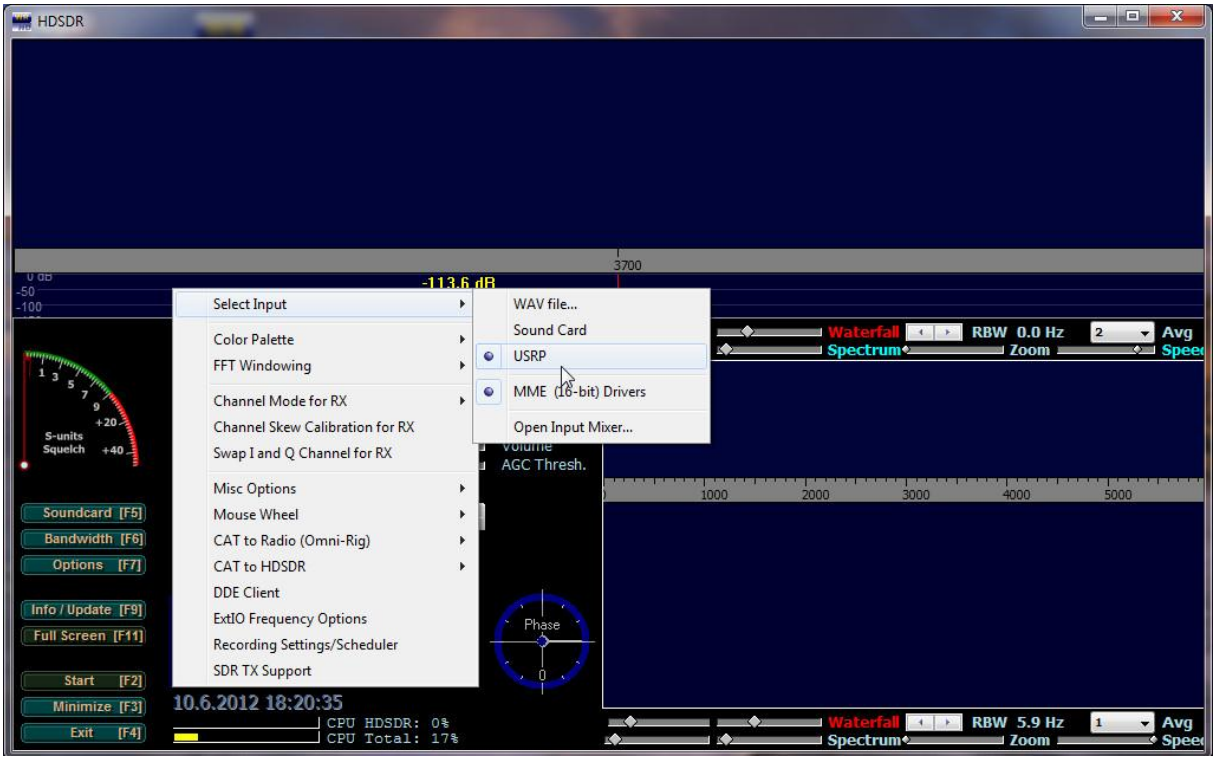
Nyní můžeme spustit HDSDR, plugina ExtIO_USRP se sice načte automaticky, ale musí se ještě **nastavit** pro práci s tunerem.



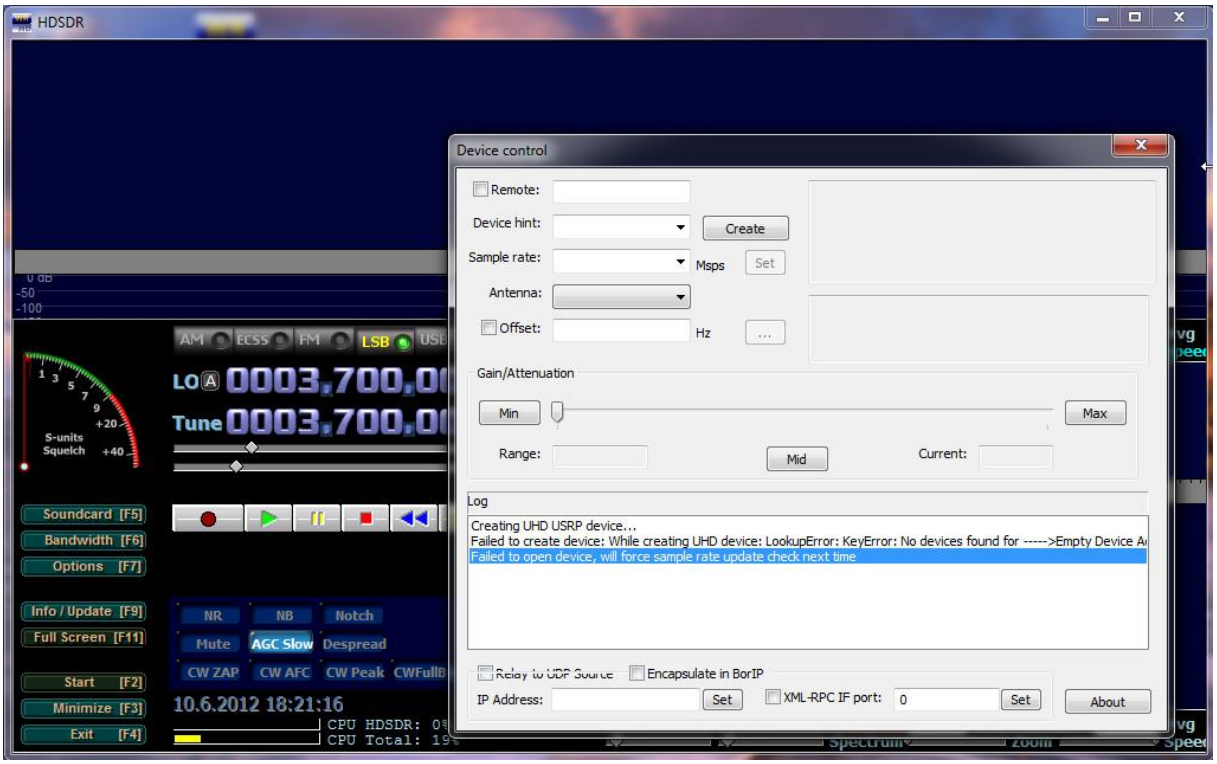
Klikneme na **OK**

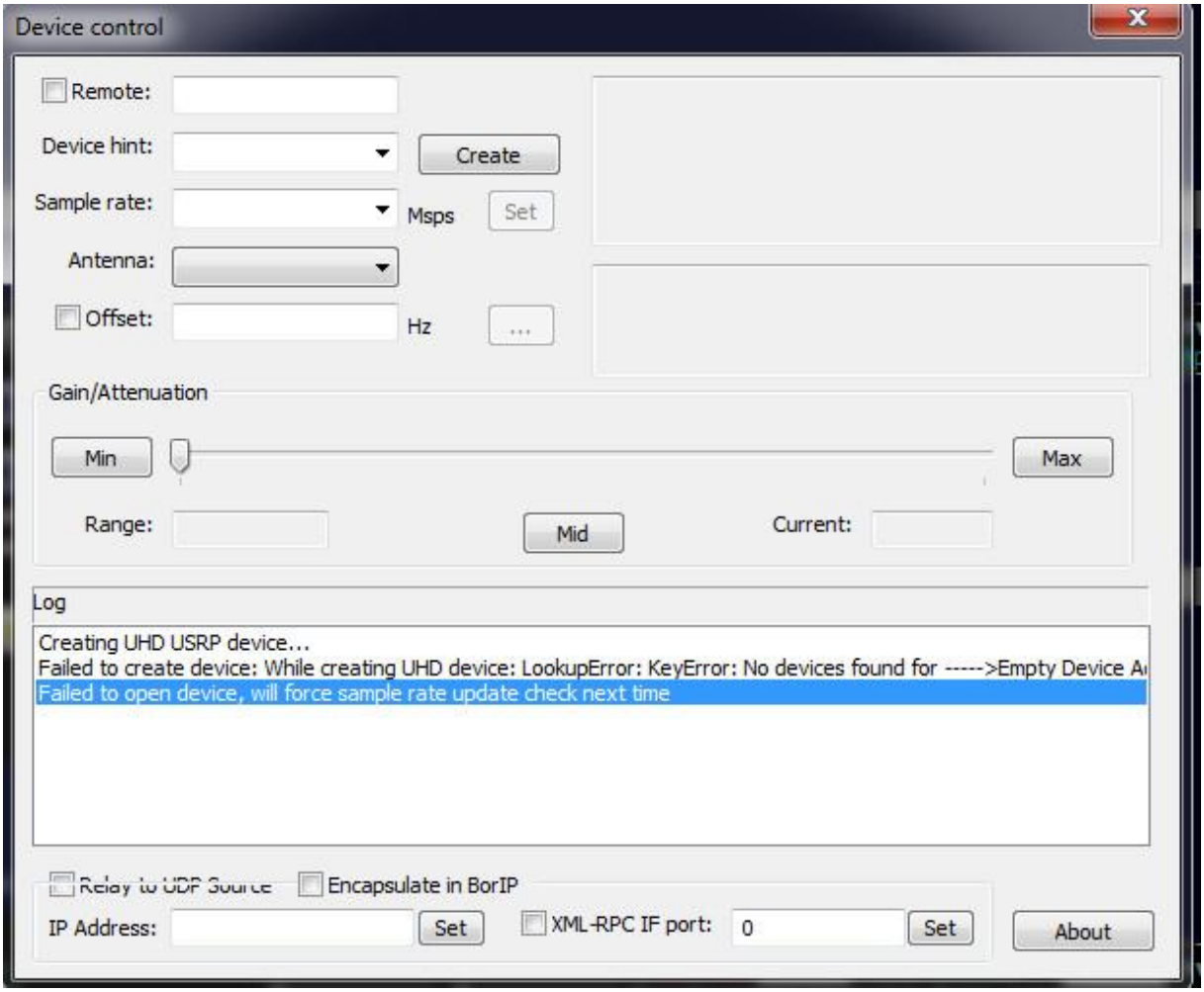


Klikneme na **OK**

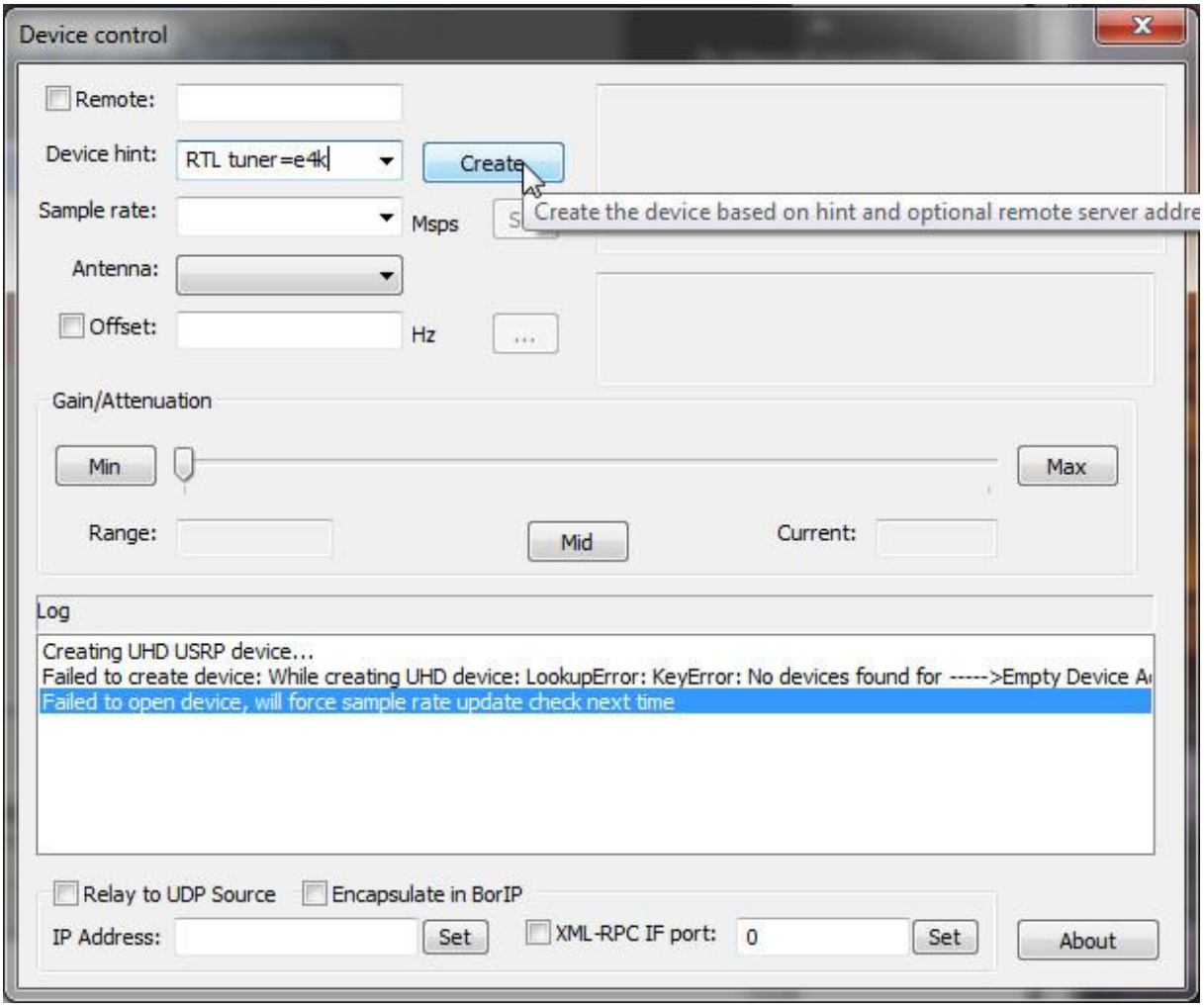


Vybíráme **Options – Select Input – USSP**

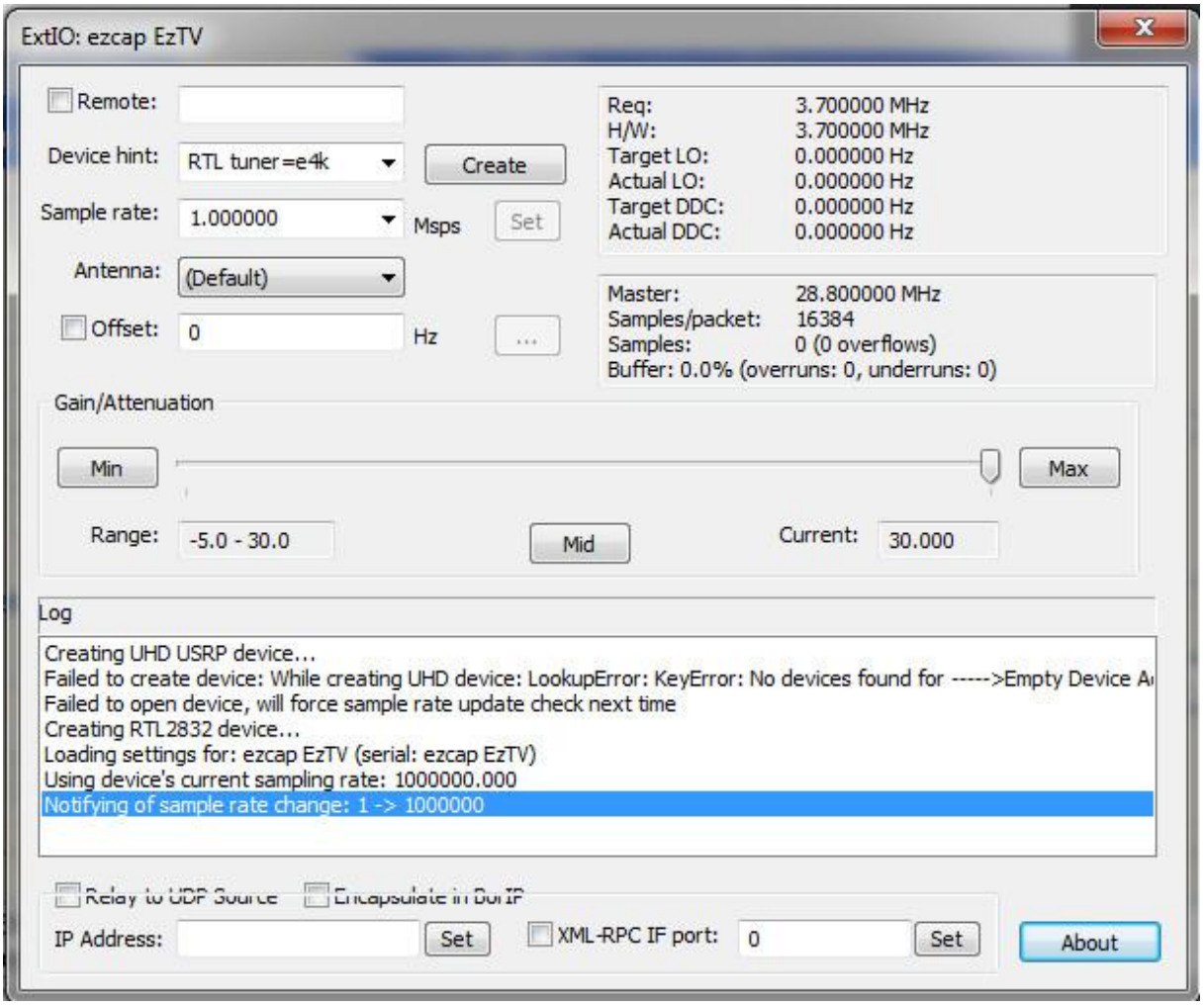




Vyplníme ručně

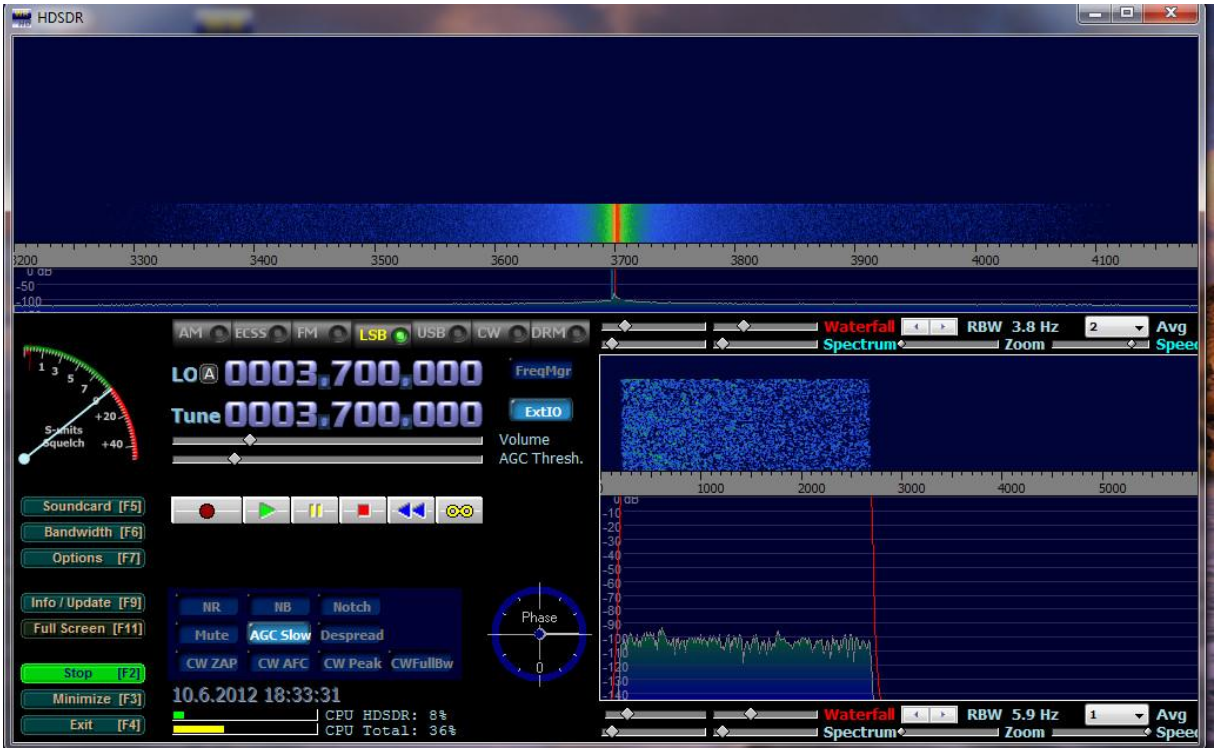


Klikneme na **Create**. Dostaneme něco takovéto



Toto okno zavřeme, máme nyní

Pokud klikneme na **START**, sw začne „přijímat“

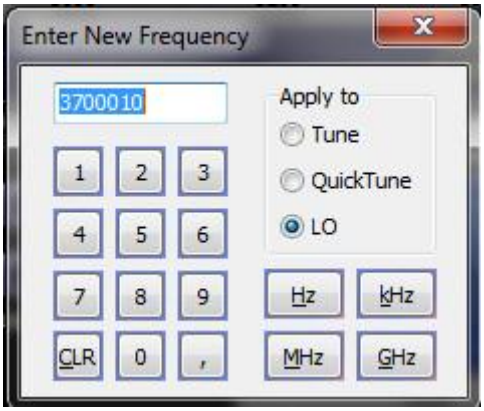


Slovo přijímat jsem napsal do uvozovek, neboť sw nyní „přijímá“ v pásmu 3,5 MHz, což je mimo rozsah 64 – 1700MHz našeho TV tuneru.Navíc máme teď nastaven příjem LSB. Proto nyní přeladíme do 70cm pásma, které používáme pro CANSATy a přepneme na FM provoz

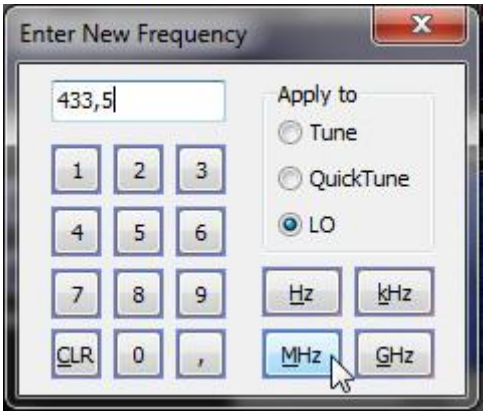
Klikneme na **FM**, dostaneme

Ještě nastavíme kmitočet v pásmu 433 MHz

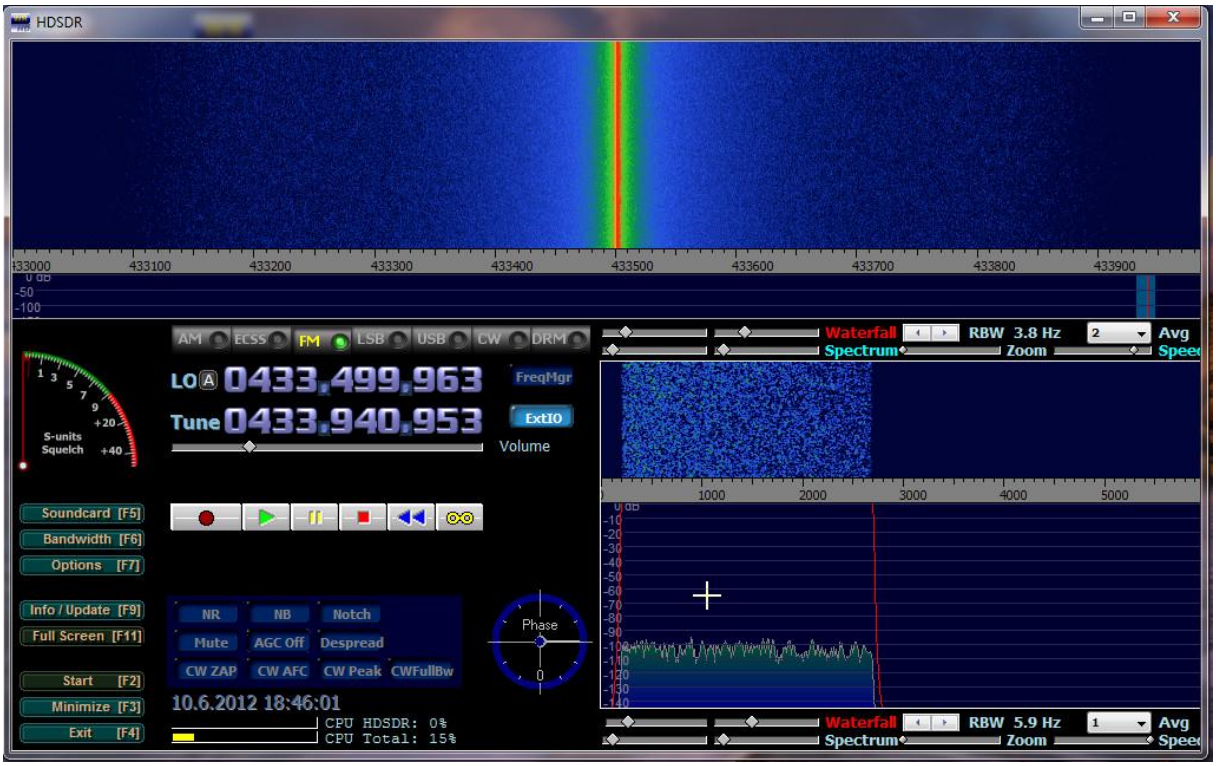
Klikneme na **LO** , dostaneme



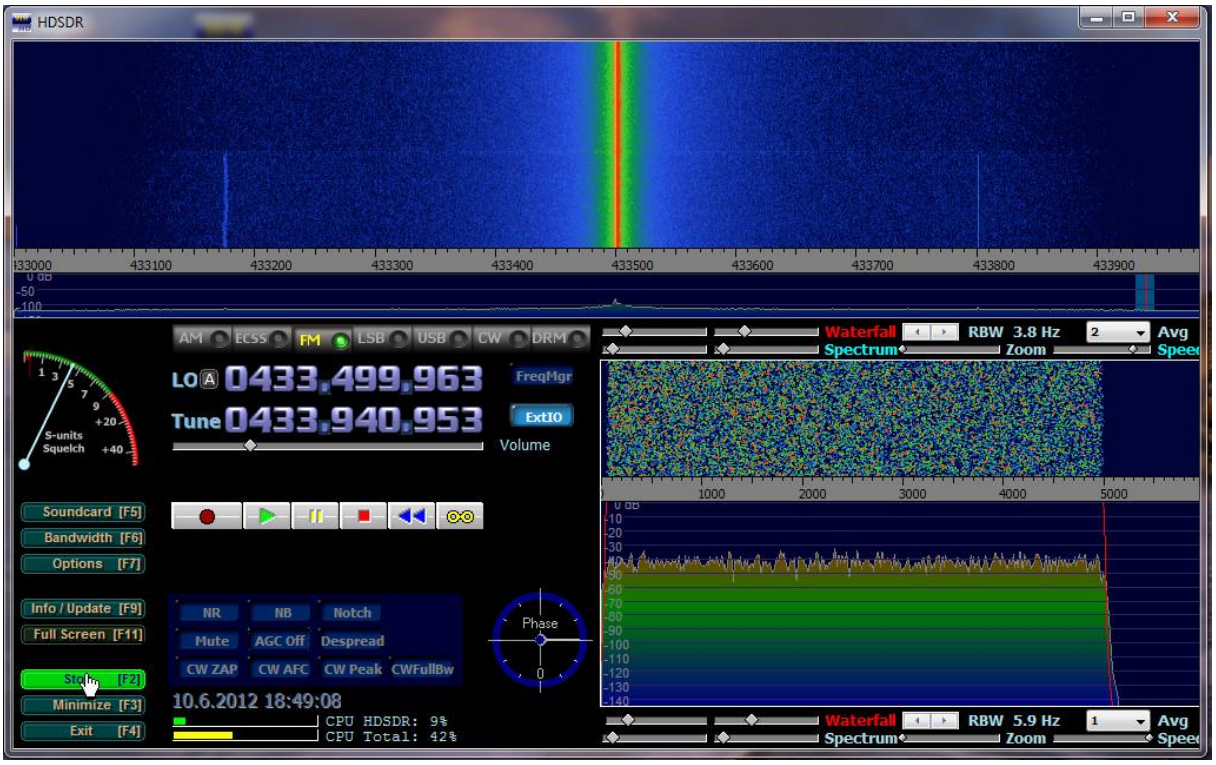
A zvolíme požadované pásmo, např



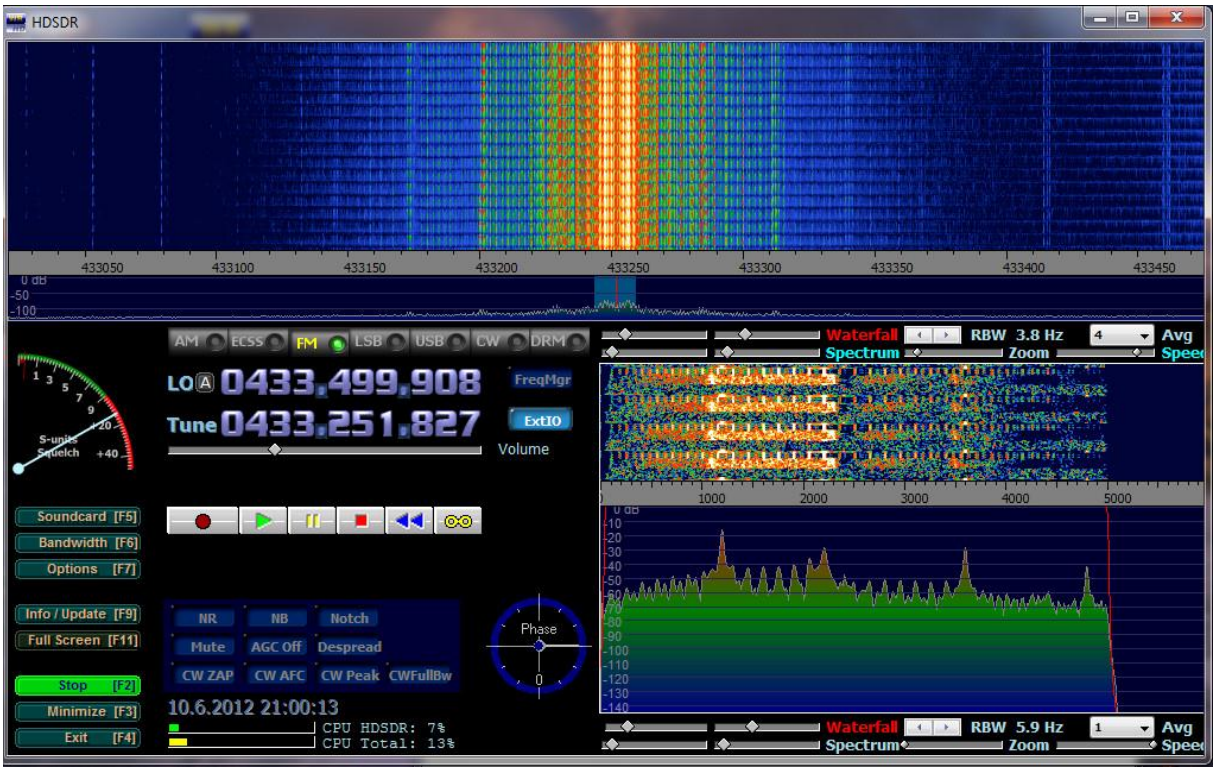
Dostaneme



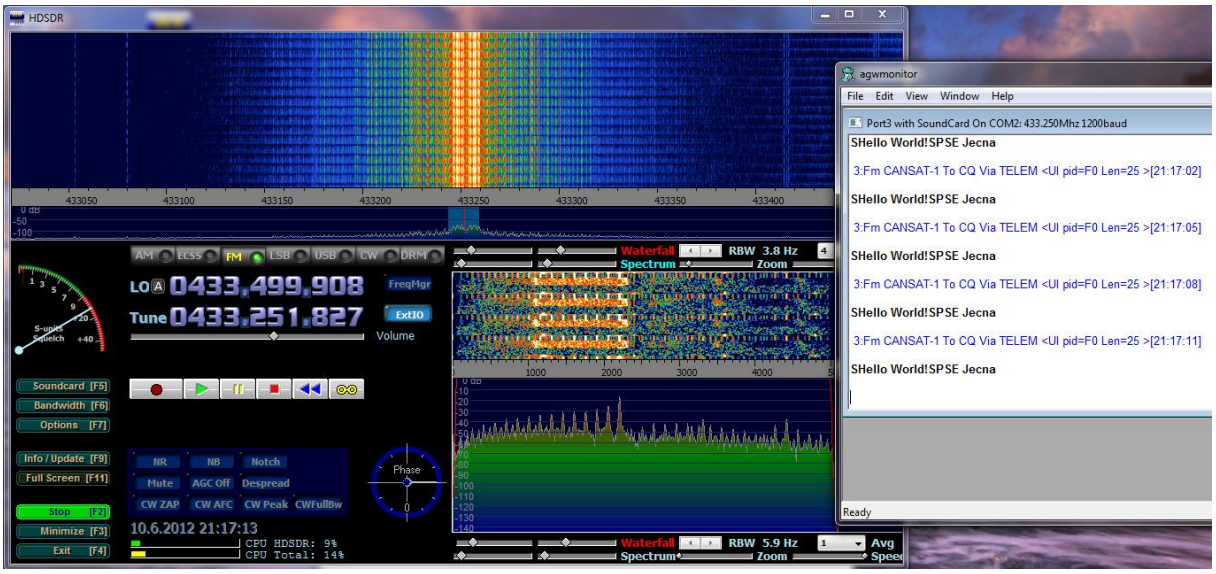
Vidíme, že 433,5 MHz máme uprostřed stupnice. Klikneme na **START** a budeme přijímat signály v pásmu 70cm. Nyní se ještě naučíme s SDR pracovat.

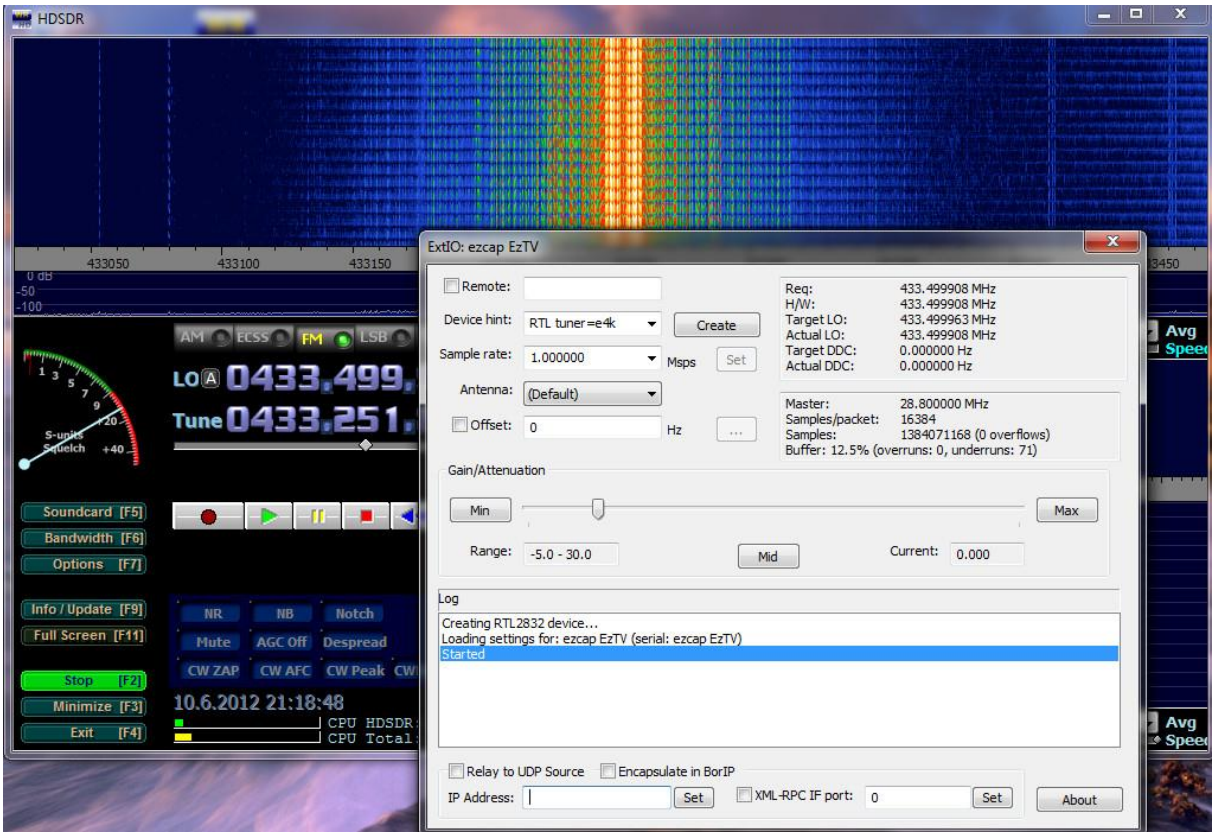


Nyní naladíme na signál vysílače našeho CANSATu

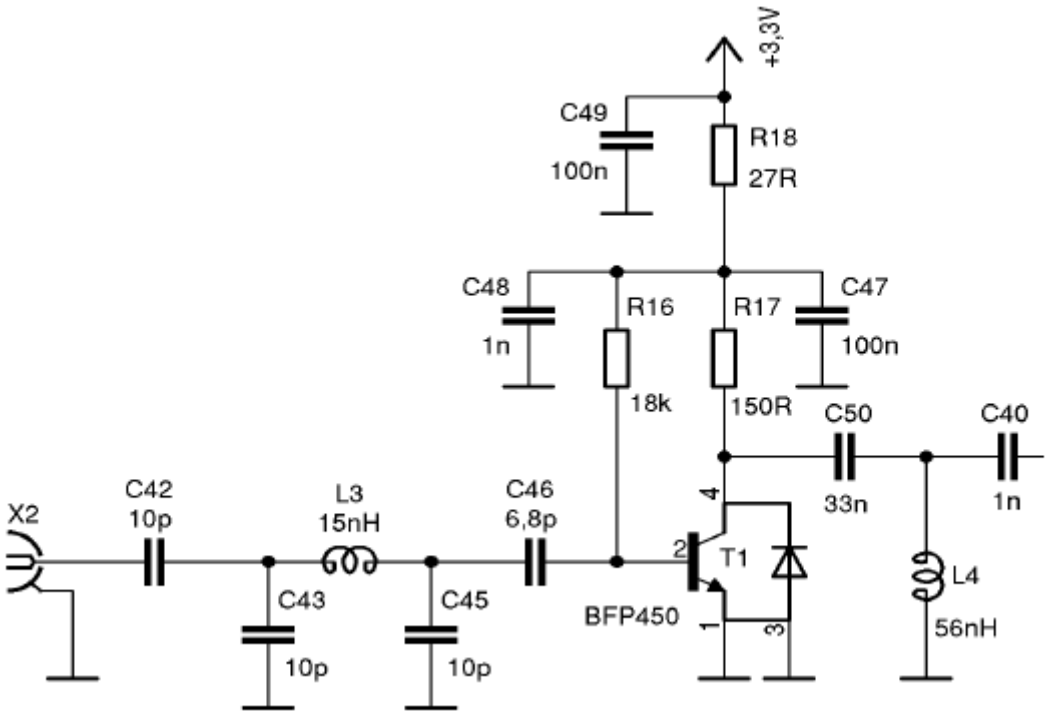


Nyní můžeme použít např. AGW engine a AGW monitor a zobrazovat obsah přijímaných packetů a současně je zapisovat do souboru pro další zpracování jejich obsahu – přenášených dat. Lze přitom nf výstup zvukové karty kabelem spojit s jejím mikrofonním vstupem. Lepší je jako input zdroj ve zvukové kartě zadat mixer a přepnout vstup z mikrofonu na wav.





Před TV tuner pracující spolu s PC jako SDR bude vhodné vřadit nějaký laděný vf. zesilovač, např.



5. Zdroje:

my remarks: *CanSat Book for Students* – part.3 - 2012

[1.1] Váňa V.: *CanSat. Výukový material pro student středních škol*. Díl 1., Praha 2011,
[1.2] Váňa V.: *CanSat. Výukový material pro student středních škol*. Díl 2. – Programování palubního počítače s STM32, Praha 2011,
[2.1] http://www.czechduino.cz/?co-je-to-arduino,29
[3.1] http://cs.wikipedia.org/wiki/MEMS
[3.2] http://www.memsnet.org/mems/what_is.html
[3.3] http://martingurtner.blogspot.cz/
[4.1] http://blog.ok1cdj.com/2012/04/levne-sdr-z-dvb-t-dongle.html
[4.2] http://sdr.osmocom.org/trac/wiki/rtl-sdr
[4.3] http://wiki.spench.net/wiki/USRP_Interfaces#Download
[4.4] http://www.hdsdr.de/