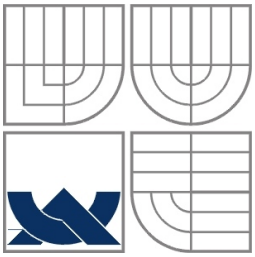
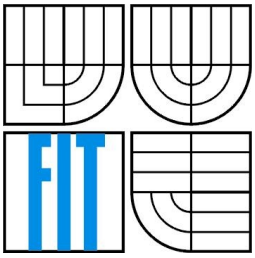


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MONITORING TEPLOTY

TEMPERATURE MONITORING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Aleš Kotačka

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Ján Kubek

BRNO 2008

Abstrakt

Diplomová práce se zabývá využitím FITkitu pro dlouhodobé monitorování teploty bez nutnosti napojení na PC. Popisuje možnosti připojení různých typů teplotních senzorů k FITkitu včetně jejich fyzické realizace. Dále se zabývá měřením teploty, zobrazením teploty na displeji a jejím ukládáním do paměti, rozdělením paměti na bloky. Definuje komunikační protokol mezi FITkitem a PC pro hromadné stahování dat. Také jsou řešeny problémy při ukládání naměřených dat do textových souborů a zpracování dat v tabulkových procesorech.

Klíčová slova

USB rozhraní, sériový port, FITkit, teplotní čidla, měření teploty, teplotní stupnice, MCU, MSP430F168, FPGA, XC3S50-4PQ208C, USB-UART převodník, FT2232C, KTY81-210, SMT 160-30, DS18B20, komunikační protokol.

Abstract

This master’s thesis deals use FITkit for longtime temperature monitoring without necessity connection on PC. Describe possibilities connection different heat-sensitive elements to FITkit inclusive physical realization. Further this thesis deals with measuring of temperature, displaying temperature and saving to memory, blocks-partitioning of memory. Is defined communication protocol between FITkit and PC for bulk downloading of data. So are solving problems with saving measure data to text files and next processing in table processors.

Keywords

USB interface, serial port, FITkit, heat-sensitive element, temperature detection, temperature scale, MCU, MSP430F168, FPGA, XC3S50-4PQ208C, USB-UART converter, FT2232C, KTY81-210, SMT 160-30, DS18B20, communication protocol.

Citace

Kotačka Aleš: Monitoring teploty. Brno, 2008, diplomová práce, FIT VUT v Brně.

Monitoring teploty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jána Kubeka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Aleš Kotačka
18.1.2008

Poděkování

Chtěl bych poděkovat vedoucímu diplomové práce Ing. Jánu Kubekovi za cenné rady a doporučení, dále své přítelkyni za pochopení a toleranci a také svým rodičům.

Obsah

Obsah 1
1 Úvod 3
2 FITkit 4
2.1 Komponenty FITkitu 4
2.1.1 Mikrokontrolér (MCU) 5
2.1.2 Hradlové pole – FPGA 5
2.1.3 USB-UART převodník 6
2.1.4 Klávesnice a LCD displej 7
2.1.5 Paměť SDRAM 8x8mbit 7
2.2 Rozhraní USB 8
2.2.1 Rychlost USB 8
2.3 Blokové schéma FITkitu 9
2.3.1 Princip programování MCU a FPGA 10
2.3.2 Napájení FITkitu 11
3 Měření teploty 12
3.1 Teplota 12
3.2 Teplotní čidla 13
3.2.1 Odporová teplotní čidla 13
3.2.2 Termistory 14
3.2.3 Převodník teplota/střída 15
3.2.4 Digitální teplotní čidlo 15
3.2.5 Termočlánky 16
4 Návrh propojení HW 17
4.1 Blokové schéma 17
4.1.1 Rozhraní SPI 17
4.1.2 LCD kontrolér 20
4.1.3 Řadič klávesnice 20
4.1.4 Generátor hodinového signálu 1 Hz 20
4.1.5 Řadič teplotních senzorů 20
4.1.6 SDRAM 25
4.2 Přesnost hodinového signálu 25
4.3 Fyzická realizace zapojení 26
5 Návrh řešení a komunikace 27
5.1 Princip ukládání teploty 27

5.1.1 Volba datového typu 27
5.1.2 Rozvržení paměti SDRAM 27
5.1.3 Výpočet maximální doby záznamu 28
5.1.4 Možnosti prodloužení max. doby záznamu 29
5.2 Komunikační protokol 32
5.2.1 Návrh komunikačního protokolu 32
5.2.2 Struktura příkazu 32
5.2.3 Přehled definovaných příkazů 34
5.2.4 Rozdělení paměti SDRAM na bloky 40
5.2.5 Princip komunikace mezi FITkitem a PC 42
6 Návrh a implementace software 44
6.1 SW pro FITkit – MCU 44
6.1.1 Soubory projektu 44
6.1.2 Výpočet a zobrazení teploty 45
6.1.3 Ovládání pomocí klávesnice 47
6.1.4 Zpracování příkazů komunikačního protokolu 48
6.1.5 Záznam teplot do paměti 49
6.2 SW pro PC 49
6.2.1 Připojení na sériový port 50
6.2.2 Zpracování komunikačního protokolu 51
6.2.3 Stahování naměřených dat 51
6.2.4 Ovládání programu 52
6.2.5 Zpracování naměřených dat v tabulkových procesorech 53
6.2.6 Ukládání dat do souboru 54
6.2.7 Nastavení programu 55
6.2.8 Zhodnocení naměřených dat během zkušebního provozu 56
7 Závěr 58
Literatura 59
Seznam příloh 60

1 Úvod

Monitorování teploty je klasická úloha, jejíž výsledky mohou sloužit k různým účelům – ať již jde čistě o statistiku pro meteorology nebo pro účely analýzy poruch zařízení v závislosti na teplotě. Diplomová práce se zabývá využitím platformy FITkit pro dlouhodobé monitorování venkovních nebo pokojových teplot z několika teplotních senzorů. Naměřená data je možné hromadně stáhnout do počítače za období několika měsíců až let v závislosti na rychlosti vzorkování. Nyní následuje stručný obsah jednotlivých kapitol.

Druhá kapitola se zabývá architekturou FITkitu a popisuje jeho jednotlivé periferie s možností využití pro danou aplikaci. Ve zkratce jsou také uvedeny informace o universální sériové sběrnici – USB, která se masově používá v poslední době.

Následující kapitola rozebírá možnosti teplotních senzorů, jejich vlastnosti a principy činnosti a zkoumá možnosti připojení těchto čidel k platformě FITkit. V úvodu jsou stručně popsány převody mezi různými teplotními stupnicemi.

Čtvrtá kapitola se věnuje převážně oblasti hardwaru. Zde jsou popsány funkční bloky zapojení celé aplikace, jejich vzájemné propojení a synchronizace. Také je řešena otázka přesnosti hodinového signálu. Závěr kapitoly definuje fyzické připojení teplotních senzorů na rozhraní FITkitu.

Pátá kapitola řeší problematiku ukládání teplot do paměti, rozdělení paměti na oddělené oblasti a možnosti kódování teplot za účelem maximalizovat dobu záznamu. V druhé části je definován komunikační protokol mezi FITkitem a osobním počítačem včetně seznamu příkazů.

Předposlední kapitola je věnována implementaci softwaru jak pro mikrokontrolér, tak i pro osobní počítač. Zabývá se principem výpočtu teploty, minimalizací chyb při měření a vůbec celkovým zpracováním teplotních údajů. Na konci je uvedeno zhodnocení zkušebního provozu.

Samotný závěr práce shrnuje dosavadní zkušenosti a poznatky získané při řešení tohoto projektu a zvažuje další možnosti rozšíření a vylepšení aplikace.

První dvě kapitoly (FITkit a Měření teploty) byly převzaty ze Semestrálního projektu a byly doplněny o nové poznatky a informace. Převážná část práce byla vypracována od začátku, protože pro řešení bylo potřeba nastudovat podrobněji zejména architekturu FITkitu.

2 FITkit

FITkit je vývojový kit, obsahující řadu HW komponent, určený pro praktickou výuku zejména hardwarových předmětů na fakultě informačních technologií. Částečnou představu získáme pohledem na obr. 2.1 a také v [1]. Jedná se tedy o vestavěný systém (anglicky *Embedded System*), jehož význam roste v poslední době. Řada aplikací, běžící na stolních počítačích, se v poslední době přesouvá do oblasti vestavěných systémů.

Obr. 2.1 FITkit - čelní pohled

2.1 Komponenty FITkitu

Jak již bylo řečeno FITkit obsahuje řadu periferií a rozhraní:

- MCU MSP430F168 (*Texas Instruments*)
- FPGA Spartan 3 XC3S50-4PQ208C (*Xilinx*)
- USB-UART převodník FT2232C
- DRAM 8x8mbit
- Klávesnice 4x4
- Znakový LCD displej 1 x 16 znaků
- Rozhraní USB (konektor typ B)

- Sérové rozhraní RS-232 (konektor CANNON 9)
- 2x konektor PS2
- Rozhraní VGA (konektor CANNON 15)
- Audio vstup a výstup (konektor JACK 3,5 mm stereo)

2.1.1 Mikrokontrolér (MCU)

Na kitu je osazen nízkopříkonový signálový mikrokontrolér **MSP430F168** [13] od firmy Texas Instruments. Tento typ je určen speciálně pro provoz na baterie.

Přehled jeho vlastností:

- Nízký příkon (5 režimů příkonu)
  - 330 µA v aktivním režimu při 1 MHz a 2,2 V
  - 1,1 µA ve stand-by režimu
  - 0,2 µA v režimu vypnuto
- 16-bitová RISC architektura
- Instrukční cyklus 125 ns
- Paměť
  - Flash 48 kB + 256 B
  - RAM 2 kB
- 2 x 16-bitový časovač
- 12-bitový 8-kanálový A/D převodník
- 12-bitový 2-kanálový D/A převodník
- 2 x asynchronní sériové rozhraní UART

Programování mikrokontroléru se provádí klasicky v jazyce C a zdrojové kódy se překládají překladačem MSPGCC [8] od firmy Texas Instruments. Pouzdro čipu má 64 vývodů. Pro úspěšné naprogramování MCU musíme mít nainstalované ovladače [9] pro USB-UART převodník a správně nastavené čísla sériových portů převodníku v souboru …`\FITkit\base\`**`Settings.inc`**.

2.1.2 Hradlové pole – FPGA

Programovatelné hradlové pole FPGA (anglicky *Field-Programmable Gate Array*) **XC3S50-4PQ208C** řady **Spartan 3** od firmy Xilinx tvoří spolu s MCU srdce FITkitu. FPGA obsahuje funkční bloky a propojovací sítě, s jejichž pomocí lze vytvořit požadovanou funkci [4]. Konfigurace FPGA (nebo jen její část) se může měnit i za běhu aplikace.

- Rozhraní
  - Až 124 uživatelských vstupů/výstupů
  - Podpora až 23 různých I/O standardů

- 192 konfigurovatelných logických bloků (CLBs) uspořádaných do matice o 16 řádcích a 12 sloupcích
- 1728 logických buněk
- 50 000 logických hradel
- Paměť RAM
  - 12 kb distribuovaných
  - 72 kb v jednom bloku
- 2 jednotky pro správu hodin (DCMs)
- 4 násobičky 18x18 bitů

2.1.2.1 VHDL

FPGA se programuje ve jazyce VHDL (***V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage*), což je jazyk pro popis hardwaru. Tento jazyk je standardizován organizací IEEE a poskytuje tak širokou kompatibilitu [7].

Strukturu HW ve VHDL můžeme popsat třemi způsoby:

- Popis chování (behavioral)
- Popis struktury (architecture)
- Popis datový toků (dataflow)

Všechny komponenty se pak propojí přes jejich rozhraní, které definuje vstupy a výstupy komponent. K vývoji a překladu zdrojových kódu v jazyce VHDL lze použít vývojové balíky Xilinx ISE 8.1i a ModelSim XE III 6.0d. Vývojový SW je možné zdarma stáhnout ze webových stránek společnosti Xilinx (je však nutná registrace). Aktuální odkazy a návody na instalaci SW a zprovoznění naleznete v [1] v sekci návody.

Vygenerování samotné konfigurace ze zdrojových kódů, kterou je možné nahrát do FITkitu, trvá až několik minut v závislosti na rozsahu projektu.

2.1.3 USB-UART převodník

Převodník **FT2232C** od firmy FTDI (*Future Technology Devices International Ltd.*) zprostředkovává komunikaci mezi MCU a PC přes sběrnici USB. Jak bylo již dříve uvedeno MCU je vybaven dvěmi asynchronními sériovými porty a samotné USB rozhraní nemá. V mnoha případech se USB rozhraní převede na asynchronní sériové rozhraní RS-232, které je na starších i novějších mikrokontrolérech dostupné a také SW podporované. V počítači se připojené zařízení přes USB detekuje jako dva USB sériové porty COM *x* a COM *y*. Podmínkou je nainstalování příslušných ovladačů, které byly v době psaní této práce ke stažení z [9]. Návod na instalaci ovladačů naleznete v [1] v sekci návody.

Obslužná aplikace na PC nebude rozlišovat, zda komunikuje se skutečným sériovým portem nebo s virtuálním sériovým portem přes USB sběrnici, protože v obou případech se využívá volání funkcí z jádra operačního systému. Často používané funkce jsou např. `CreateFile(…)`, `CloseHandle(…)`, `ReadFile(…)`, `WriteFile(…)` deklarované v hlavičkovém souboru `windows.h`.

2.1.4 Klávesnice a LCD displej

Klávesnici 4x4 klávesy lze použít pro ovládání a výběr požadovaných funkcí k jednotlivým čidlům, které se budou zobrazovat na 1-řádkovém LCD displeji:

- maximální/minimální teplota
- průměrná teplota
- název/umístění čidla
- zobrazení volné/obsazené paměti
- zobrazení intervalu měření

K ovládání displeje a získání stavu klávesnice jsou již vytvořeny obslužné rutiny, takže odpadá práce s programování těchto periferií. Zobrazovací možnosti displeje jsou na dnešní dobu omezené, protože se jedná pouze o znakový jednořádkový displej. Kdyby byl na FITkitu přítomen plně grafický displej s odpovídajícím rozlišením, bylo by možné zobrazovat současně teplotu z více čidel, případně i krátkodobý průběh – např. za posledních 24 hodin.

2.1.5 Paměť SDRAM 8x8mbit

Na FITkitu je osazen blok paměti o celkové velikosti 8 MB (8 388 608 bajtů). Tato paměť je typu SDRAM, tzn. že se jedná o synchronní dynamickou paměť (*Synchronous Dynamic Random Access Memory*), která potřebuje v pravidelných intervalech obnovovat data v ní uložené, jinak dojde ke ztrátě těchto dat. K obnově celé paměti potřebujeme 4096 cyklů a obnova musí být provedena do 64 ms. Paměť je možné synchronizovat hodinovým kmitočtem 100 MHz, ale podle praktických zkušeností spolehlivě funguje pouze na frekvenci 50 MHz.

K paměti je možné přistupovat jak z MCU, tak i z FPGA. Rovněž jsou implementovány dva typy radičů paměti pro MCU – jeden na nízké úrovni a druhý na vysoké úrovni. V rámci přehlednosti a jednoduchosti použijeme raději vysokoúrovňový řadič paměti, jenž umožňuje čtení a zápis do paměti SDRAM po bajtech. Paměť využijeme pro ukládání naměřených dat.

Při výpadku napájecího napětí ztratíme všechna naměřená data, proto je dobré systém napájení nějakým způsobem zálohovat – např. záložním zdrojem.

2.2 Rozhraní USB

USB – Universal Serial Bus, je v současné době nejrozšířenější sběrnice pro připojení externích zařízení k počítači.

Hlavní výhody:

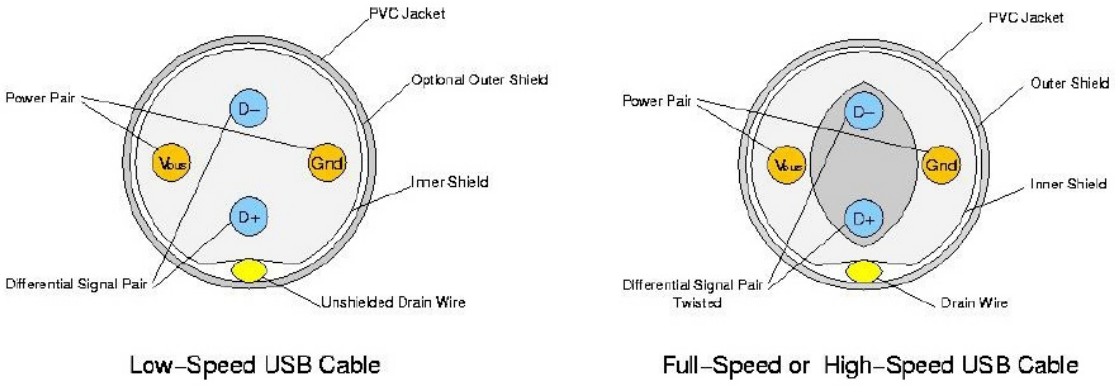
- Jednoduchá sériová obousměrná komunikace
- Napájení zařízení přímo ze sběrnice
- Vysoká rychlost komunikace až 480 Mb/s (USB 2.0)
- Nízká cena
- Podpora více zařízení
- Podpora Hot-Plug
- Automatická detekce a konfigurace zařízení
- Možnost rozšíření sběrnice

2.2.1 Rychlost USB

Standard USB definuje tři rychlosti přenosu:

- Low Speed – 1,5 Mb/s (USB 1.x)
- Full Speed – 12 Mb/s (USB 1.x)
- High Speed – 480 Mb/s (USB 2.0)

Pro komunikaci FS (Full Speed) nebo HS (High Speed) by zařízení měla být propojena patřičným kabelem viz obr. 2.2 vpravo (diferenciální datové vodiče jsou kroucené). Maximální délka tohoto typu kabelu může být 5 m (propagační zpoždění 26 ns), pro LS kabel je maximální délka 3 m (zpoždění 18 ns).



Obr. 2.2 Uspořádaní vodičů uvnitř USB kabelu

Zařízení jsou s hostitelským PC propojena čtyřmi vodiči:

- Napájení +5 V
- D+

- D-
- GND

Právě vodiče D+ a D- jsou datové komunikační vodiče, jenž tvoří diferenciální pár. Diferenciální vedení redukuje šum (projeví se na obou vodičích stejně), ale zároveň se tím omezí komunikace pouze na half-duplex (komunikace pouze jedním směrem v jednom časovém okamžiku).

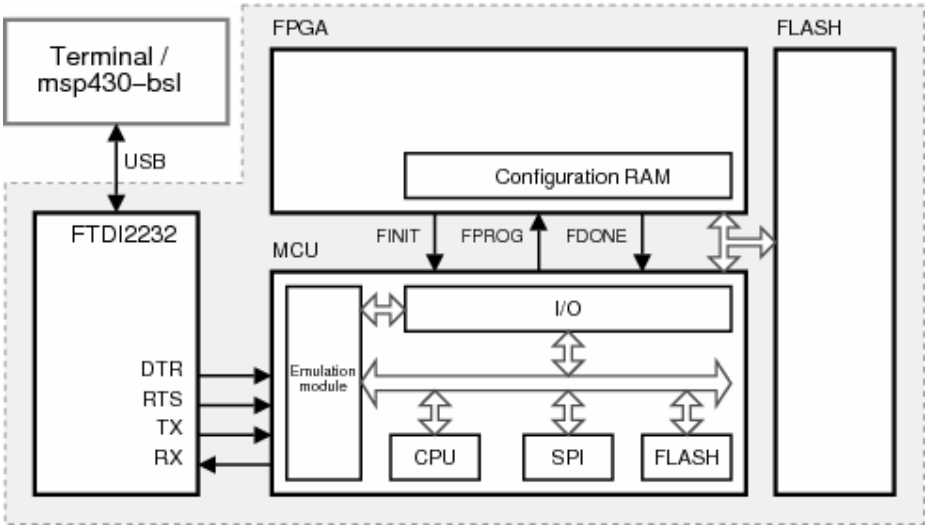
Ovladač komunikuje se zařízením prostřednictvím rour – komunikačních kanálů (pipes), přes které jdou čtyři druhy přenosů:

- Izochronní (*Isochronous*)
- Přerušovací (*Interrupt*)
- Řídící (*Control*)
- Hromadné (*Bulk*)

Jelikož ovladače čipu FT2232C převádí USB komunikaci na RS-232 (viz kapitola 2.1.3) nemá smysl se jí podrobněji zabývat, více informací naleznete v [2] a [10].

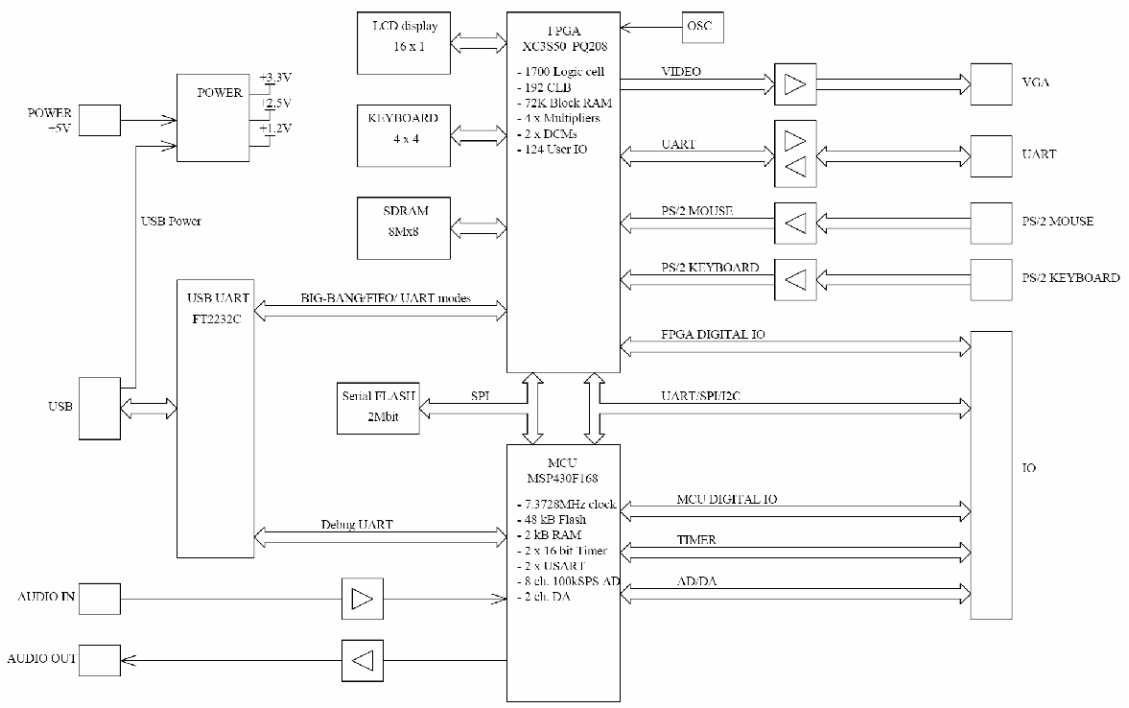
2.3 Blokové schéma FITkitu

Zjednodušené blokové schéma FITkitu je vyobrazeno na obr. 2.3. Zmenšená podoba podrobného blokového schématu je uvedena na obr. 2.4, plnou velikost blokového schématu naleznete v příloze 1.



Obr. 2.3 Blokové schéma FITkitu

Na obr. 2.4 jsou zobrazeny jednotlivé periferie FITkitu a jejich propojení.



Obr. 2.4 Podrobné blokové schéma FITkitu

2.3.1 Princip programování MCU a FPGA

Pro nahrání programu do MCU zadáme příkaz **`make load`** v adresáři se zdrojovým kódem v jazyce C (pro SVN je to `…/sw`) . Přeložený kód se zapíše do FLASH paměti v bloku MCU (viz obr. 2.3).

Pro nahraní konfigurace FPGA musíme ze zdrojového souboru `*.vhd` vytvořit (syntetizovat) binární soubor (`output.bin`) příkazem **`make`** v adresáři `…/top`. Poté spustíme komunikační terminál (*HyperTerminal* nebo *PComm*) pro komunikaci po sériové lince. V komunikačním programu nastavíme parametry sériových portů na: 460800 Bd, 8 bitů, 1 stop bit, bez parity. Otevřeme sériový port s vyšším číslem – do terminálu by se měla vypsat informace o FITkitu. V terminálu zadáme příkaz **`prog fpga`** a následně odešleme binární soubor (`output.bin`) protokolem *Xmodem* (1KCRC). A pokud nedošlo k chybě tak by mělo být FPGA naprogramované a požadovaný program běžet. Má to však jednu zásadní chybu – po odpojení napájení se konfigurace FPGA ztratí. Aby program pracoval i po opětovném připojení napájení musí se konfiguraci FPGA zapsat do FLASH paměti příkazem v terminálu **`flash w fpga`** a odeslat binární soubor. Pak ještě příkazem **`prog fpga flash`** nebo restartováním FITkitu zapíšeme konfiguraci z FLASH paměti do FPGA (to se také děje automaticky při startu FITkitu).

2.3.2 Napájení FITkitu

FITkit je možné napájet jak z USB sběrnice, tak i externím napájecím zdrojem o napětí +5 V. Externí napájení by se mělo používat v případě aplikací náročnějších na odběr proudu. Měření teploty mezi náročnější aplikace nespadá, poněvadž příkon teplotních senzorů je minimální, (max. odběr jednoho čidla je 200 μA). Při počtu 32 senzorů by celkový odběr proudu neměl přesáhnout 6,4 mA. Přitom FITkit s touto aplikací má proudový odběr 150 mA.

Připojením externího napájecího zdroje se asi nevyhneme, protože pokud má FITkit zaznamenávat teplotu během několika měsíců, těžko bude napájen jen z USB sběrnice. V takovém případě by nesmělo dojít k vypnutí počítače. A pokud vezmeme v potaz i spotřebu elektrické energie, tak jistě má menší spotřebu externí adaptér než celý počítač.

Problém nastává pokud je FITkit napájen z externího zdroje a současně z USB sběrnice. Po vypnutí počítače se totiž část energie z napájecího systému FITkitu dostává přes USB sběrnici zpět do PC. To lze u některých počítačů diagnostikovat slabě svítícími signalizačními LED na různých zařízeních. Např. se mně stalo, že notebook po vypnutí už nešel znovu zapnout. Zapnutí bylo možné až po vytažení USB kabelu. Proto se domnívám, že napájení FITkitu ze dvou zdrojů není tak úplně dořešeno.

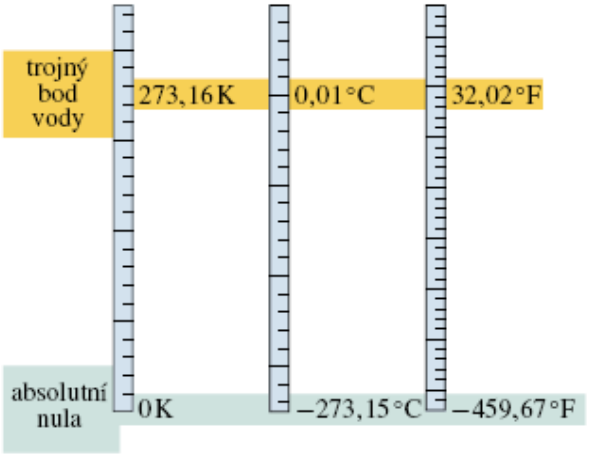
3 Měření teploty

Měření teploty je proces relativně pomalý, pokud bereme v úvahu běžné prostředí jako je např. teplota v místnosti nebo venkovní teplota vzduchu či vody. V takovémto prostředí stačí odečítat vzorky např. jednou za 10 sekund nebo i pomaleji. Na pomalé měření stačí menší výpočetní výkon (a tím také nižší spotřeba zařízení). Výpočetní výkon FITkitu je v tomto ohledu plně dostačující.

Aplikace není navrhována pro průmyslové použití, protože tam se měří teplota v daleko větším rozsahu (stovky až tisíce °C) a na přesnosti zase tak moc nezáleží. Také jsou kladeny důrazy na mechanickou odolnost a odolnost vůči elektromagnetickému rušení.

3.1 Teplota

Trojný bod vody – kapalná voda, pevný led a vodní pára mohou být spolu v tepelné rovnováze pouze při jediné teplotě a tlaku. Podle mezinárodní dohody je teplota trojného bodu stanovena na **273,16 K** [Kelvin]. Tato teplota se používá pro kalibraci teploměrů. Na obr. 3.1 je uvedeno srovnání teplotních stupnic Kelvinovy, Celsiovy a Fahrenheitovy [3].



Obr. 3.1 Srovnání stupnic teploty Kelvinovy, Celsiovy a Fahrenheitovy

V našich zeměpisných šířkách se teplota udává ve stupních Celsia [°C]. Přepočet z Kelvinovy stupnice na stupnici Celsiovu udává rov. 3.1:

$$T_C = T - 273{,}15°C \text{, [°C; K]}$$

Rov. 3.1 Přepočet teploty z Kelvinovy stupnice na Celsiovu

V Americe se pro změnu užívá stupnice ve Fahrenheitech [°F], přepočet na Fahrenheity ze °C určuje rov. 3.2:

$$T_F = \frac{9}{5} T_C + 32 \text{, [°F; °C]}$$

Rov. 3.2 Přepočet teploty z Celsiovy stupnice na Fahrenheity

3.2 Teplotní čidla

Teplotní čidla teplotu převádějí obvykle na jiné fyzikální veličiny: odpor, napětí, frekvenci nebo střídu. Některé z nich vyžadují napájecí napětí, jiné jsou zase zcela pasivní. Při výběru teplotních senzorů jsem vycházel z katalogu firmy GME [14], která nabízí elektronické součástky.

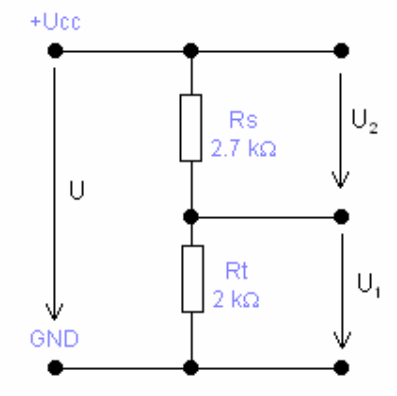
Teplotní rozsah polovodičový čidel se pohybuje v rozmezí -50 °C…+150 °C a přesnost bývá od ±0,5 °C do ±5 °C. Rozsah bývá zpravidla menší u senzorů s integrovaným čipem, protože při vyšších teplotách nelze zaručit správnou funkci křemíkové technologie.

3.2.1 Odporová teplotní čidla

Jsou teplotně závislé součástky, jejichž odpor roste téměř lineárně se stoupající teplotou. Jsou poměrně levná a dosahují přesnosti obvykle ±1 °C…±2 °C, což je z hlediska přesnosti vyhovující pro nenáročné aplikace. Teplotní rozsah: -55 °C…+150 °C (+175 °C). Jsou vyráběna v různých pouzdrech SOD70, TO92 mini, DO 34 a SOT-23. Typicky mají odpor 1 kΩ nebo 2 kΩ při 25 °C.

Čidlo lze nejednodušším způsobem k FITkitu připojit jako odporový dělič viz obr. 3.2, kde by se napětí U_1 přivedlo na vstup jednoho kanálu AD převodníku. R_S je sériový odpor, R_t představuje odporové teplotní čidlo, např. KTY81-210. Takovéto zapojení vyžaduje o něco náročnější výpočet teploty, než kdyby se čidlo připojilo do série se zdrojem konstantního proudu, protože musíme uvažovat i velikost proudu protékající čidlem, který se s teplotou mění.

K FITkitu je možné připojit až 8 takovýchto čidel (MCU má 8-kanálový 12-bitový A/D převodník). Vzhledem k tomu, že referenční napětí AD převodníku může být pouze 3,3 V, nebyl by rozsah naměřených hodnot optimální (využilo by se jen asi 65% hodnot). A při použití dlouhých přívodních vodičů by mohlo případné rušení značně ovlivnit naměřenou hodnotu.



Obr. 3.2 Připojení odporového teplotního čidla na FITkit

Vztah pro výpočet teploty podle obr. 3.2 je uveden v rov. 3.3. Hodnoty konstant jsou následující:

$$a = 7{,}88 \cdot 10^{-3} K^{-1};\ b = 1{,}937 \cdot 10^{-5} K^{-2};\ R_{25} = 2000\,\Omega;\ R_s = 2700\,\Omega$$

$$T = \left(25 + \frac{\sqrt{a^2 - 4*b + 4*b*\frac{R_S * U_1}{R_{25} * (U - U_1)}} - a}{2*b} \right) [°C]$$

Rov. 3.3 Výpočet teploty ve °C podle předchozího zapojení

3.2.2 Termistory

Jsou teplotně závislé součástky – s teplotou se mění jejich odpor, ať již pozitivně nebo negativně. Teplotní charakteristika není lineární. Teplotní rozsah termistorů: -55 °C…+125 °C.

3.2.2.1 Termistory s negativní teplotní závislostí (NTC)

Hodnota odporu klesá se vzrůstající teplotou – proto negativní teplotní závislost. V nabídce jsou termistory s hodnotou od 100 Ω do 100 kΩ. Hodnota nominálního odporu se udává při 25 °C a může se lišit až o 20 %, což vyžaduje kalibraci.

Termistory jsou sice levné, ale zase vyžadují kalibraci kvůli velkým odchylkám od nominální hodnoty. Připojení k FITkitu je možné pouze přes jednoduchý odporový dělič – převod teploty na napětí (měření AD převodníkem) nebo vytvořit převodník na frekvenci či střídu, což by představovalo značné komplikace a také by to vnášelo další nepřesnosti do výsledné hodnoty.

3.2.2.2 Termistory s pozitivní teplotní závislostí (PTC)

Hodnota odporu roste nelineárně se vzrůstající teplotou. Jelikož nejsou k dostání, tak se dále jimi zabývat nebudu.

3.2.3 Převodník teplota/střída

Toto čidlo, **SMT 160-30**, na výstupu generuje obdélníkový signál o frekvenci 1-4 kHz, jehož střída se mění lineárně s teplotou. Pracuje na napětí +5 V a má příkon menší jak 0,1 mW. Úroveň generovaného signálu je kompatibilní s logickými úrovněmi TTL a CMOS, takže jej můžeme přímo připojit na digitální vstup mikroprocesoru nebo na vstup FPGA. Střída (*DC – Duty Cycle*), tj. poměr šířky impulsu k době periody, se mění podle následujícího vztahu:

$$DC = 0{,}32 + 0{,}0047t \text{ , [-; °C]}$$

Rov. 3.4 Vztah pro výpočet střídy v závislosti na teplotě ve °C

Teplotní rozsah: -45 °C…+130 °C, přesnost: ±0,7 °C, nelinearita převodu: 0,2 °C. Čidlo je kalibrováno při výrobě, takže odpadá problém s nastavením správné teploty. Více informací je uvedeno v pramenech [5] a [6].

Počet těchto čidel připojitelných k FITkitu je omezen především počtem vstupních pinů. V této aplikaci by čidla mohla být taktéž přepínána multiplexorem implementovaným v FPGA – tím by stačil jen jeden blok pro měření střídy.

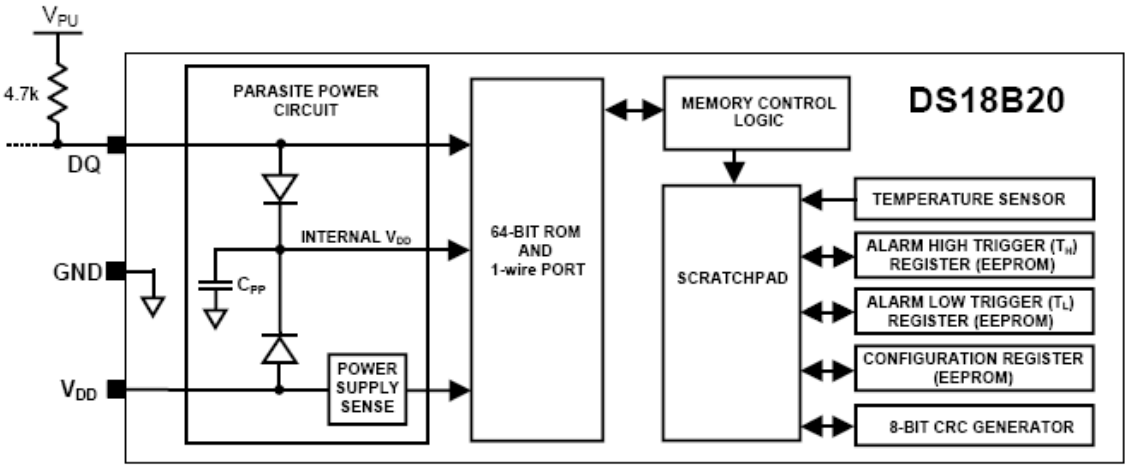
3.2.4 Digitální teplotní čidlo

Firma DALLAS Semiconductors vyrábí teplotní čidla s digitálním přenosem teploty do cílového zařízení přes sběrnici **1-Wire**. Máme k dispozici dva typy: **DS18B20** a **DS18S20**.

DS18**S**20 – 9-bitové rozlišení teploty

DS18**B**20 – programovatelné rozlišení 9–12 bitů

Několik takovýchto čidel můžeme napojit paralelně na 3vodičové vedení (+U_{CC}, data, GND), čímž ušetříme na kabeláži. Rozsah měřené teploty: -55 °C…+85 °C (+125 °C), napájecí napětí: +3,0 V…+5,0 V, přesnost ±0,5 °C v rozmezí teplot -10 °C…+85 °C. Připojení k FITkitu připadá na vstupy FPGA, ve kterém budeme muset implementovat komunikační protokol 1-Wire. Převod teploty v 12-bitovém rozlišení trvá 750 ms. Minimální doba vzorkování by pak byla limitována počtem připojených senzorů na jednom vedení, protože se musí zaručit spolehlivý přenos teploty ze všech senzorů v dané periodě.



Obr. 3.3 Blokové schéma senzoru DS18B20

3.2.4.1 Sběrnice 1-Wire

Sběrnice **1-Wire** je unikátní v tom, že po jediném datovém vodiči probíhá komunikace oběma směry. Na sběrnici musí být jeden prvek *BUS-Master*, který sběrnici řídí a jeden nebo více prvků *slave*. Čidla se mezi sebou rozlišují pomocí svého unikátního 64-bitového čísla zapsaného v ROM (viz obr. 3.4).

64-BIT LASERED ROM CODE Figure 6

8-BIT CRC		48-BIT SERIAL NUMBER		8-BIT FAMILY CODE (28h)	
MSB	LSB	MSB	LSB	MSB	LSB

Obr. 3.4 Struktura kódu

Pro komunikaci je velmi důležité časování a dodržování ochranných intervalů.

3.2.5 Termočlánky

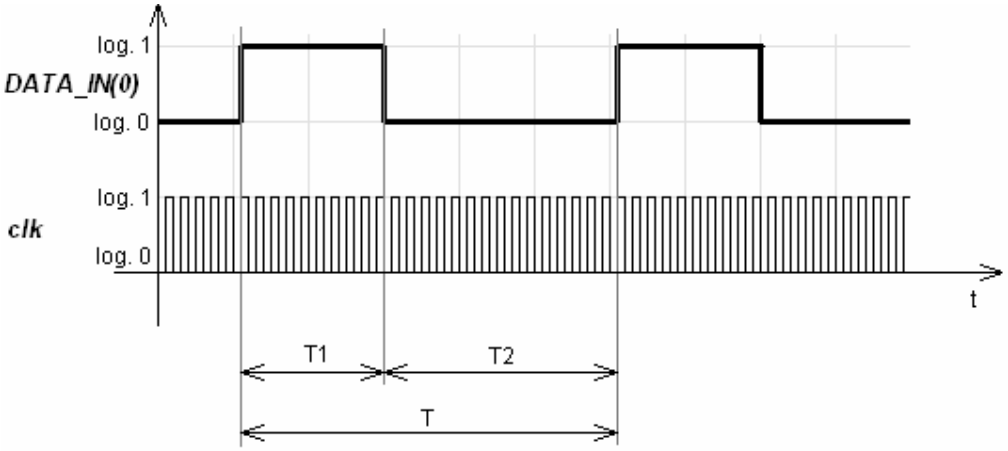
Termočlánek je složen ze dvou kovů zapojených do série a pokud mají tyto kovy navzájem různou teplotu, tak na jejich rozhraní vzniká elektrické napětí. Toto napětí dosahuje řádu milivoltů, citlivost se pohybuje v řádech několika desítek µV/°C. Termočlánků existuje více druhů (podle typu přechodu) a také se liší rozsahem teplot. Teplotní rozsah se pohybuje od -270 °C až po +2300 °C v závislosti na typu [15].

Termočlánky se používají zejména v průmyslu pro měření technologických procesů, ale najdeme je také např. u digitálních multimetrů – jako externí teplotní senzor.

Pro připojení k FITkitu by termočlánky vyžadovali nějaký převodník, protože jejich výstupní napětí je příliš nízké pro přímé napojení na vstupy A/D převodníku. Také tento převodník by navyšoval cenu už tak drahého termočlánku.

4 Návrh propojení HW

Tato kapitola se zabývá především konfigurací FPGA a hardwarovému připojení teplotních čidel k FITkitu. Po zvážení všech pro a proti jsem se rozhodl pro teplotní senzory typu SMT 160-30, jenž převádí teplotu na střídu výstupního signálu. Střídu lze jednoduše měřit pomocí čítačů implementovaných v FPGA.



Obr. 4.1 Princip měření střídy pomocí čítače

Na obr. 4.1 jsou zobrazeny časové průběhy dvou signálů – DATA_IN(0) představuje výstupní signál z teplotního převodníku (senzoru) a *clk* je hodinový vzorkovací signál o frekvenci 50 MHz.

Střída obdélníkového signálu (*DC – Duty Cycle*) je definována jako poměr doby T1 ku celkové době periody T. Signál DATA_IN(0) na obr. 4.1 má střídu 38 %. Teplotní čidlo SMT 160-30 podle katalogových údajů generuje výstupní signál o frekvenci 1 až 4 kHz. Když tento signál budeme vzorkovat hodinovým kmitočtem 50 MHz, dostaneme po dobu trvání jedné periody signálu DATA_IN(0) 12 500 až 50 000 impulsů signálu *clk*. Tento číselný rozsah lze uložit na 16 bitů.

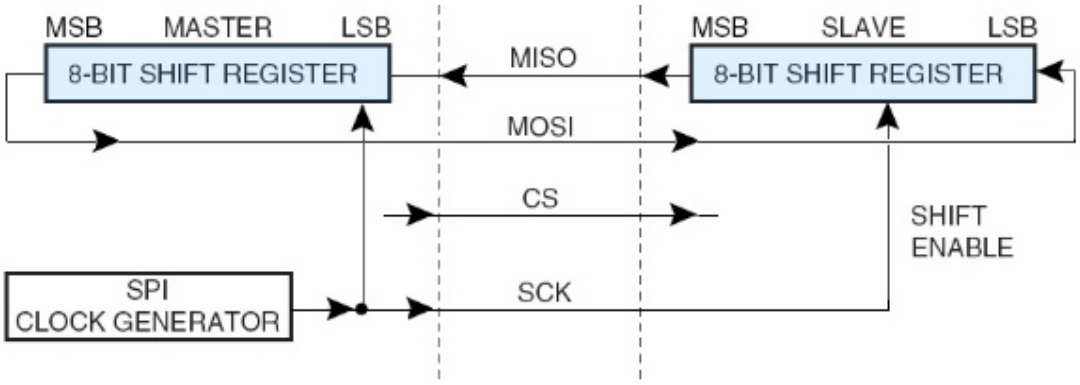
4.1 Blokové schéma

Na obr. 4.5 je znázorněno celkové blokové schéma aplikace. Architektura aplikace je založena na rozhraní `tlv_ide_ifc`, protože potřebujeme přistupovat přímo na vývody FPGA čipu, což rozhraní `tlv_bare_ifc` jednoduše neumožňuje. Než přejdeme k samotnému blokovému schématu, je potřeba ve stručnosti popsat komunikační rozhraní SPI a komponenty v FPGA.

4.1.1 Rozhraní SPI

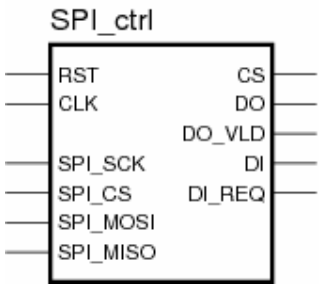
Všechna zařízení jsou prostřednictvím SPI dekodérů připojeny na vnitřní sériovou sběrnici, která je následně připojena na sběrnici SPI (*Synchronous Peripheral Interface*). SPI pracuje na principu vý-

měny obsahu dvou 8bitových registrů – viz obr. 4.2. Zápisem do posuvného registru dojde automaticky k zahájení přenosu. Změna signálu CS (*Chip Select*) do log. 0 značí zahájení přenosu. Současně probíhá zápis i čtení, které končí po osmi taktech hodinového signálu SCK, jehož generování zajišťuje master. Maximální frekvence signálu SCK je rovna 3,686 MHz, což dává maximální přenosovou rychlost 460,8 kB/s.

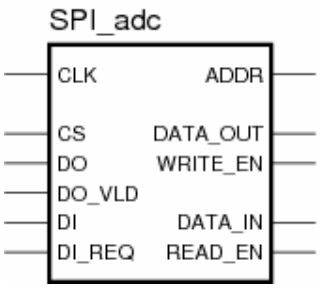


Obr. 4.2 Architektura SPI

MCU na FITkitu je zařízení typu master, FPGA, FLASH a ostatní připojená zařízení jsou typu slave. Na SPI sběrnici je připojen jeden SPI řadič (*SPI_ctrl*), který převádí SPI protokol na vnitřní sériovou sběrnici. Rozhraní řadiče je zobrazeno na obr. 4.3, na levé straně jsou signály rozhraní SPI a na pravé straně se nachází signály vnitřní sériové sběrnice. Na tuto sběrnici lze připojit téměř libovolný počet zařízení, nejčastěji se ale připojují SPI dekodéry – viz obr. 4.4.

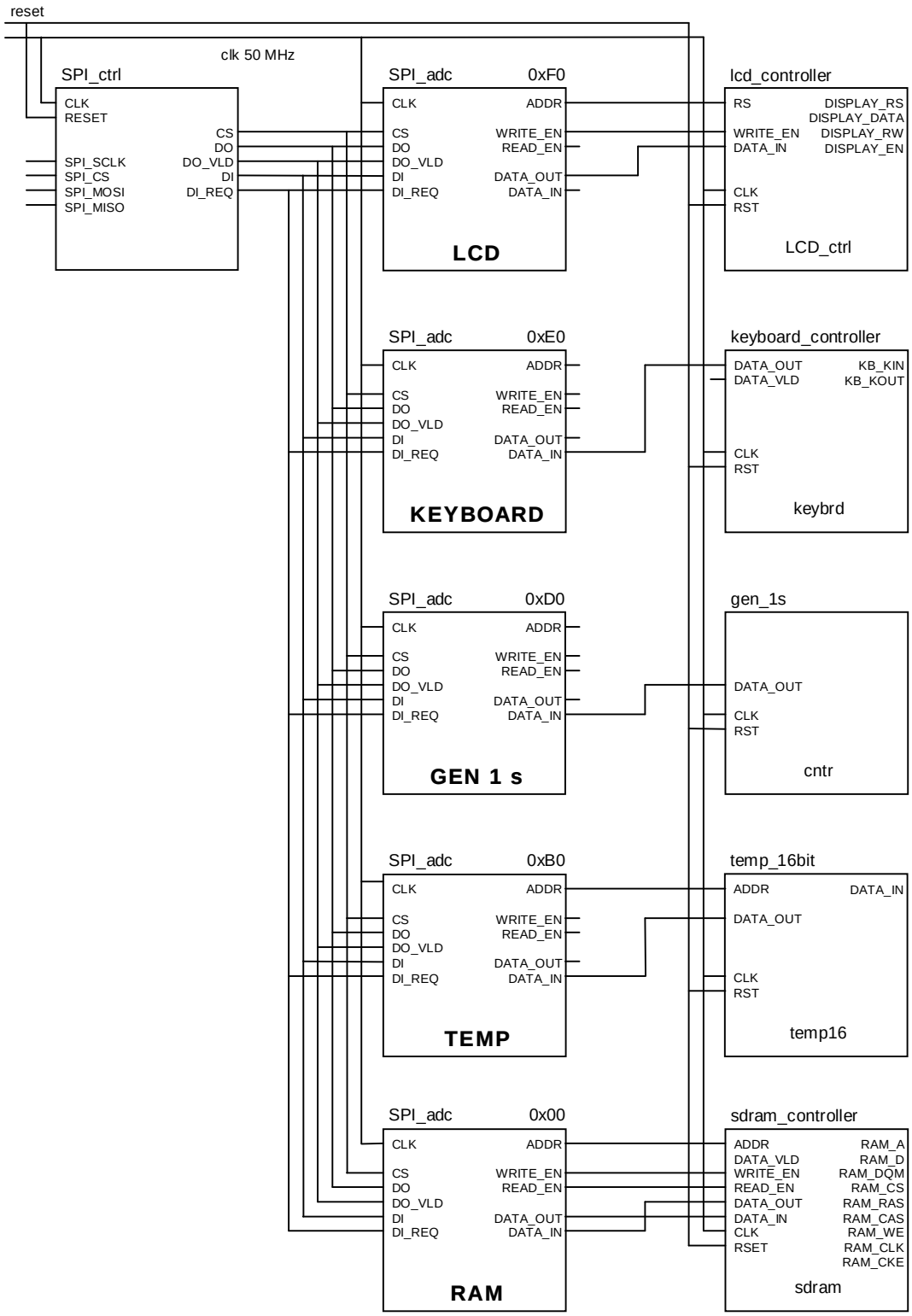


Obr. 4.3 Rozhraní SPI řadiče



Obr. 4.4 Rozhraní SPI dekodéru

SPI dekodér (*SPI_adc*) je synchronní komponenta, která převádí protokol vnitřní sériové sběrnice na jednoduché paralelní rozhraní. SPI dekodér má svoji bázovou adresu a zařízení k němu připojené lze taktéž adresovat pomocí signálu ADDR. Šířka adresy může být prakticky libovolná. Šířku dat lze nastavit také libovolně, ale je shodná pro vstup i výstup. Signál CS (*Chip Select*) aktivuje SPI dekodér jen tehdy, shoduje-li se začátek adresy s bázovou adresou SPI dekodéru. Signál WRITE_EN signalizuje, že na sběrnici DATA_OUT jsou platná data. Obdobný význam má bit READ_EN, ten je generován při požadavku čtení dat a v následujícím taktu budou do mikroprocesoru přenesena data ze sběrnice DATA_IN.



Obr. 4.5 Blokové schéma aplikace

4.1.2 LCD kontrolér

Řadič displeje jsem převzal z ukázkových aplikací „Řízení LCD Displeje z MCU“ a „Čítač zobrazující stav na displeji“. Knihovní funkce pro zobrazování textu na displeji jsou jednoduché a přehledné. Bylo však potřeba změnit bázovou adresu řadiče tak, aby nekolidovala s ostatními zařízeními. V ukázkových aplikacích jsou bázové adresy řadičů většinou nastavovány na `0x00`. V této aplikaci je více zařízení a jejich adresový prostor je nutné rozdělit na disjunktní oblasti. Proto jsem adresu změnil na `0xF0` a to je také potřeba také změnit v souboru …`\FITkit\sw\libs\`**`display.h`**:

```
#define BASE_ADDR_DISPLAY       0xF0
```

4.1.3 Řadič klávesnice

Zapojení řadiče klávesnice je též převzato z ukázkové aplikace …`\FITkit\apps\`**`app_key_int`**. Je použita varianta bez přerušení, protože mezi MCU a FPGA neexistuje žádné propojení, jenž by umožnilo přenést signál o přerušení. Toto propojení se dělá vždy externím kabelem. Opět je změněna bázová adresa řadiče klávesnice na adresu `0xE0`. Definice bázové adresy klávesnice se nachází v souboru `...\FITkit\sw\libs\`**`keyboard_4x4.h`**:

```
#define BASE_ADDR_KEYBOARD      0xE0
```

4.1.4 Generátor hodinového signálu 1 Hz

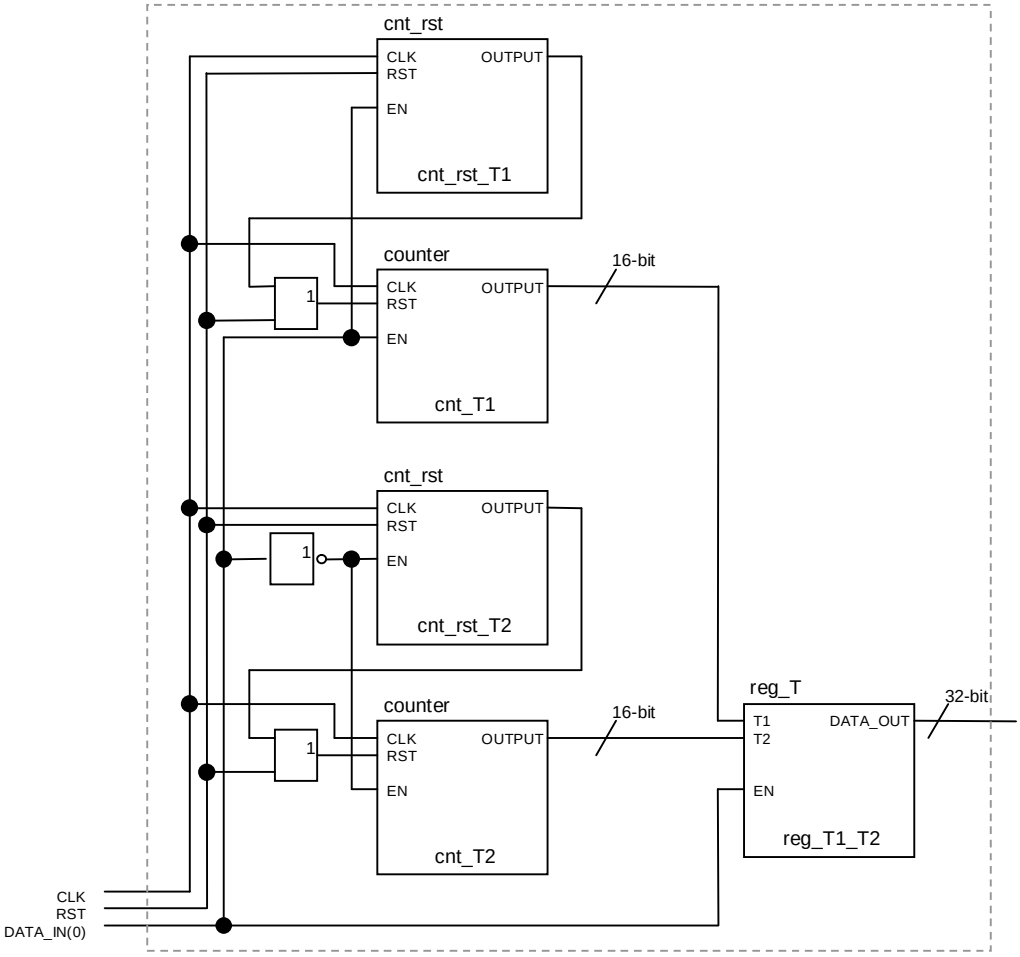
Tento generátor slouží jako výchozí hodinový signál pro zaznamenávání vzorků do paměti SDRAM. Frekvence výstupního signálu je 1 Hz. Zdrojový kód se nachází v souboru **`gen_1s.vhd`**.

Základem je čítač, jehož vstup je napojen na hodinový signál *clk* s frekvencí 50 MHz. Hodnota čítače se zvýší po každé náběžné hraně vstupního hodinového signálu *clk*. Po dosažení hodnoty přibližně 25 mil. impulsů se na všechny výstupní bity nastaví na úroveň log. 1. K nulování hodnoty čítače a výstupu dochází po načítání 50 mil. impulsů nebo signálem ***RST***. Tím se výstup mění v intervalu 1 s a jeho změnu je možné sledovat blikáním žluté LED diody D4 na FITkitu. Šířka výstupního signálu je 8 bitů, protože SPI dekodér přenáší data vždy po bajtech. Bázová adresa SPI dekodéru je nastavena na `0xD0`.

4.1.5 Řadič teplotních senzorů

Řadič teplotních čidel představuje jádro celé aplikace. Vnitřní blokové schéma pro jeden kanál je zobrazeno na obr. 4.6. Je složeno ze čtyř čítačů a dvojitého registru. Čítač `cnt_T1` počítá počet náběžných hran hodinového signálu (50 MHz) po dobu log. 1 na vstupu DATA_IN(0). Čítač `cnt_T2` naopak počítá náběžné hrany po dobu log. 0 na vstupu DATA_IN(0). Počty impulsů T1 a T2 se na základě změny vstupního signálu DATA_IN(0) přenesou do příslušné části registru `reg_T1_T2`.

V registru `reg_T1_T2` jsou počty impulsů pořád uloženy a k aktualizaci hodnot dojde při změně vstupního signálu DATA_IN(0) – na sestupnou hranu se aktualizuje hodnota registru T1, na vzestupnou hranu zase hodnota registru T2. Výstup registru je 32bitový, bity 31:16 jsou vyhrazeny pro T2 a bity 15:0 jsou vyhrazeny pro T1.



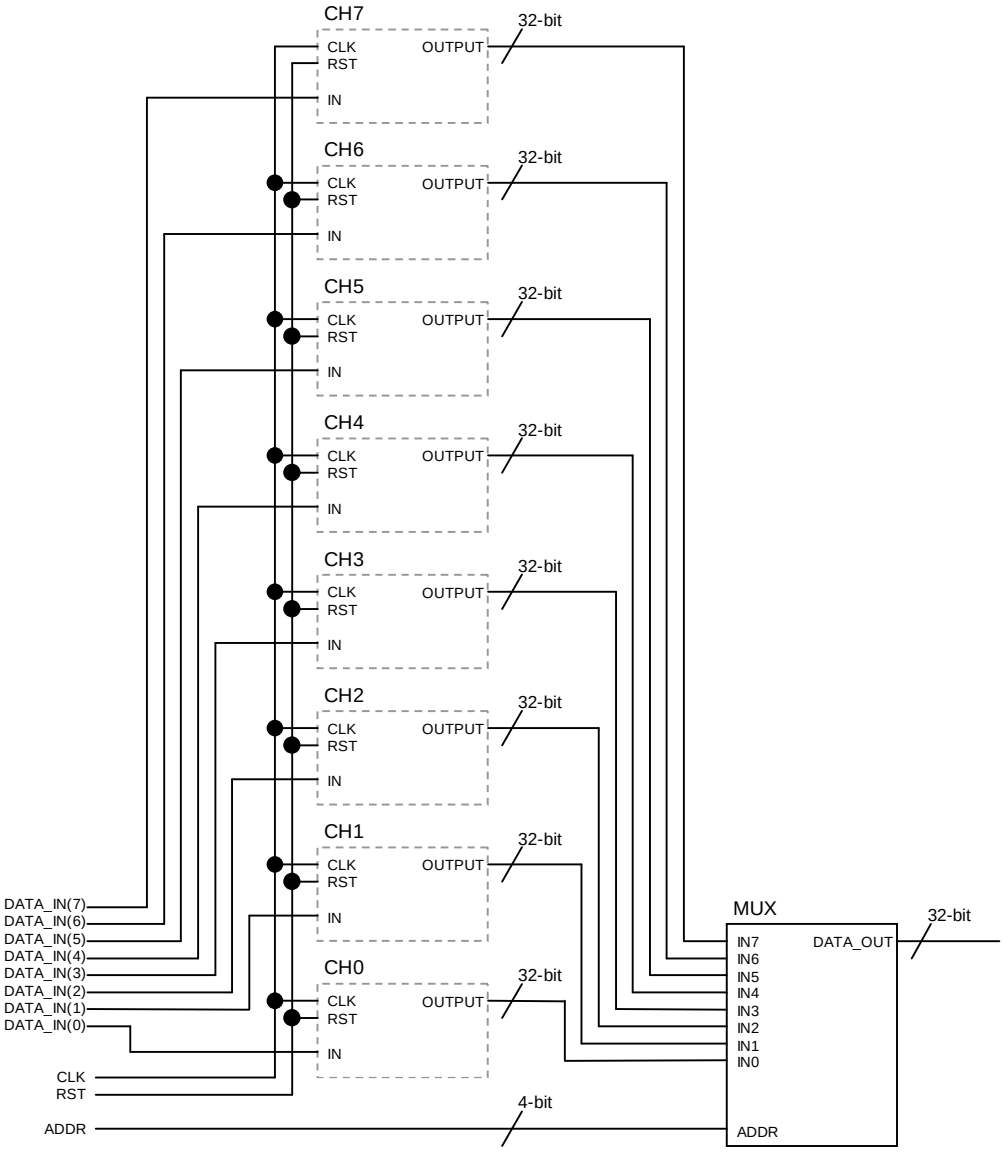
Obr. 4.6 Blokové schéma jednoho kanálu

Čítače `cnt_rst_T1`, resp. `cnt_rst_T2` slouží k nulování čítače `cnt_T1`, resp. `cnt_T2`. Jedná se 3bitové čítače, jejichž výstup se po dosažení hodnoty 7 uvede do stavu log. 1. Čítače by bylo možné nahradit nějakým zpožďovacím členem, ale ten se mně nepodařilo implementovat v jazyce VHDL tak, aniž by překladač nehlásil chybu při generování konfigurace. Čítače jsou kromě resetu (RST) nulovány také log. 1 na vstupu EN.

Obr. 4.7 znázorňuje celkové vnitřní zapojení komponenty `temp_16bit`. Jednotlivé bloky CH0 až CH7 představují zapojení z obr. 4.6. Pomocí nejnižších tří bitů signálu ADDR se adresuje číslo kanálu. Nejvyšší bit je rezervován pro rozšíření na 16 kanálů. Počet bitů v názvu komponenty může být zavádějící, protože výstup je 32bitový. 16 bitů značí délku jednoho ze dvou registrů (T1 a T2), což dohromady dává právě 32 bitů. Komponenta má 8 vstupních signálů pro připojení senzorů,

rozšíření na více jak 16 vstupů už představuje větší zásah do celé konfigurace. Musela by se změnit šířka adresní sběrnice z SPI dekodéru a také by se musely kompletně změnit adresy SPI dekodérů. Pak by bylo teoreticky možné rozšířit počet čidel až na 46 (což je počet uživatelských vstupů/výstupů na konektoru JP10).

Komponenta `tem_16bit` je v jazyce VHDL zapsána genericky jako 8prvkové pole komponent uvedených na obr. 4.6. Z tohoto hlediska je rozšíření na více kanálů poměrně snadné.

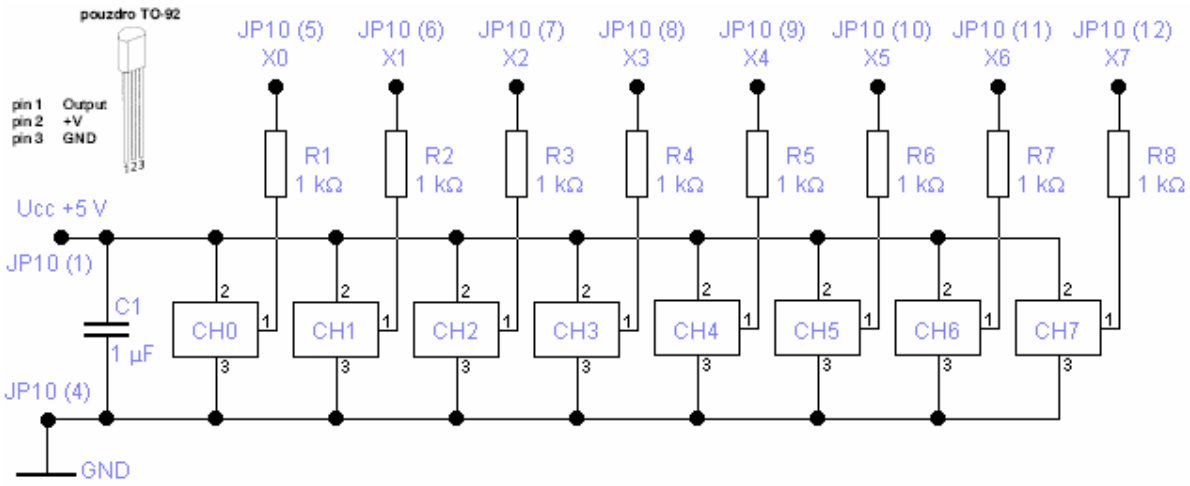


Obr. 4.7 Vnitřní blokové zapojení komponenty temp_16bit

4.1.5.1 Připojení senzorů k FITkitu

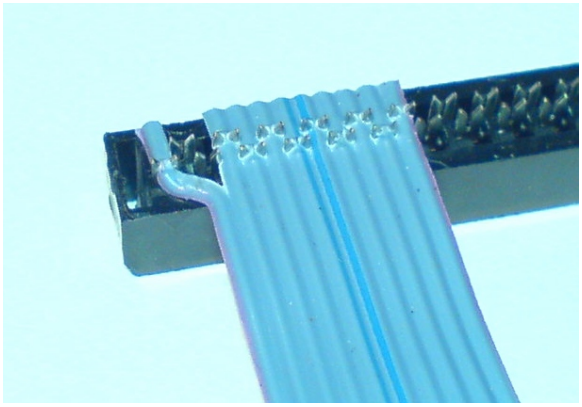
Pro připojení teplotních senzorů typu SMT160-30 jsem se rozhodl využít rozhraní X (ve VHDL označováno jako IDE), protože je přímo spojené s vstupně-výstupními vývody FPGA. Na desce FITkitu je to konektor JP10. Při návrhu jsem se zaměřil na tu část konektoru, kde je k dispozici napájecí napětí +5 V, které teplotní senzor vyžaduje. Také jsem zohlednil to, aby vybrané signály X0…X7 nebyly sdílené s jinými zařízeními na FITkitu, takže signál ze senzoru nebude nic ovlivňovat. Připojení senzorů je zobrazeno na obr. 4.8.

Odpory R1 až R8 jsou zapojeny kvůli přizpůsobení napěťových úrovní, protože teplotní čidlo má výstupní napětí 5 V a FPGA je napájeno pouze na 3,3 V. Hodnota 1 kΩ byla zvolena experimentálně a mohla by být vyšší. Spolehlivě funguje i 10 kΩ. Poměrně důležitý je filtrační kondenzátor C1. V případě, že čidla budou umístěna na dlouhém vedení, měl by být filtrační kondenzátor umístěn co nejblíže pouzdru senzoru a jeho kapacita by měla být asi 100 nF (pro každé čidlo). Kapacitu filtračního kondenzátoru nelze příliš zvyšovat, protože příliš vysoká kapacita způsobuje přetížení napájecího systému po zapnutí FITkitu a FITkit odmítá naběhnout. Tato situace nastala při kapacitě 100 µF. Pokud senzor na dlouhém vedení nebude mít filtrační kondenzátor, bude hodnota teploty prudce kolísat a v některých případech může ukazovat naprosto nesmyslné informace.



Obr. 4.8 Připojení teplotních senzorů na konektor JP10

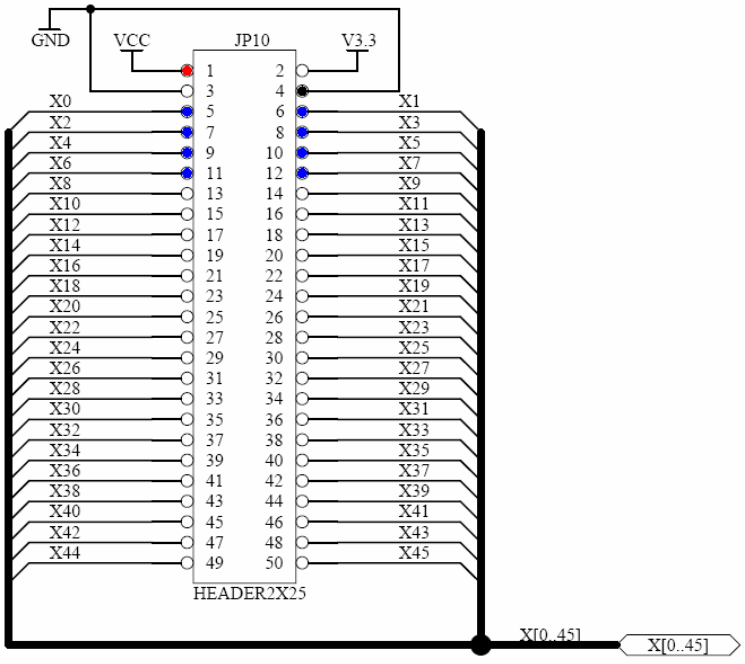
8 teplotních senzorů lze k FITkitu připojit plochým 10žilovým kabelem. Vyžaduje to však malou úpravu v lisování kabelu na konektor viz obr. 4.9. První vodič zleva oddělíme od zbývajících vodičů a zapojíme na pin 1 konektoru (Vcc +5 V). Další dvě pozice vynecháme (+3,3 V a GND), poté následuje zem (GND) a datové vodiče. Smysl to má v tom, že na druhém konci kabelu můžeme použít standardní konektor 2 x 5 pinů a signály jsou uspořádány v pořadí 1 – V_{cc}, 2 – GND, 3 – CH0, 4 – CH1, …, 10 – CH7. Také nám stačí 10žilový kabel místo 12žilového.



Obr. 4.9 Úprava lisování 10žilového kabelu na konektor

Konektor, jenž se připojuje k FITkitu na JP10, může být zkrácen z delšího konektoru (např. 2 x 7 nebo 2 x 8 pinů). Standardní, o něco delší konektory nelze nasadit, protože jejich konce jsou příliš široké. Mohli bychom také použít konektor 2 x 25 pinů, ale ten by byl využit jen z velmi malé části a hlavně při manipulaci vyžaduje daleko větší sílu a mohlo by snadno dojít k ohnutí pinů na FITkitu.

Na obr. 4.10 jsou modrou barvou zvýrazněny signály od teplotních senzorů, červeně je označeno kladné napájecí napětí a černá barva značí nulový potenciál – zem. Ostatní vývody nejsou použity.



Obr. 4.10 Znázornění obsazení konektoru JP10

4.1.6 SDRAM

O řízení přístupu k paměti SDRAM na FITkitu se stará řadič *sdram_controller* připojený k SPI dekodéru. SDRAM vyžaduje synchronizaci hodinovým kmitočtem 50 MHz. A právě kvůli synchronizaci této komponenty se musely všechny zbývající zařízení podřídit. Největší problém změny hodinového signálu z 7,37 MHz na 50 MHz dělá řadiči klávesnice.

SPI dekodér pro paměť SDRAM má největší šířku adresy – 32 bitů. Ale šířka přenášených dat je pouze 8 bitů. Také zpoždění mezi požadavkem dat a jeho vystavením na sběrnici musí být z důvodu spolehlivosti nastaveno na 20 taktů.

4.2 Přesnost hodinového signálu

Přesnost hodinového signálu závisí na kvalitě krystalu, tak především na stabilitě napájecího napětí. Při napájení FITkitu ze sběrnice USB napájecí napětí často kolísá v důsledku proměnlivého zatížení počítače.

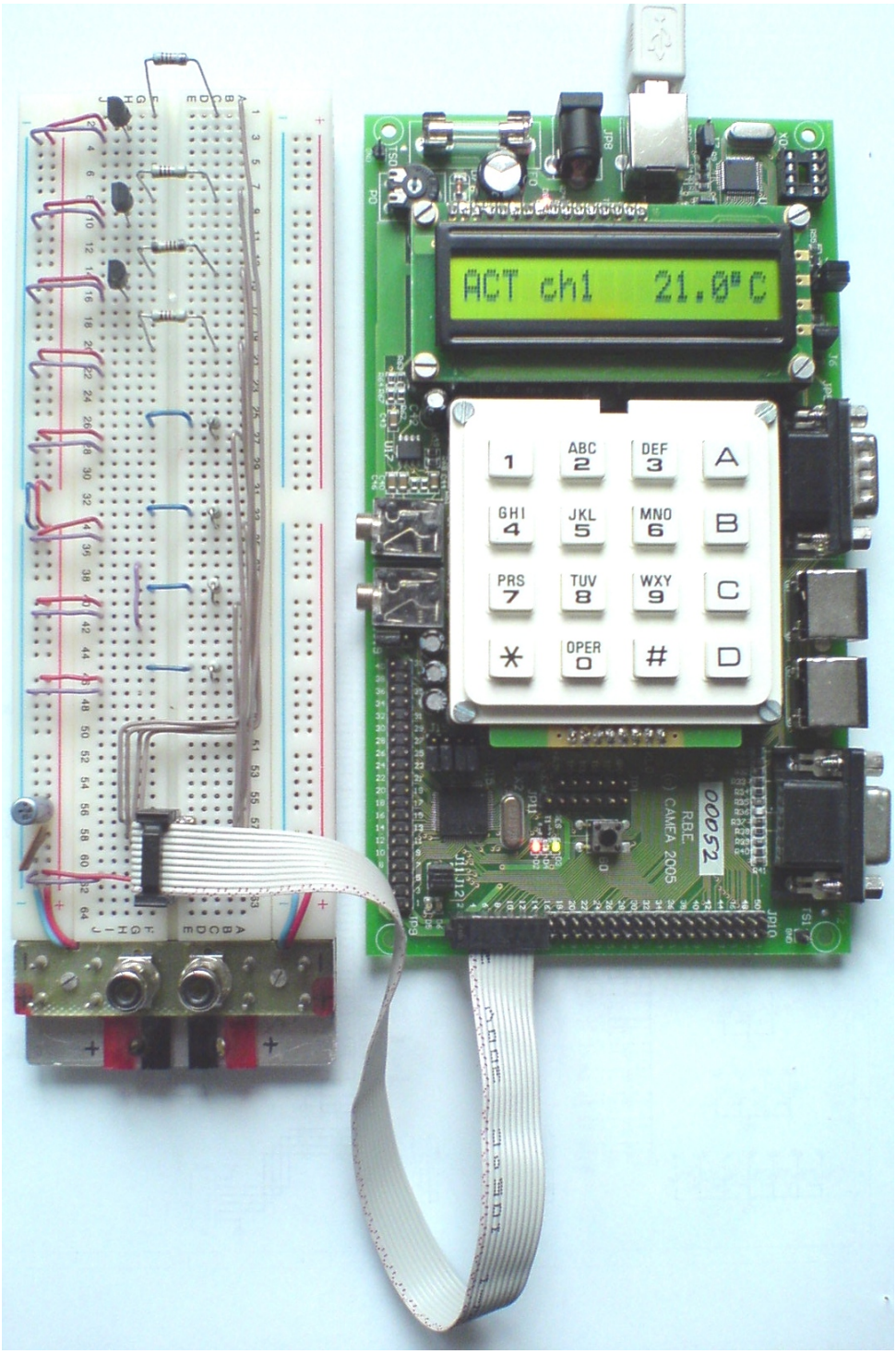
Při testování jsem zjistil, že počet uložených vzorků se mírně liší proti teoretickému výpočtu za dané časové období. Poté mě napadla myšlenka ověřit přesnost hodinového signálu, jestli generátor hodinového kmitočtu 50 MHz generuje skutečně přesně 50 MHz. K měření takovéto frekvence však nemám potřebné vybavení. Když ale tento signál vydělíme dostatečně velkým číslem, získáme velmi nízký kmitočet, jehož periodu lze změřit běžně dostupnými přístroji. K tomuto účelu jsem použil upravené elektronické stopky s rozlišením jedné setiny sekundy a měřil jsem délku periody v řádech několika desítek minut. Měření jsem několikrát opakoval a pokaždé vyšla jiná hodnota. Aritmetickým průměrem jsem stanovil dělicí koeficient na hodnotu 49 770 680, tak aby odchylka od frekvence 1 Hz byla minimální a zanedbatelná. Ovšem tato konstanta se vztahuje pro FITkit se sériovým číslem 00052 a na jiných deskách se může lišit. Také se může frekvence lišit v závislosti na velikosti napájecího napětí a provozní teploty.

Přesnost hodinového signálu nelze podceňovat, protože zařízení je navrženo pro dlouhodobý provoz a jediná možnost jak zjistit datum a čas uloženého vzorku, je z násobku vzorkovací periody a čísla vzorku. Osobně si netroufám odhadovat, jaký bude rozdíl času po dvou letech zaznamenávání.

Po uplynutí testovacího období 30 dní se hodiny opozdily o 30 sekund, což představuje odchylku menší jak $1{,}16 \cdot 10^{-3}$ %. FITkit byl napájen převážně externím adaptérem s napětím 4,64 V a průměrná teplota okolí se pohybovala okolo 22,5 °C.

4.3 Fyzická realizace zapojení

Teplotní senzory jsem se rozhodl zapojit na nepájivém kontaktním poli, protože takové zapojení má spoustu výhod. Nepájivé kontaktní pole je určeno pro experimentální zapojení obvodů, což tento případ bezesporu je. Umožňuje snadnou a rychlou modifikaci zapojení. Výsledek je na obr. 4.11.



Obr. 4.11 Zapojení senzorů na nepájivém kontaktním poli a připojení na FITkit

5 Návrh řešení a komunikace

5.1 Princip ukládání teploty

Teplota se bude měřit v rozsahu -45 °C až +130 °C s rozlišením 0,1 °C. Vetší rozlišení se u teploměrů nepoužívá a digitální teploměry se kterými jsem se setkal měly rozlišení 0,1 °C nebo 0,5 °C. Naměřenou teplotu dostáváme ve formě desetinného čísla v poměrně malém rozsahu hodnot. Nastává otázka, zda toto desetinné číslo ukládat jako 32-bitové číslo v plovoucí řádové čárce podle standardu IEEE nebo zvolit nějaký jiný, úspornější datový typ.

MCU má implementovanou podporu pro práci s čísly v plovoucí řádové čárce, avšak 32 bitů pro uložení jednoho vzorku teploty v daném teplotním rozsahu je zbytečné plýtvání pamětí a dynamický rozsah datového typu FLOAT nebude zdaleka využit. Navíc veškeré operace s tímto datovým typem jsou časově mnohem náročnější než operace s celočíselnými datovými typy.

Když bychom chtěli použít celočíselný datový typ pro uložení naměřené hodnoty teploty, musíme nějak převést číslo v plovoucí řádové čárce na celé číslo. Protože uvažuji rozlišení pouze 0,1 °C, za desetinou čárkou bude pouze jedna významová číslice. Jestliže naměřenou hodnotu vynásobíme 10krát a vezmeme pouze celou část před desetinnou čárkou, získáme celočíselnou hodnotu teploty bez ztráty přesnosti. Příklad převodu teploty je uveden v rov. 5.1.

$$21{,}7°C * 10 = 271$$

Rov. 5.1 Převod teploty z desetinného čísla na celé číslo

5.1.1 Volba datového typu

Nyní, když už máme desetinné číslo převedeno na celé, musíme zvolit vhodný datový typ pro uložení v paměti. Uvažujme mírně rozšířený teplotní rozsah -50 °C až +150 °C, celkově tedy 200 hodnot. Po vynásobení 10 dostáváme 2000 hodnot (rozsah -500 až +1500). Na uložení takového množství hodnot je potřeba 11 bitů, $2^{11} = 2048$.

Avšak MCU nepodporuje žádný datový typ o šířce 11 bitů, musí se zvolit nejbližší vyšší, tedy 16 bitů. Teplota může být i záporná, takže jako nejvhodnější datový typ se jeví **integer** na 16 bitech (**int**) s rozsahem -32768…+32767. V závislosti na typu překladače může být tento datový typ označován také jako **short**.

5.1.2 Rozvržení paměti SDRAM

Na FITkitu je osazen modul paměti SDRAM o velikosti 8 MB. Když tuto paměť podělíme počtem připojitelných teplotních čidel, dostáváme přesně 1 MB/kanál. Paměť však lze rozdělit také podle

počtu aktuálně připojených teplotních čidel po zapnutí FITkitu. O problematice rozdělení paměti pojednávají následující dvě podkapitoly.

5.1.2.1 Statické rozdělení paměti SDRAM

Celá paměť se rozdělí na *n* stejně velkých oblastí, kde *n* je počet kanálů. Počet kanálů musí být znám již v době překladu. Statické rozdělení paměti je programově jednoduší a přehlednější, ale přináší sebou jednu podstatnou nevýhodu – paměť nepřipojených teplotních čidel zůstává nevyužita.

Na druhou stranu statické rozdělení umožňuje připojit teplotní čidla kdykoliv během měřeného období bez nutnosti nulování dosud naměřených hodnot. Např. v případě nutnosti výměny nebo přeložky kabeláže k čidlu ztratíme jen pár naměřených hodnot, které ve výsledku můžeme aproximovat průměrnou hodnotu před výpadkem měření a po něm.

Paměť je nejvhodnější podělit mocninou o základu 2 – tím dostaneme optimální rozdělení a paměť bude plně využitelná. Pro adresování takto rozdělené paměti lze s výhodou využít instrukci **and** s patřičnou binární maskou. To také sníží počet možných chyb v kódu a případné přetečení adresy nebude ovlivňovat data ostatních kanálů.

5.1.2.2 Statické rozdělení paměti SDRAM s volbou počtu kanálů

Toto rozdělení je obdobné jako předchozí, ale počet kanálů je možné měnit za běhu aplikace (není potřeba celou aplikaci znovu překládat). Změna počtu kanálů a tím také změna rozložení paměti však způsobí vynulování dosud naměřených hodnot. Počet kanálů by měl být opět mocninou čísla 2, jinak musíme všechny výsledky zaokrouhlovat směrem dolů.

Počet kanálů by se mohl automaticky zvolit ihned po zapnutí FITkitu. V návrhu se však nepočítá s žádným mechanismem, který by detekoval připojené nebo nepřipojené čidlo. Jediná možnost detekce čidla je připojení zátěžového odporu mezi vstup a zem. Tento odpor by musel mít velkou hodnotu, aby příliš nezatěžoval výstup z teplotního senzoru. Poté by detekce nepřipojeného čidla spočívala v tom, že vypočtená teplota by byla mimo nominální rozsah senzoru.

5.1.3 Výpočet maximální doby záznamu

Uvažujme statické rozdělení paměti na 8 kanálů, tedy 1 MB/kanál a 16 bitů na vzorek (2 bajty). Maximální počet zaznamenaných vzorků se podle rov. 5.2 rovná $2^{19} = 524288$.

$$n = \frac{\frac{8MB}{8ch}}{2B} = \frac{\frac{2^{23}}{2^3}}{2} = \frac{2^{20}}{2} = \underline{\underline{2^{19}}} = \underline{\underline{524288}}$$

Rov. 5.2 Výpočet maximálního počtu vzorků na kanál

Všechny dosud uvedené hodnoty jsou konstantní a doba záznamu závisí jen na délce vzorkovací periody. Platí zde přímá úměra – čím delší bude vzorkovací interval, tím delší bude doba záznamu. V tab. 5.1 je uveden vzorkovací interval a k němu příslušná maximální doba záznamu v závislosti na

počtu připojitelných senzorů. Vzorkovací interval byl zvolen tak, aby bylo možné zaznamenávat jak rychlejší změny teploty, tak i ty pomalé.

Vzorkovací perioda	Maximální doba záznamu											
	1 čidlo		2 čidla		4 čidla		8 čidel		16 čidel		32 čidel	
	4194304	vzorků/kanál	2097152	vzorků/kanál	1048576	vzorků/kanál	524288	vzorků/kanál	262144	vzorků/kanál	131072	vzorků/kanál
1 sek.	49	[dny]	24	[dny]	12	[dny]	6	[dny]	3	[dny]	1,5	[dny]
2 sek.	97		49		24		12		6		3,0	
3 sek.	146		73		36		18		9		4,6	
5 sek.	243		121		61		30		15		7,6	
10 sek.	1,3	[roky]	243		121		61		30		15	
12 sek.	1,6		291		146		73		36		18	
15 sek.	2,0		364		182		91		46		23	
20 sek.	2,7		1,3	[roky]	243		121		61		30	
30 sek.	4,0		2		364		182		91		46	
45 sek.	6,0		3		1,5	[roky]	273		137		68	
1 min.	8,0		4		2		364		182		91	
2 min.	16		8		4		2	[roky]	364		182	
3 min.	24		12		6		3		1,5	[roky]	273	
4 min.	32		16		8		4		2,0		364	
5 min.	40		20	[roky]	10		5		2,5		1,2	
10 min.	80		40		20	[roky]	10		5,0		2,5	
12 min.	96		48		24		12	[roky]	6,0		3,0	
15 min.	120		60		30		15		7,5	[roky]	3,7	
20 min.	159		80		40		20		10,0		5,0	[roky]
25 min.	199		100		50		25		12,5		6,2	
30 min.	239		120		60		30		15,0		7,5	
45 min.	359		179		90		45		22,4		11,2	
1 hod.	478		239		120		60		29,9		15,0	

Tab. 5.1 Max. doba záznamu v závislosti na vzorkovací periodě při statickém rozdělení paměti

Kratšího intervalu jak jedna sekunda by bylo jen těžko dosažitelné, protože není použito přerušení (od hodinového signálu) a také výpočty a uložení dat trvá nezanedbatelnou dobu. Navíc by desetinné číslo, vyjadřující periodu, způsobilo komplikaci při kódování intervalu, poněvadž interval je zadáván přímo v sekundách jako celé číslo bez znaménka – typ **`WORD`**.

Osobně se domnívám, že velikost paměti je dostačující a vzorkovací perioda dostatečně krátká i pro náročnější měření. Nejdelší možný interval, který lze zadat, je omezen rozsahem datového typu **`WORD`**, tedy 65535 s, což představuje přibližně 18 hodin a 12 minut.

5.1.4 Možnosti prodloužení max. doby záznamu

Může se stát, že doba záznamu nebude dostačující např. vlivem zvýšení počtu připojených senzorů na 16 nebo 32. I v této situaci je ponechán relativně velký prostor pro další úspory paměti a navýšení maximální doby záznamu. Jednak můžeme paměť uspořit změnou kódování nebo zavést kompresi hodnot, příp. kombinace obojího.

5.1.4.1 Změna kódování

Změnou kódování bychom mohli ušetřit ¼ paměti nebo i více, pokud by se ukládalo pouze 11 bitů z původních 16 bitů/vzorek. Jestliže budeme uvažovat, že hodnotu teploty vynásobenou 10krát uložíme na 12ti bitech – nejvyšší bit bude představovat znaménko a zbývajících 11 bitů bude představovat teplotu, pak úspora paměti činní právě 25 % (4 bity z 16 zůstanou volné).

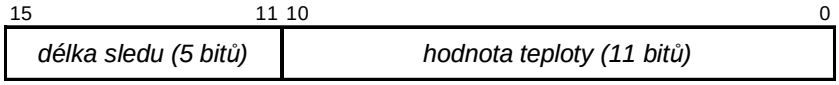
Do paměti SDRAM se musí zapisovat vždy dva po sobě jdoucí vzorky, aby velikost zapisované paměti byla dělitelná 8 (*12+12 bitů = 24 bitů; 24 bitů / 8 bitů = 3 bajty*).

Při ukládání jen 11 bitů by úspora paměti činila 31,25 %, ale museli bychom se vypořádat se zakódováním znaménka do hodnoty. To lze provést tak, že se celý teplotní rozsah -50..150 °C posune o 50 °C nahoru, tedy na 0..200 °C. Při dekódování zase odečteme 50 °C, abychom dostali původní rozsah. Mnohem větší komplikace však způsobí zápis hodnot do paměti SDRAM. Pro zápis je potřeba 8 po sobě jdoucích vzorků, aby celková zapisovaná oblast paměti byla dělitelná 8 (*8*11 bitů = 88 bitů; 88 bitů/8 bitů = 11 bajtů*). Práce s takto velkou strukturou by znamenala složitější programový kód, než v při kódování na 12 bitů.

5.1.4.2 Kódování délek sledů – RLE

Metoda kódování délek sledů nebo také *Run Length Encoding* spočívá v tom, že pokud se vyskytuje za sebou *n* naprosto stejných hodnot *t*, do výstupního toku se zapíše dvojice *nt*; *n* se pak nazývá délka sledu [11]. V tab. 5.2 je názorná ukázka vhodných (vlevo) a nevhodných (vpravo) dat pro tuto kompresní metodu.

V našem případě může být obzvlášť výhodné, že hodnotu teploty můžeme zakódovat pouze na 11 bitů (viz podkapitola 5.1.4.1). V 16-ti bitovém slově pak zbývá horních 5 bitů pro uložení délky sledu, jak naznačuje obr. 5.1. Při takovémto modelu rozložení bitů nebude kompresní poměr nikdy větší než 1, tzn. že téměř vždy dojde ke kompresi. Jediný případ, kdy by ke kompresi nemuselo dojít, je ten, že sousední vzorky teploty se budou vždy lišit v minimálně jednom bitu. Takový případ může nastat např. vlivem elektromagnetického rušení, kdy se neustále mění nejméně významný bit.



Obr. 5.1 Rozložení bitů pro kompresní metodu RLE

Na 5 bitech, vyhrazených pro délku sledu, můžeme zakódovat maximálně 64 po sobě jdoucích stejných vzorků. Sled délky 0 totiž lze považovat za sled délky 1, protože sled délky nula nemá smysl. I tak se domnívám, že sled o maximální délce 64 vzorků v praxi ani nenastane a zpravidla bude kratší.

Teplota	Délka sledu	Hodnota teploty
26,1 °C	12	26,1 °C
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,0 °C	3	26,0 °C
26,0 °C		
26,0 °C		
26,1 °C	1	26,1 °C
26,0 °C	2	26,0 °C
26,0 °C		
26,1 °C	6	26,1 °C
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		
26,1 °C		

Kompresní poměr $= \frac{5}{24} =$ 20,83 %

Teplota	Délka sledu	Hodnota teploty
24,0 °C	1	24,0 °C
24,1 °C	1	24,1 °C
24,2 °C	2	24,2 °C
24,2 °C		
24,3 °C	1	24,3 °C
24,1 °C	1	24,1 °C
24,2 °C	1	24,2 °C
24,1 °C	1	24,1 °C
24,0 °C	3	24,0 °C
24,0 °C		
24,0 °C		
24,1 °C	2	24,1 °C
24,1 °C		
24,2 °C	2	24,2 °C
24,2 °C		
24,0 °C	1	24,0 °C
24,1 °C	3	24,1 °C
24,1 °C		
24,1 °C		
23,9 °C	1	23,9 °C
24,0 °C	1	24,0 °C
24,1 °C	1	24,1 °C
24,0 °C	1	24,0 °C
23,9 °C	1	23,9 °C

Kompresní poměr $= \frac{17}{24} =$ 70,83 %

Tab. 5.2 Různá data pro kompresní metodu RLE

Kompresní metoda kódování délek sledů může dávat velmi dobrý kompresní poměr, když se bude teplota měnit jen velmi pozvolna nebo vůbec a jednotlivé vzorku nebudou zatíženy chybou nejméně významného bitu.

5.1.4.3 Diferenciální kódování

Diferenciální kódování (nebo také relativní kódování) je výhodné tehdy, když se sousední hodnoty (vzorky) od sebe výrazně neliší. Menší změna se může uložit na menší počet bitů než velké rozdíly mezi vzorky. Právě v případě měření teploty jsou si sousední hodnoty velmi podobné a tato kompresní metoda může dávat dobrý kompresní poměr. Počáteční vzorek se musí uložit na plný počet bitů, vzorky následující za ním už stačí ukládat pouze kladný nebo záporný rozdíl od předchozího vzorku. Dekodér ale musí nějak rozeznat, zda jde o hodnotu uloženou v plném formátu nebo zda jde o diferenci. To se obvykle řeší příznakovým bitem, který tyto dva stavy rozlišuje.

Počet bitů, na které by se měl ukládat rozdíl teplot je dobré zvolit v závislosti na délce intervalu měření a na velikosti změn teploty mezi sousedními vzorky. Při krátké vzorkovací periodě lze očekávat i menší rozdíly mezi sousedními vzorky – dají se tedy zakódovat na menší počet bitů. Při prudké změně teploty by se tato změna mohla rozložit na několik málo (např. 2-3) maximálních nebo mini-

málních diferencí a mohlo by to ušetřit zápis vzorku na plný počet bitů. Takovou úpravu můžeme uvažovat jen za předpokladu, že nikomu nebude vadit, že změna teploty ve výsledku nebude tak razantní jako ve skutečnosti.

5.2 Komunikační protokol

Jedním z požadavků zadaní je funkce stahování naměřených dat z FITkitu do PC. K tomuto účelu je nutné vytvořit obousměrný komunikační protokol. Sice se může zdát, že data půjdou pouze jedním směrem – z FITkitu do PC, ale musíme mít možnost nějak změnit vzorkovací periodu nebo definovat začátek měření. K těmto a dalším činnostem potřebujeme opačný směr – z PC do FITkitu.

Zařízení mezi sebou budou komunikovat po sériovém rozhraní na principu klient-server. Klient bude v tomto případě PC a FITkit bude server. Někomu se může zdát zvolení FITkitu jako serveru nepochopitelné, ale má to svůj účel. Server (FITkit) bude pasivně čekat na požadavky od klienta a na jejich základě bude posílat odpovědi. Klient (PC) bude aktivně posílat požadavky a čekat na odpovědi ze serveru.

5.2.1 Návrh komunikačního protokolu

Veškerá data se budou přenášet po sériové lince RS-232. Tímto standardem se nejčastěji přenášejí data o velikosti 8 bitů. Potřebujeme však přenášet data daleko většího objemu než je 8 bitů. Musíme také rozlišit, zda jde o příkaz nebo samotná data. Nabízí se dvě možnosti přenosu dat a příkazů: textově a binárně.

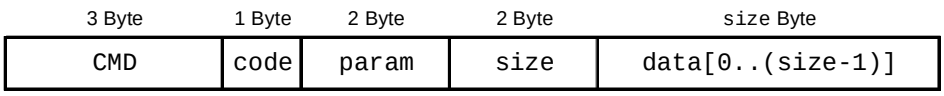
Textový komunikační protokol se může zdát na první pohled jednodušší – nějaké znaky by uvozovaly příkaz a jiné znaky zase samotná data. Začátek a konec příkazu by byl označen speciální sekvencí znaků. Přenášená teplota by musela být v textové reprezentaci a zabírala by tak mnohem větší objem přenesených dat než forma binární. Právě s ohledem na množství přenášených dat a přenosovou rychlost komunikačního kanálu, která není nijak závratná, jsem se přiklonil k binárnímu přenosu dat.

Binární komunikační protokol představuje menší objem přenesených dat a jeho režie je podstatně nižší než u textově orientovaného přenosu. Opět musíme rozlišit příkaz a data. Můžeme to udělat také tak, že příkaz se bude posílat vždy a data budou následovat hned za ním. Velikost dat bude poznačena v nějakém parametru příkazu a samotná data budou volitelnou částí příkazu.

5.2.2 Struktura příkazu

Struktura příkazu je vidět na obr. 5.2. Základní velikost příkazu je 8 bajtů. Obsahuje-li příkaz data, pak je celková velikost zvětšena právě o velikost přenášených dat. Data se přenášejí pouze ve směru

FITkit **à** PC, v opačném směru není přenos dat potřeba; resp. data se dají vložit přímo do příkazu (pole `param`).



Obr. 5.2 Formát příkazu

Význam jednotlivých polí:

- **`CMD`**: toto pole jednoznačně určuje, zda jde o začátek příkazu. Je to identifikátor příkazu a má vždy konstantní hodnotu „`CMD`“ (*command*).
- **`code`**: určuje kód příkazu.
- **`param`**: parametr příkazu, nejčastěji se zde přenáší číslo kanálu.
- **`size`**: udává velikost přenášených dat v bajtech následujících hned za příkazem. Je-li hodnota pole nulová – žádná data se nepřenáší.
- **`data`**: volitelná část příkazu – obsahuje přenášená data. Velikost dat musí odpovídat hodnotě v poli `size`.

Velikost příkazu je záměrně zvolena jako mocnina čísla 2, aby se do vyrovnávacích bufferů sériového rozhraní vlezl celistvý počet příkazů. Počet typů příkazů může být až 256, ale využita je jen malá podmnožina. Strukturu jsem se snažil navrhnout s ohledem na jednoduchost, přehlednost a účelně k požadavkům zadaní. Také je počítáno s tím, že FITkit bude možné ovládat i z jiného SW než ten, který je přiložen.

Délka příkazu ve směru PC **à** FITkit by mohla být zkrácena z 8 bajtů na 6 bajtů vynecháním pole **`size`**. Toto pole má trvale nulovou hodnotu, protože data se ve směru PC **à** FITkit neposílají. Ovšem z důvodu zachování kompatibility zdrojových kódu, jsem toto pole ponechal – může být využito pro budoucí použití. Obslužné funkce pro posílání a přijímání příkazů jsou pro MCU a PC velmi podobné a zkrácením příkazu na 6 bajtů by vznikly další komplikace.

Jelikož se v příkazu vyskytují pole o velikosti větší než jeden bajt, je nutné se zabývat uspořádáním dat v paměti a pořadím odesílání jednotlivých bajtů. Data v paměti budou uložena v kódování Little endian (nejméně významný bajt se ukládá na nejnižší adresu paměti) – typické pro platformu Intel x86.

Pořadí odesílání dat:

- první se odesílá nejvíce významný bajt slova (**`short`**, **`WORD`**, **`DWORD`**)
- naposled se posílá nejméně významný bajt slova (**`short`**, **`WORD`**, **`DWORD`**)
- pole dat (pouze ve směru FITkit **à** PC) – první se odesílá bajt s nejnižším indexem

Pořadí odesílání jednotlivých bajtů není nutné příliš řešit, pokud se pro posílání příkazů použije funkce **`RS232_Send_CMD`** definovaná v hlavičkovém souboru …`\sw\main.h`:

```
int RS232_Send_CMD(BYTE code, WORD param, WORD size, BYTE * data);
```

5.2.3 Přehled definovaných příkazů

V následujících podkapitolách jsou uvedeny nejdůležitější příkazy komunikačního protokolu. Na obrázcích jsou ilustrovány hodnoty parametrů, které jsou předávány obslužným rutinám pro odeslání, resp. přijímání příkazů. Přesné pořadí odesílaní jednotlivých bajtů příkazu naleznete ve zdrojových kódech aplikace.

5.2.3.1 Zjištění aktuální teploty konkrétního kanálu

Obousměrný příkaz, aplikace jej přímo nepoužívá.

code: `0x10, READ_ACTUAL_TEMPERATURE`

param: číslo kanálu (0..7)

size: 0 ve směru PC **à** FITkit,
2 ve směru FITkit **à** PC

data: aktuální teplota * 10; datový typ **short**

CMD	0x10	0x0007	0x0000

Obr. 5.3 Příklad požadavku aktuální teploty ze strany PC na kanál 7

CMD	0x10	0x0007	0x0002	0x016A

Obr. 5.4 Příklad odpovědi FITkitu na předchozí požadavek (teplota 36,2°C)

5.2.3.2 Zjištění maximální teploty konkrétního kanálu

Obousměrný příkaz, aplikace jej přímo nepoužívá. Kontrola maximální teploty se provádí přibližně jednou za sekundu nezávisle na vzorkovací periodě, proto se může lišit od maximální hodnoty uložené v paměti SDRAM.

code: `0x11, READ_MAX_TEMPERATURE`

param: číslo kanálu (0..7)

size: 0 ve směru PC **à** FITkit,
2 ve směru FITkit **à** PC

data: maximální teplota * 10; datový typ **short**

CMD	0x11	0x0007	0x0000

Obr. 5.5 Příklad požadavku max. teploty ze strany PC na kanál 7

CMD	0x11	0x0007	0x0002	0x01E5

Obr. 5.6 Příklad odpovědi FITkitu na předchozí požadavek

5.2.3.3 Zjištění minimální teploty konkrétního kanálu

Obousměrný příkaz, aplikace jej přímo nepoužívá. Kontrola minimální teploty se provádí přibližně jednou za sekundu nezávisle na vzorkovací periodě, proto se může lišit od minimální hodnoty uložené v paměti SDRAM.

code: `0x12, READ_MIN_TEMPERATURE`

param: číslo kanálu (0..7)

size: 0 ve směru PC **à** FITkit,

2 ve směru FITkit **à** PC

data: minimální teplota * 10; datový typ **short**

CMD	0x12	0x0007	0x0000

Obr. 5.7 Příklad požadavku min. teploty ze strany PC na kanál 7

CMD	0x12	0x0007	0x0002	0x01E5

Obr. 5.8 Příklad odpovědi FITkitu na předchozí požadavek

5.2.3.4 Zjištění průměrné teploty konkrétního kanálu

Obousměrný příkaz, aplikace jej přímo nepoužívá. Průměrná teplota se počítá aritmetickým průměrem ze všech teplot ukládaných do paměti SDRAM – závisí tedy na vzorkovací periodě.

code: `0x13, READ_AVG_TEMPERATURE`

param: číslo kanálu (0..7)

size: 0 ve směru PC **à** FITkit,

2 ve směru FITkit **à** PC

data: průměrná teplota * 10; datový typ **short**

CMD	0x13	0x0007	0x0000

Obr. 5.9 Příklad požadavku prům. teploty ze strany PC na kanál 7

CMD	0x13	0x0007	0x0002	0x00E5

Obr. 5.10 Příklad odpovědi FITkitu na předchozí požadavek

5.2.3.5 Zjištění aktuálních teplot ze všech kanálů

Obousměrný příkaz, aplikace jej používá v monitorovacím módu k obnově hodnot aktuálních teplot všech kanálů přibližně jednou za sekundu.

code: `0x14, READ_ACTUAL_TEMPERATURE_ALL_CH`

param: 0

size: 0 ve směru PC **à** FITkit,

16 ve směru FITkit **à** PC (8 kanálů * sizeof(**short**) = 16 bajtů)

data: pole aktuálních teplot * 10 ; datový typ **short[8]**

CMD	0x14	0x0000	0x0000

Obr. 5.11 Příklad požadavku ze strany PC na aktuální teplotu všech kanálů

0 B	CMD		0x14	0x0000	0x0010
8 B	CH 0	CH 1		CH 2	CH 3
16 B	CH 4	CH 5		CH 6	CH 7

Obr. 5.12 Příklad odpovědi FITkitu na předchozí požadavek s ukázkou rozložení kanálů

5.2.3.6 Zjištění maximálních teplot ze všech kanálů

Obousměrný příkaz, aplikace jej používá v monitorovacím módu k obnově hodnot maximálních teplot všech kanálů, obnova na straně PC aplikace se provádí přibližně jednou za 10 sekund.

code: 0x15, *READ_MAX_TEMPERATURE_ALL_CH*

param: 0

size: 0 ve směru PC **à** FITkit,

16 ve směru FITkit **à** PC (8 kanálů * sizeof(**short**) = 16 bajtů)

data: pole maximálních teplot * 10 ; datový typ **short[8]**

CMD	0x15	0x0000	0x0000

Obr. 5.13 Příklad požadavku ze strany PC na max. teplotu všech kanálů

0 B	CMD		0x15	0x0000	0x0010
8 B	CH 0	CH 1		CH 2	CH 3
16 B	CH 4	CH 5		CH 6	CH 7

Obr. 5.14 Odpověď FITkitu na předchozí požadavek s ukázkou rozložení kanálů

5.2.3.7 Zjištění minimálních teplot ze všech kanálů

Obousměrný příkaz, aplikace jej používá v monitorovacím módu k obnově hodnot minimálních teplot všech kanálů, obnova na straně PC aplikace se provádí přibližně jednou za 10 sekund.

code: 0x16, *READ_MIN_TEMPERATURE_ALL_CH*

param: 0

size: 0 ve směru PC **à** FITkit,

16 ve směru FITkit **à** PC (8 kanálů * sizeof(**short**) = 16 bajtů)

data: pole minimálních teplot * 10 ; datový typ **short[8]**

CMD	0x16	0x0000	0x0000

Obr. 5.15 Příklad požadavku ze strany PC na min. teplotu všech kanálů

0 B	CMD	0x16	0x0000	0x0010
8 B	CH 0	CH 1	CH 2	CH 3
16 B	CH 4	CH 5	CH 6	CH 7

Obr. 5.16 Odpověď FITkitu na předchozí požadavek s ukázkou rozložení kanálů

5.2.3.8 Zjištění průměrných teplot ze všech kanálů

Obousměrný příkaz, aplikace jej používá v monitorovacím módu k obnově hodnot průměrných teplot všech kanálů, obnova na straně PC aplikace se provádí přibližně jednou za 10 sekund.

code: 0x17, *READ_AVG_TEMPERATURE_ALL_CH*

param: 0

size: 0 ve směru PC **à** FITkit,

16 ve směru FITkit **à** PC (8 kanálů * sizeof(**short**) = 16 bajtů)

data: pole minimálních teplot * 10 ; datový typ **short[8]**

CMD	0x17	0x0000	0x0000

Obr. 5.17 Příklad požadavku ze strany PC na prům. teplotu všech kanálů

0 B	CMD	0x17	0x0000	0x0010
8 B	CH 0	CH 1	CH 2	CH 3
16 B	CH 4	CH 5	CH 6	CH 7

Obr. 5.18 Odpověď FITkitu na předchozí požadavek s ukázkou rozložení kanálů

5.2.3.9 Zjištění počtu zaznamenaných vzorků

Obousměrný příkaz, odpověď na něj vrací počet zapsaných vzorků do paměti SDRAM. Aplikace na PC potřebuje znát tento údaj pro stanovení využití (zaplnění) paměti na FITkitu.

code: 0x20, *READ_SAMPLE_COUNT*

param: 0

size: 0 ve směru PC **à** FITkit,

4 ve směru FITkit **à** PC

data: počet zaznamenaných vzorků v paměti; datový typ **DWORD**

CMD	0x20	0x0000	0x0000

Obr. 5.19 Příklad požadavku na počet uložených vzorků ze strany PC

CMD	0x20	0x0000	0x0004	0x00003400

Obr. 5.20 Příklad odpovědi FITkitu na předchozí požadavek

5.2.3.10 Nulování počtu zaznamenaných vzorků

Jednosměrný příkaz ve směru PC **à** FITkit, po jeho odeslání dojde k vynulování počtu zaznamenaných vzorků a nové vzorky teplot začnou od začátku paměti přepisovat ty původní. Dále se nastaví výchozí hodnoty pro maximální, minimální a průměrnou hodnotu všech kanálů.

code: 0x21, *CLEAR_SAMPLE_COUNT*

param: 0

size: 0

CMD	0x21	0x0000	0x0000

Obr. 5.21 Příklad příkazu na vynulování počtu zaznamenaných vzorků

Na tento příkaz FITkit neposílá žádnou odpověď zpět do PC.

5.2.3.11 Přečtení časového intervalu zaznamenávání

Obousměrný příkaz, v poli **param** se přenáší hodnota vzorkovací periody v sekundách. Datová část odpovědi není použita, protože se perioda vleze do datového typu **WORD**.

code: 0x22, *READ_TIME_DIVISOR*

param: 0 ve směru PC **à** FITkit,

perioda vzorkování [v sekundách] ve směru FITkit **à** PC; datový typ **WORD**

size: 0

CMD	0x22	0x0000	0x0000

Obr. 5.22 Příklad požadavku na přečtení vzorkovací periody

CMD	0x22	0x0005	0x0000

Obr. 5.23 Příklad odpovědi na předchozí požadavek, perioda 5 s

Aplikace na PC využívá vzorkovací periodu ke stanovení datumu zaplnění paměti.

5.2.3.12 Zápis časového intervalu zaznamenávání

Jednosměrný příkaz, v poli **param** se přenáší hodnota vzorkovací periody v sekundách. Přenos je definován jen ve směru PC **à** FITkit a není nijak potvrzován – informace o změně vzorkovací periody se zobrazí jen krátce na displeji.

code: 0x23, *WRITE_TIME_DIVISOR*

param: **perioda vzorkování** [v sekundách]; datový typ **WORD**

size: 0

CMD	0x23	0x000A	0x0000

Obr. 5.24 Příklad požadavku na zápis vzorkovací periody 10 s

5.2.3.13 Zjištění stavu FITkitu

Jednosměrný příkaz ve směru PC **à** FITkit. Tímto příkazem chceme zjistit, že zařízení připojené na sériový port je FITkit, který má v MCU a FPGA naprogramovanou aplikaci *Monitoring teploty*. FITkit na tento požadavek kladně odpoví příkaze **`STATUS_OK`** (`code 0x25`). Pokud odpověď nedojde do definované doby, je to považováno za chybu a uživatel PC aplikace je o tom informován na monitoru.

`code`: `0x24, READ_STATUS`

`param`: 0

`size`: 0

CMD	0x24	0x0000	0x0000

Obr. 5.25 Příklad požadavku na zjištění stavu FITkitu

5.2.3.14 Odeslání stavu FITkitu – stav OK

Jednosměrný příkaz ve směru FITkit **à** PC. Tento příkaz se posílá na základě předchozího požadavku ze strany PC – **`READ_STATUS`**.

`code`: `0x25, STATUS_OK`

`param`: 0

`size`: 0

CMD	0x25	0x0000	0x0000

Obr. 5.26 Odeslaní stavu FITkitu - stav OK

5.2.3.15 Změna zobrazení hodnoty na displeji

Jednosměrný příkaz ve směru PC **à** FITkit, mění na displej FITkitu typ zobrazované teploty (aktuální, minimální, maximální a průměrnou) a také číslo kanálu, ke kterému se teplota vztahuje.

`code`: `0x28, SET_DISPLAY_MODE`

`param`: Hight Byte: číslo kanálu (0..7)

Low Byte: typ teploty 0 – aktuální,
1 – maximální,
2 – průměrná,
3 – minimální

`size`: 0

CMD	0x28	0x0702	0x0000

Obr. 5.27 Příklad požadavku na změnu zobrazení na displeji – průměrnou teplotu z kanálu 7

5.2.3.16 Přenos bloku paměti

Obousměrný příkaz, slouží k přenosu dat z paměti SDRAM na FITkitu do PC. PC odešle tento příkaz s číslem požadovaného bloku paměti. FITkit na to zareaguje tím, že odešle ten samý příkaz zpět do PC a ihned za ním celý požadovaný blok paměti. Přenosu dat z FITkitu a rozdělení paměti na bloky bude věnována samostatná kapitola v dalším textu.

code: `0x30, READ_RAM_BLOCK`

param: číslo bloku paměti SDRAM (0..8191)

size: 0 ve směru PC **à** FITkit,
1024 ve směru FITkit **à** PC

data: pole hodnot teploty * 10, datový typ **short[512]**

CMD	0x30	0x01A7	0x0000

Obr. 5.28 Příklad požadavku čtení bloku paměti SDRAM č. 423 z FITkitu

0 B	CMD		0x30	0x01A7	0x0400
8 B	data[0]	data[1]	data[2]	data[3]
16 B	data[4]	data[5]	data[6]	data[7]
24 B	data[8]	data[9]	data[10]	data[11]
	•	•	•	•
	•	•	•	•
	•	•	•	•
1016 B	data[504]	data[505]	data[506]	data[507]
1024 B	data[508]	data[509]	data[510]	data[511]

Obr. 5.29 Příklad odpovědi na předchozí požadavek

5.2.4 Rozdělení paměti SDRAM na bloky

Aplikace musí umožňovat hromadné stahování naměřených dat. Množství těchto dat závisí na třech faktorech:

- době zaznamenávání
- vzorkovací periodě
- počtu povolených (vyžádaných) kanálů

Uvažujme nejhorší možnou variantu, tzn. budeme potřebovat z FITkitu přenést celou velikost paměti, tj. 8 MB. Rychlost přenosového kanálu je nastavena na 460800 Bd, vyšší rychlost (921600 Bd) bohužel nefunguje. Musí se ještě odečíst režie asynchronního sériového přenosu, která při konfiguraci 1 start bit, 1 stop bit, 8 datových bitů, bez parity, činí 20 %. Tím dostáváme čistou přenosovou rychlost 368640 bitů/s, což se rovná přesně 45 kB/s. Čistě teoreticky by se celých 8 MB paměti při rychlosti 45 kB/s přeneslo asi za dobu 3 minut. Tato doba však nezahrnuje celou řadu nezanedbatelných zpoždění, jenž vznikají při přenosu dat mezi FITkitem a PC. Cílem je zkrátit tyto zpoždění na minimální mez, tak aby celková doba přenosu byla co nejkratší.

Paměť SDRAM na FITkitu je možné adresovat (přistupovat do ní) pouze po bajtech, ovladač paměti neumožňuje přečíst najednou více jak jeden bajt. Kdyby měla aplikace na straně PC posílat příkaz pro přečtení jediného bajtu, vznikla by komunikační režie téměř 90 %, viz tab. 5.3. A to je uvažována pouze režie při odesílání dat z FITkitu, nikoliv při jejich požadování ze strany PC. Taková režie je neúnosná a přenos dat by trval extrémně dlouhou dobu.

Proto je cílem snížit režii na co nejnižší úroveň, např. tím způsobem, že přenos paměti bude blokově orientovaný. Čím větší bude velikost bloku v porovnání s velikostí příkazu (8 bajtů), tím bude režie menší. To dává myšlenku rozdělit paměť SDRAM na FITkitu na bloky. Toto rozdělení je jen pouze pro účely snížení režie při přenosu dat z FITkitu. Rozdělení jinak nemá žádný jiný význam. V tab. 5.3 jsou uvedeny možné varianty velikosti bloků v bajtech, jejich celkový počet a počet bloků připadajících na kanál při osmi kanálech.

Velikost bloku [B]	Celkový počet bloků	Počet bloků na kanál	Režie [%]
1	8388608	1048576	88,89
2	4194304	524288	80,00
4	2097152	262144	66,67
8	1048576	131072	50,00
16	524288	65536	33,33
32	262144	32768	20,00
64	131072	16384	11,11
128	65536	8192	5,88
256	32768	4096	3,03
512	16384	2048	1,54
1024	**8192**	**1024**	**0,78**
2048	4096	512	0,39
4096	2048	256	0,19
8192	1024	128	0,10
16384	512	64	0,05
32768	256	32	0,02

Tab. 5.3 Režie přenosu bloku při konstantní délce příkazu 8 bajtů

Velikost bloku by měla být co největší, aby režie byla co nejnižší, ale musíme také brát v úvahu velikost paměti v mikrokontroléru. Ten má vnitřní paměť jen 2 kB, takže největší možná velikost paměti, která se dá alokovat je 1024 bajtů. Větší oblast paměti odmítá překladač přeložit. Paměť pro blok bychom v mikrokontroléru ani nemuseli alokovat, kdybychom četli přímo z paměti SDRAM a ihned odesílali přečtenou hodnotu. Funkce pro čtení paměti je však koncipována obecně a není nerychlejší. Proto jsem vytvořil vlastní funkci pro čtení dat z paměti SDRAM, která je rychlejší. Přečte zadaný blok z paměti SDRAM a uloží ho na alokované místo v paměti mikrokontroléru. Funkci pro odeslání příkazu se potom předá jen ukazatel na takto načtený blok paměti.

5.2.5 Princip komunikace mezi FITkitem a PC

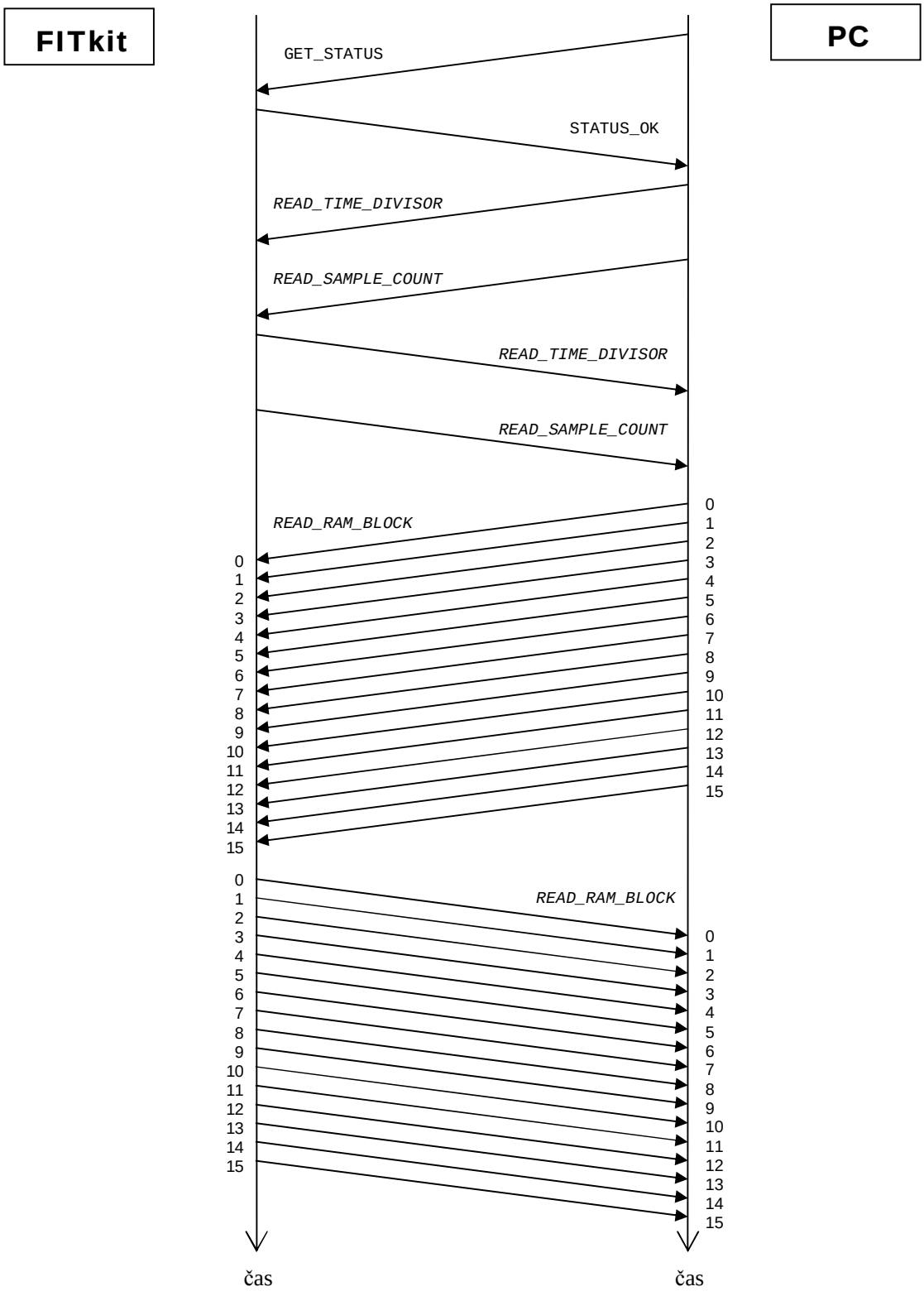
Po spuštění aplikace na PC a připojení na příslušný sériový port dojde k následujícím událostem:

- PC vyšle příkaz *GET_STATUS*
- FITkit odpoví příkazem *STATUS_OK*
- PC pošle příkaz *READ_TIME_DIVISOR*
- PC pošle příkaz *READ_SAMPLE_COUNT*
- FITkit pošle odpověď *READ_TIME_DIVISOR*
- FITkit pošle odpověď *READ_SAMPLE_COUNT*
- Poté následuje přenos dat nebo monitoring kanálů; případně nic z uvedeného

Příklad začátku komunikace je uveden na obrázku Obr. 5.30, poté následuje požadavek na přenos prvních 16-ti bloků paměti SDRAM. FITkit má implementovány knihovny pro práci se sériovým rozhraním, ve kterých jsou pevně nastaveny velikosti vyrovnávacích bufferů pro příjem a odesílání dat. Velikost bufferu pro příchozí data jsem nastavil na maximální možnou velikost 128 bajtů. Nastavení větší velikosti již způsobovalo chyby při přenosu. Obecně se dá říct, že knihovna pro práci se sériovým portem na FITkitu je pro tuto úlohu značně poddimenzována, což se týče především velikosti bufferů a datových typů pro práci s portem.

Jelikož přijímací buffer na FITkitu má kapacitu jen 128 bajtů, můžeme nárazově poslat maximálně 16 příkazů o délce 8 bajtů (16*8 = 128 bajtů), aby nedošlo k přeplnění bufferu a následné ztrátě dat. Velikost obou vyrovnávacích bufferů je nutné změnit na hodnotu 128 bajtů také v souboru …\DIP\FITkit\sw\libfitkit**rs232_0.h**:

```
#define LengthBuffers_Rx   128
#define LengthBuffers_Tx   128
```



Obr. 5.30 Příklad komunikace po připojení k sériovému portu a přenos prvních 16 bloků

6 Návrh a implementace software

Návrh SW je složen ze dvou, resp. tří částí. Jednu část tvoří zdrojové kódy v jazyce C pro MCU a zdrojové kódy v jazyce VHDL pro FPGA. Konfigurace pro FPGA je především HW záležitostí, která byla řešena ve 4. kapitole. Druhou část tvoří aplikace běžící na PC, která je programovaná v jazyce C++ a grafickém vývojovém prostředí C++ Builder 6.0 od firmy Borland.

Obě části mají obecně podobný kód pro komunikaci po sériovém rozhraní, které řeší přijímání a odesílání příkazů. Nicméně platformy jsou to odlišné, proto musí být kód programu napsán tak aby vyhovoval cílové platformě.

Všechny zdrojové kódy jsou uloženy v složce …`\FITkit\apps\thermo_monitor`. Je tedy zařazena jako nová aplikace FITkitu rev2.27. Při použití vyšší revize knihoven FITkitu nemusí jít aplikace přeložit. Jednak se musí změnit některé hodnoty konstant v hlavičkových souborech (nezávisle na revizi), ale hlavně můžou být přejmenovány nebo upraveny některé funkce. Proto se musí kontrolovat rozdíly ve zdrojových kódech při každém zařazení aplikace *Monitoring teploty* do nové verze knihoven FITkitu.

Zdrojové kódy pro jednotlivé překladače jsou rozděleny následovně:

- **`thermo_monitor\pc`** – GUI aplikace pro PC
- **`thermo_monitor\sw`** – SW pro MCU
- **`thermo_monitor\top`** – konfigurace pro FPGA

6.1 SW pro FITkit – MCU

Základ tvoří nekonečná smyčka, ve které se volají jednotlivé funkce pro obsluhu periferií. Před vstupem do této smyčky se provedou nezbytné inicializace všech řadičů a proměnných. V každém průchodu smyčkou se zkontroluje stav hodinového signálu a pokud se jeho hodnota změní na úroveň log. 1, tak se provede odečtení aktuálních hodnot všech kanálů a jejich uložení do paměti SDRAM.

V aplikaci je vyřazena funkce terminálu – není tedy možné přes terminálový program komunikovat s FITkitem prostřednictvím definovaných příkazu. Tato zvláštnost je způsobena tím, že komunikační protokol je binárně orientovaný. Proto při nahrávání konfigurace pro FPGA do paměti FLASH je potřeba první nahrát nějakou jinou aplikaci, např. *testled*, která má zapnutu podporu terminálových příkazů.

6.1.1 Soubory projektu

Zdrojové kódy pro mikrokontrolér jsou uloženy v adresáři:

…\FITkit\apps\thermo_monitor\sw, tento adresář obsahuje 4 soubory se zdrojovými kódy a *Makefile* pro přeložení a nahrání aplikace do MCU.

- `main.c`
- `main.h`
- `Thermo_SMT_160_30.c`
- `Thermo_SMT_160_30.h`

V souboru *Thermo_SMT_160_30.c* jsou definovány především funkce pro načtení a zpracování teploty z registrů FPGA. Jsou zde definovány funkce pro přepočet střídy na teplotu, převod desetinného čísla na řetězec pro zobrazení na displeji, čtení a zápis vzorkovací periody z/do paměti FLASH, optimalizovaná funkce pro čtení a zápis z/do paměti SDRAM.

Konstanty jsou definovány v hlavičkovém souboru *Thermo_SMT_160_30.h* a také v souboru *…\FITkit\apps\thermo_monitor\pc\thermo_command.h*, kde jsou společné konstanty jak pro MCU, tak pro aplikaci na PC. V tomto souboru jsou nadefinovány konstanty pro práci s pamětí SDRAM na FITkitu a čísla příkazů komunikačního protokolu mezi FITkitem a PC.

Ze souboru *main.c* se volají všechny potřebné funkce. Je zde hlavní smyčka a inicializační procedury.

6.1.2 Výpočet a zobrazení teploty

6.1.2.1 Výpočet teploty

Výpočet teploty se provádí na základě zjištění střídy z teplotního senzoru. Princip měření střídy signálu popisuje začátek 4. kapitoly. Symboly T_1, T_2 a T se taktéž vztahují k obr. 4.1. na straně 17.

$$t = \frac{\frac{T_1}{T} - 0{,}320}{0{,}00470} = \frac{\frac{T_1}{T_1 + T_2} - 0{,}320}{0{,}00470}, \text{ [°C]}$$

Rov. 6.1 Výpočet teploty ve °C ze střídy signálu

Rov. 6.1 určuje konečný vztah pro výpočet teploty ve °C, je odvozena z rov. 3.4 na straně 15. Jelikož výsledky jednoho výpočtu, resp. jednoho měření nebyly uspokojivé, muselo se měření několikrát opakovat a z naměřených hodnot vypočítat aritmetický průměr. Počet měření pro výpočet jedné hodnoty teploty lze ovlivnit změnou konstanty **`N_SAMPLE`**, deklarované v hlavičkovém souboru *…*`\FITkit\apps\thermo_monitor\sw\`**`Thermo_SMT_160_30.h`**. Hodnota konstanty by se měla pohybovat v rozmezí 24 až 64. Vyšší počet měření již nepřináší svůj efekt a pouze prodlužuje dobu výpočtu. Abychom minimalizovali chybu měření, je z výpočtu průměrné hodnoty odečten vzorek s maximální a minimální hodnotou. Tyto extrémy by mohly značně ovlivnit výpočet průměrné hodnoty, kdyby se značně lišily od ostatních hodnot. Veškeré hodnoty teploty se kterými se na FITki-

tu následně pracuje jsou počítány podle rov. 6.2, kde $N = N\_SAMPLE$, MAX – hodnota maxima ze všech vzorků, MIN – hodnota minima ze všech vzorků.

$$T = \frac{\left(\sum_{n=0}^{N-1} t[n]\right) - MAX - MIN}{N-2}, [°C]$$

Rov. 6.2 Průměrná hodnota z opakovaných měření

Nutno ještě podotknout, že prakticky většina matematických operací v MCU se provádí v celočíselné aritmetice a pracuje se s hodnotami vynásobenými 10krát, poněvadž výpočty v plovoucí řádové čárce trvají mnohem delší dobu.

6.1.2.2 Zobrazení teploty na displeji

Teplota na displeji je zobrazena ve formě desetinného čísla. Bohužel funkce `sprintf` nemá podporu pro formátovaný tisk čísel v plovoucí řadové čárce, tudíž ji neleze použít a je nutné vytvořit vlastní funkci. Obecně jsou standardní funkce jazyka C pro tento procesor velmi omezené. Proto jsem vytvořil funkci `void TempX10ToFloatStr(int value, char *str)`, která do alokovaného bufferu `str` zapíše teplotu ve formě desetinného čísla včetně znaménka a jednotky °C. Buffer by měl mít alokováno 8 bajtů. Hodnota teploty, `value`, se zadává vynásobená 10krát (287 = 28,7 °C). Výsledek je vidět na obr. 6.1. AVG značí průměrnou teplotu (*average*), ch1 je označení kanálu 1 (*channel*). Znak ° je ve zdrojovém kódu nahrazen znakem β.

Obr. 6.1 Zobrazení průměrné hodnoty na displeji

V rámci testování jsem provedl jak test maximální měřitelné teploty 130 °C, tak i test minimální teploty. Minimální teploty jsem v domácích podmínkách nedosáhl, takže musí stačit jen -20,6 °C v mrazničce. Maximální teplota byla měřena na rezistoru o výkonu 6 W, ke kterému byly stahovací páskou připevněna i další dva termočlánky z multimetrů umožňující měření teploty. Velmi však záleželo na teplotním přechodovém odporu mezi termočlánkem a zdrojem tepla. Proto nebylo možné stanovit zda maximální teplota zcela přesně odpovídá realitě. Ukázky maximální, resp. minimální teploty jsou na obr. 6.2, resp. obr. 6.3.

Obr. 6.2 Maximální dosažená teplota

Obr. 6.3 Minimální dosažená teplota

6.1.3 Ovládání pomocí klávesnice

K obsluze klávesnice je použita obslužná rutina, která nevyužívá přerušení. Tuto variantu jsem zvolil jenom kvůli tomu, aby nebylo potřeba přidávat žádný další vodič, jenž by propojoval FPGA a MCU. Obecně by bylo vhodnější použít variantu s přerušením, protože varianta bez přerušení je poněkud náročná na výpočetní výkon a zjištění stavu klávesnice trvá poměrně dlouhou dobu. Vzhledem k této situaci bylo nutné volání obsluhy klávesnice omezit na minimum, ale zároveň by měla být zachována přijatelná rychlost reakce na stisk klávesy.

Setkal jsem se také s problémem spolehlivosti klávesnice. V ukázkové aplikaci klávesnice funguje bez problémů a spolehlivě. Je to dáno tím, že celé zapojení je taktováno na mnohem nižší frekvenci (asi 7,34 MHz), než která je použita v aplikaci Thermo monitor. Kvůli synchronizaci paměti SDRAM, která vyžaduje 50 MHz a také kvůli přesnosti měření bylo nutné synchronizovat celý systém právě na 50 MHz. Domnívám se, že právě vyšší frekvence způsobuje zhoršenou spolehlivost klávesnice – ta se projevuje tím, že zejména klávesy ***** a **#** se často detekují jako jiná klávesa (obvykle to bývá **0**). Tím pádem jsem těmto klávesám nepřiřazoval žádnou funkci. Ostatní klávesy fungují do jisté míry spolehlivě.

Pomocí klávesnice lze na FITkitu měnit informace zobrazující se na displej. Především jde o změnu čísla kanálu a typ teploty, která se k tomuto kanálu vztahuje. Význam jednotlivých kláves:

- **0** – aktivuje kanál 0
- **1** – aktivuje kanál 1
- **2** – aktivuje kanál 2
- **3** – aktivuje kanál 3

- **4** – aktivuje kanál 4
- **5** – aktivuje kanál 5
- **6** – aktivuje kanál 6
- **7** – aktivuje kanál 7
- **8** – bez funkce
- **9** – zobrazí stav využití paměti
- ***** – bez funkce
- **#** – bez funkce
- **A** – změní typ zobrazované hodnoty vybraného kanálu na **aktuální** teplotu (**ACT**)
- **B** – změní typ zobrazované hodnoty vybraného kanálu na **maximální** teplotu (**MAX**)
- **C** – změní typ zobrazované hodnoty vybraného kanálu na **průměrnou** teplotu (**AVG**)
- **D** – změní typ zobrazované hodnoty vybraného kanálu na **minimální** teplotu (**MIN**)

Ovládání je navrženo jen pro 8 kanálů, navýšením počtu kanálů by se musel změnit způsob přepínání jednotlivých kanálů, protože na přímou volbu máme jen 10 kláves. Jiný způsob přepínání kanálů by se mohl být ve stylu kurzorových kláves, tak jak je tomu např. na mobilním telefonu.

6.1.4 Zpracování příkazů komunikačního protokolu

Délka příkazu odesílaného z PC je pevně stanovena na 8 bajtů. Takže teprve po přečtení 8 bajtů se může dekódovat příkaz. Musíme také brát v úvahu, že přijatá data nemusí být korektní příkaz a také délka dat nemusí být přesně 8 bajtů. S tímto stavem je nutné počítat a umět na něj patřičně zareagovat. Třeba tím, že se na displeji zobrazí chybová hláška a celý obsah přijímacího bufferu na FITkitu se vyprázdní. Pokud po vyprázdnění přijímacího bufferu přijde ještě méně jak 8 bajtů a za nimi hned platný příkaz, je tento příkaz zahozen, protože není možno správně stanovit začátek příkazu.

Komunikační protokol předpokládá spolehlivý přenos přes sériové rozhraní a příkazy ani data nejsou opatřena žádným kódem pro detekci nebo opravu chyb. Vzdálenost mezi FITkitem a PC nemůže být velká a na krátké vzdálenosti by chyby vznikat neměly. Navíc jde o přímé propojení kabelem bez žádných dalších zařízení. Jediné co je kontrolováno, je délka příkazu a dat a u některých příkazů také rozsahy hodnot jednotlivých polí příkazu.

Pokud by byl požadován spolehlivý přenos, tak by se za každý odeslaný příkaz včetně dat zařadil 16bitový kontrolní součet (CRC). Tím by samozřejmě vzrostla režie přenosu a také by se prodloužila doba přenosu dat, protože by se musel počítat kontrolní součet nad velkým objemem dat, což by jistě netrvalo krátkou dobu.

6.1.5 Záznam teplot do paměti

V hlavní smyčce se neustále zjišťuje stav sekundového generátoru z FPGA a na náběžnou hranu se zvýší počitadlo sekundových taktů. Dosáhne-li toto počitadlo nastaveného intervalu vzorkování, dojde k nulování počitadla a ke zjištění teploty a následnému zápisu teplot do paměti SDRAM. Zápis do paměti je signalizován rozsvícením červené LED diody D6. Do paměti se zapisují hodnoty teploty vynásobené 10krát. Ze zapisované teploty do paměti se počítá průměrná teplota (prostým aritmetickým průměrem ze všech zapsaných hodnot).

Pokud dojde k přetečení čítače, který počítá zapsané vzorky do paměti, provede se nulování min., max. a průměrné hodnoty všech kanálů. Čítač začne čítat od nuly a tím dojde k přepisování hodnot od začátku paměti.

Nezávisle na tom se měří teplota a zobrazuje se na displeji. Toto měření je signalizováno rozsvícením zelené LED diody D5. Z těchto hodnot se také zjišťuje minimální a maximální teplota.

Hodnota vzorkovací periody se zapisuje do paměti FLASH, aby se po zapnutí FITkitu nemusela znovu zadávat. Perioda se ukládá ve FLASH paměti na adresu 0x000203A0 (stránka 500). Adresa je zvolena z takové oblasti, do které nezasahují žádná důležitá data (jako je např. konfigurace pro FPGA). Vzorkovací perioda se do FLASH paměti zapisuje při změně periody z PC aplikace. Čtení této hodnoty se provádí automaticky po zapnutí FITkitu.

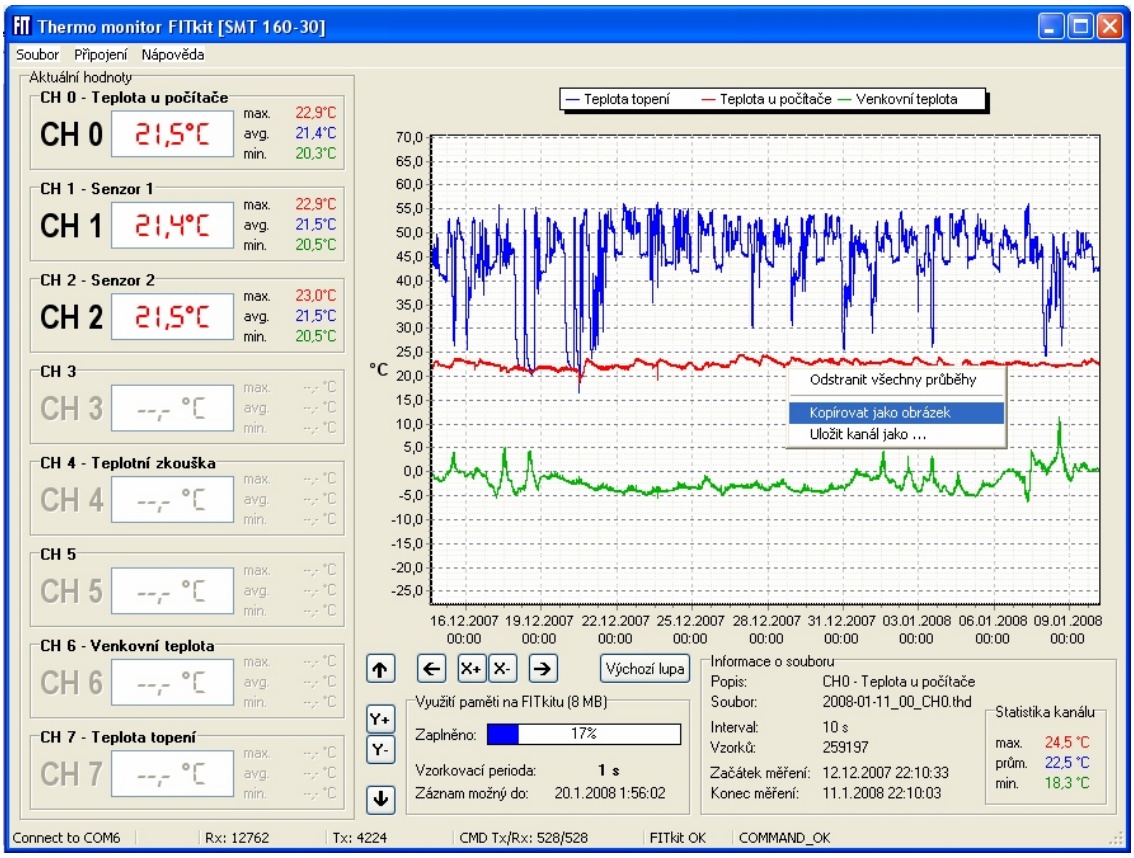
6.2 SW pro PC

Aplikace pro PC je do jisté míry principiálně podobná aplikaci pro MCU. Má však odlišnosti, jenž jsou typické pro platformu osobních počítačů a v oblasti programování mikroprocesorů se nimi nesetkáme.

Hlavní cíle PC aplikace:

- Umět komunikovat s FITkitem pomocí komunikačního protokolu
- Umožnit hromadné stahování naměřených dat
- Zobrazovat aktuální stav teploty ze všech kanálů
- Měnit druh hodnoty zobrazené na displeji
- Zobrazit orientační grafický průběh stažených dat
- Ukládání naměřených dat do souborů v různých formátech pro další zpracování

Vzhled aplikace je možno shlédnout na obr. 6.4.



Obr. 6.4 Hlavní okno aplikace

6.2.1 Připojení na sériový port

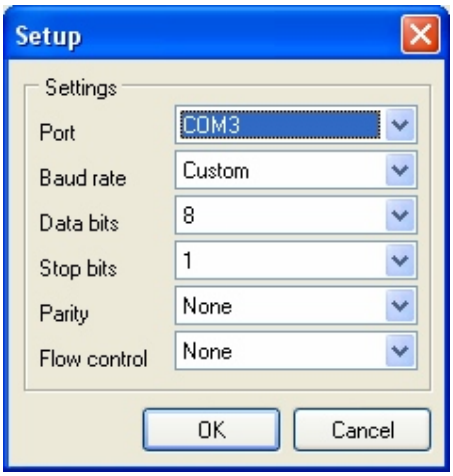
Práci se sériovým portem zajišťuje komponenta CPort, která je volně ke stažení z [12]. Obsluha sériového portu s touto komponentou je na velmi dobré úrovni. Vše je řízeno událostmi. Mezi nejčastěji používané události patří:

- `OnRxChar` – událost na přijatý znak
- `OnAfterOpen` – po otevření sériového portu se volá tato událost
- `OnAfterClose` – po zavření sériového portu se volá tato událost

Pro nastavení parametrů portu je již vytvořen jednoduchý formulář, viz obr. 6.5. Na tomto formuláři se nastavují základní parametry sériového portu jako je číslo portu, přenosová rychlost, počet datových bitů, počet stop bitů, parita a řízení toku. Autor komponenty však nezařadil do nabídky přenosových rychlostí rychlost větší než 256 000 Bd. Přidal ale položku „Custom“, která umožňuje nastavit vlastní přenosovou rychlost – bohužel jen přímo v kódu programu. I to se dá řešit tak, že rychlost bude uložena v inicializačním souboru, který se bude nahrávat při startu aplikace. Taktéž nastavení sériového portu lze snadno uložit do inicializační souboru a opět načíst.

Po úspěšném otevření sériového portu se provedou nezbytné inicializace proměnných, které souvisí s komunikačním protokolem a stahováním dat. Následně se odešle dotaz na stav FITkit. Stav

připojení se zobrazuje v dolním informačním řádku hlavního okna. Aplikace se může na port připojit manuálně (z menu „Připojení“), anebo automaticky ihned po spuštění programu – to záleží na nastavení. Informace o využití paměti a hodnota vzorkovací periody se aktualizují vždy po připojení na port.



Obr. 6.5 Nastavení parametrů sériového portu

6.2.2 Zpracování komunikačního protokolu

Přijímání a následné zpracování příkazů je pro PC aplikaci složitější než pro MCU, jelikož PC musí umět přijímat i data. Pomocí stavového automatu musíme rozlišit, zda očekáváme příkaz nebo data. Primárně se očekává příkaz, jehož význam lze dekódovat až po přijetí 8 bajtů. Jedná-li se o příkaz, za nímž následují data, přepne se automat do režimu přijímání dat. V tomto režimu setrvá tak dlouho, dokud nepřijme všechna očekávaná data (hodnota **`size`**). Po přijetí všech dat se automat přepne zpět do režim očekávající příkaz.

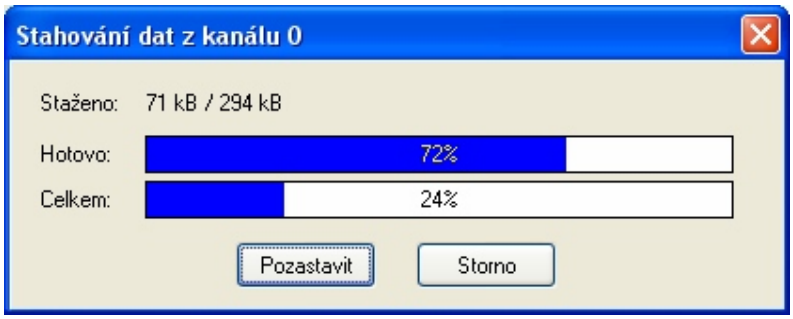
Velikost bufferu pro příjem a odesílání dat přes sériový port lze nastavit na téměř jakoukoliv velikost. Tudíž nehrozí podobný problém jako v knihovně `libfitkit`, kde jsou vyrovnávací buffery kriticky malé. Na příchozí znaky reaguje událost `OnRxChar`. V obsluze této události se nejprve nasbírá potřebný počet bajtů. Poté následuje, v případě očekávaného příkazu, rozdělení na jednotlivé parametry příkazu. Tyto parametry (a případně i data) se následně předají funkci `Decode_CMD(BYTE code, WORD param, WORD size, BYTE * data)`, která už se postará o vykonání potřebné akce.

6.2.3 Stahování naměřených dat

Stahování naměřených dat z FITkitu do PC může trvat poměrně dlouhou dobu. Záleží na množství dat (době zaznamenávání a periodě) a také na počtu povolených (aktivních) kanálů v PC aplikaci. Nemá totiž smysl stahovat data z kanálů, na kterých nejsou osazena teplotní čidla. Doba stahování se

může pohybovat od pár sekund až po maximálně 30 minut – při plném využití paměti a všech povolených kanálech.

Jelikož doba může být velmi dlouhá a proměnlivá, musí být uživatel informován o průběhu stahování. K zobrazení průběhu stažených dat jsem použil komponentu CGauge, která zobrazuje stav přímo v procentech. Protože máme k dispozici až 8 kanálů, je vhodné použít dvě tyto komponenty – jedna bude zobrazovat průběh stahování aktuálního kanálu a druhá bude zobrazovat celkový průběh. K tomu také přidáme informace o množství stažených kilobajtů. Ukázku stahování dat můžete shlédnout na obr. 6.6.



Obr. 6.6 Průběh stahování dat z FITkitu

Stahování je možné pozastavit a pak ve stahování pokračovat. Reakce na toto tlačítko „Pozastavit“ však nemusí být okamžitá, protože tímto tlačítkem se zastavuje odesílání požadavků na přenos bloku a nikoliv přijímání. Takže po stisku tlačítka se může přečíst až 16 bloků o velikosti 1 kB. Odesílání požadavků na přečtení dalších bloků je implementováno ve vlákně, aby bylo možné kdykoliv zrušit nebo pozastavit tuto činnost. Další výhodou vlákna je možnost aktualizace průběhu stahování, což by bez vlákna nebylo možné.

6.2.4 Ovládání programu

Program umožňuje zobrazení aktuálních teplot, aktivace a deaktivace se provádí v menu „Připojení **à** Monitor“. Ekvivalentní klávesová zkratka je CTRL+M. Před stahováním dat však dojde k vypnutí monitorování, protože by to mohlo narušit průběh stahování dat.

Změna typu zobrazované hodnoty na displeji FITkitu se provádí stiskem pravého tlačítka v oblasti ukazatele aktuální teploty příslušného kanálu.

6.2.4.1 Práce s grafem

Data se po stažení do PC automaticky zobrazí v grafu. Nezávisle na sobě lze měnit měřítko časové i teplotní osy pomocí tlačítek nebo kláves + a -. Pro úpravu měřítka časové osy musíme stisknout klávesu CTRL. Posouvání v grafu se děje pomocí kurzorových šipek nebo tlačítek.

Graf je možné taktéž zkopírovat do schránky jako obrázek – pomocí kontextového menu, které se vyvolá stiskem pravého tlačítka myši v oblasti grafu. Otevřením dalšího souboru se do současného zobrazení přidá další průběh.

6.2.4.2 Stažení naměřených dat

Pro stažení dat musí být program připojen na FITkit. Pak se kliknutím na menu „Připojení **à** Stáhnout data do PC“ odstartuje stahování dat. Ekvivalentní klávesová zkratka je CTRL+D (*Download*). Poté se zobrazí okno s průběhem stahování. Po dokončení stahování je uživatel dotázán, zda chce data uložit do souboru. V případě kladné odpovědi jsou data z každého povoleného kanálu uložena do souboru ve složce …`\FITkit\apps\thermo_monitor\pc\`**`Data`**. Jména souborů jsou generována automaticky.

Stažená data můžeme uložit také v jiném než binárním souboru. Všechny stažené kanály jdou do jednoho soboru exportovat pouze hned po stažení. Při dalším spuštěním programu a otevřením binárních souborů je možné exportovat do jiného formátu pouze každý kanál zvlášť. Je to dáno vnitřní strukturou uložení dat v aplikaci.

6.2.5 Zpracování naměřených dat v tabulkových procesorech

Naměřená data lze také ukládat do textových souborů. Uložení v textovém formátu však zabírá několikanásobně více místa na pevném disku než uložení ve formátu binárním. Textový formát je určen především pro následné zpracování v jiných programech. Těmito programy mohou být tabulkové procesory typu Microsoft Excel nebo Open Office Calc, případně jiné. Taktéž mohou být data z textového souboru načtena aplikací Matlab.

Pro načtení dat do tabulkových procesorů je nutné se zabývat formátem uložení desetinného čísla. V závislosti na lokálním nastavením operačního sytému se jako oddělovač desetinných čísel používá desetinná tečka (.) nebo desetinná čárka (,). Pro českou lokalizaci je typická desetinná čárka a pro anglickou lokalizaci zase desetinná tečka. Právě tento na první pohled banální rozdíl může způsobovat nemalé problémy při načítání a zpracování dat v tabulkových procesorech. Pokud např. bude v textovém souboru uloženo desetinné číslo s desetinou tečkou a soubor budeme chtít otevřít v Excelu na operačním systému s českou lokalizaci, budou načtená čísla chápána jako text a nikoliv jako číslo. Samozřejmě, že lze hromadně nahradit desetinou tečku za čárku, ale to zase vyžaduje další práci. Proto má uživatel možnost v nastavení programu zvolit, zda bude používat desetinnou tečku nebo desetinnou čárku.

Dále by měl uživatel brát v úvahu maximální počet řádků, jenž umí tabulkový procesor zpracovat. Počet řádků v tabulkovém procesoru může být omezen např. na 65 536, ale naměřených vzorků může být až 524 288, což je 8krát víc. Také zpracování takového množství dat může trvat nečekaně dlouhou dobu. Z toho vyplývá, že vzorkovací perioda by měla být volena podle možností SW, ve kterém se budou data dále zpracovávat.

6.2.6 Ukládání dat do souboru

Naměřená data lze ukládat do tří typů souborů. Jeden z nich má binární formát, zbývající dva jsou v textovém formátu. Binární formát je jediný formát, který je možné později v aplikaci otevřít. Textové formáty jsou určeny výhradně pro export do jiných aplikací.

6.2.6.1 Struktura binárního souboru *.thd

Přípona souboru *.thd znamená ***th**ermo **d**ata*. Pro každý kanál se vytváří samostatný soubor a počet vytvářených souborů závisí na počtu povolených kanálů v nastavení aplikace. Na začátku souboru se nachází hlavička souboru s pevně danou velikostí a hned za ní následují samotná data.

Hlavička souboru je definována jako struktura `ThermoChannelHeader` o velikosti 64 bajtů v souboru …`\FITkit\apps\thermo_monitor\pc\`**`Unit1.h`**:

```
typedef struct ThermoChannelHeader{
    DWORD       file_type;              // typ souboru "TH08"
    BYTE        channel;                // číslo kanálu
    BYTE        value_size;             // šířka datového typu v bajtech, (sizeof(short)=2)
    WORD        time_divisor;           // dělící koeficient časové základny (rychlost vzorkování)
    char        title[TITLE_SIZE];      // název kanálu (titulek), TITLE_SIZE = 32
    DWORD       samples;                // počet uložených vzorků
    TDateTime   t_begin;                // začátek ukládání dat
    TDateTime   t_end;                  // konec ukládání dat
}ThermoChannelHeader;
```

Význam jednotlivých parametrů:

- **`file_type`**: jednoznačně identifikuje typ souboru, TH = THermo, 08 = 8 kanálů
- **`channel`**: číslo kanálu, ze kterého byla data stažena
- **`value_size`**: na kolika bajtech je uložena hodnota jednoho vzorku
- **`time_divisor`**: vzorkovací interval v sekundách
- **`title`**: název kanálu, max. 32 znaků, kratší názvy jsou automaticky doplněny ‘\0‘
- **`samples`**: počet uložených vzorků
- **`t_begin`**: vypočítaný začátek ukládání dat ve formátu TDateTime, `sizeof(TDateTime)` = 8 bajtů
- **`t_end`**: datum a čas konce ukládání dat ve formátu TDateTime – okamžik začátku stahování dat aplikací (zjištěno z lokálního času počítače)

Po této struktuře už následují samostatná data, která jsou uložena jako datový typ **`short`**. Teplota je tedy uložena jako celé číslo se znaménkem na 16 bitech. Pro získání reálné teploty je potřeba tuto hodnotu vydělit číslem 10. Datový typ **`short`** jsem zvolil z důvodu úspory místa na disku. Další místo na disku lze ušetřit komprimací binárního souboru v nějakém komprimačním programu, čímž se lze dostat i na desetinu původní velikosti souboru.

6.2.6.2 Struktura textových souborů

Textové soubory jsou ve dvou formátech: *.txt a *.csv. Obsahově se liší pouze v oddělovačích. Formát *.txt používá jako oddělovač sloupců znak tabulátoru, formát *.csv používá znak středník (;). Pro import do tabulkových procesorů je primárně určen soubor s příponou *.csv.

V nastavení programu lze nakonfigurovat jaké položky se budou do textového souboru zapisovat. V hlavičce souboru (1. řádek) může být zapsáno označení kanálu (např. „CH4“) nebo hlavička nemusí být zapsána vůbec. Na dalším řádku se podle konfigurace zapíše nebo nezapíše jmenovka kanálu. Do prvního sloupce lze zapsat číslo vzorku nebo datum a čas vzorku, anebo se první sloupec zcela vynechá. Další sloupce jsou již hodnoty teplot jednotlivých kanálů. Za hodnotu teploty lze podle konfigurace zapsat také jednotku „°C“, ale to nemusí být v mnoha případech vhodné.

Data z vybraných kanálů mohou být ukládána jak do jednoho souboru společně, tak zároveň i každý kanál do samostatného souboru – opět záleží na konfiguraci.

6.2.6.3 Vytváření jmen souborů

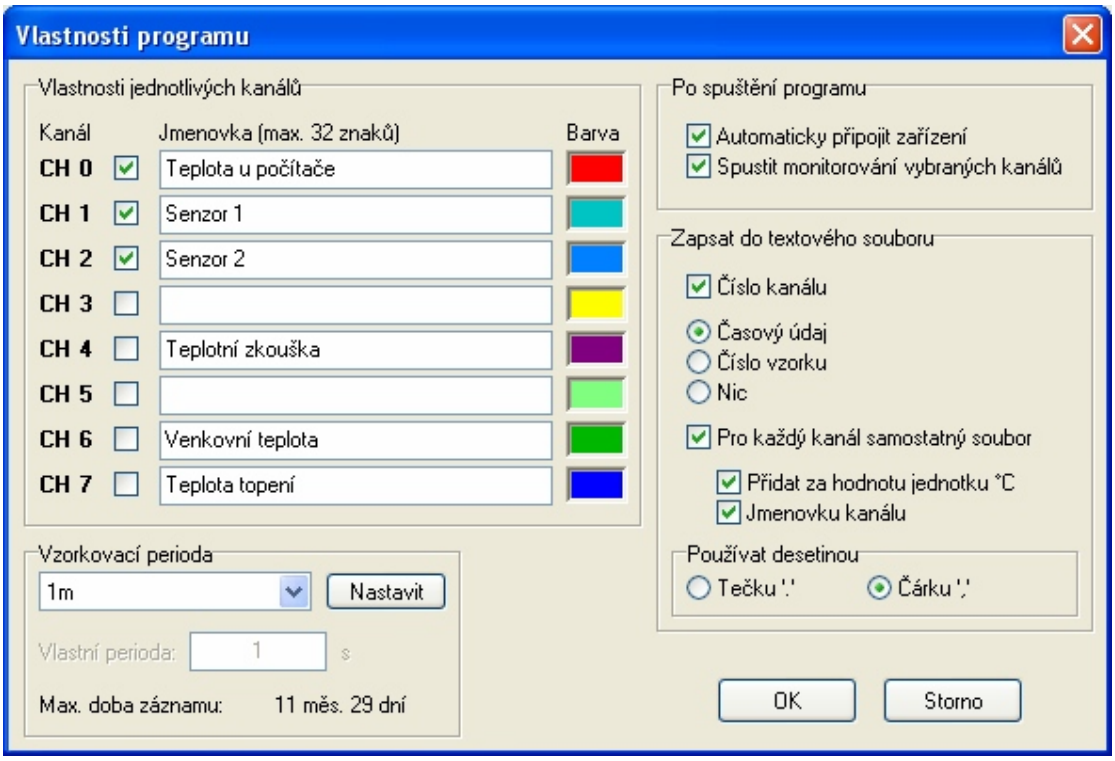
Jména souborů jsou generována automaticky. Začátek jména souboru tvoří datum ve formátu „YYYY-MM-DD“, poté následuje dvojciferné pořadové číslo souboru – pro případ, že by stahování dat provádělo vícekrát za den. Za tímto číslem se nachází označení kanálu (např. „CH4“) a přípona podle aktuálního typu.

Jméno textového souboru, do kterého se budou zapisovat všechny vybrané kanály současně, lze zadat plně uživatelsky ve standardním dialogu.

6.2.7 Nastavení programu

Některé vlastnosti programu lze měnit v menu Připojení **à** Vlastnosti. Zdejší hodnoty jsou ukládány do konfiguračního souboru „ThermoMonitor.ini“, který musí být umístěn ve stejném adresáři jako aplikace.

Nastavují se zde jmenovky kanálů, povolení kanálu, jaké operace se mají provést po spuštění programu a možnosti zápisu textových souborů, viz obr. 6.7.



Obr. 6.7 Přizpůsobení vlastností programu

6.2.7.1 Změna vzorkovací periody

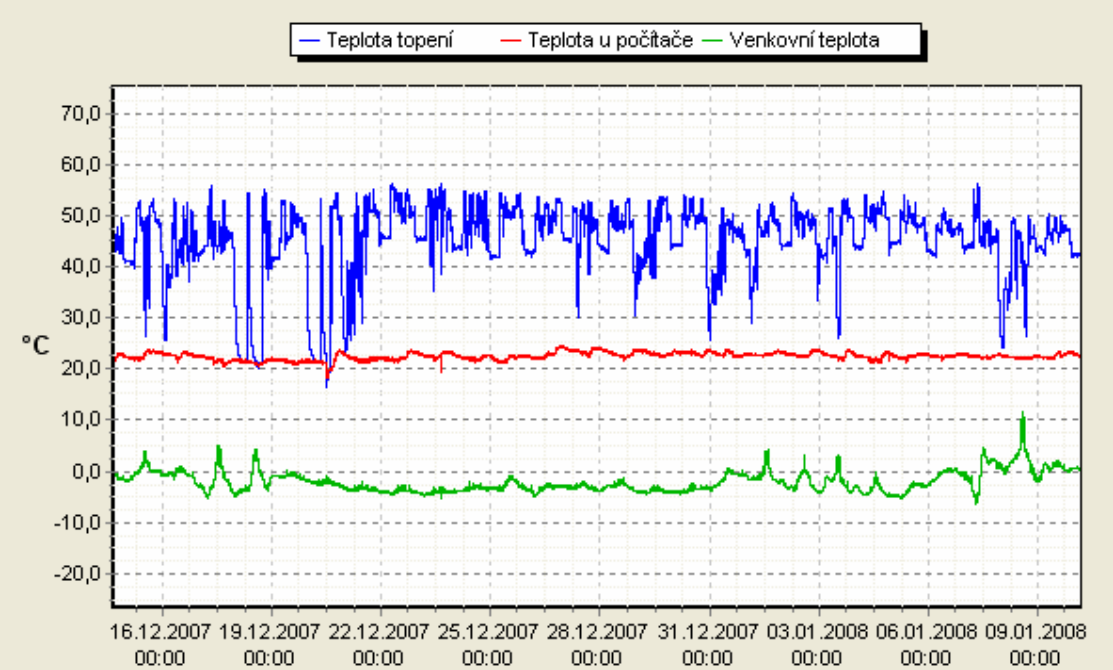
Změnu vzorkovací periody je možné provést jen v pokud je aplikace připojena k FITkitu. Před samotnou změnou však doporučuji stáhnout naměřená data, poněvadž změna způsobí také vymazání dosud zaznamenaných dat. Vzorkovací periodu je možné vybrat jak z přednastavených možností, tak definovat svoji vlastní – v rozsahu datového typu **`WORD`**. Zároveň se při každé změně počítá předpokládána maximální doba záznamu.

6.2.8 Zhodnocení naměřených dat během zkušebního provozu

Pro testování jsem použil 3 teplotní čidla připojené na kanály 0, 6 a 7. Teplotní čidlo na kanálu 0 bylo umístěno v místnosti blízko FITkitu a měřilo pokojovou teplotu. Čidlo na kanálu 6 bylo umístěno venku a teplota byla někdy ovlivňována přímým slunečním zářením. Poslední čidlo na kanálu 7 jsem připevnil k přívodní trubce od ústředního topení a měřilo teplotu vody v topení panelového bytu. Senzory na kanálu 6 a 7 byly připojeny k FITkitu plochým 4žilovým telefonním kabelem o délce 12 m, kanál 6 pak ještě pokračoval další 4 m do venkovních prostor. Měření probíhalo po dobu 30 dní se vzorkovací periodou 10 s.

Za tuto dobu došlo ke zpoždění hodinového signálu o 30 sekund. Časový průběh ze všech tří kanálů je zobrazen na obr. 6.8. Největších změn dosahuje modrý průběh z kanálu 7 – teplota topení. Naopak pokojová teplota (teplota u počítače) po celou dobu měření oscilovala kolem hodnoty

22,5 °C. Průběh venkovní teploty byl zase ovlivněn 14 dní trvajícím inverzním charakterem počasí, takže teplota se pohybovala jen malém rozmezí kolem -4°C.



Obr. 6.8 Časový průběh teploty během 30 dní

Během testování se vyskytla jedna chyba, kdy došlo k prudkém poklesu teploty. Změna byla patrná zejména na kanálu 7, kdy teplota klesla z 55,5 °C na hodnotu 38,8 °C a následně se vrátila zpět. Tato událost nastala 23.12.2007 v 15:10:34. Pokles teploty se projevil i na ostatních dvou kanálech, ale v daleko menším měřítku. Příčinou tohoto poklesu teploty by pravděpodobně mohla být nějaká napěťová porucha v elektrické rozvodné síti.

7 Závěr

Práce byla složitá v tom, že bylo potřeba naprogramovat tři různé věci ve třech naprosto odlišných prostředích a architekturách – FPGA, MCU a PC. S takto rozsáhlou problematikou jsem se doposud nesetkal, poněvadž drtivá část projektů na fakultě byla zaměřena jen na programování aplikací pro osobní počítače. Projekt tohoto typu by byl vhodný pro tým 3-4 lidí, kdy by se každý člen týmu naplno věnoval své oblasti a čtvrtý člen by mohl projekt řídit a psát dokumentaci. Tím by se podstatně zkrátila doba vývoje.

FITkit je složité zařízení a na jeho pochopení je potřeba mnoho času. Avšak bez praktického vyzkoušení nemůže člověk pochopit všechny jeho části a způsob komunikace mezi nimi. Teprve při sestavování větších celků z ukázkových aplikací se ukazuje, jak mnoho věcí spolu souvisí a vzájemně se ovlivňují. FITkit je opravdu dobrým a vhodným přípravkem pro výuku HW předmětů, protože obsahuje velké množství různých periferií na malé ploše. Navíc jednoduché připojení přes USB sběrnici dává FITkitu dlouhodobou perspektivu z hlediska konektivity k modernějším počítačům.

V oblasti programování mikrokontroléru mě asi nejvíce zarazila velmi nízká podpora standardních funkcí jazyka C a také výsledky některých matematických operací, které byly více než zarážející. Proto se musely všechny výpočty složitě ověřovat. Taktéž zcela chyběla podpora pro ladění programu na kterou jsem zvyklý u programovaní aplikací pro osobní počítače.

V jazyce VHDL jsem se setkal s velmi zásadním rozdílem mezi zdrojovým kódem pro simulaci obvodů a syntetizovatelným zdrojovým kódem pro skutečný hardware. Aby šla z VHDL kódu vygenerovat funkční konfigurace pro FPGA, vyžaduje to mnoho zkušeností a navíc řadu konstrukcí nelze vůbec použít.

Diplomová práce mně určitě rozšířila obzor o možnostech konfigurovatelného HW, ale jeho programování přenechám raději jiným odborníkům a budu se věnovat programování mikroprocesorů nebo aplikací pro osobní počítače.

Z hlediska dalšího rozšíření projektu by bylo možné rozšířit počet připojitelných senzorů, zavést kompresi hodnot, přidat reakce na dosažení teplotních limitů – např. řízení klimatizace. Nebo na VGA výstupu zobrazovat průběh teploty ze všech kanálů. Vylepšit by se dala hlavně aplikace na PC, která by mohla měřená data přenášet na webový server a umožnit tak dálkový monitoring teploty přes internetový prohlížeč.

Aplikaci by bylo možné prakticky využít např. pro monitorování teplot serverů a jejich částí přímo na fakultě nebo také pro monitorování teplot v mrazících a chladících pultech obchodních řetězců.

Literatura

[1] FITkit, url: http://merlin.fit.vutbr.cz/FITkit/

[2] Přednášky předmětu NAV, přednášející: Růžička, R., Martínek, T., Kořenek, J., 2005/2006, url: https://www.fit.vutbr.cz/study/courses/NAV/private/

[3] Halliday D., Resnick R., Walker J.: FYZIKA. ISBN 80-214-1869-9

[4] Hauck S., Dehon A.: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. ISBN 0123705223

[5] Odkaz na katalogový list SMT 160-30 http://www.omnitron.cz/pdf/SMT_160-30_CZ.pdf

[6] Informace o SMT 160-30, url: http://www.hw.cz/Produkty/Nove-soucastky/ART391-Prevodnik-teplota-strida-SMT160-30-92.html

[7] Pinker J., Poupa M.: Číslicové systémy a jazyk VHDL. ISBN 80-7300-198-5

[8] MPSGCC kompilátor, url: http://sourceforge.net/project/showfiles.php?group_id=42303

[9] Ovladače pro čip FT2232C, url: http://www.ftdichip.com/Drivers/VCP.htm

[10] HW server USB 2.0, url: http://www.hw.cz/Rozhrani/ART1232-USB-2.0---dil-1.html

[11] Přednášky předmětu KKO, přednášející: Drábek, V., 2005/2006, url: http://www.fit.vutbr.cz/study/courses/KKO/private/

[12] Komponenta Cport v3.10 pro C++ Builder, url: http://sourceforge.net/projects/comport/

[13] Katalogový list MSP430F168, url: http://www.ti.com/lit/gpn/msp430f168

[14] Katalog elektronických součástek GME, url: http://www.gme.cz

[15] Wikipedia - Termočlánek, url: http://cs.wikipedia.org/wiki/Termo%C4%8Dl%C3%A1nek

Seznam příloh

Příloha 1. Podrobné blokové schéma FITkitu

Příloha 2. Katalogový list teplotního senzoru SMT 160-30

Příloha 3. Katalogový list teplotního senzoru KTY81-210

Příloha 4. CD s programem a zdrojovými kódy