VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

NEZÁVISLÝ DATALOGGER S USB PŘIPOJENÍM

AUTONOMOUS USB DATALOGGER

SEMESTRÁLNÍ PRÁCE

SEMESTRAL THESIS

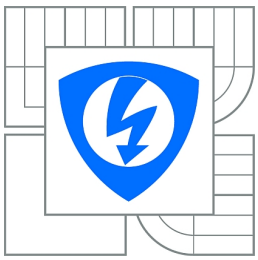
AUTOR PRÁCE Bc. Karel Románek

AUTHOR

VEDOUCÍ PRÁCE prof. Dr. Ing. Zdeněk Kolka

SUPERVISOR

BRNO, 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Karel Románek	***ID:***	72907
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Nezávislý datalogger s USB připojením

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte koncepci nezávislého dataloggeru umožňujícího dlouhodobý záznam teploty a vlhkosti okolního vzduchu s možností vyčítání dat přes USB. Zařízení bude dlouhodobě sledovat teplotu a vlhkost spolu s údajem o reálném čase, a to nezávisle na PC.
Proveďte detailní návrh zařízení s ohledem na minimální spotřebu a efektivní ukládání dat do EEPROM. Nastavování parametrů a nabíjení bude prováděno pomoví rozhraní USB. Proveďte dále návrh mechanická konstrukce odolné proti vlhkosti a mechanickému poškození.
Zařízení realizujte a vytvořte obslužný program pro PC pro konfiguraci a ukládání dat.

DOPORUČENÁ LITERATURA:

[1] AXELSON, J. USB Complete 4ed. Lakeview Research, 2009.
[2] JUNG, W. Op Amp Applications Handbook. Burlington (USA): Newnes (Elesevier), 2005.

Termín zadání:	7.2.2011	***Termín odevzdání:***	20.5.2011

Vedoucí práce: prof. Dr. Ing. Zdeněk Kolka

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Práce pojednává o koncepci nezávislého dataloggeru teploty, vlhkosti a tlaku s možností vyčítání logovaných dat pomocí USB rozhraní do PC. V práci je popsána funkce dataloggeru, návrh hardwaru dataloggeru s ohledem na spotřebu zařízení a návrh mechanické konstrukce zařízení. Dále je rozebrán komunikační protokol pro ovládání a vyčítání dat do PC. Je zde také rozebrán firmware řídící jednotky dataloggeru, tedy drivery senzorů, moduly USB, RTC a komprese logovaných dat. V další části práce je popsán software pro ovládání dataloggeru a stahování logovaných dat.

KLÍČOVÁ SLOVA

Datalogger, měření teploty, měření vlhkosti, měření tlaku, třída HID, diferenční kódování, Huffmanovo kódování, STM32 firmware

ABSTRACT

This thesis treats concept of autonomous temperature, relative humidity and pressure USB datalogger. Also there is explained datalogger function, hardware design with respect on device consumption and design of chassis. Furthermore, there is described communication protocol for control and reading out data by the PC. Furthermore, there are described firmware drivers for some used components and modules for USB communication, RTC and data compression. Lastly there is described software which is used for datalogger configuration and data read out.

KEYWORDS

Datalogger, temperature measurement, humidity measurement, pressure measurement, HID class, differential and Huffman coding, STM32 firmware

Románek, K. *Nezávislý datalogger s USB připojením*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 102 s., 8 příloh. Vedoucí diplomové práce: prof. Dr. Ing. Zdeněk Kolka.

PROHLÁŠENÍ

Jako autor diplomové práce na téma „Nezávislý datalogger s USB připojením" dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji konzultantovi diplomové práce ze společnosti ON Semiconductor Ing. Markovi Bekovi za účinnou metodickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce. Také děkuji společnosti ON Semiconductor za poskytnutí finančních a materiálových prostředků a také zázemí při práci na projektu.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Zadáním této diplomové práce je návrh a realizace nezávislého dataloggeru relativní vlhkosti a teploty s připojením k PC pomocí USB rozhraní. Hlavními cíli je vytvoření hardwaru zařízení, firmwaru řídící jednotky a ovládací aplikace pro PC.

Návrh hardwaru spočívá ve výběru metod měření atmosférických veličin teploty a relativní vlhkosti vzduchu s možným rozšířením jako je měření atmosférického tlaku. Klíčový je výběr jednotlivých komponentů zařízení, především senzorů pro snímání atmosférických veličin a integrovaných obvodů, s ohledem na funkci a spotřebu zařízení. S tímto souvisí také návrh a realizace desky plošných spojů a mechanické konstrukce zařízení.

Návrh by měl také pokrýt možnost kontroly stavu zařízení během dlouhodobých záznamů bez nutnosti připojení k PC. Spolu s koncepcí hardwaru je potřeba vytvořit návrh ukládání zaznamenávaných dat do paměti zařízení s možností jejich komprese a komunikační protokol, který bude použit pro přenos dat mezi zařízením a PC.

Firmware zařízení se skládá převážně z ovladačů senzorů, modulů pro práci se změřenými daty, RTC jednotkou a USB periferií, což koresponduje s potřebnými funkcemi a vlastnostmi zařízení.

Software pro PC musí být schopen obousměrně přenášet data mezi PC a zařízením pomocí předem definovaného komunikačního protokolu. Stěžejní je konfigurace logování, stahování zaznamenaných dat a jejich následné zpracování.

1 POPIS NAVRHOVANÉHO DATALOGGERU

Datalogger slouží k dlouhodobému zaznamenávání atmosférických veličin. Zařízení umožňuje záznam teploty, relativní vlhkosti a atmosférického tlaku vzduchu. K zařízení je navíc pomocí kabelu připojen teplotní senzor, který může být umístěn na vzdáleném stanovišti. Ze změřených hodnot teploty a tlaku je také možné přibližně určit nadmořskou výšku stanoviště.

Zaznamenaná data, včetně údajů o datu a čase záznamů, jsou ukládána v paměti zařízení, kde jsou uchována i po vybití baterie. Aktuální atmosférické veličiny mohou být změřeny a zobrazeny na LCD displeji mimo intervaly záznamu dat. Na LCD displeji mohou být také zobrazeny provozní informace - aktuální konfigurace logování, stav baterie, kapacita paměti, aktuální čas a datum.

Veškerá konfigurace logování se provádí pomocí ovládací aplikace po připojení dataloggeru pomocí USB rozhraní k PC. Ovládací aplikace synchronizuje údaj o reálném čase v dataloggeru. Je zde možné nastavit interval záznamů, datum a čas začátku a konce měření. Po připojení zařízení k PC je také spuštěno nabíjení baterie z USB.

Pomocí ovládací aplikace mohou být zaznamenaná data stažena do PC a vypsána ve formě tabulky, vykreslena ve formě grafů a exportována do souboru pro další zpracování.

2 TEORETICKÝ ROZBOR PROBLEMATIKY

V následujícím textu jsou rozebrány možnosti měření teploty a relativní vlhkosti pomocí elektrických obvodů.

2.1 Měření teploty

Teplota je skalární veličina vyjadřující úroveň tepelné energie hmoty. Pro měření teploty je možné využít několik fyzikálních jevů, a to tepelné roztažnosti materiálů, změny měrného odporu materiálu, principu termoelektrického jevu a změny intenzity hmotou vyzařované energie. Pro elektrické měření teploty prostředí se používají termočlánky a odporové čidla.

Termočlánky využívají termoelektrického jevu dvou, na obou koncích spojených, vodičů z rozdílného kovu. Je-li mezi srovnávacím a měřícím koncem vedení rozdíl teploty, dochází ke vzniku termoelektrického napětí, které je úměrné měřené teplotě. Termočlánky se dělí podle druhu použitých materiálů a rozsahu měřených teplot. V průmyslu se nejčastěji používají termočlánky typu J a K.

Odporová čidla RTD využívají změny elektrického odporu vodičů a polovodičů. Odporové snímače nejčastěji využívají platinového drátu, jelikož mezi ostatními kovy vykazuje nejlepší linearitu závislosti teplotní závislost odporu. RTD mají definovaný odpor při teplotě 0°C a 100°C. Příkladem je platinový teplotní snímač Pt100. Další skupinou RTD jsou termistory využívající polovodičové materiály. Pro měření teploty lze použít pouze termistory se záporným teplotním koeficientem elektrického odporu.

Integrovaná řešení teplotních čidel většinou obsahují samotný teplotní snímač a obvody pro zpracování signálu, kdy výstupem může být stejnosměrné napětí řádově jednotek voltů, lineárně závislé na měřené teplotě, střídavý šířkově modulovaný obdélníkový signál nebo digitální reprezentace měřené teploty přenášená po sériové lince.

2.2 Měření vlhkosti

Přístroje pro měření vlhkosti obvykle měří relativní vlhkost. Relativní vlhkost vyjadřuje množství vodní páry v určitém množství vzduchu. Relativní vlhkost je možné měřit pomocí kapacitních nebo rezistivních čidel.

Kapacitní čidla se skládají ze substrátu, na kterém je nanesena tenká vrstva polymeru nebo oxidu kovu uloženým mezi dvěma vodivými elektrodami. Kapacita čidla je takřka lineárně závislá s relativní vlhkostí.

Rezistivní čidla se skládají ze substrátu, na kterém jsou vytvořeny elektrody z drahého kovu potažené vodivým polymerem. Závislost odporu čidla na relativní vlhkosti je exponenciální.

Integrovaná řešení čidel pro měření relativní vlhkosti bývají již kalibrovaná a tepelně kompenzovaná. V některých případech dokonce disponují i teplotním snímačem. I v případě integrovaných modulů vlhkostních čidel jsou k dispozici varianty s digitálním datovým výstupem.

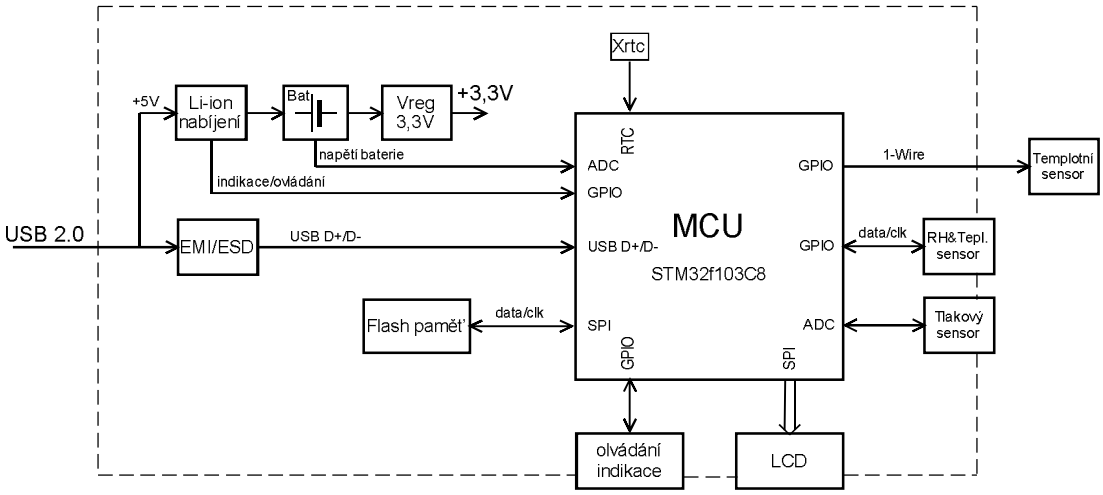
3 KONCEPCE A NÁVRH DATALOGGERU

K tomu, aby datalogger splňoval stanovené funkce, byla vytvořena koncepce a návrh obvodového zapojení. V této kapitole jsou koncepce a obvodový návrh popsány se zaměřením na výběr jednotlivých komponent.

Návrh vychází z požadavků na jednoduchost měření atmosférických veličin bez nutnosti složité kalibrace senzorů, minimální spotřebu, možnosti nabíjení baterie a přímého připojení k USB bez nutnosti použití převodníku a v neposlední řadě na malou velikost zařízení.

3.1 Blokové schéma

Blokové schéma (obr. 3.1) znázorňuje principiální zapojení dataloggeru.



Obr. 3.1 Blokové schéma dataloggeru

MCU	řídící jednotka, 32bitový mikrokontrolér STM32103Zx s jádrem ARM Cortex™-M3
Flash paměť	paměť sloužící k uložení logovaných dat
EMI/ESD	propojení PC s MCU, zajišťuje filtraci EMI linky a ochranu proti ESD
Bat	Li-Ion baterie pro napájení zařízení
Li-Ion nabíjení	vestavěná USB nabíječka Li-Ion baterie
Vreg	stabilizace napájecího napětí
Sensory	měřící senzory teploty, vlhkosti a tlaku
LCD	LCD displej zobrazující aktuální veličiny, stav baterie, kapacitu paměti

3.2 Popis funkce dataloggeru

Veškeré operace zařízení řídí a kontroluje MCU. Spouští měření atmosférických veličin pomocí senzorů, komprimuje a ukládá získaná data do flash paměti, případně zobrazuje provozní informace na LCD displeji.

Napájecí větev obvodu se skládá z nabíječky, Li-Ion baterie a napěťového stabilizátoru. MCU také sleduje stav baterie a řídí nabíjecí proces, pokud je datalogger připojen k USB lince a zajišťuje přenos dat do PC.

3.3 Výběr komponent

3.3.1 Čidlo pro měření relativní vlhkosti a teploty vzduchu

Pro měření relativní vlhkosti bylo vybráno čidlo STH15 [3]. Je složeno z „band gap" snímače pro měření teploty a kapacitního čidla pro měření relativní vlhkosti, A/D převodníku a jednotky pro zpracování signálů. Měřené hodnoty vlhkosti a teploty jsou kalibrovány a převedeny na digitální hodnoty.

Čidlo pracuje v prostředí, kde nedochází ke kondenzaci, avšak ze zaznamenaných hodnot lze teplotu rosného bodu spočítat [3].

Čidlo disponuje dvouvodičovým sériovým rozhraním pro sběr měřených dat. Pro získání reálných hodnot měřených veličin musí být data získaná z čidla převedena pomocí následujících vztahů.

Převodní vztahy [3] :

– Teplota

$$\mathrm{t}\,(°\mathrm{C}) = \mathrm{d}_1 + \mathrm{d}_2 \cdot D_\mathrm{t} = 39{,}6 + 0{,}01 \cdot \mathrm{D}_\mathrm{t} \tag{3.1}$$

kde d_1 a d_2 jsou převodní koeficienty dané výrobcem, D_t je digitální hodnota teploty přijatá z čidla.

– Relativní vlhkost

$$\mathrm{RH}\,(\%) = (t_{°\mathrm{C}} - 25) \cdot (\mathrm{t}_1 + \mathrm{t}_2 \cdot \mathrm{D}_\mathrm{RH}) + \mathrm{c}_1 + \mathrm{c}_2 \cdot \mathrm{D}_\mathrm{RH} + \mathrm{c}_3 \cdot {\mathrm{D}_\mathrm{RH}}^2$$

$$\mathrm{RH}\,(\%) = (t_{°\mathrm{C}} - 25) \cdot (0{,}01 + 0{,}00008 \cdot \mathrm{D}_\mathrm{RH}) - 2{,}0468 + 0{,}0367 \cdot \mathrm{D}_\mathrm{RH} - \tag{3.2}$$

$$-1{,}5955 \cdot 10^{-6} \cdot {\mathrm{D}_\mathrm{RH}}^2$$

kde $t_{°C}$ je aktuální teplota, t_1, t_2, c_1, c_2 a c_3 jsou převodní koeficienty dané výrobcem, D_{RH} je digitální hodnota relativní vlhkosti přijatá z čidla.

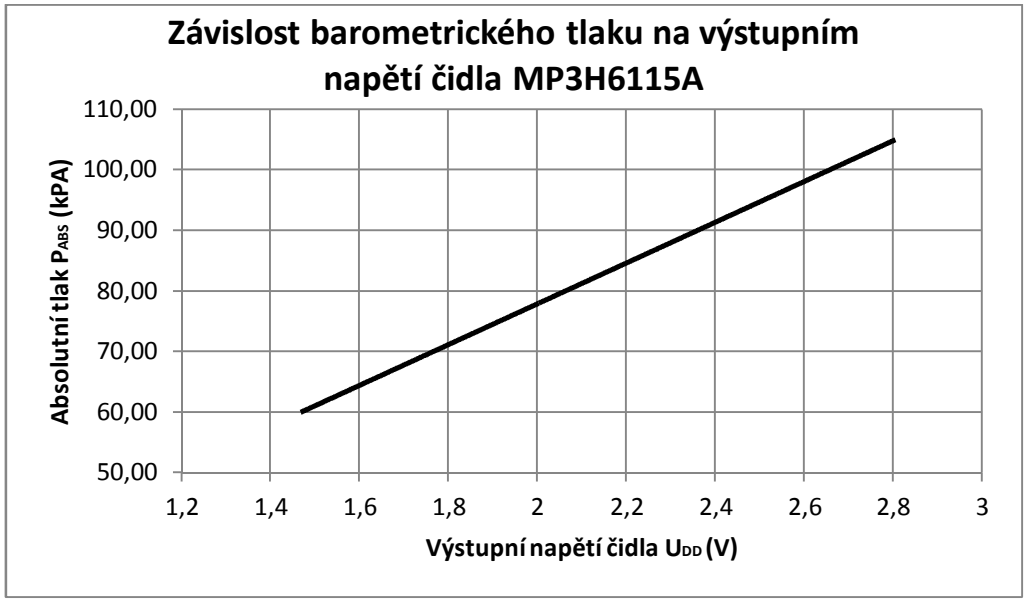
3.3.2 Čidlo pro měření atmosférického tlaku

Pro měření atmosférického tlaku bylo zvoleno čidlo MP3H6115A [4]. Jedná se o čidlo měřící absolutní tlak. Jádrem čidla je polovodičový rezistivní měřící prvek s teplotní kompenzací a výstupním zesilovačem.

Výstup čidla je analogový, výstupní napětí je přivedeno na A/D převodník MCU. Barometrický tlak je dán vztahem (3.3) [4]

$$\mathrm{P_{ABS}\ (kPa)} = \frac{\left(\frac{U_{\mathrm{OUT}}}{\mathrm{U_{DD}}} + 0{,}095\right)}{0{,}009}, \tag{3.3}$$

kde P_{ABS} je absolutní tlak, U_{OUT} je výstupní napětí čidla a U_{DD} je napájecí napětí. Závislost absolutního tlaku na výstupním napětí čidla je lineární, zobrazena na obr. 3.2.



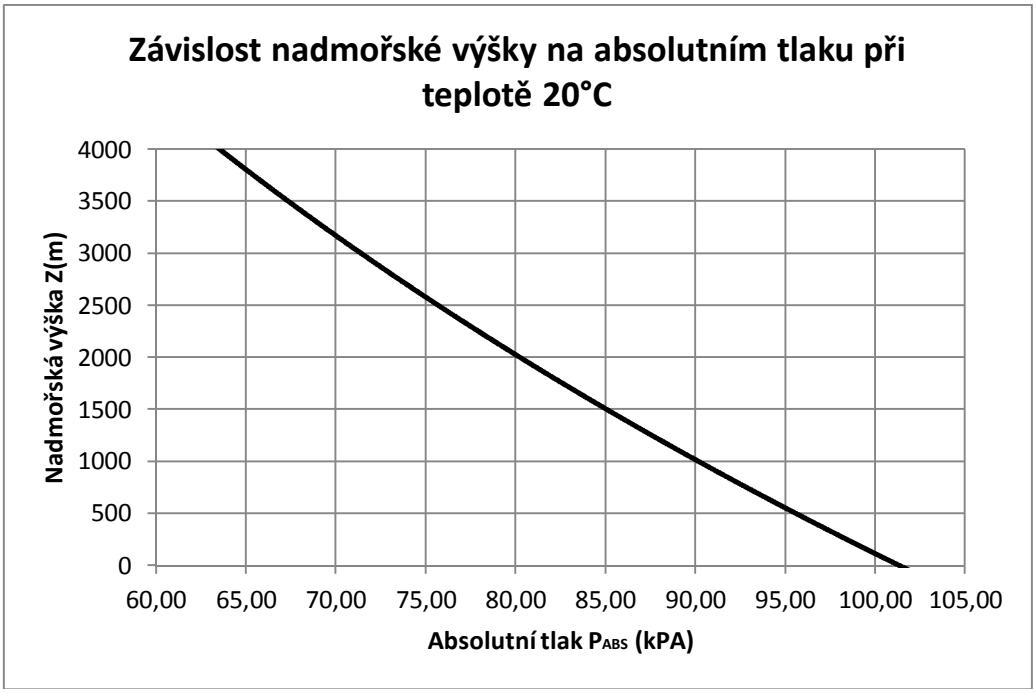
Obr. 3.2 Závislost barometrického tlaku na výstupním napětí tlakového čidla

Ze změřených hodnot atmosférického tlaku a teploty je možné vypočítat nadmořskou výšku podle vztahu (3.4) [4]

$$\mathrm{Z\ (m)} = -\frac{\mathrm{R}\cdot t}{\mathrm{g}} \cdot \ln\left(\frac{P_{\mathrm{ABS}}}{\mathrm{P_0}}\right) = -\frac{287\cdot t}{9{,}81} \cdot \ln\left(\frac{P_{\mathrm{ABS}}}{101{,}315\cdot 10^3}\right), \tag{3.4}$$

kde R = 287 $\mathrm{Jkg^{-1}K^{-1}}$ je plynová konstanta suchého vzduchu, t (°K) je měřená okolní teplota, g = 9,823 $\mathrm{ms^{-2}}$ je gravitační zrychlení, P_{ABS} je měřený atmosférický tlak a P_0 = = 101,315 kPa je referenční atmosférický tlak ve výšce 0 m n. m.

Závislost vypočtené nadmořské výšky na atmosférickém tlaku při konstantní teplotě 20 °C je zobrazena v grafu obr. 3.3.



Obr. 3.3 Závislost nadmořské výšky na atmosférickém tlaku při teplotě 20°C

3.3.3 Čidlo pro vzdálené měření teploty

Pro měření teploty na vzdáleném stanovišti bylo vybráno čidlo DS18S20 [5]. Měřená teplota je čidlem převáděna na digitální hodnotu a ukládána do interní paměti.

Čidlo disponuje 1-wire rozhraním, pro připojení k MCU je použit jeden datový vodič a zemnící vodič. Umožňuje „parazitní“ režim napájení čidla. V tomto režimu je čidlo napájeno skrze datovou linku pomocí silného pull-up rezistoru R_{13} (příloha A.4). Při vykonávání náročnějších operací čidla, jako je převod teploty nebo operace s interní pamětí, je datová linka vázána MOSFET tranzistorem Q_3 (příloha A.4) přímo k napájení dataloggeru.

3.3.4 Výběr řídící jednotky MCU

Jako řídící jednotka dataloggeru byl vybrán MCU STM32f103C8 [6]. Jedná se o 32bitový ARM procesor s jádrem Cortex-M3. Pro zařízení byl vybrán proto, že umožňuje přímé připojení k PC pomocí full-speed USB 2.0 sběrnice. Disponuje RTC jednotkou, což je potřebné pro časové záznamy při logování měřených dat. Lze použít jeden z několika režimů nízké spotřeby. Pro připojení vybrané paměti a LCD displeje je k dispozici SPI sběrnice.

3.3.5 Výběr paměti

Pro uložení měřených dat je potřebná dostatečně velká nevolatilní paměť. Byla vybrána flash paměť AT25DF161 [7]. Paměť je připojena k řídící jednotce pomocí sériového rozhraní SPI. Paměť poskytuje dostatečnou kapacitu 16Mb.

3.3.6 Výběr LCD displeje

Pro zobrazení posledních zaznamenaných údajů a informací o stavu dataloggeru byl vybrán grafický displej EADOGM132E [8]. Rozlišení displeje je 132x32 bodů, pracuje s jednotným napájením 3,3V a nízkou spotřebou 140μA v režimu bez podsvícení.

3.3.7 Napájení a nabíjení

K napájení dataloggeru byla zvolena nabíjecí Li-Ion baterie. Napětí baterie je měřeno A/D převodníkem MCU. Na vstupu A/D převodníku je napětí baterie upraveno děličem R_{20}, R_{21} s dělícím poměrem 0,5 (příloha A.5). Ze změřeného napětí je možné pomocí vybíjecí charakteristiky baterie stanovit přibližnou zbývající kapacitu baterie.

Baterie je nabíjena přímo z USB portu PC. Pro tento účel byla zvolena integrovaná nabíječka NCP1835B [9] s CCCV nabíjecím profilem.

Pomocí tohoto obvodu je nejdříve baterie nabíjena konstantním proudem (CC), dokud její napětí nepřesáhne fixní napěťovou úroveň 4,03V [9]. Poté nabíječka přechází do režimu nabíjení konstantním napětím (CV), kdy nabíjecí proud postupně klesá, dokud nedosáhne přibližně 10% nabíjecího proudu v režimu CC. V tomto stavu nabíječka přechází do stavu konce nabíjení (EOC), což signalizuje na CFLG pinu. Ve stavu EOC i nadále probíhá nabíjení v režimu konstantního napětí, dokud není dosaženo časového limitu. Pokud není v časovém limitu dosaženo EOC, nabíječka hlásí chybu na FAULT pinu.

Nabíječka také před začátkem detekuje stav kritického vybití baterie a přizpůsobuje tomuto stavu nabíjecí proud. Nedojde-li v tomto režimu k překonání napěťové úrovně kritického vybití, nabíječka na FAULT pinu hlásí chybu, což znamená poškození nebo zkrat baterie.

Jelikož bude baterie nabíjena z USB, kde je maximální proudový odběr z jednoho portu 100mA, je potřeba nastavit nabíjecí proud CC režimu. K tomu slouží rezistor R_{11} = 1,2MΩ (příloha A.5), této hodnotě odporu R_{11} přibližně odpovídá nabíjecí proud 90mA [9]. Časový limit nabíjení je nastaven kondenzátorem C_{13} = 47nF (příloha A.5), což odpovídá časovému limitu přibližně 11 hodin podle vztahu 3.5 [9]

$$\mathrm{t_{timeout}}\,(\mathrm{min}) = 14 \cdot \frac{C_{13}}{10^{-9}} = 658\mathrm{min} = 10.96\mathrm{h}. \qquad (3.5)$$

Poslední částí napájecí větve je stabilizátor napětí NCP662-D [10]. Výstupní napětí stabilizátoru, respektive napájecí napětí dataloggeru, je U_{DD} = 3,3V. Tento stabilizátor byl vybrán kvůli nízkému úbytku napětí a velmi nízké vlastní spotřebě.

Při každém logování hodnot bude měřeno napětí baterie. Pokud toto napětí klesne pod kritickou hodnotu vstupního napětí stabilizátoru potřebnou pro jeho funkci, zařízení ukončí logování dat a přejde do režimu spánku.

3.3.8 Obvod USB

Pro EMI filtraci a ESD ochranu zařízení byl zvolen obvod NUF2042XV6 [11].

Obvod s tranzistorem Q_1 (příloha A.1) připojí pull-up rezistor R9 = 1,5kΩ k USB datové lince D+, což umožní detekci rychlosti zařízení a jeho připojení v PC. Tranzistor Q_1 je sepnut pouze tehdy, pokud je připojen USB kabel.

3.4 Bilance spotřeby

Spotřeba zařízení je kritickým parametrem dataloggeru, protože bude použit pro dlouhodobé záznamy, kdy může být v provozu řádově několik měsíců bez dobíjení baterie.

Všechny komponenty byly vybírány s ohledem na jejich vlastní spotřebu. Odběr jednotlivých komponent je shrnut v tabulce 1.

Kvůli vyšší spotřebě LCD displeje a tlakového senzoru je napájení těchto komponentů spínáno MOSFET tranzistory Q_2 a Q_5 (příloha A.2 a A.4). Podsvícení LCD displeje je spínáno tranzistorem Q_4 (příloha A.2).

Tab. 1 Odběr jednotlivých komponentů z baterie

Komponent	I_{DD} (mA) během logování	I_{DD} (µA) mimo logování	I_{DD} (µA) zobrazení údajů
MCU	36	3,0	36 mA
Paměť	3,0	5,0	3 mA
Nabíječka	3 µA	3,0	3,0
Teplotní čidlo	1,0	1,0	1,0
RH&T čidlo	1,0	1,0	1,0
Tlakové čidlo	4,0	~0	~0
Stabilizátor	2,5 µA	2,5	2,5
LED	2,0	~0	~0
LCD	~0	~0	0,14 / max. 50 [1]
∑ ≈	47,0	14,5	40 / 90 [1]

[1] – režim LCD bez podsvícení / s podsvícením

3.5 Koncepce ukládání dat do paměti

Do paměti jsou ukládána logovaná data relativní vlhkosti, tlaku a teplot vzduchu spolu s údajem data a času logování. Dále budou ukládány provozní informace, tedy stav baterie a zbývající kapacita paměti. Logovaná data budou před uložením do paměti formátována a komprimována.

3.5.1 Příprava a formátování dat před uložením

Změřené hodnoty jednotlivých senzorů přenesených do MCU budou přepočítány pomocí převodních vztahů uvedených v kapitolách 3.3.1 – 3.3.3. Takto získané hodnoty budou nejdříve zaokrouhleny na jedno desetinné místo.

Jelikož bude použito entropické kódování pro následnou kompresi dat, je další fází přípravy dat diferenční kódování. Diferenční kódování spočívá v tom, že nejsou dále zpracovávány absolutní hodnoty změřených veličin, ale pouze rozdíl aktuální a předchozí hodnoty. Tímto způsobem může vzrůst počet bitů potřebných k vyjádření kódované hodnoty, ale to umožní entropickému kódování dosáhnout většího výsledného kompresního poměru.

Každá hodnota získaná diferenčním kódováním je formátována do čtyř symbolů o délce čtyři bity. První symbol vyjadřuje znaménko hodnoty, pro kladné hodnoty je kódována 0x0 a pro záporné 0xA. Další tři symboly vyjadřují desítky, jednotky a desetiny a jsou kódovány ve formě BCD kódu, tedy 0x0 až 0x9. Tento formát dat umožňuje vyjádření hodnot v rozsahu -99,9 až +99,9. Jelikož atmosférický tlak může přesahovat hodnotu 100 kPa je zavedena korekce odečtením 10 kPa od naměřené hodnoty.

Formáty a velikosti dat získaných z jednotlivých senzorů před a po formátování jsou uvedeny v tabulce 2. V tabulce jsou také uvedeny formáty dat pro uložení provozních informací - stav nabití baterie a zbývající kapacita paměti.

Tab. 2 Formáty logovaných a provozních dat

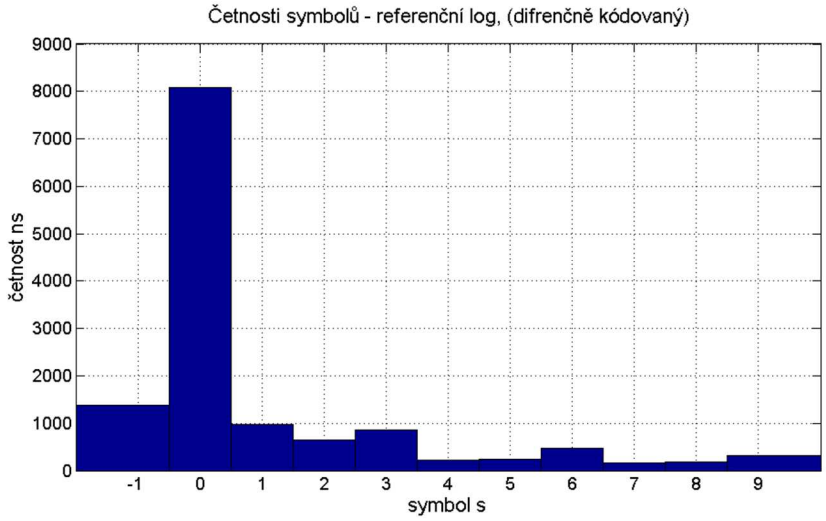
Data - čidlo	před formátováním (b)	po formátování (b)	formát
RH - SHT15	12	16	1 x S, 3 x BCD
teplota - SHT15	14	16	1 x S, 3 x BCD
teplota - DS18S20	16	16	1 x S, 3 x BCD
tlak - MP3H6115	12	16	1 x S, 3 x BCD
stav baterie	12	8	0 – 100% BIN
kapacita paměti	24	8	0 – 100% BIN

[1]- S – znaménko, BCD – znak kódovaný v BCD, BIN – binární reprezentace

3.5.2 Komprese dat před uložením

Pro kompresi dat bylo vybráno Huffmanovo kódování. Jedná se o entropické kódování, které spočívá v nahrazení vstupních symbolů binárními posloupnostmi různé délky, přičemž symboly s vyšší pravděpodobností výskytu jsou kódovány pomocí kratších posloupností. V dataloggeru bude kódováno jedenáct symbolů posloupnostmi délky nula až pět bitů.

Před samotným kódováním je potřeba vytvořit slovník symbolů a odpovídajících posloupností. V této práci je použit jednotný slovník pro kompresi hodnot všech logovaných veličin. Pro určení pravděpodobností a sestavení slovníku byl použit diferenčně kódovaný referenční log venkovní teploty z období 4.3. až 27.3. 2009 s intervalem měření 60 minut (stanoviště Starý Jičín, 295 m n. m.). Histogram četností jednotlivých symbolů je na obr. 3.4. Dosažený kompresní poměr referenčního logu byl 70%.



Obr. 3.4 Histogram četností výskytu symbolů v referenčním logu

Pro odhad pravděpodobností výskytu symbolů sloužily relativní četnosti výskytů jednotlivých symbolů v referenčním logu. Pro výpočet relativních četností byl použit vztah (3.6)

$$x_s = \frac{n_s}{n} = \frac{n_s}{\sum_s n_s}\,, \tag{3.6}$$

kde n_s je četnost výskytu jednotlivých symbolů. Relativní četnosti a odhad pravděpodobností výskytů symbolů *p* jsou uvedeny v tabulce 3.

Tab. 3 Slovník Huffmanova kódu, pravděpodobnost výskytu symbolů

Symbol	x_s [-] – ref. log	p [-] - odhad	Huffmanův kód
S – znaménko	0,10	0,10	0b010
BCD 0	0,60	0,60	0b1
BCD 1	0,07	0,07	0b0111
BCD 2	0,05	0,06	0b0110
BCD 3	0,06	0,05	0b0011
BCD 4	0,02	0,02	0b00001
BCD 5	0,02	0,02	0b00000
BCD 6	0,04	0,02	0b00011
BCD 7	0,01	0,02	0b00010
BCD 8	0,01	0,02	0b00101
BCD 9	0,02	0,02	0b00100

V tabulce 3 je také uveden Huffmanův kód pro jednotlivé symboly. Aplikace algoritmu pro určení jednotlivých kódů je uvedena v tabulce 4.

Tab. 4 Aplikace algoritmu pro určení Huffmanova kódu symbolů

s	p	h	s	p	h	s	p	h	s	p	h	s	p	h	s	p	h
0	0,60		s0	0,60		s0	0,60		s0	0,60		s0	0,60		s0	0,60	
S	0,10		sS	0,10		sS	0,10		sS	0,10		sS	0,10		sS	0,10	
1	0,07		s1	0,07		s1	0,07		s1	0,07		s4567	0,08		s389	0,09	
2	0,06		s2	0,06		s2	0,06		s2	0,06		s1	0,07		s4567	0,08	
3	0,05		s3	0,05		s3	0,05		s3	0,05		s2	0,06		s1	0,07	**1**
4	0,02		s89	0,04		s89	0,04		s89	0,04		s3	0,05	**1**	s2	0,06	**0**
5	0,02		s4	0,02		s67	0,04		s67	0,04	**1**	s89	0,04	**0**			
6	0,02		s5	0,02		s4	0,02	**1**	s45	0,04	**0**						
7	0,02		s6	0,02	**1**	s5	0,02	**0**									
8	0,02	**1**	s7	0,02	**0**												
9	0,02	**0**															

s	p	h	s	p	h	s	p	h	s	p	h
s0	0,60		s0	0,60		s0	0,60		s0	0,60	**1**
s12	0,13		s3456789	0,17		s123456789	0,23	**1**	sS123456789	0,4	**0**
sS	0,1		s12	0,13	**1**	sS	0,17	**0**			
s389	0,9	**1**	sS	0,10	**0**						
s4567	0,8	**0**									

s – kódovaný symbol / skupina symbolů

p – pravděpodobnost výskytu kódovaného symbolu / skupiny symbolů

h – přiřazený bit Huffmanova kódu

3.5.3 Ukládání dat do paměti

Komprimovaná data budou ukládána do paměti flash po blocích. Bloky budou složeny z hlavičky a datové části. Konkrétní struktura bloku dat je uvedena v tabulce 5.

Hlavička se skládá z dvou bytů SB určení začátku bloku, jednoho bytu SREG - status registru[1], osmi bytů údaje o datu a čase prvního referenčního logu v bloku, dvou bytů údajů o stavu baterie a zbývající kapacity paměti, tří bytů časového intervalu mezi jednotlivými logy, jednoho bytu počtu logů v bloku a dvou bytů udávajících počet zakódovaných paměťových buněk.

Datová část obsahuje komprimovaná data proměnné délky, referenční data, tedy absolutní hodnoty změřených veličin, a diferenčně kódovaná a komprimovaná data.

Posledním bytem bloku je údaj o počtu zakódovaných bitů v poslední paměťové buňce s Huffmanovým kódem. Význam tohoto bytu je dále vysvětlen v kap. 5.9.7.

Maximální délka bloku se odvíjí od nastaveného počtu měření, jež budou uložena v paměti. Tato délka je stanovena na 200 logů jakékoliv kombinace měřených veličin. Nemusí být tedy zaznamenávány všechny veličiny, ale například jen jedna z nich. Každý blok je možné ukončit a následně správně dekódovat např. při změně parametrů logování nebo detekování kritického stavu baterie.

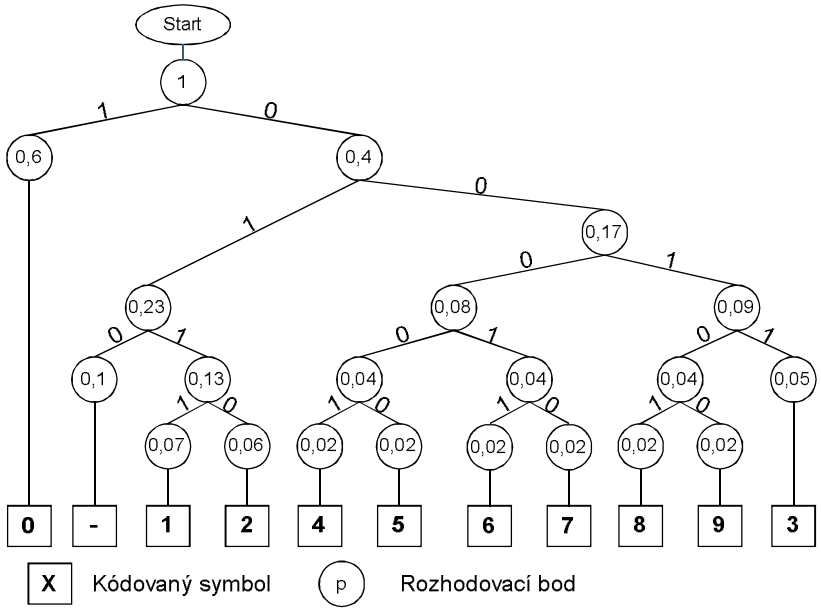
[1] Uživatelské registry – viz kap. 5.7.1.

Tab. 5 Blok dat ukládaný do paměti flash

	Flash paměť Bit 0 - 7	poznámka
Hlavička	SB – 0x00	určení začátku bloku
	SB – 0xFF	
	SREG	status registr
	YY – rok	časový údaj začátku bloku YY.MM.DD rok.měsíc.den hh:mm:ss hodina:minuta:sekunda
	MM – měsíc	
	DD – den	
	hh – hodina	
	mm – minuta	
	ss – sekunda	
	BAT	stav baterie
	MC	zbývající kapacita paměti
	perioda měření – MSB	časový interval mezi logy vyjádřený v sekundách
	perioda měření	
	perioda měření – LSB	
	BLOCK_COUNT	počet logů v bloku
	BYTE_COUNT	počet bytů komprimovaných dat
	BYTE_COUNT	
Data	komprimovaná data	posloupnost komprimovaná Huffmanovým kódem, proměnná délka
	REM_CNT	počet bitů poslední buňky s kódem

3.5.4 Dekomprese uložených dat

Při dekompresi dat jsou data načtena z paměti, bitová posloupnost je dekódována pomocí Huffmanova stromu uvedeného na obr. 3.5.



Obr. 3.5 Huffmanův strom

Při dekódování se v každém rozhodovacím bodě, podle následujícího bitu posloupnosti, pokračuje příslušnou větví stromu. V okamžiku, kdy dekodér dojde ke kódovanému symbolu, předá symbol do bufferu dekódovaných symbolů v požadovaném formátu a započne dekódování dalšího symbolu následujícím bitem vstupní posloupnosti.

3.6 Stanovené a předpokládané parametry zařízení

V tabulce 6 jsou uvedeny stanovené a předpokládané parametry dataloggeru a parametry dané specifikací senzorů.

Tab. 6 Stanovené a předpokládané parametry zařízení

Obecné parametry	
napájecí napětí	3,3V DC
typ baterie	Li-Ion
spotřeba – v režimu nízké spotřeby	14,5μA
spotřeba – během zaznamenávání dat	47mA
spotřeba – zobrazení provozních dat	40/90mA [1]
připojení k PC	USB 2.0
teplotní rozsah	-20 až 70°C
kompresní poměr	~ 70%
Měření relativní vlhkosti [3]	
pracovní rozsah[2]	0 až 100%
rozlišení	0,05%
přesnost	±2%
Měření teploty [3]	
pracovní rozsah	-20°C až +70°C
rozlišení	0,01°C
přesnost	±0,5%
Měření teploty – externí senzor[5]	
pracovní rozsah [2]	-55°C až +125°C
rozlišení	0,01°C
přesnost [3]	±0,5% / ±2%

[1] – režim LCD bez podsvícení / s podsvícením

[2] – všechny pracovní rozsahy můžou nabývat pouze hodnot -99,9 až 99,9 dané veličiny kvůli použitému systému ukládání dat, viz. kap. 3.5.3.

[3] – rozsah -10°C až +85°C / -55°C až +125°C

3.7 Návrh komunikačního protokolu

Nastavení parametrů logování a stahování zaznamenaných dat do PC bude provedeno pomocí sériové sběrnice USB2.0. Vybraný MCU disponuje USB2.0 full-speed

rozhraním, je tedy možné jeho přímé připojení ke sběrnici.

Vzhledem k relativně malému množství přenášených dat mezi dataloggerem a PC byla pro USB komunikaci vybrána třída HID, což výrazně ulehčuje vývoj ovládací aplikace i firmwaru MCU. Teorie ke komunikačnímu protokolu vychází z [13].

3.7.1 Teorie komunikace sběrnice USB2.0

Každá komunikace po USB sběrnici probíhá mezi hostem (obvykle PC) a zařízením. Veškerý provoz na sběrnici řídí host, zařízení vysílají data pouze na jeho požadavek. Komunikace je paketová. Každý paket obsahuje identifikátor délky jeden byte, data délky nula až jeden kilobyte a cyklický redundantní součet CRC délky dva byty pro kontrolu správnosti přenesených dat.

Jsou specifikovány čtyři druhy přenosu: kontrolní, hromadný, přerušovací a isochronní. Kontrolní přenosy slouží k vyžádání potřebných informací o připojeném zařízení, které jsou do PC odesílány ve formě deskriptorů a určují, jaký ovladač bude použit. Hromadné přenosy se používají pro velké objemy přenášených dat, kde musí být zachována jejich správnost. Tento typ přenosu využívají velkokapacitní zařízení – USB mass storage device class. Isochronní přenosy se také používají k přenosu velkých objemů dat, avšak nezaručují bezchybný přenos. Používají se například při přenosu zvuku nebo videa. Přerušovací přenosy slouží k přenosu dat malých objemů a mají garantovanou latenci. Tento typ přenosu využívají např. USB klávesnice a myši.

Kontrolér zařízení obsahuje několik bufferů nazývaných endpointy. Jsou to vyhrazené paměťové prostory, kde jsou data přijímána, případně odesílána hostu. Každý endpoint je definován svým číslem a směrem toku dat. Nultý endpoint je vždy vyhrazen pro kontrolní přenosy. Před začátkem přenosu musí host a zařízení vytvořit tzv. „pipe“, což je propojení mezi konkrétním endpointem a softwarem v případě PC hostu.

3.7.2 Třída HID

Třída Human Interface Device class (HID) je jednou z prvních tříd podporovaných operačním systémem Windows. Třída HID je primárně určena pro zařízení sloužící pro komunikaci mezi počítačem a uživatelem jako jsou myš, klávesnice, displeje, indikátory, klávesnice pro telefonní vytáčení, nejrůznější herní zařízení a mnoho dalších. HID třídu však mohou využívat i zařízení sloužící k jinému účelu, kde vyhovuje těsná specifikace tohoto standardu. Třída HID je robustní a kompaktní, výhodou z pohledu aplikační vrstvy je to, že používá vestavěné ovladače operačního systému. Třída HID definuje celou řadu deskriptorů, které se odesílají při enumeraci zařízení spolu s USB deskriptory. Toto umožňuje zmíněné načtení správného ovladače pro zařízení a definování způsobu komunikace se zařízením.

Podmínkou využití HID třídy je fixní délka dat 64 B přenášených v paketech, k čemuž se váže přenosová rychlost 64kB/s pro full-speed USB. Zařízení může využívat pouze jeden přerušovací endpoint.

Komunikace třídy HID je zajištěna pomocí tzv. reportů. Jeden report odpovídá jednomu paketu. Pro datové přenosy jsou k dispozici dva typy reportů. „InReport“ slouží k odesílání dat ze zařízení a „OutReport“ k příjmu dat zařízením. Každý report je pro jednoznačnou identifikaci povinně označen číslem „Report ID“ přenášeným v prvním bytu příslušného reportu. Toto umožňuje definování potřebného množství reportů volitelné délky a formátu určených ke konkrétnímu účelu.

3.7.3 Třída HID – deskriptory reportů

Spolu s HID deskriptory se při enumeraci zařízení odesílají deskriptory reportů. Tento deskriptor informuje host převážně o ID, délce, formátu a účelu reportu. USB standard definuje velké množství konkrétně daných reportů, které jsou obsaženy v základním HID ovladači operačního systému a jsou obsluhovány bez nutnosti instalace dalších ovladačů výrobce zařízení.

Datalogger bude využívat tři typy reportů, každý definovaný vlastním deskriptorem. První typ reportu bude sloužit k předávání jednoduchých příkazů z PC. Druhý typ reportu bude sloužit k přenosu konfiguračních dat a příkazů, které vyžadují přenos přídavných dat. Poslední typ reportu bude sloužit k přenosu zaznamenaných dat do PC, odeslání informací o stavu dataloggeru a případné potvrzování úspěšně proběhlých operací. Konkrétní podoba deskriptorů a formátu přenášených dat je popsána v kap. 5.8 zabývající se implementací komunikačního protokolu.

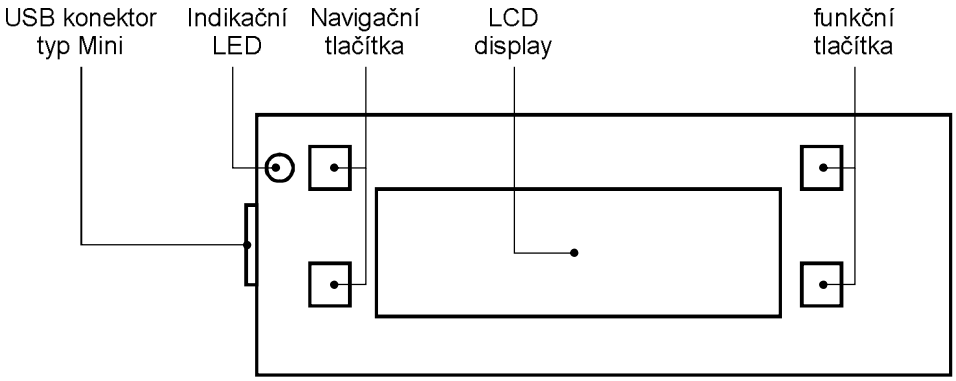
4 NÁVRH HARDWARU DATALOGGERU

Návrh hardwaru se skládá z návrhu desky plošných spojů a návrhu mechanické konstrukce zařízení.

4.1 Návrh DPS

Požadavky při návrhu DPS byly kompaktní velikost zařízení a praktické rozmístění ovládacích a informačních prvků na předním panelu mechanické konstrukce.

Na obr. 4.1 je uvedeno blokové rozmístění ovládacích a indikačních prvků.



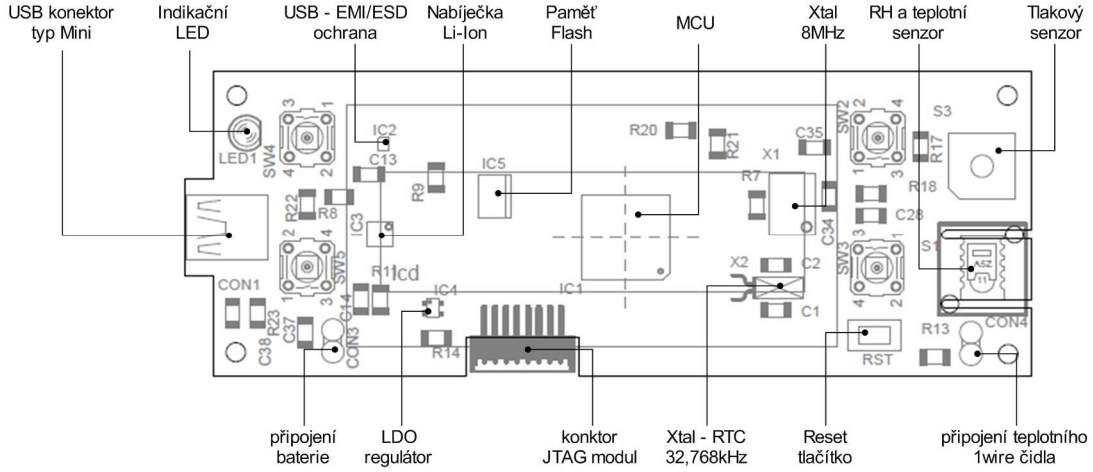
Obr. 4.1 Rozložení ovládacích a indikačních prvků

Dalším hlavním kritériem při rozmístění jednotlivých komponentů na DPS bylo její možné prohřívání. Z tohoto důvodu byla část zapojení, konkrétně napájení a nabíjení (příloha A.5), které potencionálně produkuje nejvíce tepla, umístěno co nejdále od senzorů, aby nedocházelo k jejich zahřívání, které by způsobilo zkreslení měřených veličin. Proto jsou také kolem senzoru pro měření vlhkosti a teploty STH11 vyfrézovány v DPS drážky podle doporučení z datasheatu senzoru [3].

MCU je umístěn přibližně uprostřed desky, co nejblíže k němu jsou umístěny hodinový a RTC krystaly kvůli minimalizaci zkreslení výstupního signálu a času stabilizace hodinového signálu [14]. Rozložení jednotlivých komponent na „top“ straně DPS je zobrazeno na obr. 4.2.

Deska plošných spojů je oboustranná s rozměry 93 x 35 mm. Materiál desky je FR4. Kontakty pro pájení jsou galvanicky pozlaceny. Deska je opatřena nepájivou maskou a potiskem. Deska je osazena SMD součástkami.

Kompletní návrh DPS je uveden v příloze C.1 a C.2, osazovací schéma je v příloze C.3 a C.4. K návrhu DPS byl použit program EAGLE 5.7.0 [17]. Projekt DPS je k dispozici ve složce *.//DPS/Eagle_project/*. Podklady pro výrobu DPS ve formátu Gerber RS-274X jsou k dispozici na přiloženém CD ve složce *.//DPS/Gerber_data/*.



Obr. 4.2 Rozložení komponent na DPS

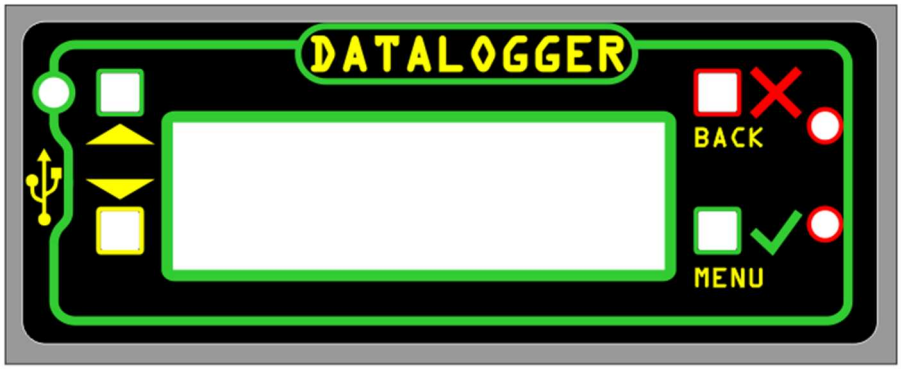
4.2 Návrh mechanické konstrukce zařízení

Pro zařízení byla navržena a vyrobena vhodná přístrojová krabička. Byl vybrán materiál texgumoid pro své dobré mechanické vlastnosti a klimatickou odolnost, především proti vlhkosti. Všechny části přístrojové krabičky jsou ošetřeny bílou základovou barvou, vnější části jsou přestříkány ochranným lakem odstínu RAL 7044.

Konstrukce se skládá ze dvou částí krytu a střední části, která slouží k uložení baterie a jejímu oddělení od DPS. V horní části krytu je vyfrézováno vybrání pro uložení fólie potisku předního panelu. Mezi horní a střední částí je uchycena deska plošných spojů. Ve střední části je také uchycen blok předpružených kontaktů pro baterii. 3D model mechanické konstrukce je uveden v příloze D.1 a výkresová dokumentace v přílohách D.2 – D.4.

4.3 Potisk předního panelu

Potisk předního panelu je proveden na průhledné fólii, která je lepením připevněna do vybrání v horním dílu přístrojové krabičky. Potisk je uveden na obr. 4.3.

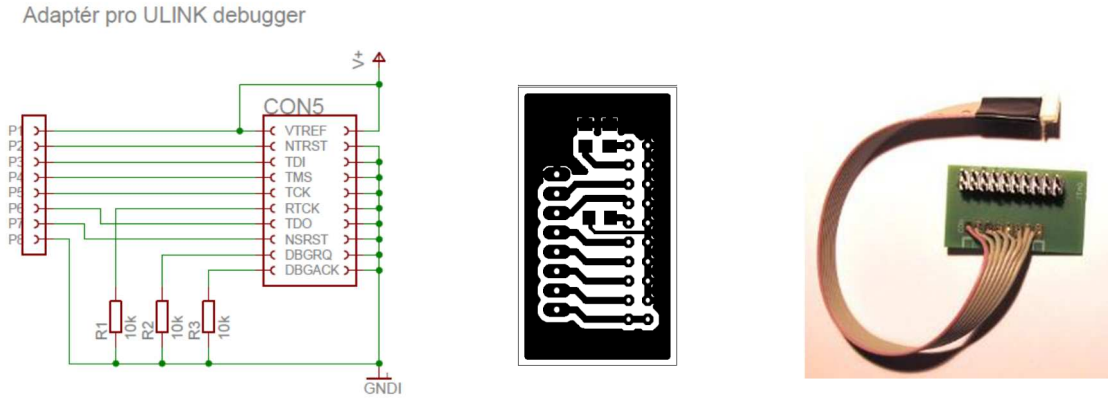


Obr. 4.3 Potisk předního panelu

4.4 Adaptér pro JTAG programátor

Z důvodu úspory místa nebylo na DPS dataloggeru vytvořeno kompletní zapojení pro JTAG programátor s 20-pinovým konektorem, ale byly pouze vyvedeny potřebné linky pomocí miniaturního konektoru CON2 (příloha A.1).

Byl vytvořený jednoduchý adaptér, který je při programování flash paměti MCU připojen ke konektoru CON2. Schéma adaptéru je na obr. 4.4a, DPS na obr. 4.4b a realizace na obr. 4.4c.



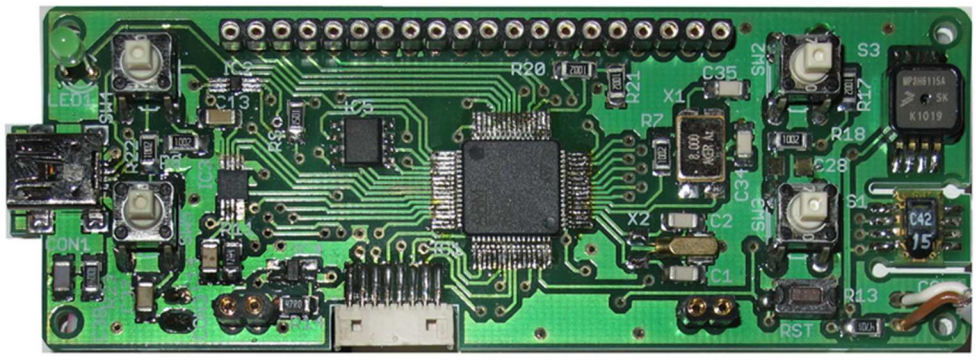
a) schéma b) DPS (M 1:1) c) realizace

Obr. 4.4 Adaptér JTAG

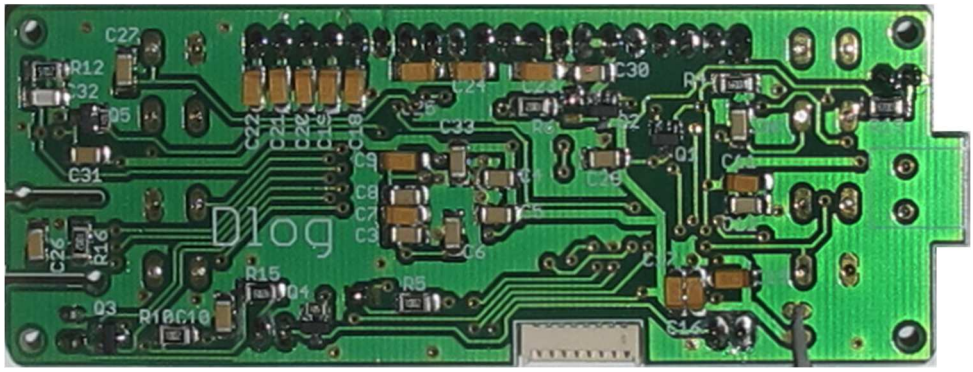
4.5 Realizace hardware

DPS byla osazena, hrubá mechanická konstrukce byla vyrobena na zakázku, upravena podle osazené DPS a nabarvena. Dále byla upravena a vlepena fólie s potiskem a zařízení bylo zkompletováno.

Pohled na DPS ze shora je na obr. 4.5, pohled ze spodu na obr. 4.6. Na obr. 4.7 je vidět kompletace DPS a mechanické konstrukce a na obr. 4.8 finální podoba zařízení.



Obr. 4.5 Realizace DPS - pohled TOP (bez LCD)



Obr. 4.6 Realizace DPS - pohled BOTTOM

Obr. 4.7 Kompletace hardware

Obr. 4.8 Zkompletovaný datalogger

5 FIRMWARE MCU

Firmware zahrnuje veškeré programové vybavení dataloggeru, implementuje moduly pro komunikaci po USB, práci s daty, power management, obsluhu RTC jednotky, drivery pro ovládání integrovaných obvodů a senzorů, LCD displeje a další potřebné funkce potřebné pro činnost dataloggeru.

Konkrétně byly vytvořeny drivery k LCD displeji EA DOGM132-5, tlakovému senzoru MP3H6115-A, vlhkostnímu a teplotnímu senzoru SHT15, teplotnímu senzoru DS18B20 a sériové flash paměti. Dále byla vytvořena obsluha dvou kanálů ADC převodníku pro měření napětí baterie a výstupního napětí tlakového senzoru a obsluha systémového časovače pro generování zpoždění. Modul pro práci s daty obsahuje funkce pro kompresi a ukládání dat do paměti, modul RTC jednotky poskytuje údaje o reálném čase a datu a také zajišťuje spouštění logování s nastavenou periodou. Modul pro práci s USB implementuje komunikační protokol pro nahrávání konfigurace, kontrolu stavu zařízení a stahování logovaných dat.

Při programování firmwaru byla použita knihovna STM32F10x Standard Peripherals Library pro práci se všemi periferiemi MCU, stěžejním materiálem byl také programovací manuál [15], který je součástí této knihovny.

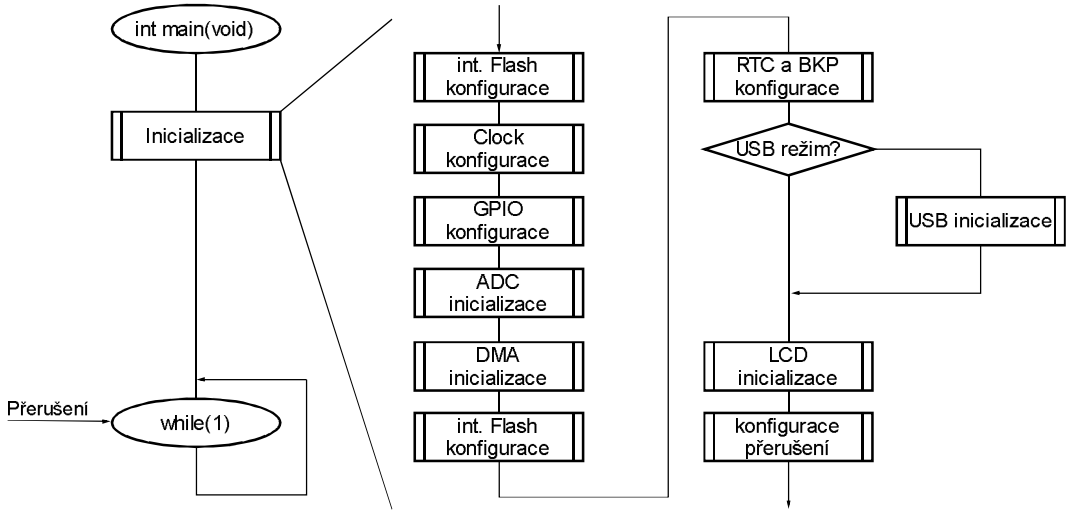
Pro vytvoření firmware pro MCU je použito vývojové prostředí Keil µVision V4.03 [16]. Projekt firmwaru je k dispozici na přiloženém CD ve složce *.//Firmware/*. K tomuto umístění také odkazují soubory uvedené v této kapitole.

5.1 Inicializace a konfigurace periferií MCU

Inicializace a konfigurace MCU je prvním krokem, který je proveden po spuštění nebo probuzení zařízení z režimu nízké spotřeby. Inicializace probíhá v hlavní funkci firmwaru *int main(void)* v souboru *./main.c*. Algoritmus hlavního programu je uveden na obr. 5.1.

Nejprve je nastavena latence interní flash paměti programu z důvodu nižší přístupové rychlosti paměti oproti taktování jádra MCU[6]. Dále jsou nastaveny hodinové kmitočty jádra, periferií a USB linky. Následuje inicializace GPIO linek a počáteční nastavení jejich úrovní, inicializace kanálů A/D převodníků spolu s inicializací DMA kanálu pro ukládání převáděných hodnot do datové paměti MCU. Následuje inicializace SPI periferie pro přenos dat s externí flash pamětí. Dále je nastaven a spuštěn systémový časovač a inicializována jednotka RTC a povolen přístup k BKP doméně. Pokud je kontrolér v režimu pro práci s USB, je inicializována USB periferie. Na závěr je inicializován LCD displej a nastaven kontrolér přerušení.

Další běh celého programu je řízen přerušeními vyvolanými RTC jednotkou, tlačítky, případně USB periferií.



Obr. 5.1 Algoritmus hlavního programu

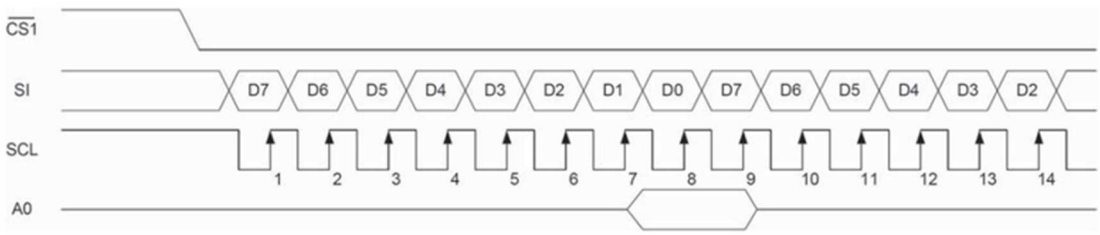
5.2 Driver LCD displeje EA DOGM132-5

Driver k displeji obsahuje funkci pro komunikaci MCU s LCD displejem, funkci pro inicializaci a samotné vykreslovací funkce pro obrazce a text.

5.2.1 Komunikace MCU s LCD

Pro komunikaci MCU s LCD displejem byla vytvořena jednoduchá softwarová implementace SPI komunikace. Po lince je vždy odesílán jeden byte, příkaz nebo data pomocí funkce *void LCD_SendByte(uint8_t Tx_byte)* definované v souboru *./code/drivers/lcd_dog132/lcd_dogm132_com.c*.

Komunikace začíná uvedením chip select pinu LCD_CS[2] displeje do stavu log 0. Následuje 8 impulsů hodinového signálu SPI_SCL[2]. Při každém hodinovém impulsu se mění také úroveň signálů SPI_SI[2] podle odesílaného bytu od MSB k LSB. Při osmém SPI_SCL[2] impulsu je displejem přečtena úroveň pinu LCD_A0[2], čímž je vybrán typ přenášených dat, log 0 pro příkazy a log 1 pro data. Průběh komunikace je zobrazen na obr. 5.2.



Obr. 5.2 Sériová komunikace MCU a LCD

[2] Viz příloha A.1 a A.2.

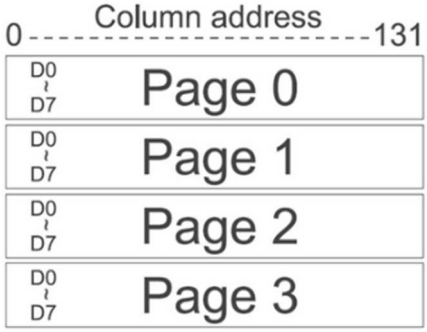
5.2.2 Inicializace displeje

Inicializace displeje spočívá v odeslání několika příkazů, sloužících k nastavení parametrů modulu displeje, kterými jsou např. počáteční řádek displeje, řízení napájení, mód normální či reverzní, kontrast a na závěr aktivace displeje. Přehled všech příkazů a možností nastavení je uveden v datasheatu [8].

5.2.3 Organizace a adresování paměti řadiče displeje

Jednotlivé pixely jsou vykreslovány na displej odesláním a uložením obrazových dat do paměti řadiče displeje. Zobrazovací prostor je rozdělen podle obr. 5.3. Jsou to tedy čtyři stránky (pege) 8bitových dat, na každé stránce je 132 sloupců (column). Určení daných pixelů na displeji odpovídá struktuře paměti řadiče.

Při adresování zobrazovací paměti se nejdříve odesláním série příkazů vybere adresa strany (0-3) a adresa sloupce (0-131). Od adresy sloupce na dané straně bude probíhat zápis dat. Dále jsou již odesílána obrazová 8bitová data. Při úspěšném přijetí každého bytu obrazových dat je inkrementována adresa sloupce paměti.



Obr. 5.3 Struktura paměti řadiče displeje

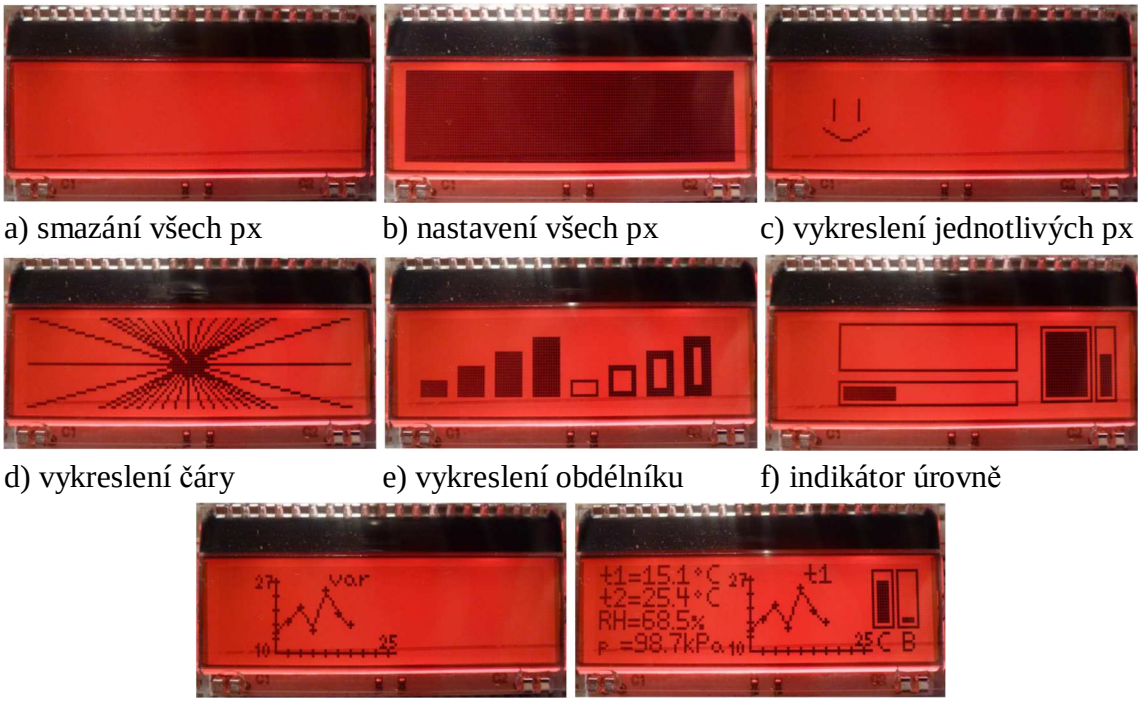
Všechny dále popsané vykreslovací funkce (vykreslovaní pixelů, čar, znaků, atd.) vždy pouze připraví potřebná data do zobrazovacího bufferu displeje v MCU. Buffer displeje je 32bitové pole o délce 132 prvků. Při vykreslování jsou jednotlivé prvky bufferu bitovým maskováním a rotací rozděleny na 8bitová data, která se odesílají do řadiče displeje. Toto zajišťuje funkce *extern void LCD_redraw_buffer(void)* definovaná v *./code/drivers/lcd_dog132/lcd_dogm132.c*.

5.2.4 Vykreslovací funkce displeje

Funkce pro vykreslování jsou shrnuty v tabulce 7. Všechny tyto funkce jsou definovány v souboru *./code/drivers/lcd_dog132/lcd_dogm132.c*. Ukázky vykreslovacích funkcí jsou uvedeny na obr. 5.4.

Tab. 7 Přehled vykreslovacích funkcí pro displej

Název funkce	Účel funkce	Odpovídající obrázek
extern void LCD_init(void)	inicializace displeje	
extern void LCD_clear(void)	smazání displeje	Obr. 5.4 a
extern void LCD_set_all(void)	vykreslení všech pixelů displeje	Obr. 5.4 b
extern void LCD_draw_px(params)	vykreslení jednoho pixelu	Obr. 5.4 c
extern void LCD_draw_line(params)	vykreslení přímky	Obr. 5.4 d
extern void LCD_draw_rect(params)	vykreslení obdélníku	Obr. 5.4 e
extern void LCD_progress_bar(params)	vykreslení indikátoru průběhu	Obr. 5.4 f
extern void LCD_print_string(params)	vykreslení základního textu	Obr. 5.4 h
extern void LCD_redraw_buffer(void)	vykreslení zobrazovacího bufferu	Obr. 5.4
extern void LCD_graph(params)	vykreslení jednoduchého grafu	Obr. 5.4 g



a) smazání všech px b) nastavení všech px c) vykreslení jednotlivých px

d) vykreslení čáry e) vykreslení obdélníku f) indikátor úrovně

g) jednoduchý graf h) text, aplikace funkcí

Obr. 5.4 Příklad vykreslování jednotlivých funkcí LCD displeje

5.3 Driver tlakového senzoru MP3H6115A

Tlakový senzor má analogový výstup, který je připojený k analogové vstupní lince prvního kanálu A/D převodníku MCU. Inicializace senzoru není součástí driveru. Spočívá ve společné inicializaci A/D převodníku a DMA kanálu, jak je uvedeno v kap. 5.1.

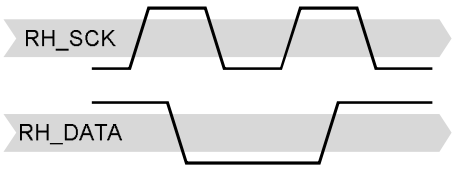
Driver obsahuje pouze jednu funkci *float psens_measure(void)* definovanou v souboru *./code/drivers/MP3H6115A/psens_MP3H6115A.c*. Tato funkce spouští A/D převod, následně čeká, dokud není pomocí DMA kanálu zapsána převedená hodnota do paměti dat MCU. Dále je načtena převedená hodnota a pomocí vztahu (3.3) je vypočten atmosférický tlak, který funkce vrací ve formátu float.

5.4 Driver vlhkostního a teplotního senzoru SHT15

Driver pro vlhkostní a teplotní senzor obsahuje funkce pro sériovou komunikaci mezi MCU a senzorem, funkce pro zápis a čtení status registru senzoru, funkci pro softwarový reset, spuštění měření, přenos a výpočet měřených hodnot vlhkosti a teploty. Všechny funkce pro obsluhu senzoru jsou definovány v souboru *./code/drivers/SHT15/RHsens_SHT15.c*.

5.4.1 Reset a inicializace komunikace

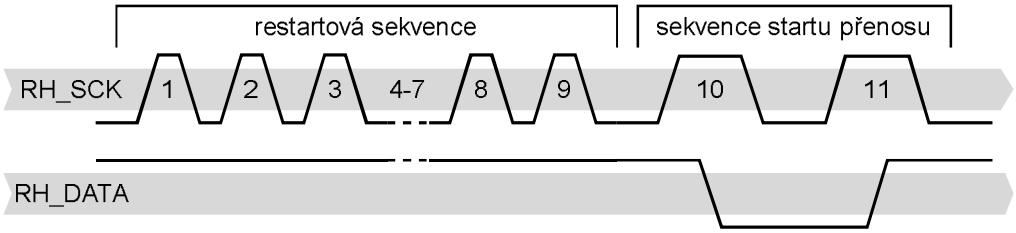
Při připojení napájení projde senzor vlastní inicializací a po jedenácti milisekundách se přepíná do sleep módu [3]. Během uvedené doby nesmí být senzoru odeslán žádný příkaz po sériové lince. Následně může být sekvence pro start přenosu dat (obr. 5.5) odeslána. Po odeslání této sekvence může být linkou odeslán příkaz (tabulka 8).



Obr. 5.5 Sekvence pro start přenosu dat [3]

Odeslání této sekvence provádí funkce *void RHsens_transStart(void)*.

Pokud dojde k chybě přenosu, nebo senzor přestane odpovídat, je možné po lince odeslat sekvenci, která restartuje komunikační rozhraní senzoru. Sekvence je zobrazena na obr. 5.6. Jedná se o devět impulsů hodinového signálu RH_SCK[3] při úrovni log 1 signálu RH_DATA[3] následovaných sekvencí pro start přenosu. Restart komunikačního prostření zajišťuje funkce *void RHsens_conInit(void)*.



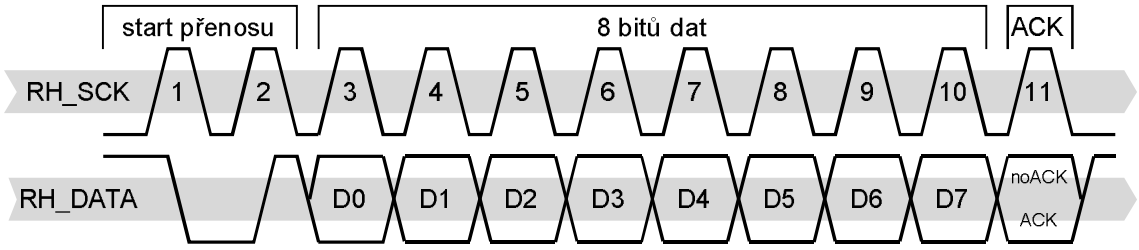
Obr. 5.6 Sekvence pro restart rozhraní senzoru [3]

[3] Viz. příloha A.1 a A.4.

Restart samotného senzoru je možný odesláním příkazu 0x1E, který restartuje komunikační rozhraní a také nastaví výchozí hodnoty ve status registru senzoru. Opět je nutné vyčkat minimálně jedenáct milisekund před odesláním dalšího příkazu. Restart senzoru obstarává funkce *uint8_t RHsens_softreset(void)*. Pokud nastane chyba, funkce vrací *unsigned char* hodnotu 0x01, jinak vrací hodnotu 0x00.

5.4.2 Zápis bytu dat sériové linky

Pro zápis bytu dat sériové linky senzoru byly vytvořeny funkce *uint8_t RHsens_writeByte(uint8_t value)*. Funkce pro zápis dat odešle sekvenci 8 bitů dat. Každý bit dat je přečten senzorem při průchodu hodinového signálu. Poté je GPIO linka MCU RH_DATA[3] nastavena jako vstupní a je udržována ve stavu log 1 pomocí pull-up rezistoru. Během následujícího synchronizačního impulsu senzor signalizuje úspěšný přenos nastavením úrovně linky na log 0. Poté je GPIO linka zpět nastavena jako výstupní s hodnotou log 1. Časový diagram je zobrazen na obr. 5.7.



Obr. 5.7 Sekvence odeslání bytu dat [3]

Pokud nebyl detekován ACK signál, funkce vrací *unsigned char* hodnotu 0x01, jinak vrací hodnotu 0x00.

Pomocí této funkce je možné odeslat senzoru příkaz. Možné příkazy jsou shrnuty v tabulce 8.

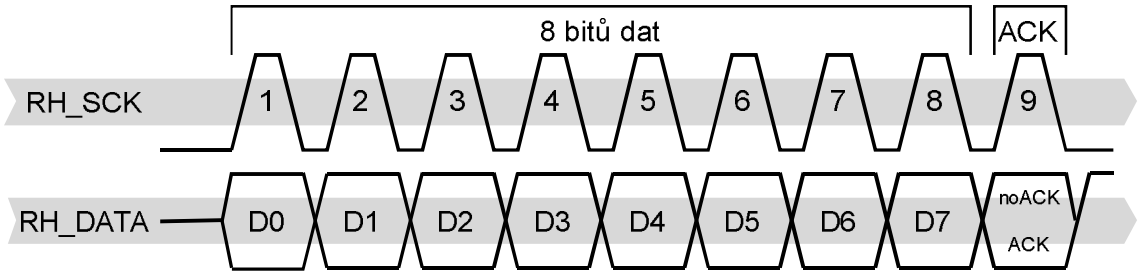
Tab. 8 Příkazy pro senzor SHT15 [3]

Příkaz	Hodnota
Start měření RH	0b00011 (0x03)
Start měření teploty	0b00101 (0x05)
Přečíst status registr	0b00111 (0x07)
Zapsat status registr	0b00110 (0x06)
Softwarový restart	0b11110 (0x1E)

5.4.3 Čtení bytu dat sériové linky

Pro zápis bytu dat sériové linky senzoru byly vytvořeny funkce *uint8_t RHsens_readByte(uint8_t ack)*. GPIO linka MCU RH_DATA je nastavena jako vstupní, postupně je vysláno osm synchronizačních impulsů RH_SCK. Při každém synchronizačním impulsu je přečtena úroveň RH_DATA a uložena na patřičnou pozici bytu. GPIO linka je nastavena jako výstupní a při devátém synchronizačním impulsu

MCU nastaví úroveň RH_DATA na log 1 v případě, že má být komunikace ukončena a senzor přechází do režimu spánku. Pokud je nastavena úroveň na log 0, potom senzor odesílá další byte dat, případně čeká na další příkaz. Časový diagram je zobrazen na obrázku 5.8.

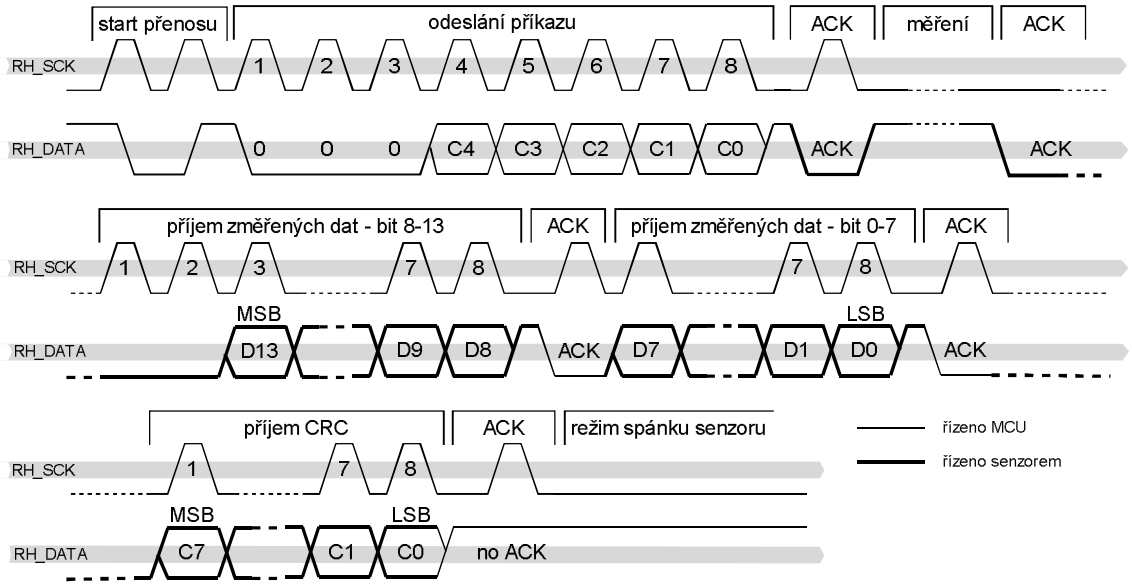


Obr. 5.8 Sekvence čtení bytu dat ze senzoru [3]

Funkce vrací *unsigned char* hodnotu přečtených dat.

5.4.4 Měření a přenos dat RH a teploty

Pro měření a přenos změřených dat byla vytvořena funkce *uint8_t RHsens_measure(uint16_t *val, uint8_t *checksum, uint8_t mode)*. Funkce spustí přenos pomocí *void RHsens_transStart(void)* uvedené v kap. 5.4.1, dále odešle příkaz pro měření teploty případně vlhkosti pomocí funkce *uint8_t RHsens_writeByte(uint8_t value)* a MCU čeká, dokud senzor nepotvrdí změření požadované veličiny, tedy nastavení datové linky RH_DATA do stavu log 0. Poté jsou přečteny dva byty naměřené veličiny pomocí funkce *uint8_t RHsens_readByte(uint8_t ack)* a jeden byte kontrolního součtu a senzor přechází do režimu spánku. Přijatá data jsou pomocí rotací a maskování sloučena do 16bitové hodnoty a uložena do paměti dat MCU. Časový diagram je uveden na obrázku 5.9.



Obr. 5.9 Časový diagram měření dané veličiny

5.4.5 Přepočet RH a teploty z přijatých dat senzoru

Tento přepočet obstarává funkce *void RHsens_calc(float *RH ,float *Temp, uint16_t measRH, uint16_t measTemp)*. Tato funkce realizuje přepočet z přijatých dat ze senzoru podle vztahu (3.1) pro RH (%) a vztahu (3.2) pro teplotu (°C), které jsou uvedeny v kap. 3.3.1.

5.4.6 Získání hodnot měření

Hlavní funkcí driveru je *uint8_t RHsens_getMeasurement(float *RH, float *Temp)*, která volá ostatní popsané funkce driveru, a to spouští měření vlhkosti, poté teploty a zavolá funkci pro přepočet získaných hodnot. Funkce vrací hodnotu 0x00, pokud nenastane žádná chyba v kterémkoliv kroku měření, v opačném případě vrací hodnotu ≥0x01.

5.5 Driver teplotního senzoru DS18B20

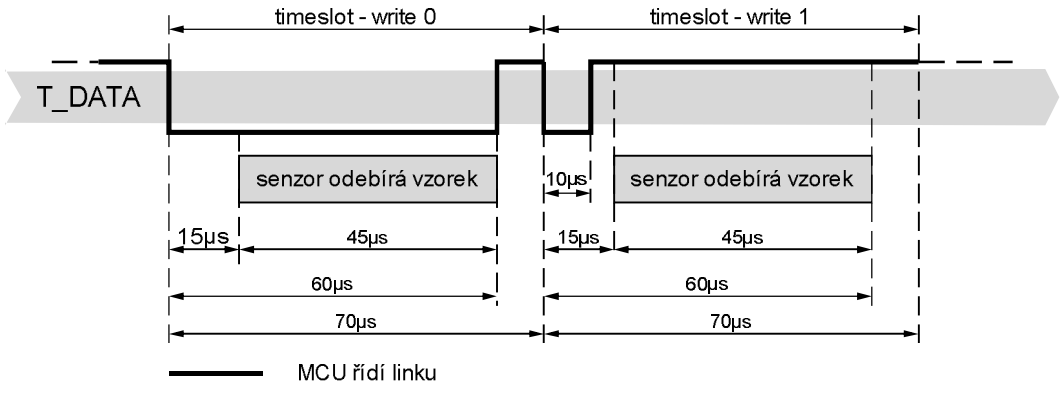
Driver teplotního senzoru DS18B20 disponuje 1-Wire sériovým rozhraním. Pro komunikaci po této lince byly vytvořeny funkce pro čtení a odeslání bytu dat *uint8_t Tsens_readByte(void)* a *void Tsens_writeByte(uint8_t val)*. Komunikace po lince je prováděna v přesně definovaných timeslotech.

Pro inicializaci, respektive reset přenosu, byla vytvořena funkce *uint8_t Tsens_reset(void)* a pro samotné měření a převod teploty funkce *float Read_ Temperature(void)*.

Všechny funkce pro obsluhu senzoru jsou definovány v souboru *./code/drivers/DS18B20/DS18B20.c*.

5.5.1 Zápis bytu dat 1-Wire linky

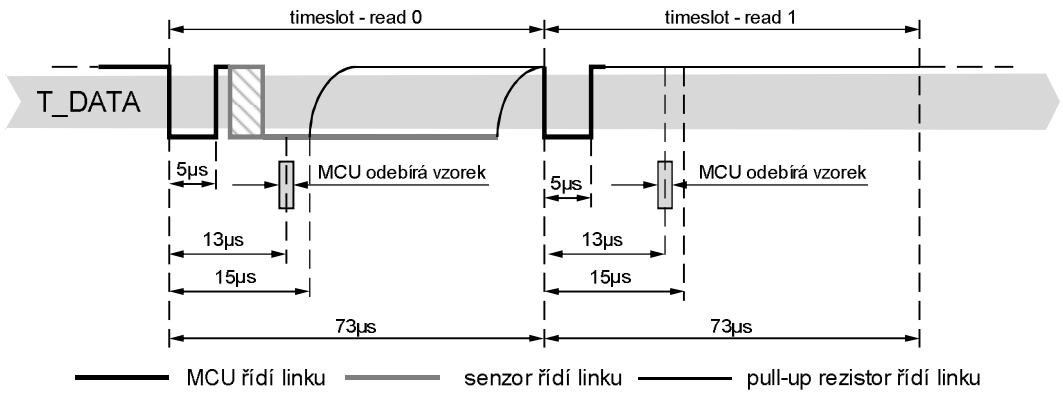
Zápis bytu dat na linku je uskutečněn osmi zápisovými timesloty typu write 0 nebo write 1. Jeden timeslot tedy odpovídá jednomu bitu odesílané hodnoty. Pořadí timeslotů je od LSB k MSB odesílané hodnoty[5]. Podoba a časování jednotlivých write timeslotů je uvedeno na obr. 5.10. Odeslání a korektní časování jednoho timeslotu zajišťuje funkce *void Tsens_writeBit(uint8_t val)*.



Obr. 5.10 Časování timeslotů write 0 a write 1

5.5.2 Čtení bytu dat 1-Wire linky

Čtení bytu dat na linku je uskutečněno osmi čtecími timesloty typu read 0 nebo read 1. Jeden timeslot tedy odpovídá jednomu bitu čtené hodnoty. Pořadí timeslotů je od LSB k MSB přijaté hodnoty[5]. Podoba a časování jednotlivých read timeslotů je uvedeno na obr. 5.11. Čtení a korektní časování jednoho timeslotu zajišťuje funkce *uint8_t Tsens_readBit(void)*.

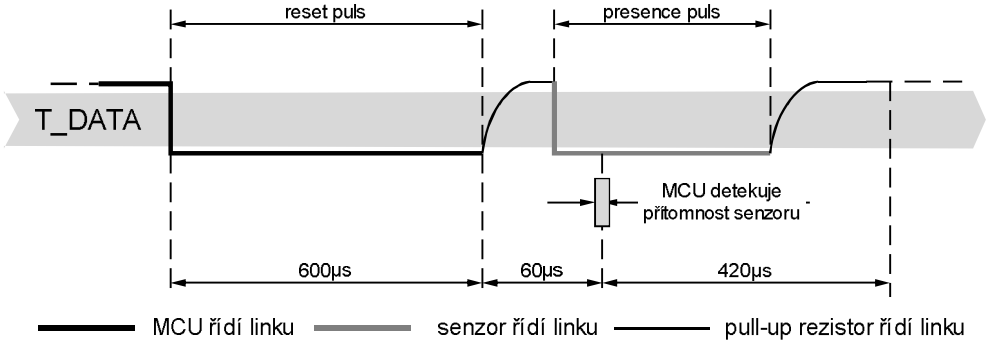


Obr. 5.11 Časování timeslotů read 0 a read 1

5.5.3 Inicializační procedura

Veškerá komunikace se senzorem je započata inicializační procedurou[5]. Tuto proceduru vykonává funkce *uint8_t Tsens_reset(void)*. MCU nejdříve vyšle reset puls a poté čte stav linky. Na reset puls senzor odpovídá presence pulsem, z čehož MCU zjistí, zda je senzor k lince připojen. Pull-up rezistor R_{13} poté vrací linku do stavu log 1.

Pokud je detekována přítomnost senzoru, funkce vrací hodnotu 0x00, v opačném případě 0x01. Časování inicializační procedury je uvedeno na obr. 5.12.



Obr. 5.12 Časování inicializační procedury

5.5.4 Měření, přenos a přepočet dat teploty

Měření, přenos a přepočet dat teploty zajišťuje funkce *float Read_Temperature(void)*. Zdrojový kód této funkce je uveden v příloze E.1. Jelikož datalogger využívá na 1-Wire lince pouze jeden senzor, není třeba jej identifikovat pomocí unikátního 64bitového identifikačního čísla senzoru [5].

Proces měření začíná inicializační procedurou. Proběhne-Ĺi úspěšně, je odeslán příkaz pro přeskočení identifikace senzoru SKIP-ROM (odeslání bytu 0xCC) a příkaz pro spuštění konverze teploty CONVER_T (odeslání bytu 0x44). Senzor využívá režimu parazitního napájení datovou linkou, proto je datová linka T_DATA na dobu 750 ms vázána k napájení dataloggeru MOSFET tranzistorem Q_3 (příloha A.4). Po uplynutí uvedené doby je převod teploty ukončen a změřená hodnota je uložena v paměti senzoru.

Následuje druhá inicializační procedura a přeskočení identifikace senzoru. Je odeslán příkaz pro přečtení paměti senzoru READ_SP (odeslání bytu 0xBE), MCU přečte z linky 9 bytů dat odpovídající paměti senzoru. V prvním přečteném bytu se nachází LSB, ve druhém MSB změřené hodnoty.

Proces převodu teploty spočívá v konverzi vyčtených dat na hodnotu typu *float*. Z přečtených LSB a MSB je vytvořena 16bitová hodnota t_{BIN}. Výpočet hodnoty teploty typu float t_{float} je proveden podle vztahu (5.1) pro kladné hodnoty a vztahu (5.2) pro záporné hodnoty teploty[5].

$$t_{float} = \frac{t_{BIN}}{16}, \qquad [°C, -] \tag{5.1}$$

$$t_{float} = -\frac{t_{BIN}+1}{16}, \qquad [°C, -] \tag{5.2}$$

5.6 Driver SPI Flash paměti AT25DF161

Driver zajišťuje především funkce pro zápis, čtení a mazání logovaných dat externí flash paměti. Bylo nutné také definovat funkce umožňující odstranění ochrany proti zápisu do paměti, inicializaci SPI periferie MCU, obsluhu režimu nízké spotřeby paměti a další podpůrné funkce potřebné k vykonávání jednotlivých operací.

Všechny funkce pro obsluhu senzoru jsou definovány v souboru *./code/drivers/AT25DF161/AT25DF161.c*.

5.6.1 Inicializace SPI periferie

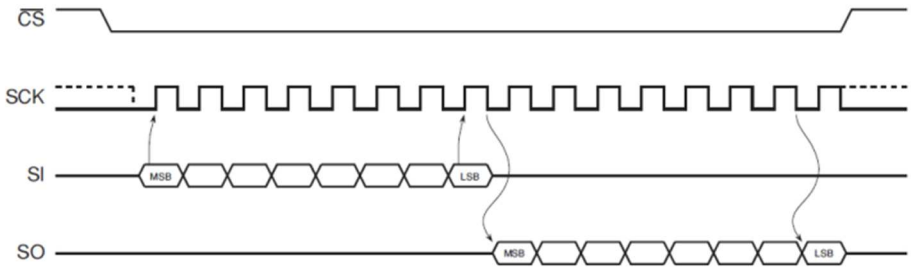
Inicializace SPI periferie spočívá v povolení taktovacího kmitočtu periferie nutného pro její funkci a nastavení následujících parametrů:

- směr přenosu: 2 linky, plně duplexní
- mód MCU: master
- datový rámec: 8 bitů
- SPI mód: 3
- baudRate dělička: /4
- první bit rámce: MSB.

5.6.2 SPI komunikace

Jeden komunikační cyklus MCU s pamětí pomocí SPI rozhraní je zobrazen na obr. 5.13 a zajišťuje ho funkce *uint8_t Mem_sendByte(uint8_t byte)*. Jedná se vždy o zápis a

přečtení jednoho bytu přenášených dat. Při jakékoliv komunikaci musí být chip select signál MEM_CS (příloha A.3 a A.1) na úrovni log 0.



Obr. 5.13 Komunikační cyklus SPI [7]

MCU nejdříve kontroluje, zda je výstupní Tx datový buffer SPI periferie vyprázdněn a odešle osm bitů dat. Poté MCU čeká, dokud není vstupní Rx datový buffer SPI periferie naplněn přijatými daty. Jakmile k tomu dojde, funkce přečte hodnotu přijatého bytu a vrací hodnotu *unsigned char* přijatého bytu dat.

5.6.3 Čtení status registru

Status registr paměti udává aktuální stav paměti, například stav povolení zápisu, ochrany proti zápisu, hlášení o chybě posledního zápisu nebo mazání dat a také informaci o zaneprázdnění paměti vnitřními operacemi. Status registr je rozdělen do dvou bytů[7].

MCU musí být schopno při práci s pamětí číst tento status registr. K tomu slouží funkce *uint16_t Mem_readSREG(void)*. Tato funkce odesílá příkaz pro čtení status registru paměti MEM_READ_SREG (byte 0x05). Dále MCU odesílá dva dummy byty MEM_DUMMY (byl zvolen byte 0xA5, nesmí odpovídat žádnému jinému příkazu paměti). Na každý dummy byte paměť odpovídá odesláním hodnot bytu 1 a 2 status registru. Funkce vrací hodnotu status registru v podobě 16bitové hodnoty *unsigned short*.

5.6.4 Zápis bytu 0 status registru

MCU musí být schopno měnit hodnoty ve status registru paměti. K tomuto účelu je vytvořena funkce *void Mem_writeSREG_byte1(uint8_t SREGval)*.

Funkce odesílá příkaz MEM_WRITE_SREG_B1 (byte 0x01) a byte požadované hodnoty status registru.

5.6.5 Povolení zápisu do paměti

Před každým zápisem nebo mazáním dat paměti musí být odeslán příkaz pro povolení zápisu. K tomuto účelu slouží funkce *void Mem_writeEnable(uint8_t status)*. Funkce odesílá do paměti příkaz MEM_WCMD_ENABLE (byte 0x06). Touto funkcí lze také zakázat zápis, a to odesláním příkazu MEM_WCMD_DISABLE (byte 0x04).

5.6.6 Čekání na dokončení operace zápisu do paměti

Po každém zápisu a mazání dat paměti musí MCU čekat na potvrzení dokončení vnitřních operací paměti. Až poté mohou být vykonávány další operace s pamětí. K tomuto účelu slouží funkce *void Mem_waitForWriteEnd(void)*.

Nejdříve je provedena funkce čtení status registru[4]. Dále MCU opakovaně odesílá dummy byte MEM_DUMMY. Na každý dummy byte paměť odpovídá střídavým odesíláním hodnoty status registru bytu 0 a 1. První bit v status registru bytu 0 i 1 je RDY/BSY stav. Je-li tento bit status registru v log 0, indikuje ukončení vnitřních operací paměti a program může pokračovat.

5.6.7 Úsporný režim paměti

Paměť disponuje režimem nízké spotřeby, do kterého může být přepnuta, když není potřeba k paměti přistupovat. K přechodům do a z úsporného režimu slouží funkce void Mem_PWR(uint8_t status).

Pro uspání paměti funkce odesílá příkaz MEM_SLEEP (byte 0xB9) a pro probuzení z režimu nízké spotřeby příkaz MEM_RESUME (byte 0xAB). Odeslaný příkaz odpovídá vstupnímu parametru funkce *status*.

5.6.8 Mazání dat v paměti

Paměť lze smazat buď celá nebo jen po blocích definovaných délek. Datalogger používá pouze funkci pro mazání celé paměti *void Mem_eraseAll(void)*.

Tato funkce nejdříve povoluje zápis do paměti voláním funkce *Mem_writeEnable()*[5], dále odesílá příkaz pro smazání celé paměti MEM_ERASE_ALL (byte 0xC7) a čeká na ukončení operace voláním funkce *Mem_waitForWriteEnd( )*.

5.6.9 Čtení dat z paměti

Čtení dat uložených v paměti zajišťuje funkce *void Mem_readArray(uint32_t ReadAddr, uint8_t* pBuffer, uint16_t NumByteToRead)*. Vstupními parametry funkce jsou adresa v rozsahu 0x0000 0000 až 0x001F FFFF prvního bytu, který má být přečten, ukazatel na paměťový prostor MCU, kam mají být přečtená data uložena, a na závěr počet bytů dat, která mají být přečtena.

Funkce odešle příkaz pro čtení z paměti MEM_READ (byte 0x03) a následně je opakovaně odesílán dummy byte. Počet opakování odpovídá počtu bytů dat, která mají být přečtena. Na každý dummy byte paměť odpovídá odesláním bytu dat příslušné paměťové buňky a ta je následně ukládána do paměťového prostoru MCU.

5.6.10 Zápis dat do paměti

Paměť je organizována na stránky o délce 256 bytů. Jedním příkazem pro zápis lze zapisovat vždy jen do rozsahu jedné stránky. K tomuto účelu slouží funkce *void Mem_writePage(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite)*.

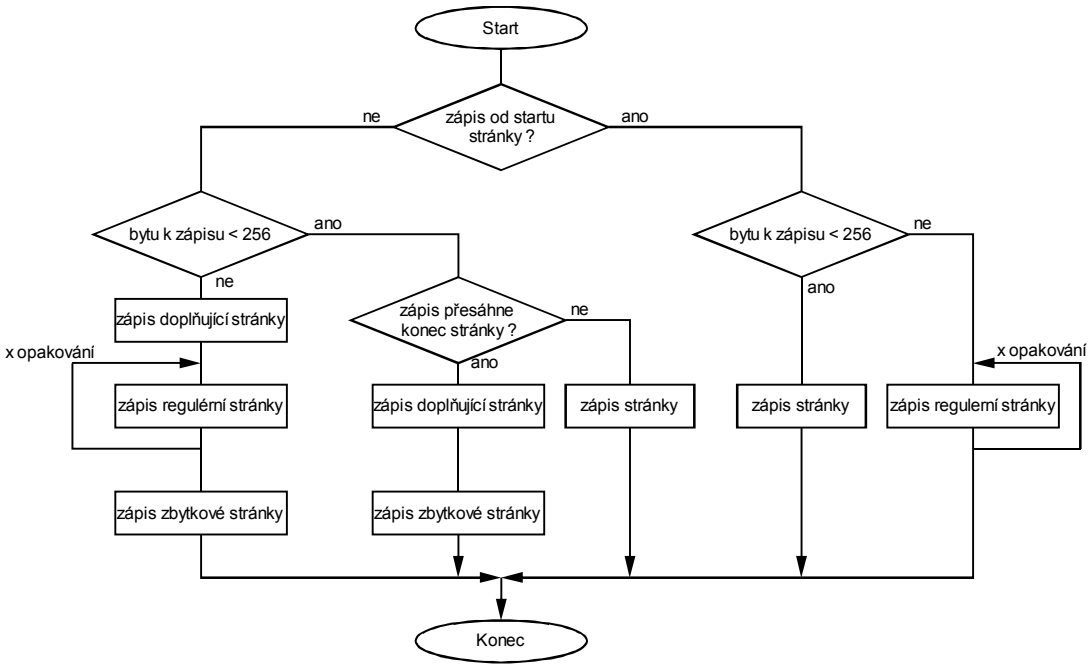
Vstupními parametry této funkce jsou ukazatel na paměťový prostor MCU, kde se

[4] Čtení status registru - viz. kap. 5.6.3.
[5] Povolení zápisu do paměti – viz. kap. 5.6.5.

nacházejí data, která mají být zapsána, počáteční adresa, od které bude probíhat zápis a počet bytů, které mají být zapsány.

Tato funkce nejdříve povoluje zápis do paměti voláním funkce *Mem_writeEnable()*[6] a odesílá příkaz pro smazání celé paměti MEM_WRITE (byte 0x02). Poté je odeslána adresa rozdělená do tří bytů, postupně od MSB k LSB. Dále jsou odesílány byty zapisovaných dat, dokud se nedosáhne počtu bytů k zápisu a čeká se na skončení operace zápisu do paměti voláním funkce *Mem_waitForWriteEnd( )*[7].

Pro snadnější zápis dat bez omezení rozsahu stránek paměti byla vytvořena funkce *void Mem_writeBuffer(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite)*. Zdrojový kód této funkce je uveden v příloze E.2. Tato funkce využívá výše popsanou funkci pro zápis stránky paměti a pouze upravuje její vstupní parametry. Postup rozhodování, jak bude volána funkce zápisu stránky, je uveden v diagramu na obr. 5.14.



Obr. 5.14 Diagram rozhodování funkce zápisu dat do paměti

5.7 Jednotka reálného času RTC a BKP doména

Datalogger využívá RTC jednotku ke dvěma základním úkolům - implementaci reálného času a data a časování logování. Všechny dále popsané funkce modulu jsou definovány v souboru *./code/drivers/RTC/RTC.c*.

Jádrem RTC jednotky je 32bitový čítač. RTC jednotka je taktována hodinovým krystalem X2 (příloha A.1) o frekvenci f = 32,768kHz. RTC čítač je inkrementován každou sekundu a z jeho hodnoty je možné určit aktuální čas a datum.

[6] Povolení zápisu do paměti – viz kap. 5.6.5.
[7] Čekání na dokončení zápisu do paměti – viz kap. 5.6.6.

Jednotka RTC se nachází v BKP doméně MCU[8].

5.7.1 BKP doména MCU, uživatelské registry

BKP doména je periferie MCU, která může být napájena separovaně od zbytku MCU skrze pin VBAT (příloha A.1). BKP doména zůstává aktivní ve všech režimech nízké spotřeby MCU i při odpojení hlavního napájení MCU. V BKP doméně se nalézá RTC jednotka se svými registry a dále se zde nacházejí uživatelsky definovatelné registry. Vybraný MCU disponuje deseti uživatelskými registry BKP_DR1 až BKP_DR10. Datalogger používá pro svůj chod několik těchto registrů.

Definice jednotlivých uživatelských registrů je uvedena v souboru *./code/app/ /inc/app_defs.c* a rozebrána dále.

BKP_SREG

V tomto registru jsou uloženy provozní informace (stav měření a aktuálně měřené veličiny) a jeden konfigurační bit. Zabírá uživatelský registr MCU - BKP_DR1. Struktura je uvedena v tabulce 9.

Tab. 9 Uživatelské registry – struktura BKP_SREG

BKP_SREG[0-6]							
BKP_DR1[15-7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
-	CFG	mP	mRH	mT2	mT1	ST	M

bit 1: **M** - mód logování
- log 0 – volně běžící
- log 1 – plánované

bit 2: **ST** - stav logování
- log 0 – pozastaveno
- log 1 – spuštěno

bit 3: **mT1** – logování teploty T1 - log 0 – neaktivní / log 1 – aktivní

bit 4: **mT2** – logování teploty T2 - log 0 – neaktivní / log 1 – aktivní

bit 5: **mRH** – logování RH - log 0 – neaktivní / log 1 – aktivní

bit 6: **mP** – logování AT - log 0 – neaktivní / log 1 – aktivní

bit 7: **CFG** – inicializační konfigurace paměti
- log 0 – formátování paměti při nahrání nové konfigurace
- log 1 – ponechání dat v paměti při nahrání nové konfigurace

BKP_LASTMEMADDRx

Do tohoto registru je uložena 24bitová adresa posledního zapsaného bytu dat do externí flash paměti. Každá funkce zapisující do flash paměti upravuje hodnotu tohoto registru. Zabírá dva uživatelské registry MCU - BKP_DR2 a BKP_DR3. Struktura registrů je uvedena v tabulce 10.

[8] BKP doména – viz kap. 5.7.1.

Tab. 10 Uživatelské registry – struktura BKP_LASTMEMADDRx

BKP_LASTMEMADDR2[0-7]		BKP_LASTMEMADDR1[0-15]	
BKP_DR3[15-8]	BKP_DR3[7-0]	BKP_DR2[15-8]	BKP_DR2[7-0]
-	addr [23-16]	addr [15-8]	addr [7-0]

BKP_TIMEINTERVALx

V tomto registru je při konfiguraci uložena hodnota periody logování. Jedná se o 24bitovou hodnotu vyjádřenou v sekundách. Zabírá dva uživatelské registry MCU - BKP_DR4 a BKP_DR5. Struktura registrů je uvedena v tabulce 11.

Tab. 11 Uživatelské registry – struktura BKP_TIMEINTERVALx

BKP_TIMEINTERVAL2[0-7] (BKP_DR3)		BKP_TIMEINTERVAL1[0-15]	
BKP_DR5[15-8]	BKP_DR5[7-0]	BKP_DR4[15-8]	BKP_DR4[7-0]
-	per [23-16]	per [15-8]	per [7-0]

BKP_DATAREMAINED

Do tohoto registru je ukládáno až osm bitů zbylých dat z komprese posledního logu (data[0-7]). V horní polovině registru je uložen počet bitů délky zbylých dat(bitCnt[0-7]). Tento registr zabírá uživatelský registr MCU – BKP_DR6. Struktura registru je uvedena v tabulce 12.

Tab. 12 Uživatelské registry – struktura BKP_DATAREMAINEDx

BKP_DATAREMAINED[15-0]	
BKP_DR6[15-8]	BKP_DR6[7-0]
bitCnt[0-7]	data[0-7]

BKP_BLOCKCOUNT

Do tohoto registru je ukládán počet logů dat od začátku aktuálního bloku, viz kap. 3.5. Tento registr zabírá uživatelský registr MCU – BKP_DR7. Struktura registru je uvedena v tabulce 13.

Tab. 13 Uživatelské registry – struktura BKP_BLOCKCOUNT

BKP_BLOCKCOUNT[7-0]	
BKP_DR7[15-8]	BKP_DR7[7-0]
-	blockCnt[0-7]

BKP_BYTECOUNT

Do tohoto registru je ukládán počet bytů komprimovaných dat uložených v externí flash paměti od začátku aktuálního bloku dat[9]. Tento registr zabírá uživatelský registr MCU – BKP_DR8. Struktura registru je uvedena v tabulce 14.

[9] Struktura bloku dat – vit kap. 3.5.

Tab. 14 Uživatelské registry – struktura BKP_BYTECOUNT

BKP_BYTECOUNT[15-0]	
BKP_DR8[15-8]	BKP_DR8[7-0]
byteCnt[15-8]	byteCnt[0-7]

BKP_SUMMERTIME

Tento registr se vztahuje k reálnému času a datu. Obsahuje informaci o tom, zda se RTC údaj nachází v letním nebo zimním času. Struktura registru je uvedena v tabulce 15.

Tab. 15 Uživatelské registry – struktura BKP_SUMMERTIME

BKP_SUMMERTIME[0]		
BKP_DR9[15-8]	BKP_DR9[7-1]	[0]
-	-	S

bit 1: **S** - mód logování - log 0 – zimní čas
- log 1 – letní čas

5.7.2 Konfigurace RTC jednotky

Konfigurace RTC jednotky spočívá v povolení taktování rozhraní RTC a BKP, výběru zdroje RTC čítače (externí krystalový oscilátor X2), nastavení hodnoty registru děličky vstupního kmitočtu, tak aby byl RTC čítač inkrementován právě každou sekundu podle vztahu (5.3).

$$PRL = \frac{f_{X2}}{f_{RTC}} - 1 = \frac{32768}{1} - 1 = 32767 \tag{5.3}$$

Konfigurace RTC jednotky je zajištěna funkcí *void RTC_config(void)*. Před ukončením funkce je ještě vyčkáno na synchronizaci sběrnic BKP domény.

5.7.3 Synchronizace času RTC jednotky

Pro synchronizaci času RTC jednotky dataloggeru byla vytvořena funkce *void RTC_setTime(struct dateTime_TypeDef newDateTime)*. Vstupem této funkce je struktura *dateTime_TypeDef*. V této struktuře je uloženo synchronizační datum a čas.

Nastavení času a data spočívá ve výpočtu a uložení hodnoty registru RTC čítače. Nová hodnota RTC čítače je vypočítána jako časový rozdíl mezi synchronizačním a referenčním datem a časem vyjádřeným v sekundách. Jako referenční byl zvolen čas 00:00 a datum 1.1.2010.

Přepočet vykonává funkce *uint32_t RTC_countRegVal(struct dateTime_TypeDef dateTime)*. Nejdříve je stanoven počet dní mezi referenčním a požadovaným časem (ddCnt) s ohledem na přestupné roky a dopočítána hodnota pro RTC registr podle vztahu (5.4) a vrácena ve formátu *unsigned int*.

$$RTCregVal = ddCnt \cdot 86400 + hh * 3600 + mm * 60 + ss \tag{5.4}$$

Dalším krokem při nastavení času je zjištění, zda synchronizační čas náleží do zimního nebo letního času. Tato skutečnost je zapsána do uživatelského registru BKP_SUMERTIME[10].

5.7.4 Čtení data a času RTC jednotky

Pro čtení data a času RTC jednotky slouží funkce *void RTC_getTime(void)*. Zdrojový kód této funkce je uveden v příloze E.3. Tato funkce načte hodnotu RTC čítače a nejdříve kontroluje, zda došlo od minulého čtení času k posunu mezi letním a zimním časem. To je provedeno vyhodnocením aktuálního stavu uživatelského registru BKP_SUMMERTIME a výsledku porovnávání hodnoty RTC čítače s hodnotami v definovaných tabulkách pro přechody mezi letním a zimním časem. V případě, že od posledního načtení času došlo k časovému posunu, je adekvátně přičteno nebo odečteno 3600 s od hodnoty RTC čítače a upravena hodnota BKP_SUMMERTIME registru.

Následuje určení data pomocí smyčky, kde jsou postupně inkrementovány tři dočasné čítače pro dny, měsíce a roky podle kalendáře a dopočítána odpovídající hodnota v sekundách pro rozdíl data čítačů a referenčního data. Tato odpovídající hodnota je porovnávána s načtenou hodnotou RTC čítače a pokud ji přesáhne, smyčka je ukončena a datum z dočasných čítačů zaznamenáno. Aktuální čas je poté dopočítán z rozdílu *val* mezi odpovídající hodnotou a hodnotou RTC čítače podle vztahů (5.5) až (5.7).

$$hh = val/3600 \tag{5.5}$$

$$mm = (val\%3600)/60 \tag{5.6}$$

$$ss = (val\%3600)\%60 \tag{5.7}$$

pozn. k matematickým operacím pro vzorce (5.5) až (5.7):

/ - celočíselné dělení

% - modulo

Načtené datum a čas je uložen do globální proměnné *dateTime_TypeDef currentDateTime* a funkce je ukončena.

5.7.5 Alarm RTC jednotky

RTC jednotka disponuje třemi druhy přerušení – přetečení RTC čítače, sekundové přerušení (při inkrementování RTC čítače) a přerušení typu alarm. Pro datalogger je nejdůležitější přerušení typu alarm. Přerušení je generováno, pokud hodnota RTC čítače dosáhne hodnoty alarm registru.

RTC jednotka umožňuje také probuzení MCU z režimů nízké spotřeby v případě alarmu.

Pro nastavování hodnoty alarm registru během logování slouží funkce *void RTC_updateAlarm(void)*. Funkce uloží do alarm registru součet aktuální hodnoty RTC čítače a periody měření uložené v registrech BKP_TIMEINTERVALx.

[10] Uživatelské registry – viz kap. 5.7.1.

5.8 Implementace komunikačního protokolu

Teorie ke komunikačnímu protokolu již byla uvedena v kap. 3.7. Konkrétní implementace protokolu a funkcí s ním spojených bude probrána v následujících kapitolách.

Při implementaci komunikačního protokolu byla použita programovací knihovna STM32F10x and STM32L1xx USB Full Speed Device Library. Konkrétní základ tvořilo Custom HID Demo. V tomto demu byla přizpůsobena konfigurace endpointů, vytvořeny nové deskriptory reportů a vytvořen systém zpracování příchozích reportů.

5.8.1 Deskriptor reportů

Aplikace využívá tři typy reportů. Každý report je označen svým jednoznačným ID a popsán v deskriptoru reportů.

Implementovaný deskriptor reportů:

```
const uint8_t CustomHID_ReportDescriptor[CUSTOMHID_SIZ_REPORT_DESC] =
{
     0x06, 0x00, 0xff, // USAGE_PAGE (Vendor Defined Page 1)
     0x09, 0x01,       // USAGE (Vendor Usage 1)
     0xa1, 0x01,       // COLLECTION (Application)
     // ------ common globals -----------------------------------
     0x15, 0x00,       // LOGICAL_MINIMUM (0)
     0x26, 0x00, 0xFF, // LOGICAL_MAXIMUM (255)
     0x75, 0x08,                        //   REPORT_SIZE (8)   - bits
     // ------ input report - data MCU to HOST ------------------
     0x85, 0x02,       /*     REPORT_ID (2)                   */
     0x95, 0x3F,       // REPORT_COUNT      - bytes
     0x09, 0x01,       // USAGE (Vendor Usage 1)
     0x81, 0x02,       // INPUT (Data,Var,Abs)
     // ------ output report - cmd HOST to MCU ------------------
     0x85, 0x01,       /*     REPORT_ID (1)                   */
     0x95, 0x03,       // REPORT_COUNT      - bytes
     0x09, 0x01,       // USAGE (Vendor Usage 1)
     0x91, 0x02,       // OUTPUT (Data,Var,Abs)
     // ------ output report - config report HOST to MCU ---------
     0x85, 0x03,       /*     REPORT_ID (3)                   */
     0x95, 0x40,       // REPORT_COUNT      - bytes
     0x09, 0x01,       // USAGE (Vendor Usage 1)
     0x91, 0x02,       // OUTPUT (Data,Var,Abs)

     0xc0              // END_COLLECTION

}; /* CustomHID_ReportDescriptor */
```

Popis jednotlivých reportů je uveden dále.

Report ID 0x02

typ reportu: IN (směr přenosu dat z MCU do PC)

délka paketu: 63B

popis: Report slouží především k odesílání logovaných dat do PC. Tímto reportem je také přenášeno potvrzení operace MCU.

Report ID 0x01

typ reportu: OUT (směr přenosu dat z PC do MCU)

délka paketu: 3B

popis: Report slouží k přenosu krátkých jednoduchých příkazů.

Report ID 0x03

typ reportu: OUT (směr přenosu dat z PC do MCU)

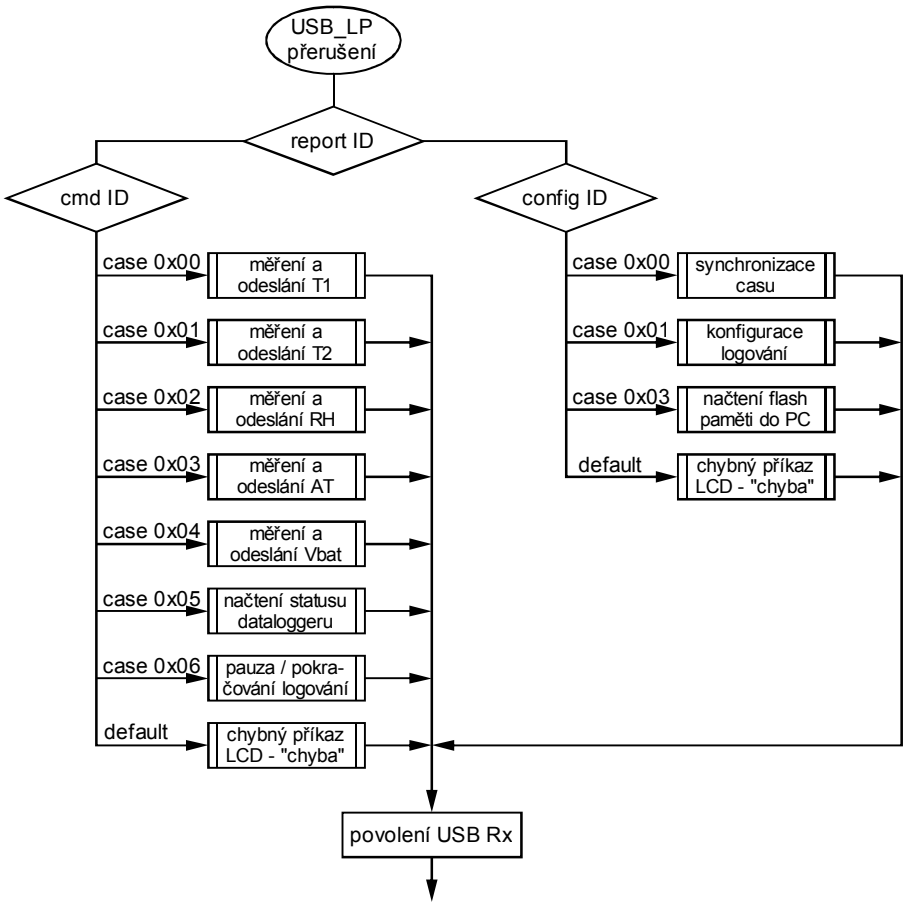
délka paketu: 64B

popis: Report slouží k přenosu konfiguračních údajů a složitějších příkazů s rozšiřujícími daty. Je použit i při načítání logovaných dat z paměti dataloggeru.

5.8.2 Zpracování příchozích reportů

Veškerou komunikaci pomocí HID třídy vyvolává PC. Data přicházející z PC naplňují příchozí Rx buffer. Po jeho naplnění je vyvoláno přerušení od USB[11]. Přerušení spouští funkci pro základní obsluhu příchozích dat endpointu 1 *void EP1_OUT_Callback(void)*.

Funkce obsahuje dvouúrovňové rozhodování. Nejdříve se funkce člení na obsluhu reportů ID 0x01 (příkazy) a poté reportů ID 0x03 (konfigurace). Další členění podléhá subID (druhý byte reportu). Popis konkrétních subID a datových rámců je uveden v kap.5.8.3. Funkce je naznačena v algoritmu na obr. 5.15.



Obr. 5.15 Algoritmus obsluhy USB přerušení

[11] USB přerušení – viz kap. 5.11.2.

Jednotlivé funkce pro obsluhu příkazů jsou uvedeny v kap 5.10 a pro obsluhu konfigurace v kap. 5.9.

5.8.3 Přenášené datové rámce

Každý přenášený report resp. datový rámec je specifikován svým report ID (první byte reportu) a subID (druhý byte reportu). Další obsah rámců se liší podle konkrétního použití. Následující struktury datových rámců jsou striktně dodržovány jak ve firmwaru pro MCU, tak v softwaru pro PC.

Report ID 0x01 – příkazy (přenos z PC do MCU)

Datový rámec pro všechny report ID 0x01 příkazy je stejný. Struktura rámce je uvedena v tabulce 16.

Tab. 16 Struktura datového rámce report ID 0x01 - příkazy

byte no.	obsah bytu	hodnota
00	reportID	0x01
01	subID	0x00 – 0x05
02	přídavná data	N/A

V tomto rámci je přenášeno pouze report ID a subID pro identifikaci příkazu, který má být vykonán. Seznam implementovaných příkazů je uveden v tabulce 17.

Tab. 17 Seznam příkazů pro report ID 0x01

identifikátor[12]	subID	popis
MEASURE_TEMP1	0x00	měření a odeslání teploty T1
MEASURE_TEMP2	0x01	měření a odeslání teploty T2 - externího senzoru
MEASURE_RH	0x02	měření a odeslání relativní vlhkosti RH
MEASURE_PRESS	0x03	měření a odeslání atmosférického tlaku
MEASURE_VBAT	0x04	měření a odeslání napětí baterie
GET_STATUS	0x05	načtení stavu dataloggeru
SET_STATUS_PAUSE	0x06	pozastavení logování
SET_STATUS_RUN	0x07	opětovné spuštění logování

Report ID 0x03 – konfigurace (přenos z PC do MCU)

Datové rámce pro každou konfigurační operaci jsou jiné. Seznam těchto operací a jejich identifikátorů je uveden v tabulce 18.

[12] Příkazy jsou definovány v souboru *./code/app/inc/app_defs.h*

Tab. 18 Seznam konfiguračních operací pro report ID 0x03

identifikátor[12]	subID	popis
CONFIG_SYNC_TIME	0x00	synchronizace času
CONFIG_LOGGING	0x01	nastavení logování
CONFIG_READ_FLASH	0x03	načtení flash paměti do PC

Report ID 0x03 – synchronizace času

V datovém rámci je přenášeno šest bytů dat obsahujících synchronizační čas a datum. Struktura datového rámce je uvedena v tabulce 19.

Tab. 19 Datový rámec synchronizace času

byte no.	obsah bytu	hodnota
00	reportID	0x03
01	subID	0x00
02	synchronizační čas - ss	DEC 0 – 60
03	synchronizační čas - mm	DEC 0 – 60
04	synchronizační čas - hh	DEC 0 – 23
05	synchronizační datum - DD	DEC 1 – 31
06	synchronizační datum - MM	DEC 1 – 12
07	synchronizační datum - YY	DEC 10 – 40[1]
08 - 63	prázdné byty	0x00

[1]- zavedena korekce údaje let odečtením hodnoty 2000

Report ID 0x03 – konfigurace logování

V datovém rámci je přenášeno šestnáct bytů dat obsahujících všechny potřebné údaje k nastavení logování. Tento datový rámec je zpracováván funkcí pro nastavení logování popsanou[13]. Struktura datového rámce je uvedena v tabulce 20.

Tab. 20 Datový rámec nastavení logování

byte no.	obsah bytu	hodnota
00	reportID	0x03
01	subID	0x01
02	perioda měření[1] - ss	DEC 0 – 60
03	perioda měření[1] - mm	DEC 0 – 60
04	perioda měření[1] - hh	DEC 0 – 23
05	perioda měření[1] - DD	DEC 1 – 190
06	status registr[2]	0x00 – 0xEF

[13] Nastavení logování – viz kap. 5.9.1.

07	čas startu logování – ss	DEC 0 – 60
08	čas startu logování – mm	DEC 0 – 60
09	čas startu logování – hh	DEC 0 – 23
10	datum startu logování – DD	DEC 1 – 31
11	datum startu logování – MM	DEC 1 – 12
12	datum startu logování – YY	DEC 10 – 40[3]
13	čas ukončení logování – ss	DEC 0 – 60
14	čas ukončení logování – mm	DEC 0 – 60
15	čas ukončení logování – hh	DEC 0 – 23
16	datum ukončení logování – DD	DEC 1 – 31
17	datum ukončení logování – MM	DEC 1 – 12
18	datum ukončení logování – YY	DEC 10 – 40[3]
19 - 63	prázdné byty	0x00

[1] – registr periody měření BKP_TIMEINTERVALx14
[2] – status registr BKP_SREG[14]
[3] – zavedena korekce údaje let odečtením hodnoty 2000

Report ID 0x03 – načtení logovaných dat z flash paměti do PC

V datovém rámci je přenášena adresa paměťové buňky, od které má být započato čtení a příkaz, zda má být odeslána pouze adresa poslední zapsané paměťové buňky (registr BKP_LASTMEMADDRx[14]) nebo data. Struktura datového rámce je uvedena v tabulce 21.

Tab. 21 Datový rámec načtení logovaných dat z flash paměti do PC

byte no.	obsah bytu	hodnota
00	reportID	0x03
01	subID	0x01
02	příkaz – odeslání adresy / dat	0x00 / 0x01
03	adresa začátku čtení - LSB	0x00 – 0xFF
04	adresa začátku čtení	0x00 – 0xFF
05	adresa začátku čtení - MSB	0x00 – 0xFF
06 - 63	prázdné byty	0x00

Report ID 0x02 – přenos dat z MCU do PC

Pro přenos dat z MCU do PC jsou definovány tři datové rámce, každý slouží k jinému účelu a jejich struktura se liší. Seznam těchto operací a jejich identifikátorů je uveden v tabulce 22.

[14] Uživatelké registry, viz kap. 5.7.1.

Tab. 22 Seznam operací pro přenos dat z MCU do PC

identifikátor[12]	subID	popis
STATUS_INFO	0x00	odeslání stavových informací
DATA_READOUT	0x01	odeslání dat z paměti flash do PC
CONFIRM_OPERATION	0xFF	potvrzení provedení operace iniciované z PC

Report ID 0x02 – odeslání stavových informací dataloggeru do PC

Tento datový rámec je odesílán jako odpověď na příchozí příkaz GET_STATUS. Obsahuje informace o aktuálním stavu dataloggeru, konkrétně o stavu zbývající kapacity paměti, stavu baterie a také obsahuje status registr dataloggeru14. Struktura datového rámce je uvedena v tabulce 23.

Tab. 23 Datový rámec pro odeslání stavových informací dataloggeru do PC

byte no.	obsah bytu	hodnota
00	reportID	0x02
01	subID	0x00
02	status register dataloggeru[1]	0x00 - 0xEF
03	obsazená kapacita paměti[2] - LSB	0x00 – 0xFF
04	obsazená kapacita paměti	0x00 – 0xFF
05	obsazená kapacita paměti - MSB	0x00 – 0xFF
06	napětí baterie – 1. digit	ASCII
07	napětí baterie – 2. digit	ASCII
08	napětí baterie – 3. digit	ASCII
09	napětí baterie – 4. digit	ASCII
10 - 63	prázdné byty	0x00

[1] – Status registr BKP_SREG[14].
[2] – Obsazená kapacita paměti odpovídá adrese posledního zápisu – registr BKP_LASTMEMADDRx[14].

Report ID 0x02 – odeslání dat z paměti flash do PC

V tomto datovém rámci jsou odesílána data načtená z flash paměti, případně adresa posledního zápisu do paměti. V tomto případě je tedy datový rámec dělen na dva druhy podle vyžádaných dat. Struktura datového rámce pro odesílání adresy posledního zápisu je uvedena v tabulce 24 a pro odeslání dat v tabulce 25.

Tab. 24 Datový rámec pro odeslání ukončovací adresy čtení

byte no.	obsah bytu	hodnota
00	reportID	0x02
01	subID	0x01
02	ukončovací adresa[1] – LSB	0x00 – 0xFF
03	ukončovací adresa	0x00 – 0xFF
04	ukončovací adresa – MSB	0x00 – 0xFF
10 - 63	prázdné byty	0x00

[1] – ukončovací adresa odpovídá adrese posledního zápisu – registr BKP_LASTMEMADDRx[14].

Tab. 25 Datový rámec pro odeslání načtených dat z flash paměti

byte no.	obsah bytu	hodnota
00	reportID	0x02
01	subID	0x01
02-62	60 bytů načtených dat z paměti	0x00 – 0xFF
63	prázdný byte	0x00

Report ID 0x02 – odeslání hodnoty měřené veličiny mimo logovací cyklus

Pro odeslání hodnoty měřené veličiny mimo cyklus logování při přímém měření aplikací v PC byl definován datový rámec, jehož struktura je uvedena v tabulce 26.

Tab. 26 Datový rámec pro odeslání měřené veličiny mimo logovací cyklus

byte no.	obsah bytu	hodnota
00	reportID	0x02
01	subID	0x02
02	hodnota veličiny – 1. digit	ASCII
03	hodnota veličiny – 2. digit	ASCII
04	hodnota veličiny – 3. digit	ASCII
05	hodnota veličiny – 4. digit	ASCII
06	hodnota veličiny – 5. digit	ASCII
07 - 63	prázdný byte	0x00

Report ID 0x02 – potvrzení provedení operace dataloggeru

Pro detekci a potvrzení déletrvajících operací v dataloggeru, jako je mazání flash paměti, je definován samostatný datový rámec. Jeho struktura je uvedena v tabulce 27.

Tab. 27 Datový rámec pro odeslání potvrzení proběhlé operace

byte no.	obsah bytu	hodnota
00	reportID	0x02
01	subID	0xFF
02 - 63	prázdný byte	0x00

5.9 Modul pro práci s logovanými daty

Funkce v tomto modulu slouží pro nastavení logování, provedení logování, následnou kompresi a uložení naměřených dat do paměti a také k odeslání obsahu paměti do PC. Všechny dále popsané funkce jsou definovány v souboru *./code/app/src/ /data_management.c*.

5.9.1 Nastavení logování

K nastavení logování je definována funkce *void configLogging(uint8_t* configBuffer)*. Tato funkce je volána při přijetí datového rámce z PC Report ID 0x03 – konfigurace logování. Funkce zpracuje jednotlivá pole těchto rámců a nastaví příslušné registry BKP_TIMEINTERVAL a BKP_SREG.

Pokud je vyžadována pouze úprava parametrů logování, aktuální blok dat ukládaných do paměti je ukončen[15].

Pokud je vyžadováno formátování paměti, je flash paměť smazána[16], od adresy 0x00000000 jsou nahrány časové údaje o startu a ukončení logování – byty sedm až patnáct datového rámce Report ID 0x03 – konfigurace logování.

Dalším krokem je nastavení hodnoty alarm registru podle požadovaných kritérií. V případě plánovaného logování je přerušení typu alarm nastaveno na čas startu měření. V případě úpravy parametrů je toto přerušení nastaveno tak, aby došlo k jeho vyvolání po dokončení této funkce pro nastavení logování.

Na závěr funkce odesílá do PC potvrzení o úspěšně proběhlé operaci – datový rámec Report ID 0x02 – potvrzení provedení operace dataloggeru.

5.9.2 Odeslání dat z flash paměti do PC

Pro odeslání logovaných dat z flash paměti do PC slouží funkce *void dataReadOut(uint8_t action, uint8_t readAddr)*. Funkce se dělí podle vstupního parametru *action* na dvě části – odeslání adresy posledního zápisu do paměti a odeslání části logovaných dat.

V případě odeslání poslední adresy zápisu je nejdříve ukončen aktivní blok, kde jsou ukládána logovaná data, a poté odeslána adresa datovým rámcem Report ID 0x02 – odeslání dat z paměti flash do PC (varianta rámce uvedená v tabulce 24).

V případě odeslání části logovaných dat jsou načtena data z paměti o délce 60 bytů[16], počínaje adresou specifikovanou vstupním parametrem funkce *readAddr*.

[15] Ukončení bloku dat – viz kap. 5.9.4.
[16] Driver flash paměti – viz kap. 5.6.

V tomto případě je použit datový rámec Report ID 0x02 – odeslání dat z paměti flash do PC (varianta rámce uvedená v tabulce 25).

5.9.3 Vytvoření a uložení hlavičky bloku dat

Pro vytvoření a uložení hlavičky bloku dat slouží funkce *void updateDataHeader(void)*. Tato funkce vytváří a ukládá hlavičku bloku dat dle struktury definované v kap. 3.5. Údaje počet logů v bloku a počet bytů komprimovaných dat jsou do hlavičky uloženy až při ukončení bloku dat.

5.9.4 Ukončení bloku dat

Pro ukončení bloku dat slouží funkce *void finalizeBlock(void)*. Blok dat je ukončen vždy při jeho naplnění 200 logy, při změnách parametrů logování, při vyčítání logovaných dat do PC, ale také v dalších nutných případech.

Tato funkce načítá obsah BKP_DATA_REMAINED a ukládá jej do posledních dvou bytů bloku. Také ukládá údaje počtu logů v bloku a počtu bytů komprimovaných dat z registrů BKP_BLOCKCOUNT a BKP_BYTECOUNT do hlavičky bloku dat. Na závěr funkce zmíněné registry nuluje.

5.9.5 Huffmanovo kódování hodnoty logované veličiny

Zakódování jedné hodnoty kterékoliv logované veličiny vykonává funkce *uint32_t encodeSingleMeasurement(float val)*. Vstupním parametrem této funkce je hodnota logované veličiny ve formátu float.

Vstupní hodnota je převedena na řetězec ASCI znaků. Jednotlivé znaky jsou uspořádány a upraveny tak, jak je definováno v kap. 3.5.1 a následně zakódovány Huffmanovým kódem.

Funkce vrací 32bitovou hodnotu *unsigned int*, která obsahuje posloupnost zakódovaných bitů a informaci o délce této posloupnosti. Struktura výstupu této funkce je uvedena v tabulce 28.

Tab. 28 Struktura výstupu funkce Huffmanova kódování hodnoty logované veličiny

bit	32 - 29	28 - 8	7 – 0
	-	zakódovaná posloupnost	délka posloupnosti

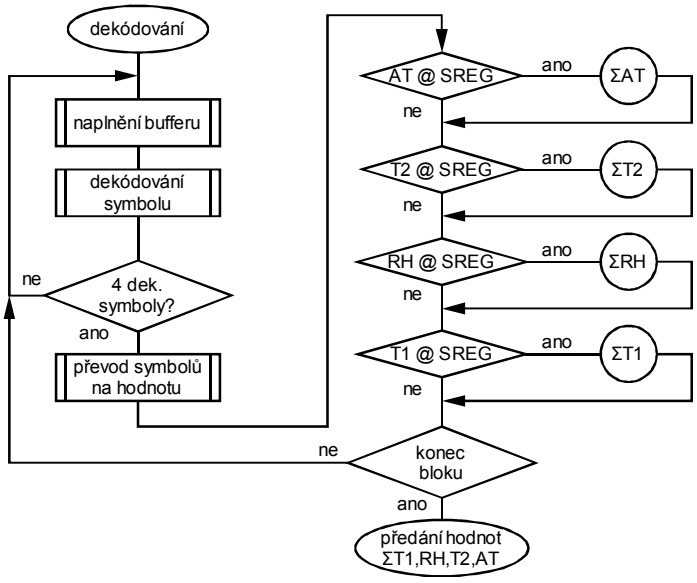
5.9.6 Funkce pro dekódování hodnot logovaných veličin

Jelikož je při kompresi dat použito diferenční kódování, je potřeba znát hodnoty posledních logovaných veličin. Jelikož vybraný MCU nedisponuje dostatkem uživatelských registrů, kde by mohly být tyto hodnoty uloženy v nekomprimované podobě, musela být vytvořena funkce, která tyto hodnoty dekóduje z bloku dat, kde jsou uloženy. K tomuto účelu slouží funkce *void decodeCurrentBlock(float *Press, float *RH, float *Temp2, float *Temp1)*.

Vstupními parametry této funkce jsou ukazatele typu *float*, kde se mají dekódované hodnoty uložit. Funkce využívá 16bitový vyrovnávací buffer. Do bufferu je postupně načítána posloupnost komprimovaných dat z flash paměti. Posloupnost

komprimovaných dat je dekódována z Huffmanova kódu principem popsaným v kap.3.5.4. Dekódované symboly jsou kumulovány a převáděny na hodnoty typu *float*. Následuje rozhodování, které veličině náleží daná hodnota podle status registru BKP_SREG. Průběžně jsou počítány sumy dekódovaných hodnot pro jednotlivé veličiny.

Sumy hodnot jednotlivých veličin odpovídají posledním logovaným hodnotám, jakmile je dekódována celá komprimovaná posloupnost. Tyto hodnoty jsou uloženy na adresy příslušných ukazatelů (vstupní parametry funkce). Stručný algoritmus této funkce je uveden na obr. 5.16.



Obr. 5.16 Algoritmus dekódování posledních hodnot veličin v bloku dat

5.9.7 Měření a uložení logovaných dat do paměti

Provedení jednoho logu dat spočívá ve spuštění měření, komprese a uložení logovaných dat do paměti. K tomuto účelu slouží funkce *void measureAndStoreData(void)*. Stručný algoritmus této funkce je uveden na obr. 5.17. Zdrojový kód této funkce je uveden v příloze E.4.

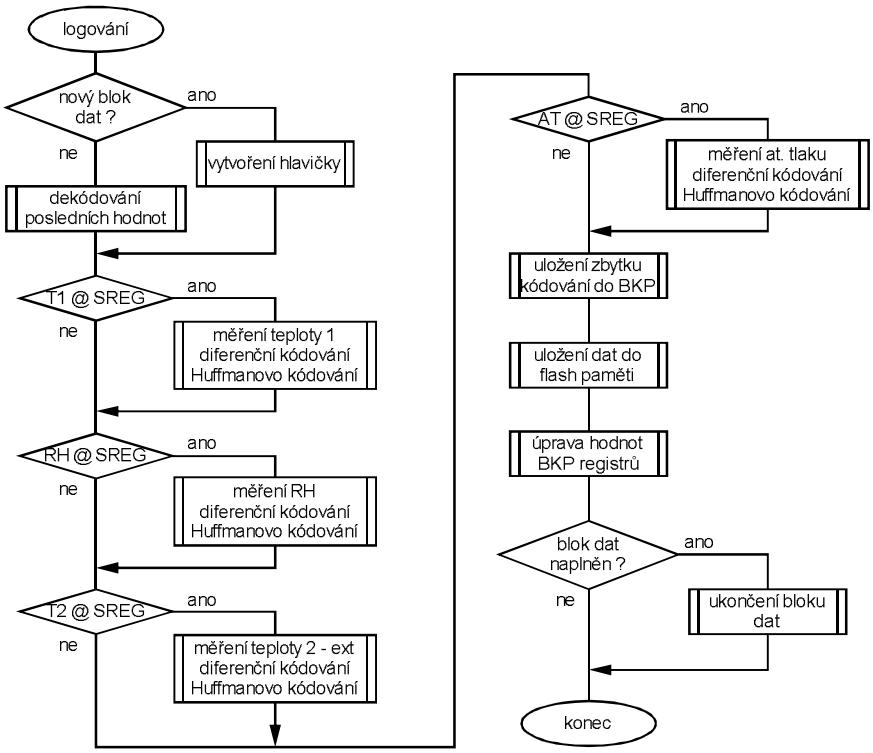
Funkce pro měření a logování dat nejdříve kontroluje, zda má být započat nový blok dat a případně vytváří hlavičku bloku. Pokud není vytvářen nový blok, proběhne dekódování posledních logovaných dat. Podle hodnoty status registru BKP_SREG jsou postupně měřeny dané veličiny pomocí příslušných funkcí driverů senzorů[17], diferenčně kódovány[18] a komprimovány Huffmanovým kódováním[19]. Zbylé bity dat po kompresi, které by neobsadily celou buňku flash paměti, a počet těchto bitů jsou uloženy pro další cyklus komprese do registru BKP_DATA_REMAINED. Komprimovaná data jsou uložena do flash paměti a hodnoty uživatelských registrů (počet logů v bloku BKP_BLOCKCOUNT a počet bytů v bloku BKP_BYTECOUNT) jsou upraveny. Na

[17] Měření teploty T1 a RH – kap.5.4.6, měření teploty T1 – kap. 5.5.4, měření atmosférického tlaku AT – kap. 5.3.

[18] Viz. kap. 3.5.1.

[19] Firmwarová funkce pro Huffmanovo kódování – kap 5.9.5.

závěr je kontrolováno, zda byl zapsán poslední log v bloku dat. Pokud tato skutečnost nastala, blok dat je ukončen[20].



Obr. 5.17 Algoritmus jednoho cyklu logování

5.10 Ostatní firmwarové funkce

Ostatní firmwarové funkce jsou definovány v souboru *./code/app/src/app_fce.c*. Tyto funkce jsou využívány jinými moduly, případně slouží k interakci s ovládací aplikací.

5.10.1 Převod integrovaného A/D převodníku

A/D převodník integrovaný v MCU slouží k měření napětí baterie a výstupního napětí tlakového čidla. Převod iniciuje funkce *void start_ADCconv(void)*. Získané hodnoty jsou uloženy pomocí DMA kanálu v globální proměnné a poté načítány ostatními funkcemi k dalšímu zpracování.

5.10.2 Odeslání měřených veličin mimo logovací cykly do PC

Pomocí ovládací aplikace je možné měření veličin mimo logovací cykly. K tomuto účelu slouží funkce uvedené v tabulce 29.

V každé z těchto funkcí je změřena příslušná veličina pomocí odpovídajícího driveru senzoru[17]. Takto změřená hodnota typu *float* je převedena na řetězec ASCII

[20] Ukončení bloku dat – kap. 5.9.4

znaků. Tento řetězec je odeslán do PC pomocí datového rámce Report ID 0x02 – odeslání hodnoty měřené veličiny mimo logovací cyklus.

Tab. 29 Funkce pro měření mimo logovací cykly

měřená veličina	zdroj přerušení
teplota T1	*void measureAndSendT1(void)*
teplota T2 – externí senzor	*void measureAndSendT2(void)*
relativní vlhkost RH	*void measureAndSendRH(void)*
atmosférický tlak AT	*void measureAndSendAT(void)*

5.10.3 Odeslání stavu dataloggeru do PC

Pro odeslání provozních informací, tedy stavu dataloggeru do PC, slouží funkce *void sendStatus(void)*. Zdrojový kód této funkce je uveden v příloze E.5. Provozní informace jsou odesílány ve formě datového rámce Report ID 0x02 – odeslání stavových informací dataloggeru do PC.

5.10.4 Pozastavení a opětovné spuštění logování

Pro pozastavení a spuštění logování je vytvořena funkce *void loggingPauseResume(void)*. Tato funkce je volána při přijetí odpovídajícího příkazu z ovládací aplikace.

5.11 Obsluha přerušení

Chod dataloggeru je řízen přerušeními. Konkrétně se jedná o přerušení vyvolané RTC jednotkou při dosažení hodnoty v alarm registru, přerušení systémového časovače, přerušení externích linek – tlačítek a přerušení vyvolaného příjmem dat z USB.

Funkce pro obsluhu jednotlivých přerušení jsou definovány v souboru *./code/app/src/ stm32f10x_it.c*.

5.11.1 Přerušení systémového časovače, generování zpoždění

Přerušení je spouštěno systémovým časovačem každou mikrosekundu. Obsluhu přerušení zajišťuje funkce *void SysTick_Handler(void)*. Toto přerušení používá funkce pro generování zpoždění.

Funkce pro generování zpoždění *void delay(uint32_t nTime, uint16_t unit)* nastaví počítadlo cyklů, které je dekrementováno funkcí *void TimingDelay_ Decrement(void)* při každé obsluze přerušení systémového časovače. Tyto funkce jsou definovány v souboru *./code/app/src/delay.c*.

Počet cyklů je stanoven vynásobením parametru *nTime* (velikost zpoždění) a parametru *unit* (násobitel zpoždění). Pro parametr *unit* jsou definována dvě makra, UNIT_MS (násobitel x1000), *nTime* odpovídá milisekundám a UNIT_US (násobitel x1), *nTime* odpovídá mikrosekundám.

Funkce pro generování zpoždění je ukončena ve chvíli kdy počítadlo cyklů dojde k hodnotě nula.

5.11.2 Přerušení příjmu dat z USB

Obsluhu přerušení zajišťuje funkce *void USB_LP_CAN1_RX0_IRQHandler(void)*. Obsluhou tohoto přerušení je volání funkce pro zpracování příchozích reportů.

5.11.3 Přerušení RTC jednotky - typ alarm

Přerušení typu alarm spouští RTC jednotka, dosáhne-li hodnota RTC čítače hodnotu uloženou v alarm registru. Obsluhu tohoto přerušení zajišťuje funkce *void RTC_IRQHandler(void)*.

Tato funkce vyhodnocuje hodnotu status registru BKP_SREG a hodnotu RTC čítače ve vztahu k údaji o ukončení měření. Na základě tohoto vyhodnocení se spouští nebo ukončuje logování, upravuje hodnota status registru BKP_SREG případně nastavuje hodnotu alarm registru pro další cyklus logování.

5.11.4 Přerušení externích linek – tlačítek

Obsluhu tohoto přerušení zajišťují tři funkce uvedené v tabulce 30. Tyto funkce pro obsluhu přerušení volají funkci pro obsluhu LCD displeje[21].

Tab. 30 Přerušení externích linek – tlačítek

zdroj přerušení	Obsluha přerušení
tlačítko OK – SW3	*void EXTI0_IRQHandler(void)*
tlačítko ESC – SW2	*void EXTI4_IRQHandler(void)*
tlačítko UP – SW4	*void EXTI9_5_IRQHandler(void)*
tlačítko DOWN – SW5	

5.11.5 Priority přerušení

Priority přerušení jsou nastaveny při inicializaci dataloggeru funkcí *void InterruptsConfig(void)* definovanou v souboru ./code/app/src/hw_config.c. Priority jednotlivých přerušení jsou shrnuty v tabulce 31.

Tab. 31 Priority přerušení

priorita	zdroj přerušení
	vnitřní chyby MCU
0	systémový časovač
1	RTC jednotka
2	tlačítko OK – SW3
2	tlačítko ESC – SW2
2	tlačítko UP – SW4
2	tlačítko DOWN – SW5
3	příjem dat USB

[21] Funkce pro obsluhu displeje – viz. kap. 5.13.

5.12 Řízení spotřeby

Spotřeba je jedním z klíčových parametrů dataloggeru. Kromě hardwarových opatření musel být přizpůsoben také firmware MCU. MCU disponuje několika režimy nízké spotřeby, pro datalogger je využit nejúčinnější z nich – StandBy.

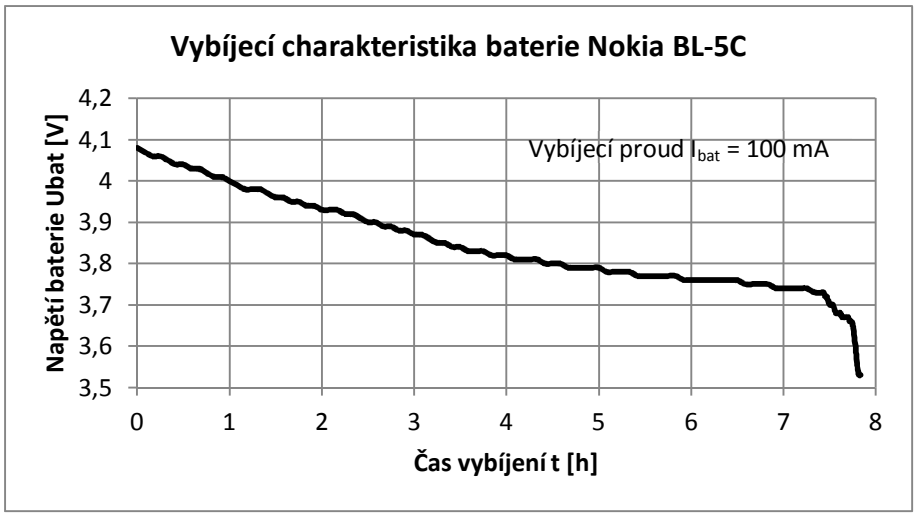
5.12.1 Driver Li-Ion nabíječky

Pro ovládání nabíječky Li-Ion baterie byl vytvořen driver, který umožňuje povolení a zakázání funkce nabíječky. K tomuto účelu byla vytvořena funkce *void Charger_SetStatus(uint8_t status)* definovaná souboru v *./code/drivers/NCP1835B/ /NCP1835B.c*. Tato funkce povoluje a zakazuje činnost nabíječky podle vstupního parametru *status.*

V driveru jsou definovány také funkce pro čtení stavu nabíjení z pinů nabíječky CFLG a FAULT (příloha A.5). K tomu slouží funkce *uint8_t Charger_GetStatus(void),* která vrací informaci o stavu CFLG pinu a funkce *uint8_t Charger_GetFault(void),* která vrací informaci o výskytu chyby při nabíjení.

5.12.2 Vybíjecí charakteristika baterie

Vybíjecí charakteristika baterie je závislost napětí baterie na čase při konstantním odběru proudu z baterie. Vybíjecí charakteristika použité baterie je zobrazena na obr. 5.18. Napětí baterie U_{bat} bylo měřeno A/D převodníkem MCU, vybíjecí proud byl nastaven na I_{bat} = 100 mA.



Obr. 5.18 Vybíjecí charakteristika Li-Ion baterie Nokia BL-5C

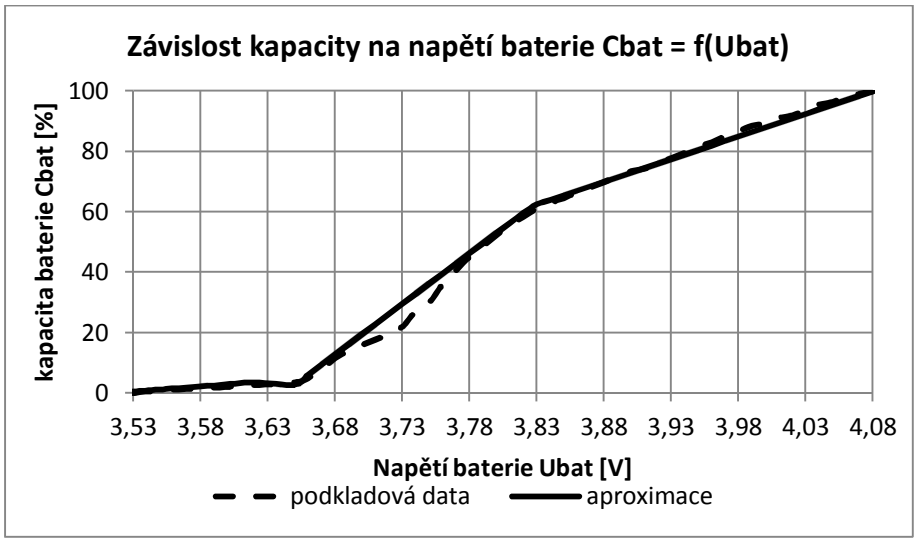
5.12.3 Pracovní režim baterie, odpojení dataloggeru

Kapacita vybrané baterie Nokia BL-5C je 1020mAh. Datalogger v aktivním režimu loguje data pouze, pokud je zbývající kapacita baterie přibližně 200 mAh, což odpovídá napětí baterie 3,55 V. Pokud je při logovacím cyklu řídící jednotkou naměřena tato hodnota datalogger ukončí probíhající logování, aby se další vybíjení zpomalilo a tím případně zabránilo kritickému vybití baterie.

5.12.4 Přepočet napětí na kapacitu baterie

Zbývající kapacita baterie je jeden z informačních údajů, které mohou být zobrazeny na LCD a také je ukládán do bloků změřených dat. Tento údaj je udáván v procentech a je odvozen od změřeného napětí baterie A/D převodníkem MCU pomocí vybíjecí charakteristiky baterie. Jak je vidět z obr.18, vybíjecí charakteristika baterie není lineární, proto musel být zaveden přepočet napětí baterie na její kapacitu udávanou v procentech.

Pro tento přepočet byla z vybíjecí charakteristiky vytvořena závislost kapacity na napětí baterie, která je vidět na obr. 5.19.



Obr. 5.19 Závislost kapacity na napětí baterie

Tato závislost byla po částech lineárně aproximována. Lineární rovnice jednotlivých částí jsou uvedeny dále.

- Pro interval napětí (3,53 ; 3,65) platí vztah (5.8)

$$C_{bat} = 150 \cdot U_{bat} - 512. \tag{5.8}$$

- Pro interval napětí (3,65 ; 3,82) platí vztah (5.9)

$$C_{bat} = 335{,}3 \cdot U_{bat} - 1219{,}8. \tag{5.9}$$

- Pro interval napětí (3,82 ; 4,08) platí vztah (5.10)

$$C_{bat} = 33{,}3 \cdot U_{bat} - 117{,}7. \tag{5.10}$$

Převod kapacity na napětí baterie lineární aproximací zajišťuje funkce *uint8_t getBatCapacity(void)*, definovaná v souboru *./code/drivers/NCP1835B/*. Funkce vrací kapacitu baterie v procentech ve formátu *unsigned char*. Tato funkce také ukončuje logování při dosažení kritické hodnoty baterie[22].

[22] Pracovní režim baterie – viz. kap. 5.12.3.

5.12.5 Režim nízké spotřeby StandBy

V době kdy datalogger není v režimu logování nebo zobrazování informací na LCD, byl jako režim nízké spotřeby MCU zvolen režim StandBy. V tomto režimu je aktivní pouze BKP doména, spotřeba MCU je nejnižší. Probuzení ze StandBy režimu je možný dvěma způsoby. Prvním způsobem je alarm RTC jednotky (spouští logování). Druhým způsobem je nástupná hrana napětí na pinu WKUP - GPIOA0 MCU, ke kterému je připojeno tlačítko OK. Při probuzení MCU tímto způsobem je spuštěn režim zobrazení informací na LCD.

Uvedení MCU do tohoto režimu zajišťuje funkce *void enterLowPower(void)* definovaná v souboru *./code/drivers/NCP1835B/*.

5.12.6 Odběr zařízení v různých režimech - měření

Proudový odběr zařízení byl změřen v režimech nízké spotřeby, režimu logování a režimu prohlížení informací na LCD. Změřené hodnoty proudového odběru jsou uvedeny v tabulce 32.

Tab. 32 Spotřeba dataloggeru

režim	Ibat [mA]
nízká spotřeba	6,8
logování	37,2
zobrazení dat na LCD[1]	86,4

[1]- s LCD podsvícením / bez LCD podsvícení

5.13 Zobrazení provozních informací na LCD

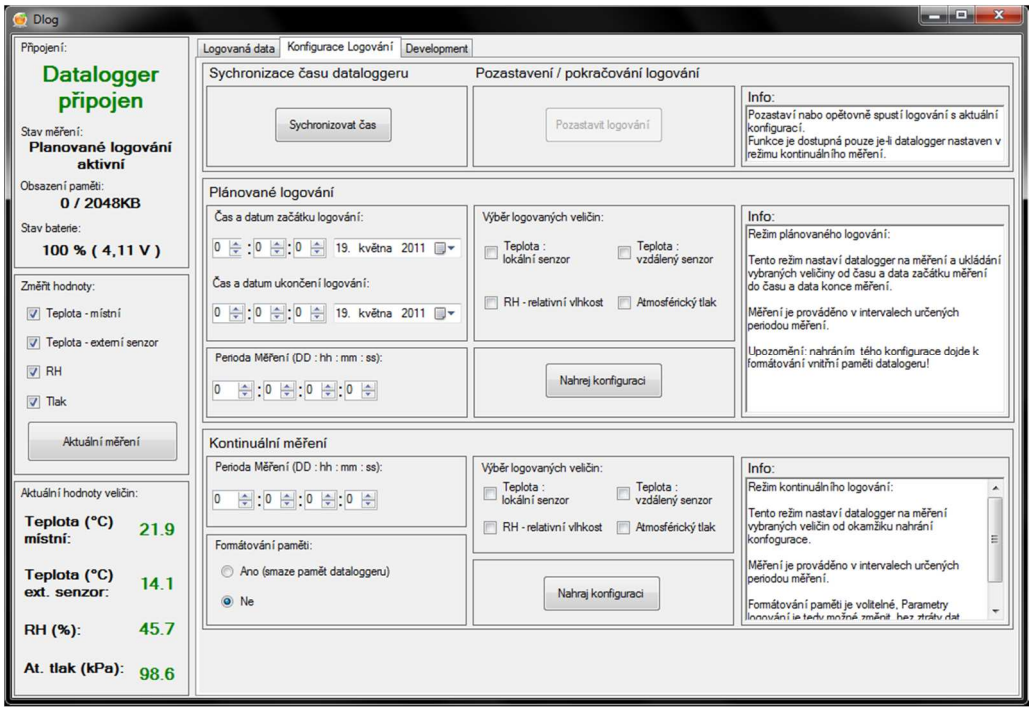
Pro zobrazení provozních informací na LCD byla vytvořena funkce *void manageLCD(uint8_t keyPressed)* definovaná v souboru *./code/app/src/lcd.c*. Vstupem této funkce je údaj, které tlačítko bylo stisknuto. Tato funkce dále vykresluje na displej provozní informace dataloggeru, stav logování, a aktuálně změřené veličiny.

6 SOFTWARE – OVLÁDACÍ APLIKACE PRO PC

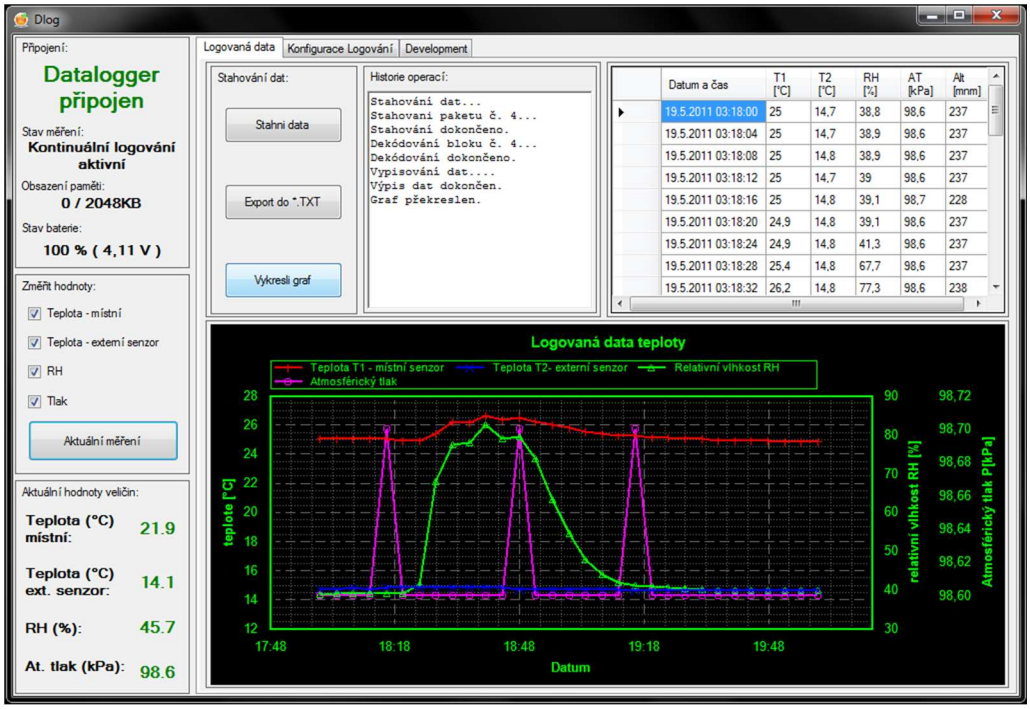
Ovládací aplikace pro PC slouží k nastavování parametrů logování dataloggeru, kontrole stavu zařízení, vyčítání dat pomocí USB sběrnice a jejich export do souboru pro další zpracování. Aplikace také umožňuje přímě měření atmosférických veličin mimo logovací cykly, pokud je datalogger připojen k PC.

GUI aplikace je typu Windows Forms, postaveno na .Net Frameworku 3.5. Aplikace je vytvořena v programovacím jazyce C#. Pro tvorbu aplikace bylo zvoleno vývojové prostředí SharpDevelop v3.2.1[18] (Instalátor je k dispozici na přiloženém CD ve složce *.//Vyvojove_nastroje/*).

GUI ovládací aplikace je rozděleno na 3 hlavní části. V levé části je panel stavu dataloggeru a měření mimo cykly logování. V pravé části je panel s přepínatelnými kartami *Logovaná data* s ovládacími prvky pro stažení a exportování dat a jejich vykreslení do grafu a *Konfigurace logování*. V kartě *Logovaná data* se nacházejí ovládací prvky pro stažení, zobrazení a exportování dat do souboru a jejich vykreslení do grafu. V kartě *Konfigurace logování* se nacházejí ovládací prvky pro uživatelský vstup konfiguračních dat a jejich nahrání do dataloggeru, ale také prvky pro synchronizaci reálného času a pozastavení nebo spuštění logování. Všechny tyto části aplikace jsou vidět na snímcích aplikace na obr. 6.1 a obr. 6.2.



Obr. 6.1 GUI ovládací aplikace – konfigurace logování



Obr. 6.2 GUI ovládací aplikace – logovaná data

Chod aplikace je zajištěn callback funkcemi vyvolanými stiskem jednotlivých tlačítek nebo časovači.

Projekt ovládací aplikace pro SharpDevelop je k dispozici na přiloženém CD ve složce *.//SOFTWARE/*. K tomuto umístění odkazují soubory uvedené v této kapitole.

Zkompilovaný release aplikace je k diposzici na přiloženém CD ve složce *.//Ovladaci_aplikace/*.

6.1 Implementace komunikačního protokolu

Pro obsluhu USB komunikace byla využita DLL knihovna *HIDLibrary.dll* (k dispozici na přiloženém CD ve složce *.//Vyvojove_nastroje/HIDLibrary-2.0.2.3-64-Bit-Fix*). Tato knihovna je vydána pod licencí LGPL, licence je také k dispozici na uvedeném umístění. Knihovna zprostředkovává základní funkce potřebné pro HID komunikaci, konkrétně enumeraci HID zařízení a odesílání a příjem reportů.

6.1.1 Enumerace a kontrola připojení dataloggeru

Enumeraci a kontrolu připojení dataloggeru zajišťuje callback funkce *void TimerTick(object sender, EventArgs e)* definovaná v souboru *./Dlog/MainForm.cs*. Tato funkce je volána časovačem každou sekundu. Enumerace a výběr obsluhovaného HID zařízení probíhá pomocí Vendor ID a Product ID. Tyto identifikátory jsou odesílány dataloggerem v HID deskriptoru. Tato funkce podle stavu připojení také ošetřuje povolování nebo zakazování tlačítka aplikace. Stav připojení je signalizován v poli „Připojení:“ v panelu stavu dataloggeru.

6.2 Panel stavu a měření mimo logovací cykly

6.2.1 Načtení stavu dataloggeru

Ke zjištění stavu dataloggeru slouží funkce *void timerStatusTick(object sender, EventArgs e)*, definovaná v souboru *./Dlog/MainForm.cs*. Tato funkce odesílá po USB Report ID 0x01 – příkaz pro načtení stavu dataloggeru a přijímá datový rámec Report ID 0x02 – odeslání stavových informací dataloggeru do PC odeslaný dataloggerem. Informace z přijatého reportu, tedy stav měření, kapacity paměti a nabití baterie, jsou zpracovány a zobrazeny ve stavovém panelu.

Zjištění stavu dataloggeru je prováděno každých pět sekund v případě, že je dataloggeru připojen a neprobíhá žádný jiný přenos dat.

6.2.2 Přímé měření veličin z aplikace mimo logovací cykly

Aplikace umožňuje přímé měření atmosférických veličin, bez záznamu do logu, pokud je dataloggeru připojen. K tomuto účelu složí callback funkce tlačítka „Aktuální měření“ *void button_StartMeasure_click(object sender, EventArgs e)*. Výběr měřených veličin se provádí zaškrtnutím políček v panelu „Změřit Hodnoty“.

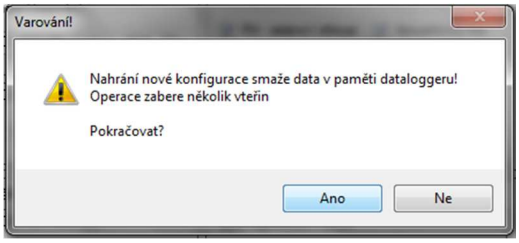
Funkce postupně odesílá příslušné příkazy datovými rámci Reporty ID 0x01 podle výběru a přijímá změřené hodnoty v datových rámcích Report ID 0x02 – odeslání hodnoty měřené veličiny mimo logovací cyklus. Z těchto datových rámců funkce vybírá ASCII řetězce změřených hodnot a zobrazuje je v panelu „Aktuální hodnoty veličin“.

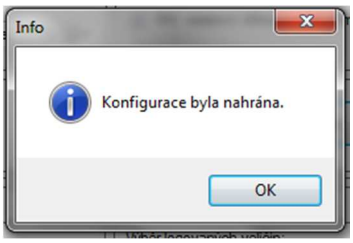
6.3 Karta konfigurace logování

6.3.1 Konfigurace plánovaného logování

Konfigurace plánovaného logování spočívá ve vytvoření a odeslání příslušného konfiguračního datového rámce Report ID 0x03 – konfigurace logování. Data odeslaná v tomto rámci jsou vytvořena z údajů zadaných v panelu „Plánované logování“.

Odeslání konfiguračního reportu zajišťuje callback funkce tlačítka „Nahraj konfiguraci“ v daném panelu *void sendNewConfig_click(object sender, EventArgs e)* definovaná v souboru *./Dlog/MainForm.cs.* Při stisku tohoto tlačítka je uživatel pomocí MessageBoxu dotázán, zda se má v operaci pokračovat, protože dojde k formátování paměti dataloggeru a ztrátě nestažených dat. Tento MessageBox je uveden na obr. 6.3a. Pokud je zvoleno pokračování, konfigurační report je odeslán a aplikace čeká, než datalogger provede všechna vnitřní nastavení a operace a odešle potvrzení - Report ID 0x02 – potvrzení provedení operace dataloggeru. Následuje zobrazení informačního MessageBoxu o úspěšném nahrání konfigurace. Tento MessageBox je uveden na obr. 6.3b.





a) Varování – formátování paměti b) Informace – úspěšné nahrání konfigurace

Obr. 6.3 použité MessageBoxy

6.3.2 Nová konfigurace a úpravy parametrů kontinuálního logování

Konfigurace kontinuálního logování neobsahuje informace o startu a ukončení logování. Oproti plánovanému logování také umožňuje pouhou změnu parametrů bez nutnosti formátování flash paměti dataloggeru. Pokud je plánované logování aktivní konfigurací dataloggeru a je provedena změna parametrů, dojde ke změně aktivní konfigurace z plánovaného na kontinuální logování. Dříve nahrané parametry startu a ukončení logování jsou dataloggerem dále ignorovány.

Odeslání konfiguračního reportu zajišťuje callback funkce tlačítka „Nahraj konfiguraci“ v daném panelu *void Button_setConfigContClick(object sender, EventArgs e)* definovaná v souboru *./Dlog/MainForm.cs.* Tato funkce pracuje na stejném principu jako funkce popsaná v předchozí kapitole, pouze s rozdílem, že odesílaná data vytváří z údajů uvedených v panelu „Kontinuální měření“.

6.3.3 Pozastavení a opětovné spuštění logování

Probíhající logování lze pozastavit, případně znovu spustit tlačítkem v panelu „Pozastavení / pokračování logování“, což zajišťuje callback funkce *void Button_startStopClick (object sender, EventArgs e)* definovaná v souboru *./Dlog/MainForm.cs.* Tato funkce je dostupná pouze v případě, kdy aktivní konfigurací dataloggeru je kontinuální logování.

Tato funkce odesílá dataloggeru datový rámec s odpovídajícím příkazem, Report ID 0x01 – příkazy.

6.3.4 Synchronizace času dataloggeru

Synchronizace času dataloggeru je zajištěna callback funkcí tlačítka „Synchronizovat čas“ *void timeSync_click(object sender, EventArgs e)*. Tato funkce načte aktuální čas operačního systému a odesílá jej dataloggeru v datovém rámci Report ID 0x03 – synchronizace času.

Tímto způsobem je provedena synchronizace času vždy při odeslání konfigurace do dataloggeru.

6.4 Karta logovaná data

6.4.1 Stažení a dekódování logovaných dat

Stažení a dekódování logovaných dat zajišťuje callback funkce tlačítka *void dataReadOut_click (object sender, EventArgs e)* definovaná v souboru *./Dlog/data_management.cs*.

Prvním krokem je načtení dat z flash paměti dataloggeru. K tomuto účelu slouží funkce *public void getData( )*. Tato funkce pomocí datového rámce Report ID 0x03 – načtení logovaných dat z flash paměti do PC vyžádá adresu posledního zápisu do paměti a načte tuto adresu z příchozího datového rámce Report ID 0x02 – přenos dat z MCU do PC. Poté je střídavě odesílán požadavek na odeslání paketu dat (zmíněný Report ID 0x03) a přijímána data (zmíněný Report ID 0x02), dokud je počet načtených bytů dat menší než adresa posledního zápisu do paměti.

V dalším kroku jsou postupně dekódovány všechny načtené bloky dat. To zajišťuje funkce *public void decodeHuffman( )*. Princip dekódování je popsán v kap.3.5.4, realizace dekódování je obdobná jako v případě dekódování poslední hodnoty v bloku dat[23] (implementace ve firmwaru). V tomto případě není průběžně vypočítávána suma všech hodnot dané veličiny v bloku dat, což vede k určení pouze poslední zakódované hodnoty, ale jsou z bloku dat diferenčně dekódovány všechny hodnoty veličin. Průběžně jsou také rekonstruovány časové údaje jednotlivých logů z údajů v hlavičkách bloků dat.

Hodnoty veličin a časového údaje získané z každého datového bloku jsou seřazovány do připravených polí typu *float* a *DateTime*.

Na závěr jsou všechna získaná data vypsána do tabulky v panelu v pravé horní části karty „Logovaná data“.

Hlášení o průběžném stavu a ukončení všech prováděných operací je zaznamenáván do panelu „Historie operací:“.

6.4.2 Export načtených dat do souboru

Načtená data mohou být vyexportována do textového dokumentu. Export dat vykonává callback funkce tlačítka „Export do *.TXT“ *void Button_exportToTxtClick(object sender, EventArgs e)* definovaná v souboru *./Dlog/data_management.cs*.

Po stisku tlačítka je spuštěn dialog pro uložení souboru, kde uživatel zvolí název souboru a umístění, kam má být soubor uložen. K samotnému vytvoření textového souboru slouží systémová komponenta StreamWriter. Do souboru jsou uloženy základní informace o stažených datech a datu stahování a následuje tabulka hodnot logovaných veličin s časovými údaji. Vytvořený textový dokument je uložen a otevřen k náhledu. Příklad exportovaného souboru je uveden v příloze G.

6.4.3 Zobrazení logovaných dat do grafu

Načtená data mohou být zobrazena v grafu přímo v aplikaci. Jako komponent pro vykreslování grafu načtených dat v ovládací aplikaci byl použit ZegGraphComponent

[23] Dekódování poslední uložené hodnoty v bloku – viz kap. 5.9.6.

z DLL knihovny ZedGraph.dll. Tato knihovna je vydána pod licencí LGPL a umístěna i s touto licencí a ostatními podklady na přiloženém CD v adresáři *.//Vyvojove_nastroje/ zedgraph_dll_v515/*.

Komponent grafu je inicializován funkcí *public void graph_init( )*. Tato funkce utváří grafickou podobu výsledného grafu, je zde nastaven systém tří Y-lonových os se základními rozsahy pro každou veličinu. Také jsou nastaveny mřížky, popisky grafu a jednotlivých os a použité barvy.

K vykreslení grafu slouží callback funkce tlačítka „Vykresli graf“ *void Button_drawGraphClick(object sender, EventArgs e)* definovaná v souboru *./Dlog/dlog_graph.cs*. Tato funkce zajišťuje vytvoření odpovídajících křivek v grafu, legendy a autoscale všech os, tak aby byly všechny průběhy viditelné.

Možnosti grafu jsou uvedeny v uživatelské příručce (příloha H).

ZÁVĚR

Na základě zadání diplomové práce byl proveden návrh koncepce dataloggeru sloužícího k měření a zaznamenávání atmosférických veličin - teploty, relativní vlhkosti a tlaku vzduchu. Primárním úkolem byl výběr jednotlivých komponent s ohledem na přesnost měření atmosférických veličin, spotřebu, možnost připojení k PC a také kompaktnost zařízení. Na základě této koncepce bylo vytvořeno obvodové zapojení zařízení.

Další částí realizace práce byl návrh desky plošných spojů, který byl proveden s ohledem na co nejmenší možné ovlivnění měření atmosférických veličin, velikost celého zařízení, ale také na pohodlné rozmístění ovládacích prvků. Zároveň s návrhem desky plošných spojů byl vytvořen návrh přístrojové krabičky, která byla zakázkově vyrobena podle vytvořené výkresové dokumentace.

Na základě těchto koncepcí a návrhů byla osazena a oživena první verze dataloggeru. V rámci oživení byly vytvořeny firmwarové drivery pro senzory, flash paměť, LCD displej a základní moduly RTC jednotky, komprese dat a komunikaci po USB. Zároveň s firmwarem byla vytvářena ovládací aplikace, pomocí které je možné volitelně nastavit parametry logování a načíst logovaná data do PC, zobrazit je ve formě tabulky a grafu. Při vytváření ovládací aplikace a firmwaru bylo objeveno několik nedostatků a chyb v návrhu, což vedlo k výrobě opravné verze DPS.

Vytvořené zařízení je schopné vykonávat všechny funkce dané zadáním práce i s některými rozšířeními. Základními rozšířeními jsou přidaný senzor tlaku a externí senzor teploty, více možností nastavení logování, ale také měření atmosférických veličin i mimo intervaly logování a zobrazení jejich hodnot na displeji nebo jejich přímý přenos do PC a zobrazení v ovládací aplikaci.

Hlavním nedostatkem práce je spotřeba zařízení. V režimech logování a zobrazení dat naměřené hodnoty odběru proudu odpovídají předpokladu. V režimu nízké spotřeby, který je pro tuto aplikaci klíčový, je změřený proudový odběr řádově větší, než byl předpokládán. Po prověření a odladění většiny možných příčin, jak hardwarových tak firmwarových, nebyl nalezen zdroj tohoto problému. Nejpravděpodobnější příčinou vysokého odběru dataloggeru je špatně detekovatelná chyba v obvodovém zapojení.

Vytvořený datalogger má potenciál pro další rozvoj, což se týká hlavně ovládání bez nutnosti připojení dataloggeru k PC, signalizaci případných poruch senzorů a ostatních komponentů. Také je možné vylepšení systému komprese ukládaných dat a dosažení vyššího kompresního poměru.

ZDROJE LITERATURY

[1] PARR, E. A.: *Industrial Control Handbook :3rd edition* [online]. Industrial Press Inc., 1999 [cit. 2010-04-12]. 896 s. Dostupný na: <http://books.google.cz/books?id=zLwtngK3T1UC>

[2] ROVETI, D. K. *Choosing a Humidity Sensor: A Review of Three Technologies*. Sensor Magazine [online]. Jul-2001 [cit. 2010-04-12]. Dostupné z WWW: <http://www.sensorsmag.com/sensors/humidity-moisture/choosing-a-humidity-sensor-a-review-three-technologies-840>

[3] SENSIRION, GENEVA : *STM32F103xC STM32F103xD STM32F103xE datasheet*, [online]. last revision 12-Dec-2008, [cit. 2010-04-15]. 11 s. Dostupný na: <http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf>

[4] FRESCALE SEMICONDUCTOR, TEMPE: *High Temperature Accuracy Integrated Silicon Pressure Sensor for Measuring Absolute Pressure, On-Chip Signal Conditioned, Temperature Compensated and Calibrated*, [online]. last revision Oct-2009, [cit. 2010-04-15]. 12s. Dostupný na: <http://www.freescale.com/files/sensors/doc/data_sheet/MP3H6115A.pdf>

[5] MAXIM INTEGRATED PRODUCTS, SUNNYVALE: *DS18S20 High-Precision 1-Wire Digital Thermometer* [online]. last revision 22-Apr-2008, [cit. 2010-04-15]. 23s. Dostupný na: <http://datasheets.maxim-ic.com/en/ds/DS18S20.pdf>

[6] STMicroelectronics NV, STAEFA: *STM32F103x8 STM32F103xB*, [online]. last revision 22-Sep-2009, [cit. 2010-4-18]. 96 s. Dostupný na: <http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00161566.pdf>

[7] ATMEL CORPORATION, SAN JOSE: *16-Megabit 2.7-volt Minimum SPI Serial Flash Memory*, [online]. last revision C Jul-2009, [cit. 2010-4-18]. 60 s. Dostupný na: <http://www.atmel.com/dyn/resources/prod_documents/doc3687.pdf>

[8] ELECTRONIC ASSEMBLY, GILCHING: *DOGM GRAPHIC SERIES 132x32 DOTS*, [online]. last revision Jun-2009, [cit. 2010-4-18]. 8 s. Dostupný na: <http://www.lcd-module.de/eng/pdf/grafik/dogm132-5e.pdf>

[9] ON SEMICONDUCTOR, DENVER: *NCP1835B Integrated Li−Ion Charger*, [online]. last revision 4 Sep-2006, [cit. 2010-4-21]. 14 s. Dostupný na: <http://www.onsemi.com/pub_link/Collateral/NCP1835B.PDF>

[10] ON SEMICONDUCTOR, DENVER: *NCP662, NCV662, NCP663, NCV663 100 mA CMOS Low Iq Low-Dropout Voltage Regulator*, [online]. last revision 2 Mar-2009, [cit. 2010-4-21]. 10 s. Dostupný na: <http://www.onsemi.com/pub/Collateral/NCP662-D.PDF>

[11] ON SEMICONDUCTOR, DENVER: *NUF2030XV6, NUF2042XV6 USB Upstream Terminator with ESD Protection*, [online]. last revision 2 Jul-2005, [cit. 2010-4-21]. 6 s. Dostupný na: <http://www.onsemi.com/pub/Collateral/NUF2030XV6-D.PDF>

[12] *Universal Seriál Bus specification* [online]. revision 2.0, last revision 27-Apr-2000, [cit. 2010-4-25]. 650 s. Dostupný na: <http://www.usb.org/developers/docs>

[13] AXELSON, J. USB COMPLETE : The Developer's Guide. 4th edition. Madison : Lakeview Research LLC, 2009. 529 s.

[14] STMicroelectronics NV, STAEFA : *STM32F103xxx hardware development: getting started*, [online]. last revision 19-Oct-2010, [cit. 2010-10-12]. 28 s. Dostupný na: <http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00164185.pdf>

[15] *STM32F10x Standard Periferals Library* [knihovna firmware]. Ver. 3.4.0, [cit. 2010-20-12]. Dostupná na: <http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/stm32f10x_stdperiph_lib.zip>

[16] *µVision4* [počítačový program]. Ver. 4.03, [cit. 2010-20-12]. Vývojové prostředí firmwaru MCU.

[17] *EAGLE* [počítačový program]. Ver. 5.7.0, [cit. 2010-20-12]. Program pro návrh DPS.

[18] *SharpDevelop* [počítačový program]. Ver. 3.2.1, [cit. 2011-10-5]. Programovací prostředí pro C#.

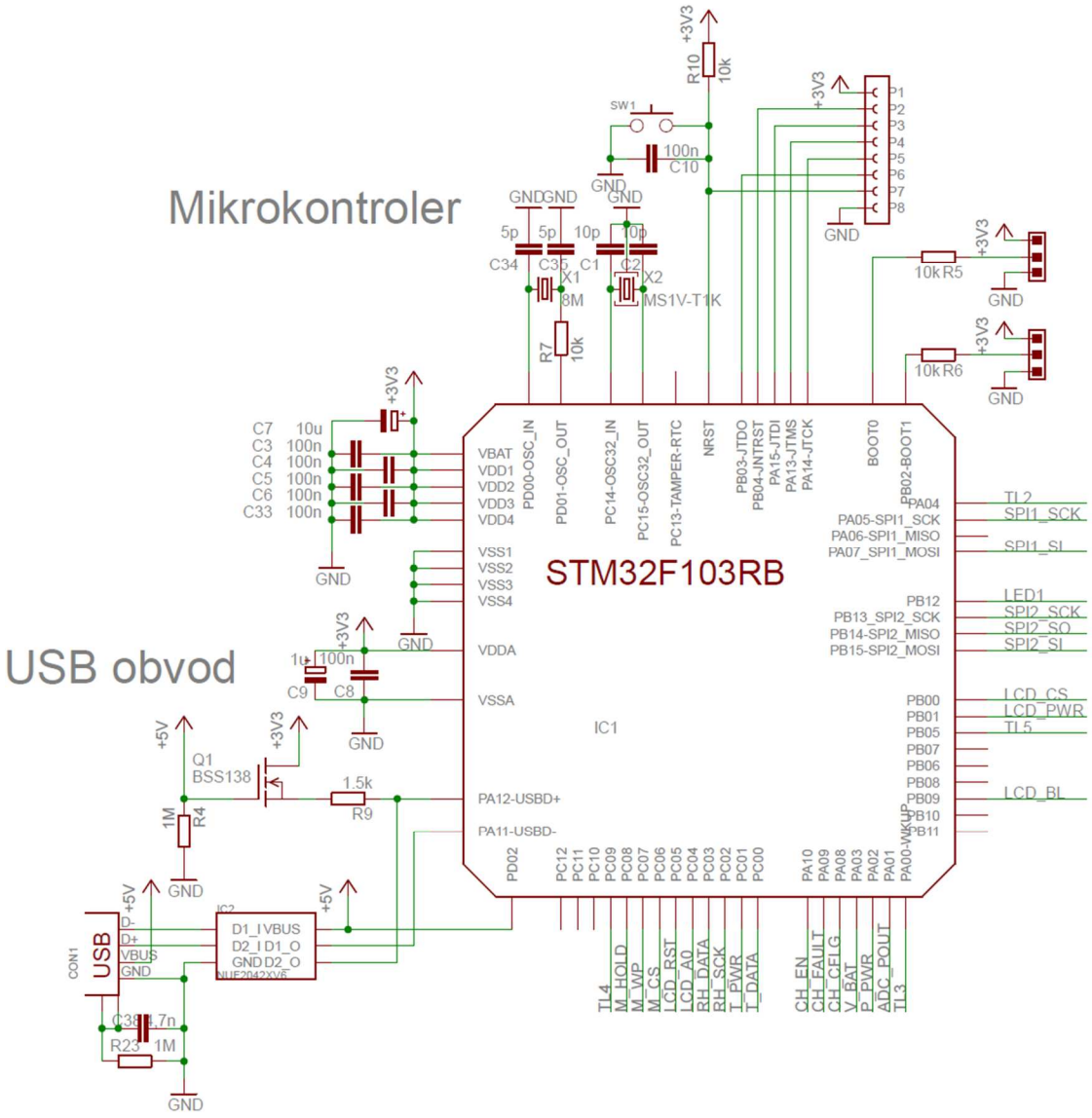
SEZNAM SYMBOLŮ A ZKRATEK

A/D	analogové / digitální
ARM	Advanced RISC Machine, 32bitová architektura jádra mikrokontrolérů s redukovanou instrukční sadou
BAT	baterie
BCD	Binary Coded Decimal, binárně vyjádřené dekadické číslo
CBL	kontrolní velikost bloku
CC	Constant Current, konstantní proud
CCCV	Constant Current Constant Voltage, konstantní proud konstantní napětí
CRC	Cyclic Redundancy Check, cyklická redundantní kontrola
CV	Constant Voltage, konstantní napětí
D_x	Data, digitální vyjádření hodnoty
EMI	Electromagnetic Interferences, elektromagnetické rušení
EOC	End Of Charge, stav ukončení nabíjení
ESD	Electro Static Discharge, elektrostatický výboj
GPIO	General Purpose Input Output, obecné vstupně / výstupní linky
HID	Human Interface Device, třída USB, rozhraní pro ovládání PC
ID	identifikační číslo
LCD	Liguid Crystal Display, displej z tekutých krystalů
Li-Ion	Lithium Ion, lithium ionový
LSB	Least Significant Bit, bit s nejnižší pozicí v binárním čísle
MC	kapacita paměti
MCU	Microcontroller unit, mikrokontrolér
MSB	Most Significant Bit, bit s nejvyšší pozicí v binárním čísle
P	Pressure, tlak
RH	Relative Humidity, relativní vlhkost
RTC	Real Time Clock, hodiny reálného času
RTD	Resistance Temperature Detectors, rezistivní termometr
RTD	Resistance Temperature Detectors, rezistivní termometr
SPI	Serial Peripheral Interface Bus, sériové periferní rozhraní
SRAM	Static Random Access Memory, statická paměť
t	Temperature, teplota
U_{DD}	Napájecí napětí
USB	Universal Seriál Bus, univerzální sériová linka

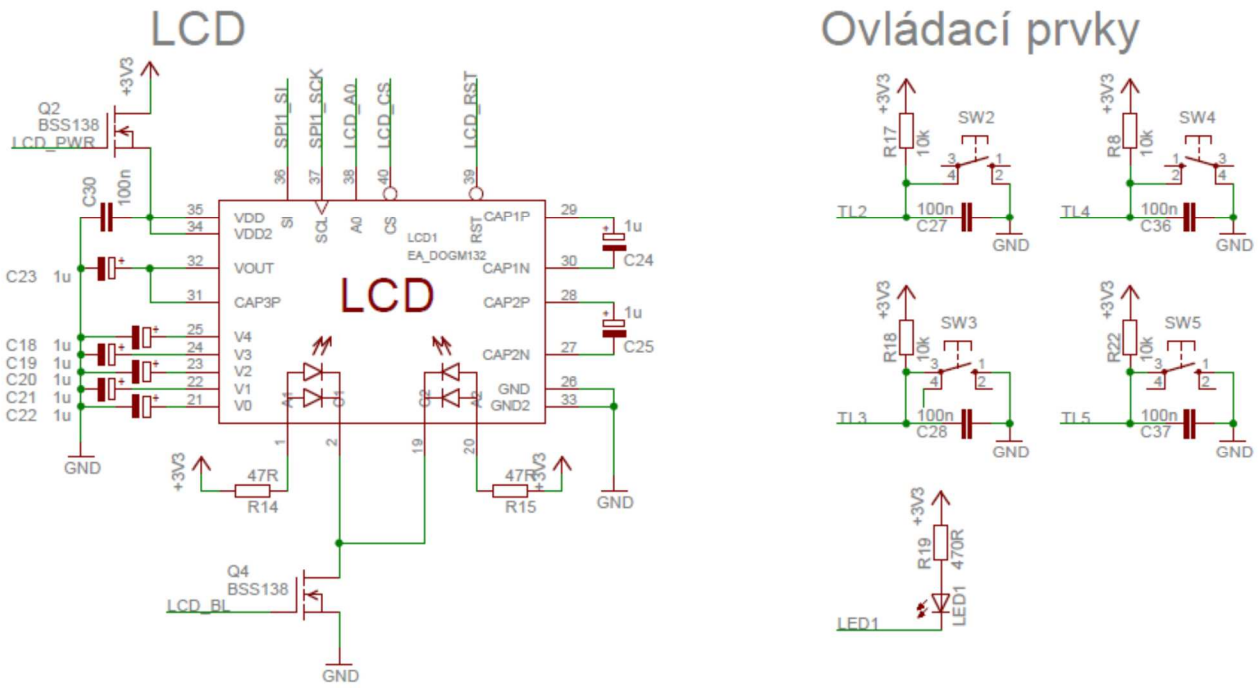
SEZNAM PŘÍLOH

A NÁVRH ZAŘÍZENÍ

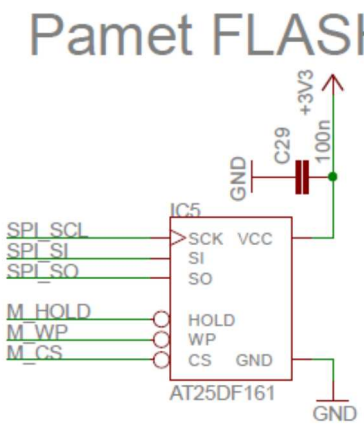
A.1 Schéma zapojení – MCU, obvod USB



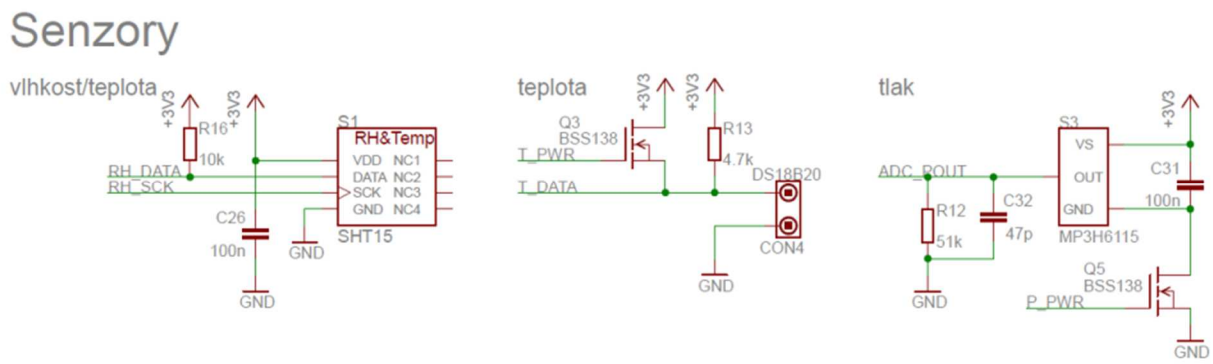
A.2 Schéma zapojení – LCD displej, ovládací prvky



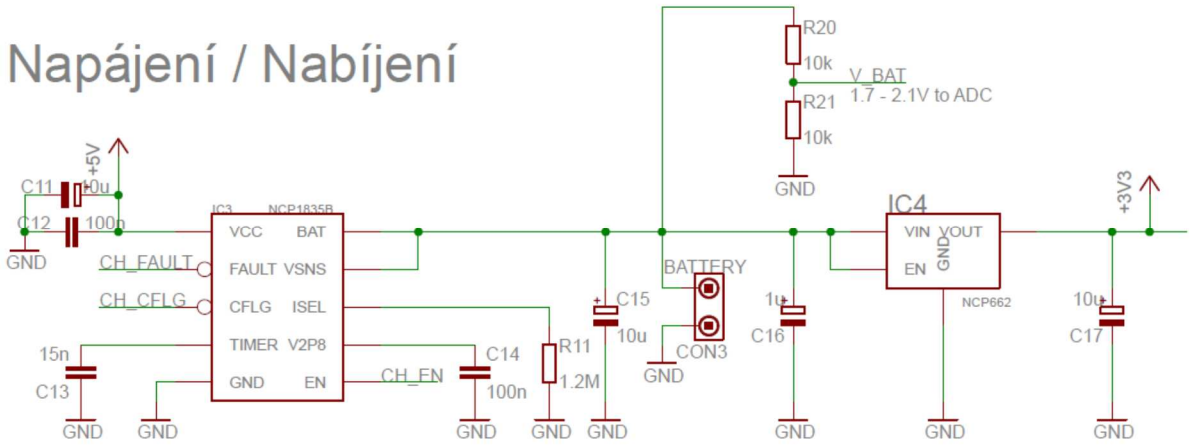
A.3 Schéma zapojení – Paměť flash



A.4 Schéma zapojení – senzory



A.5 Schéma zapojení – napájecí a nabíjecí obvod



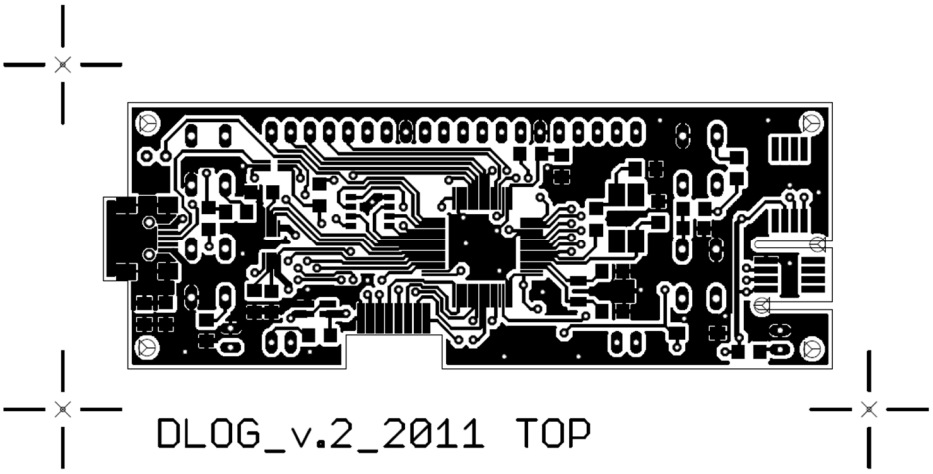
B SOUPISKA SOUČÁSTEK

Označení	Hodnota	Pouzdro	Popis	ks
Integrované obvody				
IC1	STM23F103RBT6	LQFP64	Mikrokontrolér	1
IC2	NUF2042XV6	SOT563	Ochrana USB linek	1
IC3	NCP1835	DFN3x3	Li-Ion nabíječka	1
IC4	NCP662	SC82AB	Stabilizátor napětí	1
IC5	AT25DF161	SOIC8	Paměť FLASH	1
Rezistory				
R14,R15	47R	1206	Rezistor	2
R19	470R	1206	Rezistor	1
R9	1k5	1206	Rezistor	1
R13	4k7	1206	Rezistor	1
R1,R2,R3,R5,R6,R7,R8,R10,R16, R17,R18, R20,R21,R22	10k	1206	Rezistor	14
R12	51k	1206	Rezistor	1
R4,R23	1M	1206	Rezistor	2
R11	1M2	1206	Rezistor	1
Kondenzátory				
C34,C35	5p6	1206	Keramický kondenzátor	2
C1,C2,	10p	1206	Keramický kondenzátor	2
C38	4n7	1206	Keramický kondenzátor	1
C32	47p	1206	Keramický kondenzátor	1
C3,C4,C5,C6,C8,C10,C12,C14,C26, C27,C28,C29,C30, C31,C33,C36,C37	100n	1206	Keramický kondenzátor	17
C7,C11,C15,C17	10u	A 3216-18	Keramický kondenzátor	4
C9,C16,C18,C19,C20,C21,C22, C23,C24,C25	1u	A 3216-18	Keramický kondenzátor	10
C13	15n	1206	Keramický kondenzátor	1
Senzory				
S1	SHT15		Modul senzoru vlhkosti a teploty	1
S2	DS18B20+		1-wire senzor teploty	1
S3	MP3-H6115A		tlakový senzor	1
Konektory				
CON1	Lumberg 2410 07		USB konektor typ A	1
CON2	molex 53047-0810		JTAG spojovaci M	2
	molex 51021-0800		JTAG spojovaci F	2
	molex 50058-8000		kontakty	16
CON5	MLW20G		JTAG konektor	1
Spínače - tlačítka				
SW1	DTSM32N	SMD	Tlačítko	1
SW2,SW3,SW4,SW5	schurter 1301.9308		Tlačítko	4
cup-SW2,SW3,SW4,SW5	B32-1010		knoflík tlačítek	4
Ostatní				
LCD1	EA DOGM132		LCD Displej	1
LCD1-bl	EALED55x31		LCD podsvícení	1
Q1,Q2,Q3,Q4,Q5,Q5	BSS138	SOT23	Tranzistor	5
LED1	L-934LGD		led ø3mm	3
X1	C7S-8.000-12-3030-X	7x5SMD	Krystal 8MHz	1
X2	MS1V-T1K 32.768KHZ		Krystal 32,768kHz	1

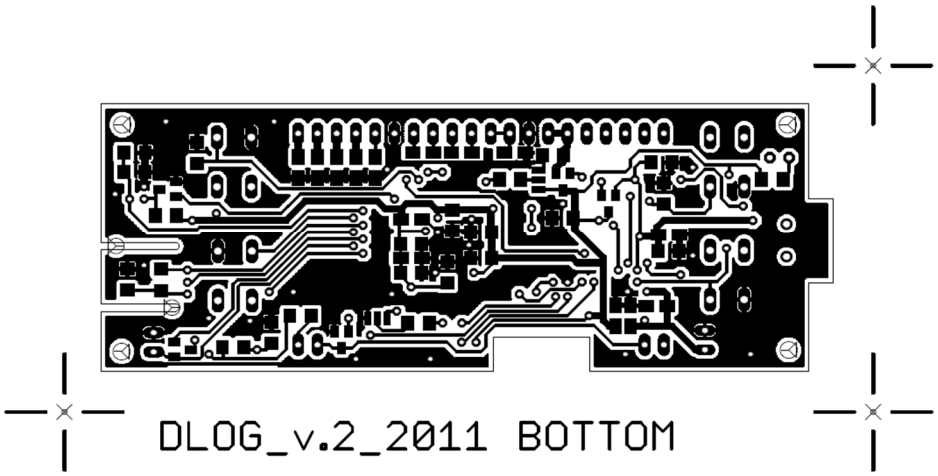
C DESKA PLOŠNÉHO SPOJE

rozměry desky: 94 x 35 mm
měřítko: 1:1

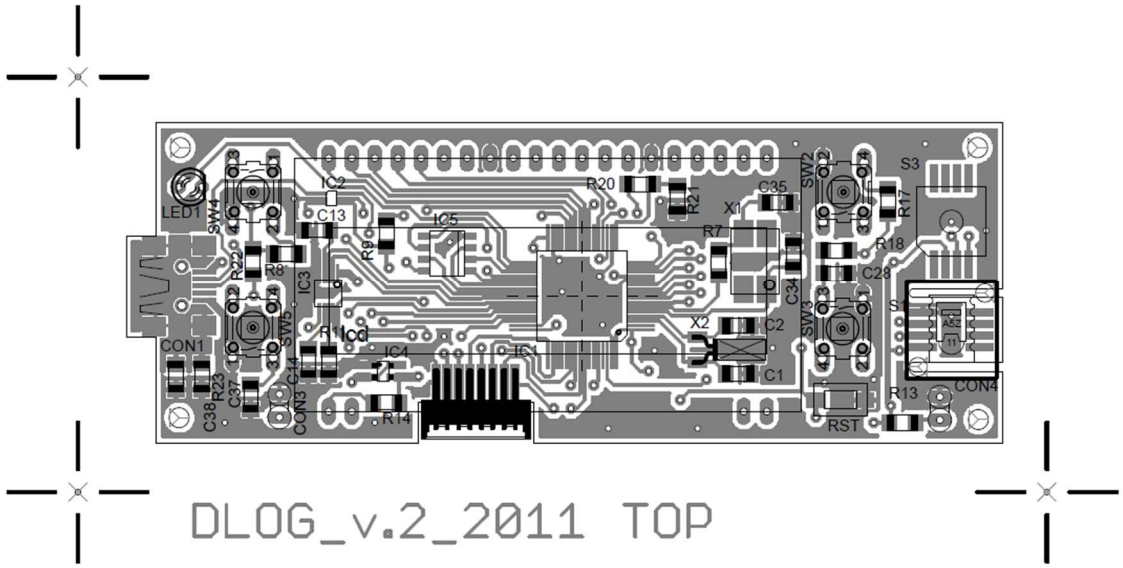
C.1 DPS – strana TOP



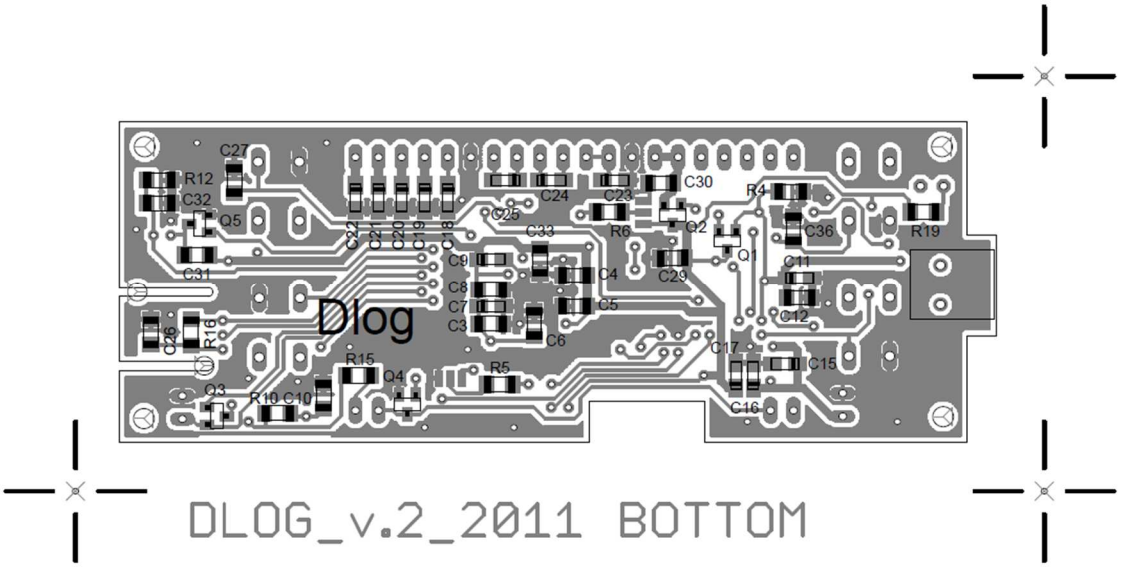
C.2 DPS – strana BOTTOM



C.3 DPS – Osazovací schéma – strana TOP

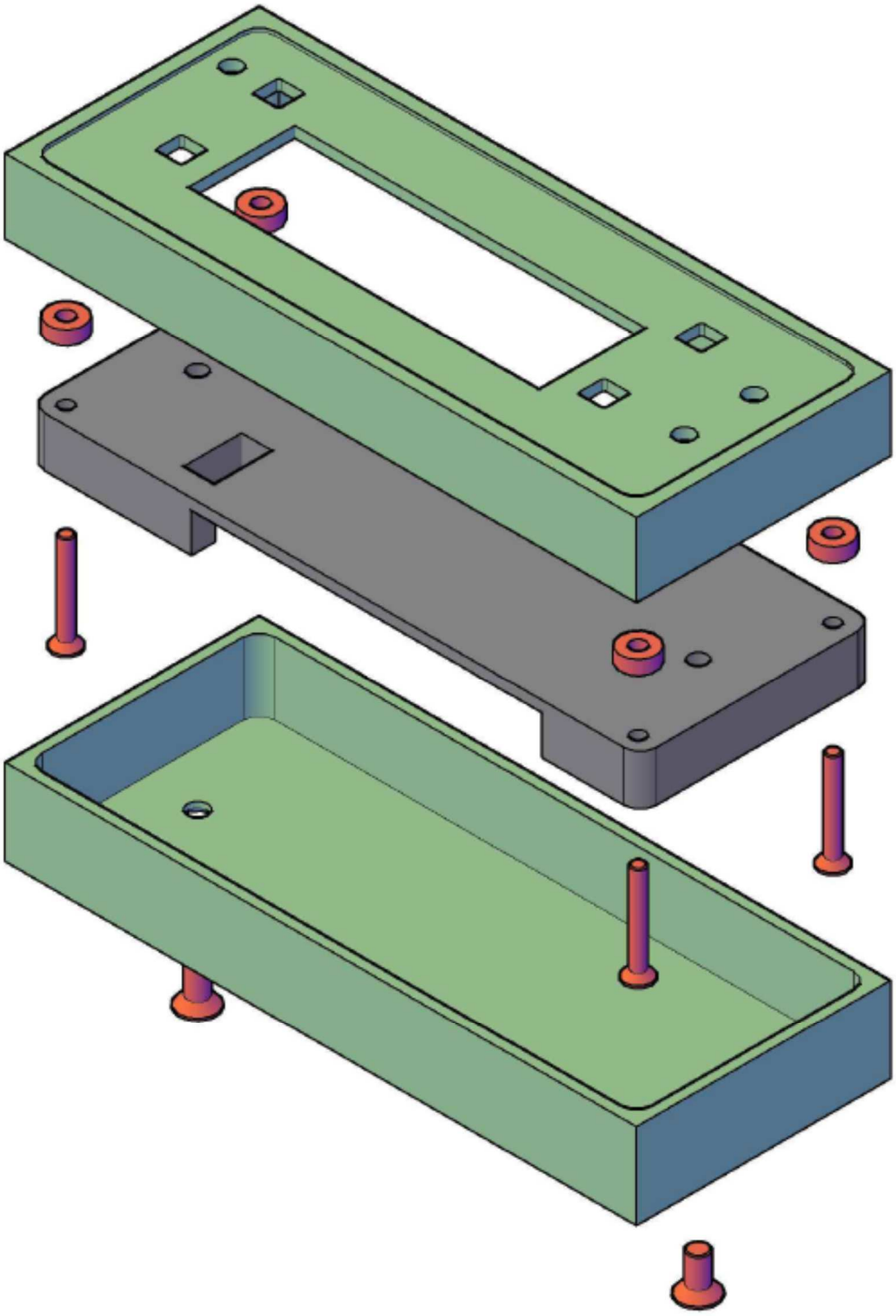


C.4 DPS – Osazovací schéma – strana BOTTOM

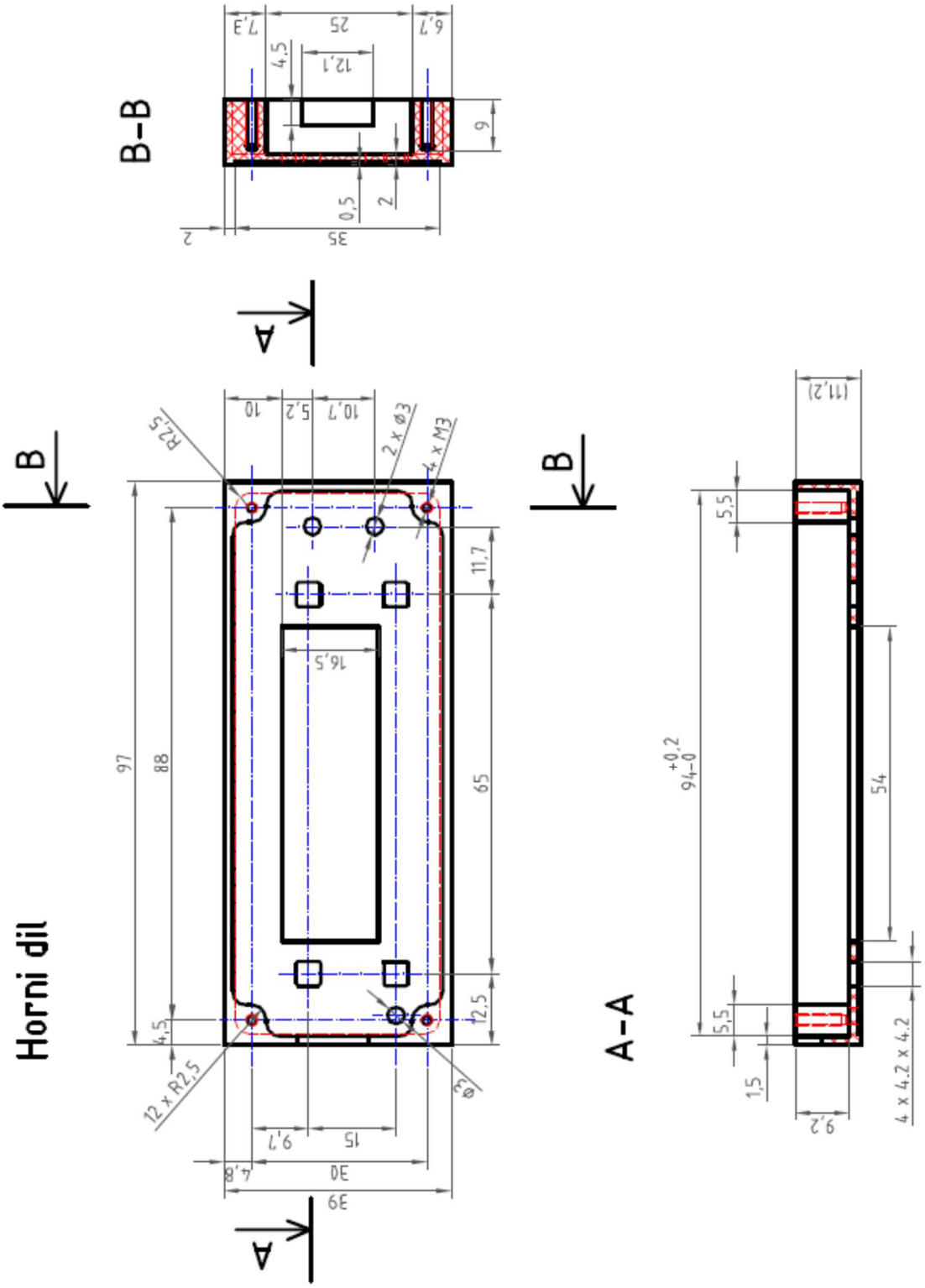


D MECHANICKÁ KONSTRUKCE

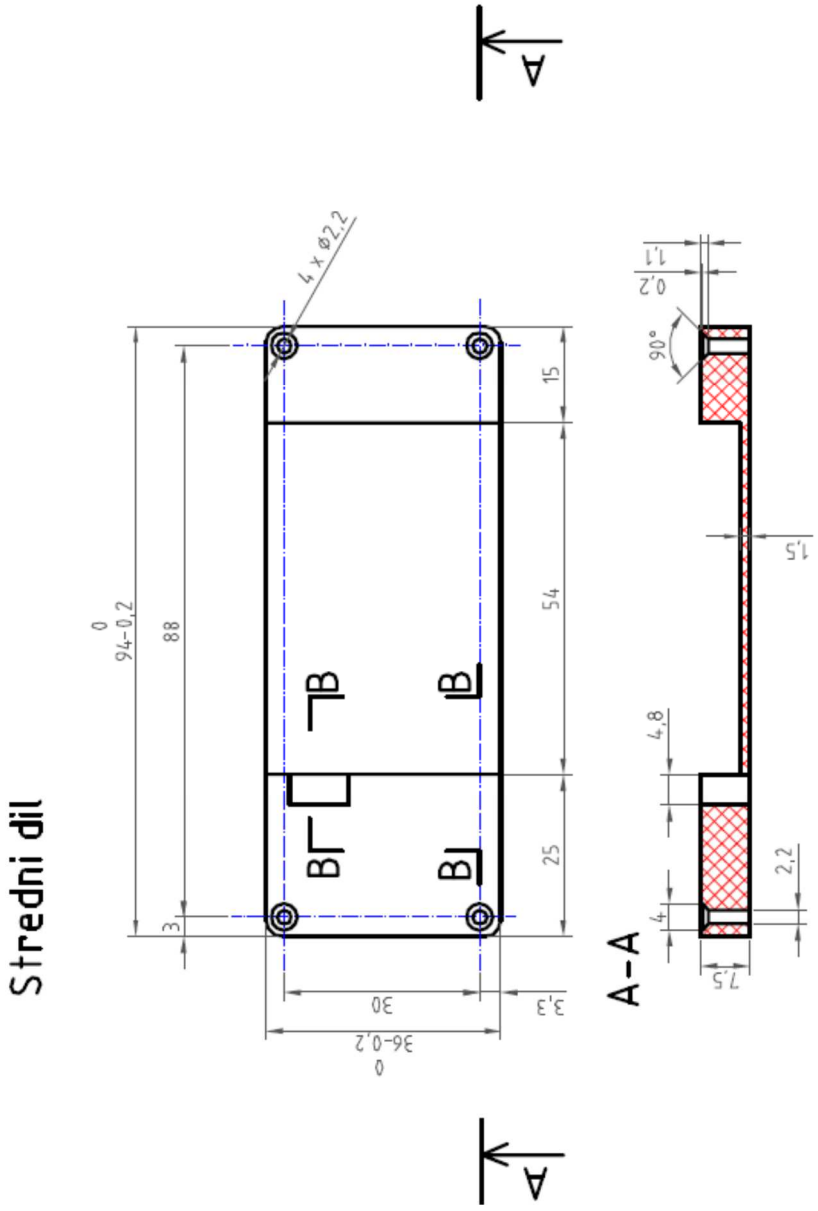
D.1 3D model přístrojové krabičky



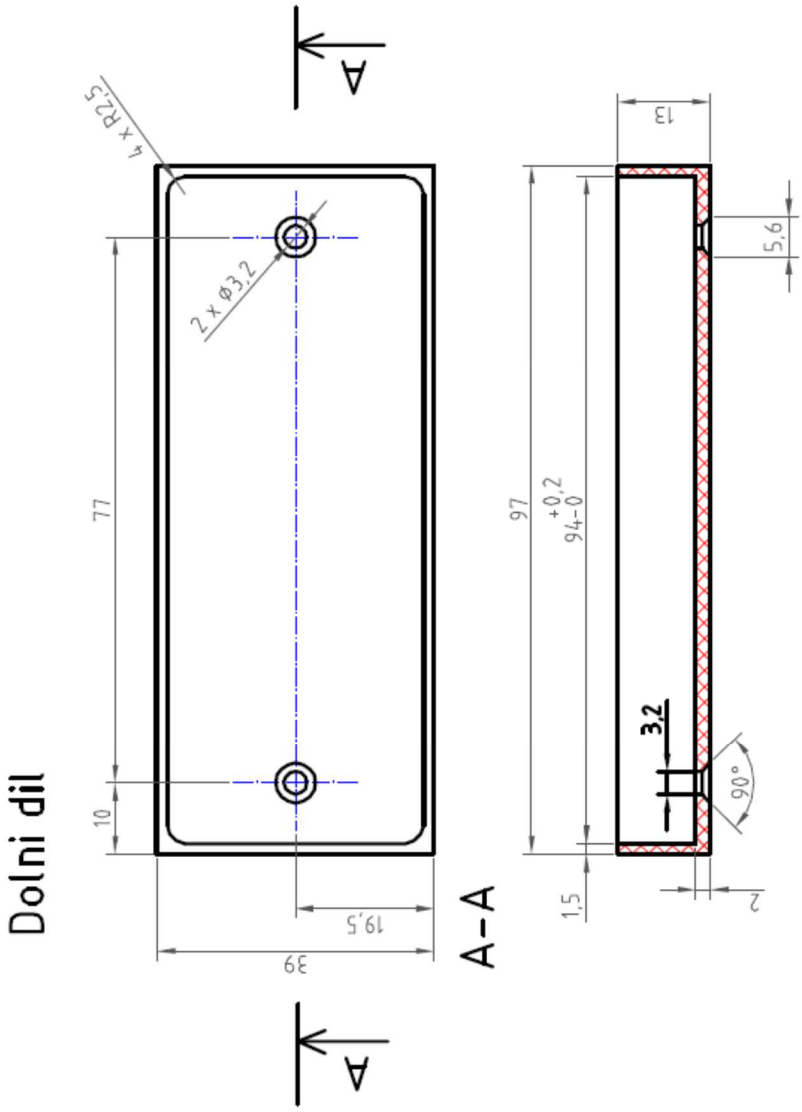
D.2 Výkresová dokumentace – Horní díl



D.3 Výkresová dokumentace – Střední díl



D.4 Výkresová dokumentace – Spodní díl



E UKÁZKY ZDROJOVÉHO KÓDU FW

E.1 Driver teplotního čidla DS18B20 – měření teploty

```
/* ------------------------------------------------------------------------
 * Jmeno funkce:              Tsens_readTemp( )
 * Popis:                     mereni a cteni teploty senzoru
 * Vstupy:                    -
 * Vystupy:           -
 * Return hodnota:    float temp - zmerena teplota
-*------------------------------------------------------------------------*/
float Read_Temperature(void)
{
  uint8_t get[9];
  uint8_t temp_lsb,temp_msb;
  uint16_t temp_C = 0;
  uint8_t sign = 0x00;
  uint8_t k;
  float temp = 0.0;
  uint8_t err = 0;

  err = Tsens_reset();              // inicializace komunikace
  if(err)
       {
              return(PRESENCE_ERR); // vraci chybu pokud neni senzor detekovan
       }
  Tsens_writeByte(SKIP_ROM);        // preskoceni enumerace
  Tsens_writeByte(CONVERT_T);       // spusteni konvezre
  T_PWR_ON;                         // parasitni napajeni pri konverzi
  delay(750,UNIT_MS);               // zpozdeni
  T_PWR_OFF;

  if(Tsens_reset())                 // inicializace komunikace
       {
              return(PRESENCE_ERR); // senzor nepripojen
       }
  Tsens_writeByte(SKIP_ROM);        // preskoceni enumerace
  Tsens_writeByte(READ_SP);         // prikaz k precteni zmerene hodnoty

  for (k=0;k<9;k++)                 // cteni zmerene hodnoty
  {
       get[k] = Tsens_readByte();
  }

  temp_lsb = get[0];                // nactena teplota LSB
  temp_msb = get[1];                // nactena teplota MSB

  temp_C = temp_msb;                // slozeni nactene hodnoty
  temp_C = temp_C << 8;
  temp_C |= temp_lsb;

  if(temp_C > 0x1000)               // uprava nactene hodnoty podle znamenka
  {
       sign = 0x01;
       temp_C = !temp_C + 1;
  }

  temp = (float)temp_C/16;          // prepocet na float
  if(sign)                          // uprava znamenka
  {
       temp = -temp;
  }
  return(temp);
}
```

E.2 Driver flash paměti – zápis dat do paměti

```
/* ----------------------------------------------------------------------------
 * Jmeno funkce:      Mem_writeBuffer( )
 * Popis:             funkce pro zapis bufferu dat libovolne delky do pameti
 * Vstupy:     uint32_t WriteAddr           - adresa pameti od ktere ma byt
                                              proveden zapis
               uint8_t* pBuffer             - ukazatel na buffer dat k zapisu
               uint16_t NumByteToWrite      - pocet bytu, ktere maji byt zapsany
 * Vystupy:
 * Return hodnota:
-*----------------------------------------------------------------------------*/
void Mem_writeBuffer(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite)
{
       uint8_t NumOfPage = 0, NumOfSingle = 0, Addr = 0, count = 0, temp = 0;
       Addr = WriteAddr % MEM_PAGE_SIZE;
       count = MEM_PAGE_SIZE - Addr;
       NumOfPage =  NumByteToWrite / MEM_PAGE_SIZE;
       NumOfSingle = NumByteToWrite % MEM_PAGE_SIZE;
  //--- adresa je zarovnana na zacatek nektere page pameti ----------
       if (Addr == 0)
       {
       //--- pocet bytu < page size ---
              if (NumOfPage == 0)
              {
                     Mem_writePage(pBuffer, WriteAddr, NumByteToWrite);
              }
       //--- pocet bytu > page size ---
              else
              {
                     while (NumOfPage--)          // zapis regulernich pages
                     {
                            Mem_writePage(pBuffer, WriteAddr, MEM_PAGE_SIZE);
                            WriteAddr +=  MEM_PAGE_SIZE;
                            pBuffer += MEM_PAGE_SIZE;
                     }
                     // zapis zbytkove page
                     Mem_writePage(pBuffer, WriteAddr, NumOfSingle);
              }
       }
  //--- adresa neni zarovnana na zacatek nektere page pameti --------
       else
       {
       //--- pocet bytu < page size ---
              if (NumOfPage == 0)
              {
              //--- pocet bytu k zapisu od adresy zapisu > page size
                     if (NumOfSingle > count)
                     {
                            temp = NumOfSingle - count;
                            Mem_writePage(pBuffer, WriteAddr, count);
                            WriteAddr +=  count;
                            pBuffer += count;
                            Mem_writePage(pBuffer, WriteAddr, temp);
                     }
              //--- pocet bytu k zapisu od adresy zapisu > page size
                     else
                     {
                            Mem_writePage(pBuffer, WriteAddr, NumByteToWrite);
                     }
              }
              //--- pocet bytu > page size ---
              else
              {
                     NumByteToWrite -= count;
                     NumOfPage =  NumByteToWrite / MEM_PAGE_SIZE;
                     NumOfSingle = NumByteToWrite % MEM_PAGE_SIZE;
                     // zapis prvni page
                     Mem_writePage(pBuffer, WriteAddr, count);
                     WriteAddr +=  count;
                     pBuffer += count;
```

```
                        while (NumOfPage--)          // zapis regulernich pages
                        {
                                Mem_writePage(pBuffer, WriteAddr, MEM_PAGE_SIZE);
                                WriteAddr +=  MEM_PAGE_SIZE;
                                pBuffer += MEM_PAGE_SIZE;
                        }

                        if (NumOfSingle != 0)        // zapis zbytkove page
                        {
                                Mem_writePage(pBuffer, WriteAddr, NumOfSingle);
                        }
                }
        }
}
```

E.3 RTC jednotka – určení času a data z RTC čítače

```
/* ----------------------------------------------------------------------------
 * Jmeno funkce:      RTC_getTime( )
 * Popis:             zjisteni data a casu z RCTcounteru
                      funkce zapisuje stanovene hodnoty do globalni promenne
                      viz. vystupy
 * Vstupy:
 * Vystupy:    currentDateTime
                      .ss     - sekundy
                      .mm - minuty
                      .hh - hodiny
                      .DD - dny
                      .MM - mesice
                      .YY     - roky
 * Return hodnota:
-*----------------------------------------------------------------------------*/
void RTC_getTime( void )
{
  uint32_t regVal = 0, tmp = SEC_PER_DAY, tmpDD = 1, tmpMM = 1, tmpYY = 10;
  weakDay = 4;

        //--- nacteni RTCcounter hodnoty aktualniho casu ----------------
        RTC_WaitForLastTask();        // cekani na skonceni posledni RTCreg operace
        regVal = RTC_GetCounter();    // nacteni hodnoty RTC counteru
        regVal = summerTimeCorrection(regVal);      // korekce letniho casu

        //--- vypocet hodnot data z hodnoty RTCCounteru -----------------
        while(tmp < (regVal+1))
        {
                if(weakDay == 6)              // nastaveni dne v tydnu na pondeli je-li nedele
                {
                        weakDay = 0;
                }
                else
                {
                        weakDay++;            // nastaveni dne v tydnu nastavit dalsi den
                }

                if( (tmpDD < daysInMonthsList[tmpMM-1]) )   // neni-li posledni den v mesici
                {
                        tmpDD++;              // pricte den
                }
                else                          // je-li posledni den v mesici
                {
                        if( (tmpYY%4 == 0 ) && (tmpMM == 2) && (tmpDD == 28) )
                        // korekce - unor prestupneho roku
                        {
                                tmpDD++;      // pricte den
                                tmpMM--;      // korekce pricitani mesice
                        }
                        else
                        {
                                tmpDD = 1;    // nastavi prvni den v mesici
                        }
```

```
                        if(tmpMM < 12)          // neni-li posledni mesic v roce
                        {
                                tmpMM++;        // pricte mesic
                        }
                        else                    // je-li posledni mesic v roce
                        {
                                tmpMM = 1;      // nastavi prvni mesic v roce
                                tmpYY++;        // nastavi dalsi rok
                        }
                }
                tmp += SEC_PER_DAY;             // pricti pocet sekund za den do pocitadla dni
        }

        //--- ulozeni zjistenych hodnot do globalni struktury aktualniho casu
        regVal = regVal % SEC_PER_DAY;          // zjisteni hodnoty aktualniho casu v dany den
        currentDateTime.hh = regVal / 3600;     // vypocet hodin z hodnoty registru
        currentDateTime.mm = (regVal % 3600) / 60;   // vypocet minut z hodnoty registru
        currentDateTime.ss = (regVal % 3600) % 60;   // vypocet sekund z hodnoty registru
        currentDateTime.DD = tmpDD;             // zapis hodnoty dni aktualniho data
        currentDateTime.MM = tmpMM;             // zapis hodnoty mesicu aktualniho data
        currentDateTime.YY = tmpYY;             // zapis hodnoty let aktualniho data
}
```

E.4 Změření a zakódování veličin jednoho cyklu logování

```
/* ---------------------------------------------------------------------------
 * Jmeno funkce:              measureAndStoreData( )
 * Popis:                     zmereni velicin, komprese a ulozeni do pameti
 * Vstupy:                    -
 * Vystupy:           -
 * Return hodnota:    -
-*---------------------------------------------------------------------------*/
void measureAndStoreData(void)
{
        uint8_t i = 0, j = 0, dataBufferIdx = 0, bitPointer = 0;
        uint8_t blockCount = 0;
        uint8_t encDataBitCount[4] = {0};
        uint8_t dataBuffer[10] = {0};
        uint16_t byteCount = 0;
        uint32_t encData[4] = {0};
        float lastValP = 0, lastValRH = 0, lastValT2 = 0, lastValT1 = 0;

        Mem_PWR(MEM_RESUME);                        // probuzeni pameti
        Mem_writeSREG_byte1(0x00);                  // povoleni zapisu do pameti

        /*-------------------------------------------------------------*/
        //--- nacteni posledni adresy zapisu do pameti ------------------
        memLastWriteAddr =    (uint32_t)BKP_ReadBackupRegister(BKP_LASTMEMADDR1);
        memLastWriteAddr |= ((uint32_t)BKP_ReadBackupRegister(BKP_LASTMEMADDR2)) << 16 ;

        //--- nacteni poctu zapsanych mereni a bytu ---------------------
        blockCount = (uint8_t)BKP_ReadBackupRegister(BKP_BLOCKCOUNT);
        byteCount  = BKP_ReadBackupRegister(BKP_BYTECOUNT);

        /*-------------------------------------------------------------*/
        //--- zacatek noveho bloku dat ----------------------------------
        if(blockCount == 0)
        {
                updateDataHeader();
                // zapis hlavicky bloku do pameti
                Mem_writeBuffer(dataHeader, (memLastWriteAddr + 1), DATA_HEADER_LENGTH-3);
                memLastWriteAddr += DATA_HEADER_LENGTH;
        }
        /*-------------------------------------------------------------*/
        //--- mereni, komprese, ulozeni komprimovanych dat do pameti ----
        decodeCurrentBlock((float*) &lastValP, (float*) &lastValRH, (float*) &lastValT2,
                           (float*) &lastValT1);
```

```
if(((uint8_t)BKP_ReadBackupRegister(BKP_SREG) & SREG_MVARS_TEMP1_MASK) != 0)
{
	// mereni T1
	RHsens_getMeasurement((float*) &mRH, (float*) &mTemp1);// zmereni dane veliciny
	mTemp1 -=lastValT1;
	encData[0] = encodeSingleMeasurement(mTemp1);// zakodovani veliciny
}

if(((uint8_t)BKP_ReadBackupRegister(BKP_SREG) & SREG_MVARS_RH_MASK) != 0 )
{
	// mereni RH
	RHsens_getMeasurement((float*) &mRH, (float*) &mTemp1);// zmereni dane veliciny
	mRH -=lastValRH;
	encData[1] = encodeSingleMeasurement(mRH);  // zakodovani veliciny
}

if(((uint8_t)BKP_ReadBackupRegister(BKP_SREG) & SREG_MVARS_TEMP2_MASK) != 0)
{
	// mereni T2
	mTemp2 = Read_Temperature();                // zmereni dane veliciny
	mTemp2 -=lastValT2;
	encData[2] = encodeSingleMeasurement(mTemp2);// zakodovani veliciny
}

if(((uint8_t)BKP_ReadBackupRegister(BKP_SREG) & SREG_MVARS_PRESS_MASK) != 0)
{
	// mereni AT
	mPress = psens_measure() - 10;       // mereni + korekce hodnot nad 100.0
	mPress -=lastValP;
	encData[3] = encodeSingleMeasurement(mPress);// zakodovani veliciny
}
/*-------------------------------------------------------------*/
//--- ulozeni zmerenych a kodovanych dat do pemeti --------------
for( i=0 ; i<4 ; i++ )
{
	encDataBitCount[i] = 0x000000FF & encData[i];
	encData[i] &=0xFFFFFF00;
	encData[i] = encData[i] >> 8;
}

dataBufferIdx = 0;

//--- nacteni bitu dat a jejich delky zbylych z minuleho zapisu -
bitPointer = (uint8_t)(BKP_ReadBackupRegister(BKP_DATA_REMAINED) >> 8);
dataBuffer[dataBufferIdx] = (uint8_t)(BKP_ReadBackupRegister(BKP_DATA_REMAINED) << 1);

//--- priprava bufferu dat pro ulozeni do pameti z kodovanych posloupnosti
for( j=0 ; j<4 ; j++ )
{
	while(encDataBitCount[j] != 0)
	{
		dataBuffer[dataBufferIdx] |= 0x01 & ((uint8_t)encData[j]);
		encData[j] = encData[j] >> 1;

		bitPointer++;

		if(bitPointer == 8)
		{
			dataBufferIdx++ ;
			bitPointer = 0;
		}
		else
		{
			dataBuffer[dataBufferIdx]  = (dataBuffer[dataBufferIdx] << 1);
		}
		encDataBitCount[j]-- ;
	}
}
dataBuffer[dataBufferIdx]  = dataBuffer[dataBufferIdx] >> 1;
/*-------------------------------------------------------------*/
//--- zapis bufferu zakodovanych dat do pameti ------------------
Mem_writeBuffer(dataBuffer, (memLastWriteAddr + 1), dataBufferIdx);
Mem_PWR(MEM_SLEAP);                             // uspani pameti
```

```
        memLastWriteAddr += dataBufferIdx;
        byteCount += dataBufferIdx;
        blockCount++;
        //--- ulozeni bitu zbylych dat a jejich delky -------------------
        //--- pro pristi zapis do BKPreg
        BKP_WriteBackupRegister(BKP_DATA_REMAINED, ((uint16_t)dataBuffer[dataBufferIdx]) |
                                ((uint16_t)bitPointer << 8) );

        //--- ulozeni posledni adresy zapisu do pameti do BKPreg --------
        BKP_WriteBackupRegister(BKP_LASTMEMADDR1, (uint16_t)memLastWriteAddr);
        BKP_WriteBackupRegister(BKP_LASTMEMADDR2, (uint16_t)(memLastWriteAddr >> 16));

        //--- ulozeni blockCount do BKPreg
        BKP_WriteBackupRegister(BKP_BLOCKCOUNT, (uint16_t)blockCount);
        //--- ulozeni byteCount do BKPreg
        BKP_WriteBackupRegister(BKP_BYTECOUNT , byteCount );
        /*-------------------------------------------------------------*/
        //--- pri zapisu posledniho mereni v bloku ----------------------
        if(blockCount == BLOCK_LENGTH)
        {
               finalizeBlock();                          // ukonceni bloku
        }
        /*-------------------------------------------------------------*/
}
```

E.5 Odeslání stavu dataloggeru po USB

```
/* ---------------------------------------------------------------------------
 * Jmeno funkce:      sendStatus( )
 * Popis:             odeslani provoznich informaci do PC
 * Vstupy:
 * Vystupy:
 * Return hodnota:
-*---------------------------------------------------------------------------*/
void sendStatus(void)
{
  extern uint32_t ADCConvertedValue[];
  float Vbat;
  char str[4];
        //--- naplneni datoveho ramce k odeslani
        Send_Buffer[0] = IN_REPORT_GD;                   // report ID
        Send_Buffer[1] = STATUS_INFO;                    // subID

        Send_Buffer[2] = (uint8_t)BKP_ReadBackupRegister(BKP_SREG);       //status registr

        Send_Buffer[3] = (uint8_t)BKP_ReadBackupRegister(BKP_LASTMEMADDR1);// kapacita pameti
        Send_Buffer[4] = (uint8_t)(BKP_ReadBackupRegister(BKP_LASTMEMADDR1) >> 8);
        Send_Buffer[5] = (uint8_t)BKP_ReadBackupRegister(BKP_LASTMEMADDR2);

        start_ADCconv();                                 // spusteni ADC prevodu
        Vbat = 2*3.3*(uint16_t)ADCConvertedValue[1]/4096;  // vypocet napeti bat. z ADC
        sprintf(str,"%1.2f",Vbat);                       // prevod na ASCII
        Send_Buffer[6] = str[0];                         // napeti baterie
        Send_Buffer[7] = str[1];
        Send_Buffer[8] = str[2];
        Send_Buffer[9] = str[3];
        //--- odeslani dat po USB ---------------------------------------
        USB_SIL_Write(EP1_IN, (uint8_t*) Send_Buffer, 64);
    SetEPTxValid(ENDP1);
}
```

F UKÁZKY ZDROJOVÉHO KÓDU SW

F.1 Enumerace a kontrola připojení zařízení

```
/*-------------------------------------------------------------------------
 * Funkce:     TimerTick( )
 * Popis:      funkce slouzi pro enumeraci a kontrole pripojeni zarizeni
 *-------------------------------------------------------------------------*/

void TimerTick(object sender, EventArgs e)
{
       // enumerace zarizeni podle VID a PID
       HidDeviceList = HidDevices.Enumerate(0x0483, 0x5750);
       if (HidDeviceList.Length > 0)        // pukud bylo nalezeno zarizeni s VID a PID
       {
              HidDevice = HidDeviceList[0]; // vybere prvni zarizeni v seznamu
              if(HidDevice.IsConnected)     // kontrola je-li zarizeni pripojeno
              {
                     label_connStat.Text = "Datalogger připojen";// zobrazeni infa na panel
                     label_connStat.ForeColor = Color.Green;     // textcolor - zelena
                     button_StartMeasure.Enabled = true;  // povoli tlacitka
                     button_setNewConfig.Enabled = true;  // spoustejici USB komunikaci
                     button_dldData.Enabled = true;
                     button_setConfigCont.Enabled = true;
                     timer_status.Enabled = true;
              }
       }
       else                                 // pokud zarizeni nebylo nalezeno
       {
              label_connStat.Text = "Datalogger odpojen"; // zobrazeni infa na panel
              label_connStat.ForeColor = Color.Red;       // textcolor - cervena
              button_StartMeasure.Enabled = false;        // zakaze tlacitka spoustejici
              button_setNewConfig.Enabled = false;        // USB komunikaci
              button_dldData.Enabled = false;
              button_setConfigCont.Enabled = false;
              timer_status.Enabled = false;
       }
}
```

F.2 Příprava a odeslání konfigurace logování

```
/*-------------------------------------------------------------------------
 * Funkce:     configReport( )
 * Popis:      nacteni a odeslani konfigurace planovaneho logovani
 *-------------------------------------------------------------------------*/
public void configReport()
{
       byte[] Send_Buffer = new byte[64];   // buffer pro odchozi komunikaci
       HidDeviceData Rec_Buffer;            // buffer pro prichozi komunikaci

       //--- hlavicka reportu ---------------------------------
       Send_Buffer[0] = 0x03;               //IN_REPORT_CONFIG    0x03
       Send_Buffer[1] = 0x01;               //CONFIG_LOGGING      0x01
       //--- perioda mereni -----------------------------------
       Send_Buffer[2] = System.Convert.ToByte(timePicker_Periode_ss.Value);
       Send_Buffer[3] = System.Convert.ToByte(timePicker_Periode_mm.Value);
       Send_Buffer[4] = System.Convert.ToByte(timePicker_Periode_hh.Value);
       Send_Buffer[5] = System.Convert.ToByte(timePicker_Periode_DD.Value);
       //--- status registr -----------------------------------
       Send_Buffer[6] = 0x01;
       if(CB_mVars_temp1.Checked == true)
       {Send_Buffer[6] |= 0x04;}
       if(CB_mVars_temp2.Checked == true)
       {Send_Buffer[6] |= 0x08;}
       if(CB_mVars_RH.Checked == true)
       {Send_Buffer[6] |= 0x10;}
       if(CB_mVars_AT.Checked == true)
       {Send_Buffer[6] |= 0x20;}
       //--- cas zacatku logovani -----------------------------
```

```
        Send_Buffer[7] = System.Convert.ToByte(timePicker_StartL_ss.Value);
        Send_Buffer[8] = System.Convert.ToByte(timePicker_StartL_mm.Value);
        Send_Buffer[9] = System.Convert.ToByte(timePicker_StartL_hh.Value);
        Send_Buffer[10] = ystem.Convert.ToByte(dateTimePicker_logStart.Value.Day);
        Send_Buffer[11] = ystem.Convert.ToByte(dateTimePicker_logStart.Value.Month);
        Send_Buffer[12] = ystem.Convert.ToByte(dateTimePicker_logStart.Value.Year - 2000);

        //--- cas zacatku logovani ------------------------------
        Send_Buffer[13] = System.Convert.ToByte(timePicker_StopL_ss.Value);
        Send_Buffer[14] = System.Convert.ToByte(timePicker_StopL_mm.Value);
        Send_Buffer[15] = System.Convert.ToByte(timePicker_StopL_hh.Value);
        Send_Buffer[16] = System.Convert.ToByte(dateTimePicker_logStop.Value.Day);
        Send_Buffer[17] = System.Convert.ToByte(dateTimePicker_logStop.Value.Month);
        Send_Buffer[18] = System.Convert.ToByte(dateTimePicker_logStop.Value.Year - 2000);

        //--- odeslani konfiguracniho reportu -------------------
        timer_status.Enabled = false;        // zakazani casovace pro kontrolu stavu
        HidDevice.Write(Send_Buffer);

        Rec_Buffer = HidDevice.Read();       // nacteni prichoziho potvrzeni

        string message = "Konfigurace byla nahrána.";
        string caption = "Info";
        if(Rec_Buffer.Data[1] == 0xFF)       // kontrola spravnosti operace
        {
                // zobrazeni dialogu potvrzeni uspesne vykonane operace
                MessageBox.Show(message, caption,
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Information);
                timeSync();                  // synchronizace času
        }
        else
        {
                // zobrazeni chyboveho dialogu pri selhani
                message = "Operace selhala.";
                caption = "Chyba!";
                MessageBox.Show(message, caption,
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
        }
        timer_status.Enabled = true;  // povoleni casovace pro kontrolu stavu
}
```

F.3 Dekomprese bloku dat

```
/*-------------------------------------------------------------------------------
 * Funkce:     decodeHuffman( )
 * Popis:      dekodovani bloku dat
 *-------------------------------------------------------------------------------*/
public void decodeHuffman()
{
        UInt16 d, Sreg = 0;
        byte decSymbol = 0;
        int i, j, k;
        UInt16 bitPointer = 0;
        int behCnt = 1, varCnt = 0, doneP = 0, doneT1 = 0, doneT2 = 0, doneRH = 0, doneCycle = 0;
        .
        .
        double[]  dP  = new double[200];
        int curBytePtr = 0, decDataPtr = 0;

        //--- nacteni prvniho vyrovnavaciho bufferu ----------------------
        bitPointer = block[ byteCount ];
        d = block[byteCount -1 ];
        Sreg = blockHeader[2];
        varCnt  = ((Sreg & 0x04) >> 2);      // SREG_MVARS_TEMP1_MASK      0x04
        varCnt += ((Sreg & 0x08) >> 3);      // SREG_MVARS_TEMP2_MASK      0x08
        varCnt += ((Sreg & 0x10) >> 4);      // SREG_MVARS_RH_MASK         0x10
        varCnt += ((Sreg & 0x20) >> 5);      // SREG_MVARS_PRESS_MASK      0x20
        curBytePtr = byteCount-2;
        decDataPtr = blockHeader[14] -1;
        // opakovani - pocet logu v bloku
```

```
for(i = 0; i < (varCnt * blockHeader[14]); i++)
{       // opakovani - ctverice symbolu
        for(j = 0; j<4 ; j++)
        {       // doplneni vyrovnavaciho bufferu
                if((bitPointer < 5) && (curBytePtr >= 0))
                {
                        d |= (UInt16)(((UInt16)block[curBytePtr]) << bitPointer) ;
                        curBytePtr--;
                        bitPointer += 8;
                }
                /*-------------------------------------------------------------*/
                //--- dekodovani jednoho znaku ---------------------------------
                if((d & 0x0001) == 1)               // symbol: 0
                { decSymbol = 0;      }
                else                                        // symboly: -123456789
                {
                  if( ((d >> 1) & 0x0001) == 1)             // symboly: -12
                  {
                    if( ((d >> 2) & 0x0001) == 1)           // symboly: 12
                    {
                        if( ((d >> 3) & 0x0001) == 1) // symbol: 1
                        { decSymbol = 1;      }
                        else                                // symbol: 2
                        { decSymbol = 2;      }
                    }
                    else                                    // symbol: -
                    { decSymbol = 0x0A;       }
                  }
                  else                                      // symboly: 3456789
                  {
                    if( ((d >> 2) & 0x0001) == 1)           // symboly: 389
                    {
                        if( ((d >> 3) & 0x0001) == 1) // symbol: 3
                        { decSymbol = 3;      }
                        else           // symboly: 89
                        {
                          if( ((d >> 4) & 0x0001) == 1)// symbol: 8
                          { decSymbol = 8;    }
                          else                          // symbol: 9
                          { decSymbol = 9;    }
                        }
                    }
                    else                                    // symboly: 4567
                    {
                        if( ((d >> 3) & 0x0001) == 1)       // symboly: 67
                        {
                          if( ((d >> 4) & 0x0001) == 1)// symbol: 6
                          { decSymbol = 6;    }
                          else                          // symbol: 7
                          { decSymbol = 7;    }
                        }
                        else                                // symboly: 45
                        {
                          if( ((d >> 4) & 0x0001) == 1)// symbol: 4
                          { decSymbol = 4;    }
                          else                          // symbol: 5
                          { decSymbol = 5;    }
                        }
                    }
                  }
                }
                digits[j] = decSymbol;        // ulozeni dekodovaneho symbolu
                bitPointer -= HuffmanTableLengths[decSymbol];
                k = HuffmanTableLengths[decSymbol];
                while(k>0)
                {
                        d = (UInt16)((0xFFFE & d) >> 0x0001);
                        k--;
                }
                /*-------------------------------------------------------------*/
        }
        decVal = 10*digits[1] + digits[2] +0.1*digits[3]; // prevod na hodnotu float
        if(digits[0] == 0x0A ){decVal = 0 - decVal;}
        /*-------------------------------------------------------------*/
```

```
                // prirazovani dekodovane hodnoty jednotlivym velicinam podle status registru
                // SREG_MVARS_PRESS_MASK 0x20
                if( (doneP == 0x00) && (doneCycle == 0x00) && ((Sreg & 0x20) != 0))
                {
                        dP[decDataPtr] = decVal;
                        doneP++;
                        doneCycle++;
                        behCnt++;
                }
                // SREG_MVARS_TEMP2_MASK 0x08
                if( (doneT2 == 0x00) && (doneCycle == 0x00) && ((Sreg & 0x08) != 0))
                {
                        dT2[decDataPtr] = decVal;
                        doneT2++;
                        doneCycle++;
                        behCnt++;
                }
                // SREG_MVARS_RH_MASK 0x10
                if( (doneRH == 0x00) && (doneCycle == 0x00) && ((Sreg & 0x10) != 0 ))
                {
                        dRH[decDataPtr] = decVal;
                        doneRH++;
                        doneCycle++;
                        behCnt++;
                }
                // SREG_MVARS_TEMP1_MASK 0x04
                if( (doneT1 == 0x00) && (doneCycle == 0x00) && (( Sreg & 0x04) != 0))
                {
                        dT1[decDataPtr] = decVal;
                        doneT1++;
                        doneCycle++;
                        behCnt++;
                }
                doneCycle = 0;
                if(behCnt == varCnt+1)
                {
                        decDataPtr--;
                        behCnt = 0x01;
                        doneT1 = 0x00;
                        doneT2 = 0x00;
                        doneRH = 0x00;
                        doneP  = 0x00;
                }
        }
        //--- diferenvni dekodovani bloku -----------------------
        for(i = 1 ; i < blockHeader[14] ; i++)      // pro pocet mereni v bloku
        {
                dP[i] = dP[i-1] + dP[i];
                dT1[i] = dT1[i-1] + dT1[i];
                dT2[i] = dT2[i-1] + dT2[i];
                dRH[i] = dRH[i-1] + dRH[i];
        }
        //--- ulozeni hodnot bloku do bloku vyslednych dat ------
        // casovy interval
        timeInterval = blockHeader[11] | (blockHeader[12] << 8) | (blockHeader[13] << 16);
        // perioda mereni z hlavicky bloku
        initDateTime = new DateTime(2000 + (int)blockHeader[3], (int)blockHeader[4],
                                (int)blockHeader[5],  // datum z hlavicky bloku
                                (int)blockHeader[6], (int)blockHeader[7],
                                (int)blockHeader[8]); // cas z hlavicky bloku
        // pro vsechny dekodovane hodnoty v bloku
        for(i = 0; i < blockHeader[14]; i++)
        {       // vytvoreni casoveho udaje logu
                tmpDateTime = initDateTime;
                tmpDateTime = tmpDateTime.AddSeconds(i * (double)timeInterval);
                timeStamp[varPtr + i] = tmpDateTime.ToOADate();
                // ulozeni hodnot velicin do bloku vyslednych dat
                Temp1[varPtr + i] = dT1[i];
                Temp2[varPtr + i] = dT2[i];
                RH   [varPtr + i] = dRH[i];
                Press[varPtr + i] = dP [i] + 10;     //korekce kodovanych dat (přesah 100)
        }
        varPtr += blockHeader[14];
}
```

G PŘÍKLAD EXPORTU SOUBORU DAT Z SW

Dlog - Exportovaná data

Datum a čas stahování: 19.5.2011 04:13:02

Začátek měření: 19.5.2011 03:18:00
Konec měření: 19.5.2011 03:20:00

Změřené hodnoty:
T1 - teplota [°C] - místní senzor
T2 - teplota [°C] - externí senzor
RH - relativní vlhkost [%]
AT - atmosférický tlak [kPa]

Datum:	Čas:	T1:	T2:	RH:	AT:
19.5.2011	03:18:00	25,00	14,70	38,80	98,60
19.5.2011	03:18:04	25,00	14,70	38,90	98,60
19.5.2011	03:18:08	25,00	14,80	38,90	98,60
19.5.2011	03:18:12	25,00	14,70	39,00	98,60
19.5.2011	03:18:16	25,00	14,80	39,10	98,70
19.5.2011	03:18:20	24,90	14,80	39,10	98,60
19.5.2011	03:18:24	24,90	14,80	41,30	98,60
19.5.2011	03:18:28	25,40	14,80	67,70	98,60
19.5.2011	03:18:32	26,20	14,80	77,30	98,60
19.5.2011	03:18:36	26,10	14,80	77,90	98,60
19.5.2011	03:18:40	26,60	14,80	82,50	98,60
19.5.2011	03:18:44	26,30	14,80	78,80	98,60
19.5.2011	03:18:48	26,50	14,70	79,30	98,70
19.5.2011	03:18:52	26,20	14,70	73,70	98,60
19.5.2011	03:18:56	26,00	14,70	63,10	98,60
19.5.2011	03:19:00	25,80	14,70	54,40	98,60
19.5.2011	03:19:04	25,50	14,70	47,70	98,60
19.5.2011	03:19:08	25,40	14,70	43,90	98,60
19.5.2011	03:19:12	25,20	14,70	41,90	98,60
19.5.2011	03:19:16	25,20	14,60	41,00	98,70
19.5.2011	03:19:20	25,10	14,60	40,80	98,60
19.5.2011	03:19:24	25,10	14,60	40,50	98,60
19.5.2011	03:19:28	25,00	14,60	40,20	98,60
19.5.2011	03:19:32	25,00	14,60	40,10	98,60
19.5.2011	03:19:36	24,90	14,60	39,80	98,60
19.5.2011	03:19:40	24,90	14,60	39,70	98,60
19.5.2011	03:19:44	24,90	14,60	39,80	98,60
19.5.2011	03:19:48	24,90	14,60	39,80	98,60
19.5.2011	03:19:52	24,80	14,60	39,70	98,60
19.5.2011	03:19:56	24,80	14,60	39,70	98,60
19.5.2011	03:20:00	24,80	14,60	39,70	98,60

---------- Konec Souboru ----------

H NÁVOD K POUŽITÍ DATALOGGERU

Nezávislý datalogger s USB připojením

Bc. Karel Románek
Brno 2011

Součásti dataloggeru

- Datalogger
- Baterie Nokia BL-5C
- Externí senzor – konektor CINCH
- Kabel USB A/mini
- Ovládací aplikace Dlog

Kompatibilita ovládací aplikace

Aplikace je kompatibilní s operačními systémy Windows 7 / Vista. V OS Windows XP vyžaduje instalaci balíku .Net Framework 3.5.

Popis dataloggeru

Nezávislý datalogger umožňuje dlouhodobé záznamy atmosférických veličin. Zařízení zaznamenává:

- Teplotu vzduchu – dva senzory
- Relativní vlhkost vzduchu RH
- Atmosférický tlak AT

Konfigurace logování a následné vyčítání dat probíhá pomocí ovládací aplikace Dlog.

Režimy dataloggeru

Datalogger může být nastaven do několika režimů:

1) Přímé měření bez připojení k PC
2) Přímé měření pomocí ovládací aplikace
3) Plánované měření
4) Kontinuální měření

První dva režimy nevyžadují žádnou konfiguraci a změřená data nejsou zaznamenávána do paměti dataloggeru.

Režim přímého měření

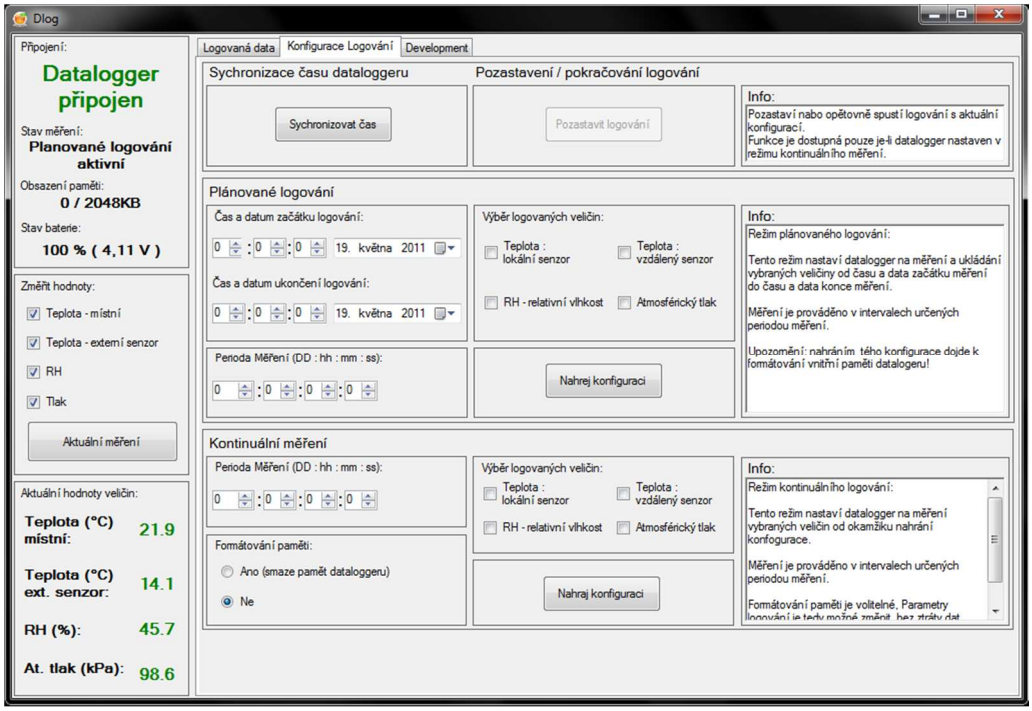
Datalogger je aktivován stiskem tlačítka **„OK“**. Druhým stiskem tlačítka **„OK“** je otevřeno menu. Potvrzením první položky „Aktualni mereni“ dojde ke změření veličin a vypsání jejich aktuálních hodnot na LCD displej.

Režim přímého měření pomocí ovládací aplikace

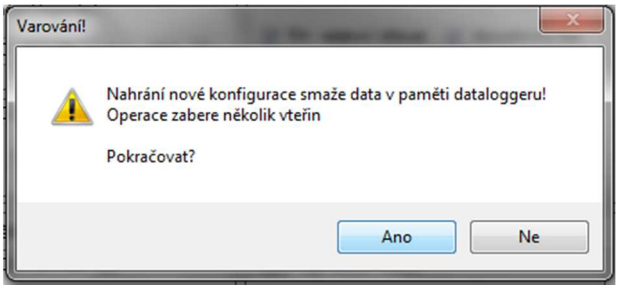
Datalogger je nutné připojit k USB pomocí kabelu USB A/mini. Stiskem tlačítka „OK“ je datalogger aktivován. Po spuštění ovládací aplikace je signalizován stav připojení dataloggeru v panelu **1**. V panelu **2** jsou zatržení políček vybrány veličiny, které mají být změřeny. Po stisku tlačítka **„Aktuální měření“** dojde ke změření požadovaných veličin a jejich zobrazení v panelu **3**.

Režim plánovaného měření

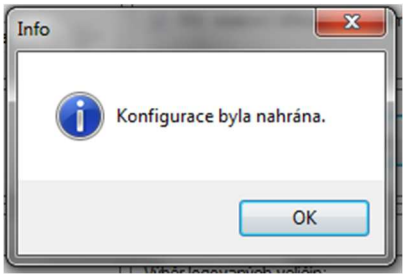
Režim plánovaného logování provádí měření ve stanovených časových periodách v definovaném období. Plánované logování je nastaveno v ovládací aplikaci v kartě **„Konfigurace Logování“**. Panel **„Plánované logování“** slouží k nastavení tohoto režimu. Zde je nastaven čas a datum začátku a konce logovacího období, perioda měření a výběr veličin, které mají být zaznamenávány.



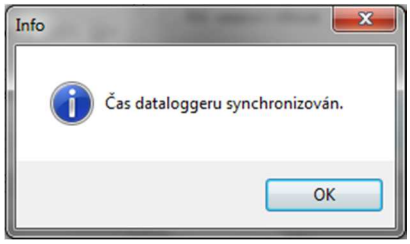
Konfigurace je do dataloggeru nahrána stiskem tlačítka **„Nahraj konfiguraci“** v příslušném panelu. Tato konfigurace vyžaduje formátování paměti dataloggeru, proto operace trvá několik vteřin. Po stisku tlačítka je zobrazeno varování o ztrátě dat z paměti s výběrem, zda má program pokračovat v nahrání konfigurace.



Pokud operace proběhne úspěšně je zobrazena informace o úspěšném nahrání konfigurace.



Při nahrání konfigurace je také synchronizován čas dataloggeru a je o této skutečnosti zobrazena informace.



Režim kontinuálního měření

V tomto režimu datalogger provádí měření ve stanovaných časových periodách od okamžiku nahrání konfigurace. Konfigurace je nastavena v kartě **„Konfigurace Logování“** v panelu **„Kontinuální měření“**. V tomto panelu je nastavena perioda měření a vybrány veličiny, které mají být zaznamenávány. Formátování paměti dataloggeru je v tomto případě volitelné. Konfigurace je nahrána do dataloggeru po stisku tlačítka „Nahraj konfiguraci“

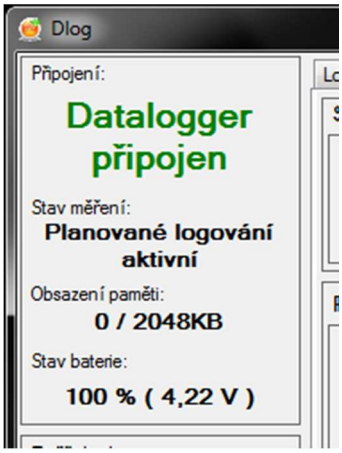
Tento konfigurační panel slouží také k úpravě parametrů probíhajícího logování. Pokud je aktivní konfigurací dataloggeru plánované měření a je provedena úprava parametrů, datalogger přechází do režimu kontinuálního měření a datum a čas začátku a ukončení logování jsou dále ignorovány.

Režim kontinuálního logování může být pozastaven případně znovu spuštěn tlačítkem v panelu **„Pozastavení / pokračování logování“**.

Synchronizace času dataloggeru může být provedena ve všech režimech dataloggeru stiskem tlačítka **„Synchronizovat čas“**.

Stav dataloggeru

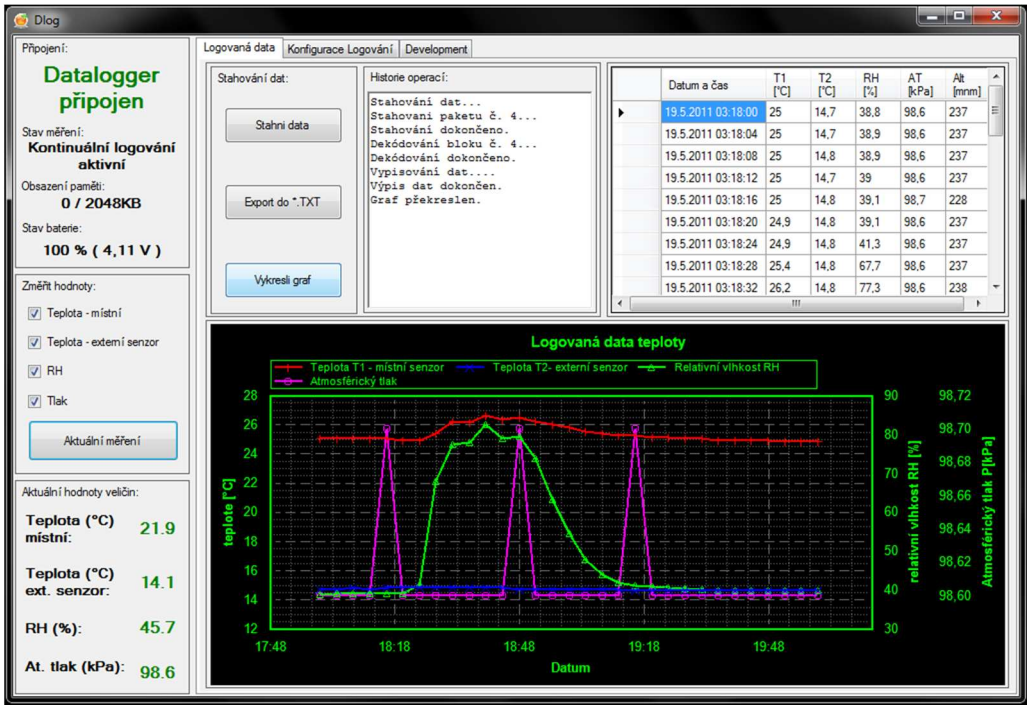
Stav dataloggeru je kontrolován a vypisován pravidelně v dvousekundových intervalech. Provozní informace jsou vypisovány v levém horním panelu.
Je zde zobrazen stav připojení k USB, obsazení kapacity paměti, stav baterie a aktuální režim dataloggeru.



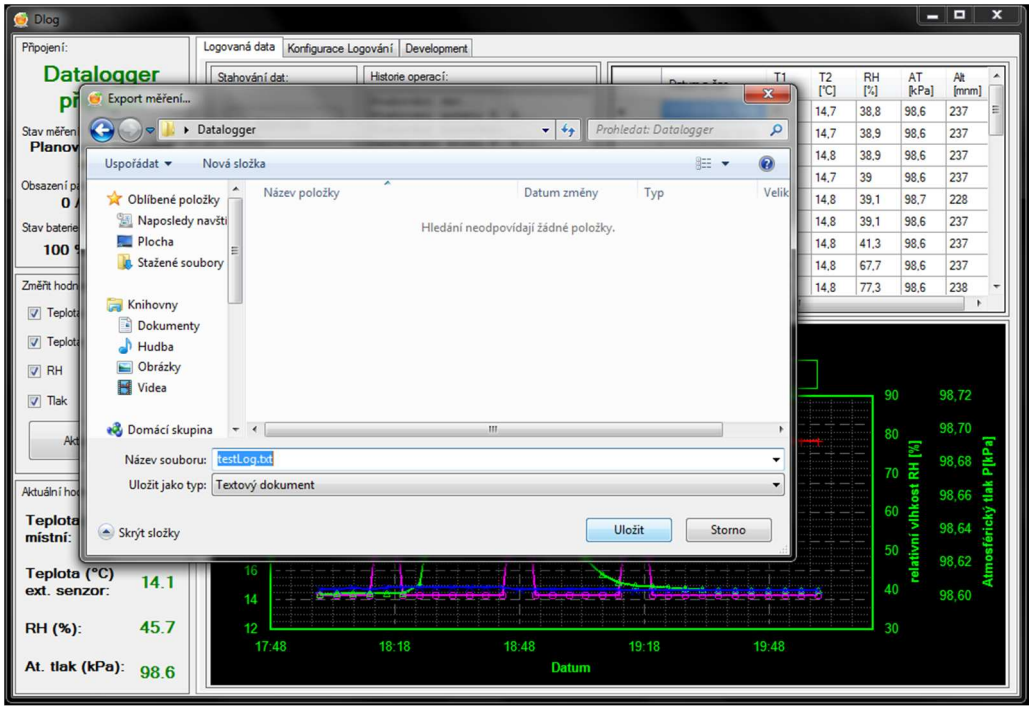
Vyčítání a zpracování logovaných dat

K tomuto účelu slouží karta **„Logovaná data“**. Zaznamenaná data z paměti dataloggeru jsou načtena po stisku tlačítka **„Stáhni data“.** Průběh operací s daty je zobrazován ve

výpisu „Historie operací“. Po úspěšném stažení jsou data vypsána do tabulky v pravém horním panelu.



Data mohou být vyexportována do textového dokumentu stiskem tlačítka **„Export do .TXT“**. Při stisku tohoto tlačítka je otevřeno dialogové okno pro výběr umístění souboru a zadání jeho názvu. Po stisku tlačítka **„Uložit“** je soubor uložen na disk a automaticky otevřen pro kontrolu.



Data mohou být zobrazena grafu po stisknutí tlačítka „Vykresli graf“. V grafu jsou zobrazeny všechny logované veličiny odlišeny barvou a značkami. Vykreslená data jsou zobrazena s automatickými osami, je tedy vidět celý průběh. Přiblížení a oddálení v grafu je možné kolečkem myši, nebo tažením myši (vybrání detailu).

V nabídce po stisku pravého tlačítka může být graf překreslen zpět do původní podoby (bez přiblížení), zkopírován do schránky, uložen jako obrázek, nebo přímo vytisknut.