TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Řídící a komunikační software geofyzikální ústředny

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Matěj Kolář**
Vedoucí práce: Ing. Lubomír Slavík Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Control and communication software of geophysical control unit

Diploma thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Matěj Kolář**
Supervisor: Ing. Lubomír Slavík Ph.D.

Liberec 2014

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2013/2014**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Matěj Kolář**
Osobní číslo:	**M11000238**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Řídicí a komunikační software geofyzikální ústředny**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Prostudujte strukturu procesorů ARM-Cortex M3 a M4.
2. Realizujte firmware geofyzikální ústředny (GU100)
   a) komunikační interface mezi měřicími moduly a motherboardem GU100,
   b) rozhraní pro konfiguraci, ovládání a lokální ukládání dat,
   c) pro komunikaci s nadřazeným systémem implementujte protokol Fiedler.
3. Ověřte firmware v reálném provozu.

Rozsah grafických prací:	**dle potřeby dokumentace**
Rozsah pracovní zprávy:	**40–50 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Praktické zkušenosti s procesory ARM Cortex M3, URL: http://www.hw.cz**

[2] **Mann, B.: C pro mikrokontroléry, BEN 2003, ISBN 80-7300-077-6**

[3] **Dokumentace výrobků firmy Fiedler s.r.o. URL: http://www.fiedler-magr.cz/**

[4] **Katalogové listy obvodů STM32F407, DP83848, FT230XQ, DS1337.**

Vedoucí diplomové práce:	**Ing. Lubomír Slavík, Ph.D.** Ústav mechatroniky a technické informatiky
Konzultant diplomové práce:	**doc. Ing. Milan Hokr, Ph.D.** Ústav nových technologií a aplikované informatiky
Datum zadání diplomové práce:	**10. října 2013**
Termín odevzdání diplomové práce:	**16. května 2014**

V. Kopecký

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

Kolář

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2013

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Abstrakt

Práce popisuje realizaci softwaru pro geofyzikální ústřednu. V úvodu představuje koncept celého zařízení a obecně popisuje hardware. Shrnuje hlavní vlastnosti platformy, pro kterou je software vyvíjen a zdůvodňuje použití programovacího jazyk C++. Dále popisuje jednotlivé hardwarové části z pohledu vývoje ovladačů. Popisuje základní principy důležitých prvků systému a jejich implementace (Ethernet, sběrnice RS485, microSD karta). Je popsán použitý operační systém freeRTOS a jeho rozhraní. Dále je popsán návrh vývoj vlastního softwaru (komunikační protokol, grafická knihovna) nebo použití softwaru třetích stran (TCP/IP stack, knihovna FAT32). V práci jsou vysvětleny použité algoritmy a popsány poznatky vzniklé během vývoje.

Abstract

This work describes the implementation of software for geophysical data logger. The introduction presents the concept of the entire facility and generally describes the hardware. It summarizes the main features of the platform for which the software is developed and justifies the use of the programming language C++. It also describes the various hardware parts from the perspective of the drivers. It describes the basic principles of critical system components and their implementation (Ethernet, RS485, microSD card). It describes the operating system freeRTOS and its interface. The following describes a proposal for the development of custom software (communication protocol, a graphics library) or using third-party software (TCP/IP stack, library FAT32). The paper used algorithms are explained and described findings arising during development.

Poděkování

Chtěl bych poděkovat své rodině za to, že mě podporovali. Dále Ing. Lubomíru Slavíkovi Ph.D. za spolupráci na konceptu celého zařízení, návrhu hardwarových částí a za konzultace k různým částem řešení. Dále pak Ing. Milošovi Hernychovi a doc. Ing. Milanovi Hokrovi za konzultace a nápady ke koncepci řešení. Také bych chtěl poděkovat doc. RNDr. Pavlu Satrapovi, Ph.D. za vytvoření šablony pro LaTeX .

Obsah

Seznam obrázků

Seznam zkratek

Embedded	Vestavný
SoC	Systém integrovaný na jednom čipu
DPS	Deska plošných spojů
CPU	Mikroprocesor
MCU	Mikrokontroler
RTOS	Operační systém reálného času
Thread-safe	bezpečné z hlediska více–úlohových systémů
Wrapper	Návrhový vzor – upravuje stávající rozhraní knihovny na jiné
Virtuální funkce	Funkce, volaná ukazatelem – polymorfismus
Scope	Oblast ve zdrojovém kódu, ve které mají platnost lokálně deklarované proměnné
Stack	Zásobník
Heap	Halda
malloc	C funkce pro alokaci paměti
free	C funkce pro dealokaci paměti
GUI	Grafické uživatelské prostředí
CRC	Kontrolní součet

Úvod

Úkolem práce je software pro nově vyvíjený funkční model univerzálního měřicího celku pro on-line monitoring geofyzikálních jevů a procesů. Systém se skládá ze sítě měřicích ústředen. Každá ústředna pak slučuje měřicí moduly různých geofyzikálních veličin. Součástí ústředny je záložní zdroj energie a lokální uživatelské rozhraní. Důraz je kladen na minimální spotřebu celého systému.

Na úkolu spolupracuji s Ing. Lubomírem Slavíkem Ph.D., který navrhuje hardwarovou část systému. Celé zařízení je vyvíjeno jako co možná nejuniverzálnější. Systém by měl být nasazen v rámci projektu dlouhodobého sledování geologických jevů v reálném prostředí granitů českého masivu pro bezpečnostní výpočty a pro návrh požadavků, indikátorů a kritérií na výběr vhodného prostředí pro hlubinné úložiště jaderného odpadu v České republice, který probíhá v bedřichovském tunelu a je veden doc. Ing. Milanem Hokrem Ph.D. Ústředna by měla sjednotit již probíhající systém měření.

Požadavky na měřicí systém jsou jeho modulárnost, jednotná koncepce pro všechny druhy měření, velký počet měřených veličin, vzdálený přístup a sběr dat, lokální zálohování dat, uživatelské rozhraní pro lokální zásah do měřicího procesu a spotřeba energie odpovídající možnosti provozu na baterie. Důležitým požadavkem je také provoz zařízení v nepříznivých podmínkách. V tunelu je kolem 10 °C a 100% vlhkost.

Komerčně dostupná řešení nabízejí požadované vlastnosti, ale ne všechny současně.

Zařízení v této aplikaci bude komunikovat s nadřazenou řídící jednotkou vyvíjenou ing. Milošem Hernychem a databázovým systémem pro sběr všech dat v řídícím centru Technické univerzity v Liberci.

Předmětem diplomové práce je několik softwarových částí. Komunikační interface mezi měřicími moduly a základní deskou, uživatelské rozhraní a komunikace s nadřazeným a vzdáleným systémem.

V první kapitole jsou popsány základní části měřicí ústředny a několik měřicích modulů, včetně základních principů měření. Druhá kapitola je úvodem do možností vývoje pro cílovou platformu. Zdůvodňuje výběr programovacího jazyka a uvádí jeho hlavní výhody a nedostatky, dále popisuje vývojové prostředí. Třetí kapitola konkrétněji popisuje jednotlivé hardwarové části systému a to zejména z pohledu

softwaru a vývoj ovladačů pro tyto periférie. Kapitola čtvrtá se zabývá operačním systémem, důvody jeho použití a základními vlastnostmi. Další kapitola se věnuje všem důležitým softwarovým částem.

Kapitola Koncepce a Vývojové prostředky je orientovaná spíše teoreticky. Kapitola Operační systém popisuje freeRTOS a zároveň představuje C++ wrapper, který vznikl v rámci práce. Kapitoly Hardware a Software popisují výsledky této práce (návrhy softwarového řešení, vyvinutý software a spolupráce na návrhu hardware), přičemž jsou u každé části uvedeny obecné informace pro pochopení základních souvislostí.

1 Koncepce systému

Celé zařízení můžeme rozdělit do několika částí:

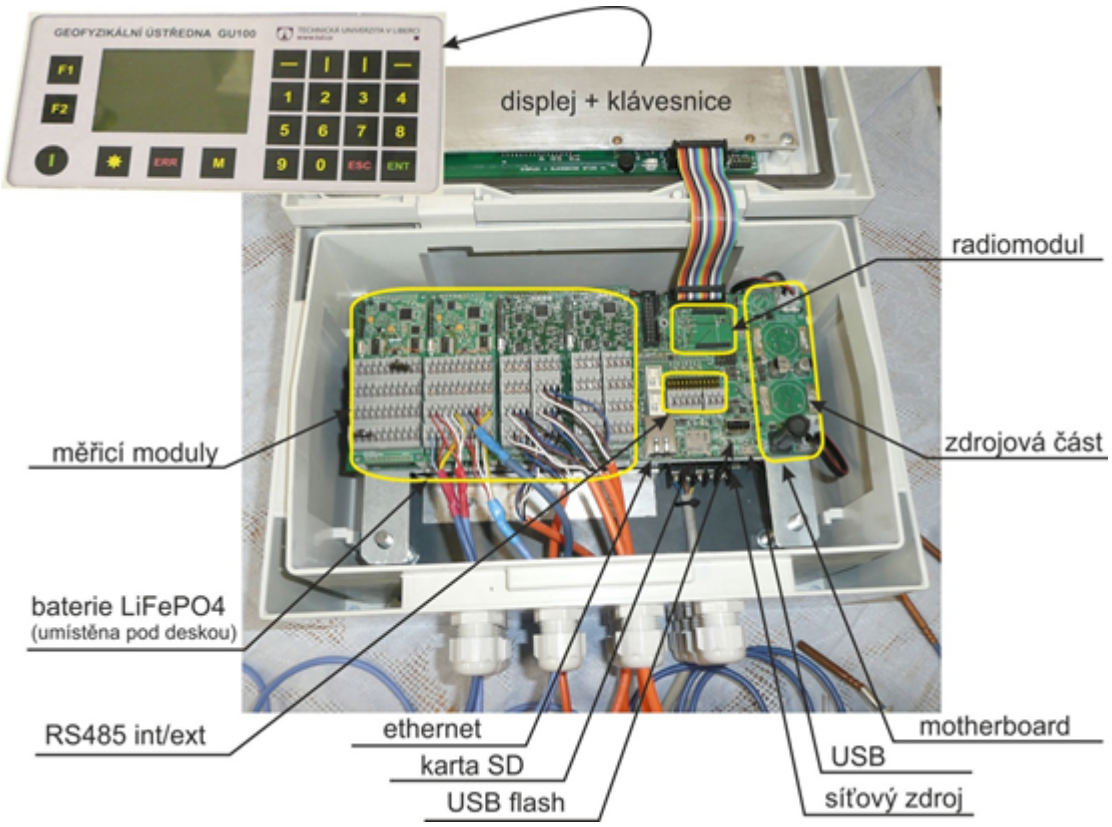
1.1 Základní deska

Základní deska obsahuje:

- hlavní mikroprocesor,
- napájecí zdroj,
- rozhraní pro komunikaci s nadřazeným (vzdáleným) systémem,
- rozhraní pro připojení měřicích modulů,
- rozhraní pro archivaci dat.

K dispozici na základní desce jsou dvě sběrnice RS485 (jedna bez galvanického oddělení, druhá s galvanickým oddělením), bezdrátové rozhraní, rozhraní Ethernet, USB host i slave a slot pro microSD kartu. Základní deska by měla řídit a spravovat celý proces měření. Měla by vyčítat data z měřicích modulů, vytvářet lokální databázi, poskytovat data nadřízenému (vzdálenému) systému. Prostřednictvím jedné sběrnice RS485 by zařízení mělo komunikovat s měřicími moduly, druhá sběrnice by měla být určena pro nadřazený systém, který bude měřicí ústřednu ovládat. Rozhraní Ethernet by mělo sloužit pro komunikaci se vzdáleným systémem pomocí TCP/IP protokolu. MicroSD karta by měla sloužit k ukládání lokální databáze a nastavení celého systému.

Další úkol hlavního mikroprocesoru je kontrola nad spotřebou energie celého zařízení. Na základní desce je několik spínaných zdrojů, které poskytují napájecí napětí 3,3 V, 12 v a 24 V. Všechna napájecí napětí je možné zapínat a vypínat na základě rozhodnutí hlavního procesoru nebo žádostí měřicích modulů. Stejně tak má hlavní procesor možnost probouzet moduly a naopak. Všechna tato opatření by měla dát dostatek hardwarových prostředků pro maximální možnou úsporu energie. Například asynchronním snímáním stavů čidel a probouzením pouze potřebných bloků na minimální dobu.



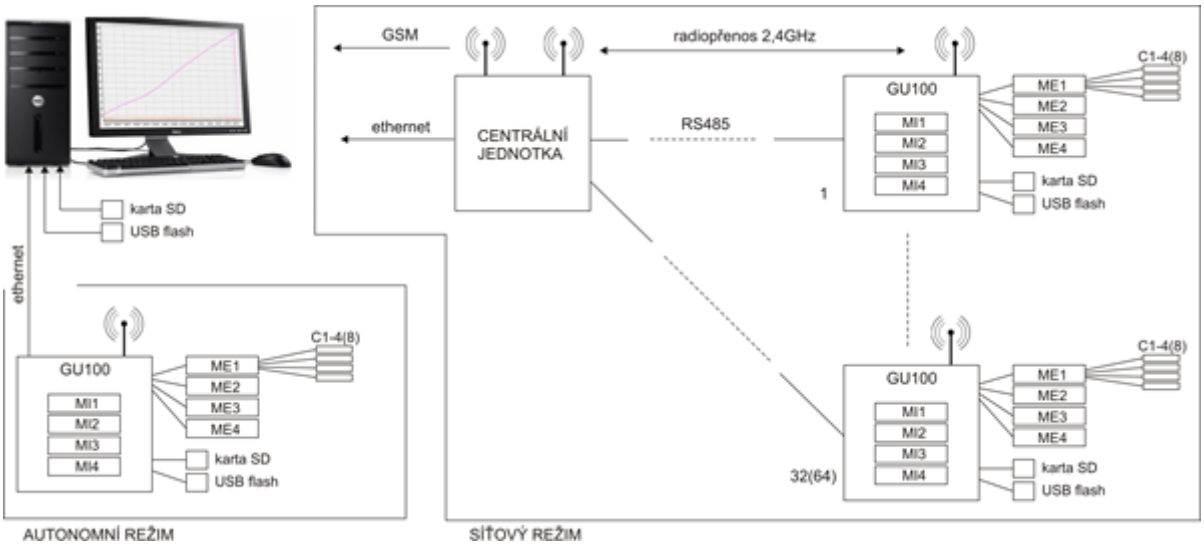
Obrázek 1.1: Měřicí ústředna.

1.2 Uživatelské rozhraní

Uživatelské rozhraní je realizováno grafickým displejem a podsvícenou kapacitní klávesnicí. Mělo by umožnit základní parametrizaci systému a odečítání aktuálních hodnot měření a stavů. Základní požadavek je možnost ovládání v úplné tmě a vlhkém prostředí s minimální spotřebou energie.

1.3 Napájení

Zařízení lze napájet ze zdroje 13,6 V (230 V / 13,6 V nebo 24 V / 13,6 V) s možností napájení zálohovat 12 V olověnou baterií. Druhá možnost je napájení z baterie. Pro větší výdrž a minimální samovybíjení byla zvolena baterie LiFePO4. V případě připojení na sběrnici RS485 se předpokládá trvalé napájení a bateriové se využije při bezdrátovém přenosu dat.



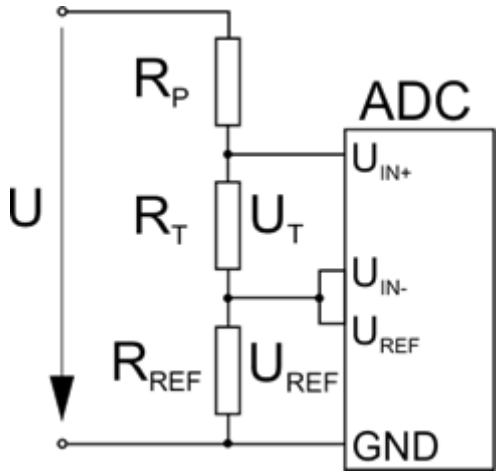
Obrázek 1.2: Schéma systému.

1.4 Měřicí moduly

Měřicí moduly jsou osazeny vlastním procesorem a komunikují společným rozhraním RS485. Moduly jsou interní, na základní desce jsou pro ně k dispozici čtyři pozice. Dále mohou být moduly externí. Obě varianty budou sdílet jednu sběrnici. Interní moduly mají navíc k dispozici napájecí napětí ze základní desky (3,3 V, 12 V a 24 V) a kontrolní signály pro probuzení hlavního procesoru nebo procesoru na modulu. Externí modul bude mít k dispozici pouze 12 V napájecí napětí. Výhodou společné sběrnice RS485 (i když pro komunikaci na jedné DPS poněkud nestandardní) je kompatibilita software a do určité míry i hardware obou variant modulů.

1.4.1 Teplotní modul

Modul je určen pro měření osmi teplot pomocí teplotních čidel Pt1000, zapojených 4vodičově, je tedy eliminován vliv přívodních vodičů. Princip měření spočívá v zapojení odporového děliče dle obrázku.



Obrázek 1.3: Princip měření teploty.

Dělič je napájen ze zdroje 3,3 V, rezistor R_p je pracovní odpor, v našem případě 8,2 $k\Omega$, který zajišťuje stabilní pracovní proud. Rezistor R_T je odpor měřicího čidla Pt1000 a rezistor R_{REF} (1,5 $k\Omega$) je referenční odpor, který určuje přesnost a stabilitu měření. Následuje analogově-číslicový převodník.

Pracovní proud děličem lze určit ze vzorce:

$$I_P = \frac{U}{R_P + R_T + R_{REF}}$$

Při předpokládaném rozsahu teplot 0 – 130°C se pracovní proud pohybuje v rozmezí 295 – 308 μA.

Napěťový signál U_{REF} je přiveden na referenční napětí analogově-číslicového převodníku a tedy úbytku napětí na R_{REF} odpovídá maximální převodní kód, tedy 65535. Hodnotu odporu teplotního čidla pak lze vypočítat dle vzorce

$$R_T = R_{REF}\frac{D_T}{2^16}$$

kde R_{REF} je hodnota referenčního odporu a představuje maximální měřenou teplotu 130 °C, D_T je hodnota výstupu z AD převodníku.

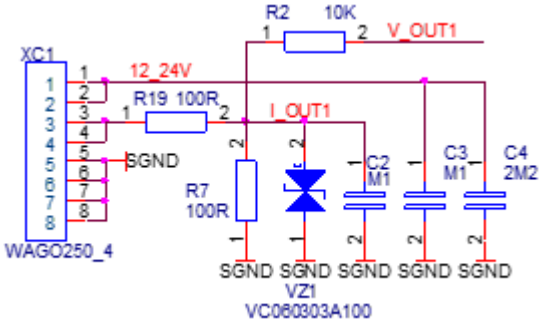
Pro měření osmi teplotních čidel jsou vloženy do zapojení dva multiplexery, aby bylo možno přepínat mezi měřeními jednotlivých čidel.

Přesnost měření je kromě přesnosti teplotních čidel dána přesností a stabilitou referenčního odporu a linearitou AD převodníku. Nelinearita AD převodníku je 3,5 ppm, což se v podstatě neuplatní. Odpor rezistoru má toleranci 0,01 % a teplotní stabilitu 5 ppm/K. Výchozí chyba elektroniky je tedy ±0,025 °C, přičemž v předpokládaném teplotním rozsahu elektroniky 0-50 °C dosáhne chyba elektroniky ±0,031 °C. V případě použití teplotních senzorů Pt1000 třídy A, která zaručuje chybu ±0,15 °C, bude tedy celková chyba měření v celém teplotním rozsahu elektroniky ±0,18 °C.

1.4.2 Modul pro proudovou smyčku 4-20 mA

Modul je určen pro zpracování signálu z proudové smyčky 4-20 mA.

Proud jdoucí z proudové smyčky je přiveden na ochranný odpor 100 Ω a do série zapojený měřicí odpor 100 Ω. Paralelně s měřicím odporem je zapojen ochranný transil a kondenzátor 100 nF pro redukci šumu.



Obrázek 1.4: Zapojení modulu pro proudovou smyčku.

Napěťový signál z měřicího odporu je veden přes ochranný odpor 10 $k\Omega$ do multiplexeru, který vybírá jeden z osmi měřicích kanálů (piny B8 – B10). Z výstupu multiplexeru je signál veden na 16bitový AD převodník (AD7790). Procesor komunikuje s AD převodníkem po sběrnici SPI. Referenční napětí je generováno napěťovou referencí 2,5 V (ADR3425). Vstupní rozsah je tedy 0 – 2,5 V, což odpovídá vstupnímu proudu 0 – 25 mA.

Přesnost měření je dána přesností a stabilitou měřicího odporu a napěťové reference. Dále se uplatní i integrální a diferenciální nelinearita AD převodníku. Odpor má přesnost 0,01 % a teplotní stabilitu 5 ppm/K. V celém předpokládaném rozsahu (5 – 50 °C) způsobí celkovou chybu měření 0,035 %. Nelinearita AD převodníku je 3,5 ppm, což se v podstatě neuplatní. Přesnost napěťového referenčního obvodu je 0,1 % a teplotní závislost je 8 ppm/°C. Vliv reference v celém teplotním rozsahu je tedy 0,14 %.

Celková teoretická přesnost měření je tedy 0,18 %. V případě výpočtu nejistoty měření předpokládáme rovnoměrné rozložení výskytu měřené veličiny ($Q = \sqrt[2]{3}$) a

koeficient rozšíření 2. Například pro měření proudové smyčky 20 mA vychází následující nejistota:

$$\Delta I = 36\mu A$$
$$U(I) = 2\frac{\Delta I}{\sqrt[2]{3}} = 42\mu A$$

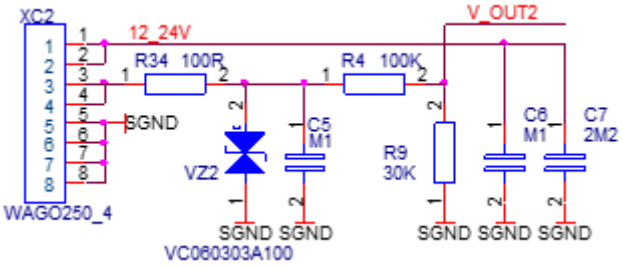
Pro napájení proudové smyčky je využito napájení základní desky, a to buď 12 V nebo 24 V. Výběr napětí je proveden pomocí jumperu. Použité napětí je indikováno na pinu procesoru B1.

1.4.3 Modul pro napěťový vstup 0 – 10 V

Modul je určen pro zpracování signálu 0 – 10 V.

Vstupní napětí je přivedeno na ochranný RC člen (100 Ω a 100 nF) a transil. Poté je signál přiveden na odporový dělič (100 $k\Omega$ / 30 $k\Omega$). Z děliče je signál přiveden na osmi vstupový multiplexer. Odtud je signál veden na AD 16bitový převodník (AD7790). Převodník komunikuje po sběrnici SPI. Referenční napětí 2,5 V pro AD převodník generuje obvod ADR3425.

Paralelně s měřicím odporem je zapojen ochranný transil a kondenzátor 100 nF pro redukci šumu.



Obrázek 1.5: Zapojení modulu pro napěťový vstup.

Napěťový signál z měřicího odporu je veden přes ochranný odpor 10 $k\Omega$ do multiplexeru, který vybírá jeden z osmi měřicích kanálů (piny B8 – B10). Z výstupu multiplexeru je signál veden na 16bitový AD převodník (AD7790). Procesor komunikuje s AD převodníkem po sběrnici SPI. Referenční napětí je generováno napěťovou referencí 2,5 V (ADR3425). Vstupní rozsah je tedy 0 – 2,5 V, což odpovídá vstupnímu napětí 0 – 10,8 V.

Přesnost měření je dána přesností a stabilitou odporového děliče, napěťové reference a nelinearitou AD převodníku.

Odporový dělič je možno vyrobit s přesností 0,01 % a teplotní stabilitou 1 ppm/K. V celém předpokládaném rozsahu (5 – 50 °C) způsobí celkovou chybu měření

0,015 %. Nelinearita AD převodníku je 3,5 ppm, což se v podstatě neuplatní. Přesnost napěťového referenčního obvodu je 0,1 % a teplotní závislost je 8 ppm/°C. Vliv reference v celém teplotním rozsahu je tedy 0,14 %. Celková teoretická přesnost měření je 0,16 %. Předpokládáme rozložení výskytu měřené veličiny ($Q = \sqrt[2]{3}$) a koeficient rozšíření 2.

Například pro měření napětí 10 V vychází následující nejistota:

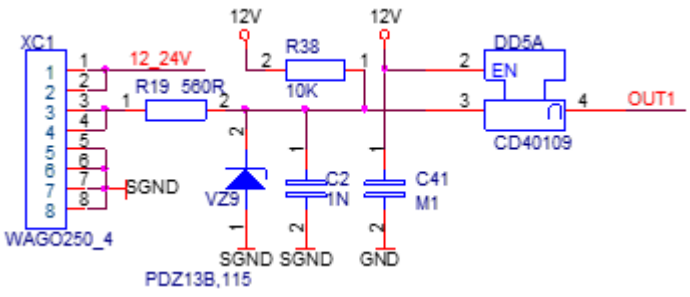
$$\Delta U = 16mV$$
$$U(U) = 2\frac{\Delta U}{\sqrt[2]{3}} = 19mV$$

Pro napájení čidla, které generuje signál 0 – 10 V je využito napájení základní desky, a to buď 12 V nebo 24 V. Výběr napětí je proveden pomocí jumperu. Použité napětí je indikováno na pinu procesoru B1.

1.4.4 Modul pro pulzní vstupy

Modul je určen pro zpracování pulzního vstupu, případně pro měření frekvence vstupního signálu. Tento modul může být použit pro měření průtoku (průtokoměry s pulzním výstupem), měření průtoku pomocí sklopek, nebo pro čítání pulzů (např. z optických závor).

Vstupní signál (aktivní je log. 0) je veden přes ochranný člen (R, Zenerova dioda, C) na vstup převodníku úrovně 12 V / 3,3 V (CD40109). Zařízení může být napájeno 12 nebo 24 V (doporučeno 12 V) a výstupní signál může být buď aktivní (generuje signál) nebo pasivní, kdy je vstupní signál napájen z elektroniky modulu a zařízením přerušován. Po konverzi napěťové úrovně je signál veden přímo na piny procesoru (A1 – A8). Všechny signály jsou zároveň vedeny na osmi kanálové hradlo NAND. Výstupní signál z hradla je veden na vstup WKUP2. Pomocí tohoto vstupu je možné procesor probudit.



Obrázek 1.6: Zapojení modulu pro pulzní vstup.

2 Vývojové prostředky

2.1 Embedded system – Vestavěný systém

Zařízení vyvíjené v rámci této práce je embedded systém, který částečně vyžaduje real–time přístup. Je postavený na SoC mikrokontrolérech s omezenými paměťovými prostředky (řádově 10-100 kB RAM). Řešení je optimální kompromis mezi požadovanými schopnostmi a minimální spotřebou a cenou. Nevýhodou oproti plnohodnotnému OS je více práce s ovládáním hardware. Výhodou je menší závislost na produktech třetích stran, lepší optimalizace a využití prostředků.

Systém by teoreticky šel provozovat bez OS, nicméně vzhledem k rozmanitosti měření a činností, které se s časem můžou rozšiřovat, je vhodné operační systém použít. Poměrně náročnou úlohou je i podpora TCP/IP protokolu, který je vhodné provozovat s OS.

2.1.1 Embedded system

Embedded systém je jednoúčelový systém, určený pro předem daný účel. Systém obsahuje pouze nutné prvky ke své funkci. Není univerzální a je pevně zabudovaný v dané aplikaci. Hlavní výhodou je optimalizace celého sytému z hlediska ceny, výkonu, spolehlivosti a bezpečnosti. Předpokladem je i provozuschopnost bez zásahu uživatele a možnost masové výroby.

Embedded systém může v zásadě představovat standardní PC s operačním systémem Windows (Linux) a specializovaným softwarem, nebo také jenom jedno–čipový 8–bitový mikroprocesor s pamětí RAM pod 1 kB. Existují i speciální verze standardních operačních systémů jako například Windows Embedded 7. Embedded systémem je například bankomat, navigace, řídící jednotka automobilu, ale například i pračka.

2.1.2 Real–time operating system

Operační systém reálného času je systém, který zaručuje okamžité (včasné) reakce na události. Systémy nemusejí být extrémně rychlé, ale doby reakcí musí být předem

známé. Využívají se zejména v embedded systémech, robotice a automatizaci.

Podstatnou funkcí operačního systému je multi–tasking. Schopnost systému vykonávat několik úloh souběžně (rychle je střídat). Systém úlohy střídá takzvaným přepínáním kontextu. Přepnutí probíhá tak, že se uloží aktuální kontext – stack (respektive stack–pointer), program–counter a registry procesoru a nahradí se již dříve uloženými daty, čímž bude pokračovat dříve přerušená úloha. Analogicky se zase přepne zpět.

Souvisejícím problémem je souběh více úloh přistupujících ke stejným prostředkům (HW, data). Příkladem může být situace, kdy příkaz inkrementace proměnné i++ mají ve stejnou chvíli vykonat dvě úlohy. Operace není atomická (taková, která je vykonaná bez přerušení – jedna instrukce procesoru), rozdělí se na dvě operace. Načtení proměnné a uložení čísla o jedna větší do proměnné. Problém nastane, pokud jedna úloha číslo načte a bude přerušena, druhá úloha načte stejné číslo a pak obě úlohy číslo zvětší o jedna a uloží je. Výsledkem bude číslo větší pouze o jedna, přitom inkrementace proběhla dvakrát.

Nejjednodušším řešením je neatomické operace provádět se zakázaným přerušením, ale takové řešení lze použít jen v ojedinělých případech, protože to může mít nepříznivý dopad na chod systému. Zpravidla přerušení zakazuje jen samotný operační systém, například při synchronizaci přístupu ke správě paměti nebo v obsluze takzvaných synchronizačních primitiv.

Synchronizační primitiva jsou dostupná rozhraní operačního systému, které slouží k synchronizaci úloh. Více v odstavci 4.2.

2.2 Programovací jazyk

Software je napsán v jazyce C++. V embedded aplikacích pro MCU není v podstatě jiná možnost, než zvolit jazyk C a případně jeho nadstavbu. V následujícím textu budou popsány hlavní rozdíly mezi jazyky a důvody, které vedly k rozhodnutí pro C++.

Jazyky jako Java nebo C# (obecně jazyky vyžadující běhové prostředí) jsou problematické velkou paměťovou náročností. Většina obdobných aplikací zabere méně paměti (programové i datové), než by zabral samotný runtime pro běh programu v takovém jazyce. Existují projekty, které umožňují použít Javu i C# na MCU, ale spotřebují poměrně velkou část paměti RAM. Dalším problémem je garbage collector a i samotná dynamická alokace paměti, která je pro časově kritické systémy problematická, protože je nedeterministická.

Jazyky jako jsou Pascal, Fortran, Ada je možné použít pro embedded i časově kritické aplikace. Například NASA používá (používala) Fortran i Adu. Problémem je malá rozšířenost a omezená kompatibilita ostatního softwaru, který je psaný v drtivé většině v jazyce C.

2.2.1 Jazyk C

Jazyk C je obecně považován za vhodný pro embedded i časově kritické aplikace stejně jako pro nízko–úrovňové programování (je schopen téměř zastoupit assembler). Je považován za ostrý meč, který je velmi mocným nástrojem, ale při špatném zacházení je velmi nebezpečný, programátorům dovolí téměř vše. Častým problémem je špatný přístup do paměti a paměťové úniky. C je strukturovaný programovací jazyk. Autorem je Dennis Ritchie a vznikl v roce 1972, poslední verze vyšla v prosinci 2011 – C11. Jazyk ObjectiveC je rozšířením jazyka C o objektově orientované programování.

2.2.2 Jazyk C++

Jazyk C++ rozšiřuje C a je považován za jeden z nejkomplexnějších, ale zároveň nejsložitějších jazyků. Jedná se o multiparadigmatický programovací jazyk. Umožňuje strukturované, objektově orientované i generické programování. Jazyk vyvinul v roce 1983 Bjarne Stroustrup. V jazyce lze programovat na stejně nízké úrovni jako v C, ale zároveň umožňuje vysokou míru abstrakce, typickou například pro Javu.

Základními prvky C++ je dědičnost, polymorfismus, šablony a výjimky. Na rozdíl od Javy nemá C++ garbage collector, ale využívá systém konstruktorů a destruktorů, který dokáže garbage collector ve většině případů nahradit. Jedná se ale o více deterministický přístup, který lze vývojářem přizpůsobit.

Pokud při srovnání budeme považovat C za jednostranný ostrý meč, kterým se při špatném zacházení můžeme pořezat, jazyk C++ bude oboustranný meč, kterým si rovnou usekneme ruku i nohu. V jazyce C++ nejde udělat chybu tak snadno jako v jazyce C, ale pokud se to povede, jsou důsledky daleko horší. Problémem je abstrakce, která při špatném použití může vytvářet nepřehlednost. Příkladem může být kromě poměrně standardního přetěžování funkcí, také možnost přetížit libovolný operátor. V C++ tedy není problém, použitím sčítacího znaménka odčítat. Velkou výhodou je možnost definovat funkci operátoru pro třídy, které nesou vícerozměrnou informaci, například souřadnice nebo komplexní číslo.

2.2.3 C++ v embedded aplikacích

Další otázkou je použití jazyka C++ pro embedded aplikace, kolem kterého existuje několik nepřesností. V průběhu vývoje docházelo ke špatným zkušenostem s překladači C++ z hlediska optimalizace kódu. Skončil tak například pokus přepsat jádro Linuxu z C do C++. V dnešní době jsou překladače na podstatně lepší úrovni, i když nejsou zcela bezchybné.

Hlavním důvodem, pro který není C++ nasazované častěji jsou vysoké nároky na vývojáře a v případě knihoven také problém se zpětnou kompatibilitou s jazykem

C. Dokonce vznikla varianta embedded C++, která zakazovala některé „nevhodné“ konstrukce. Problém byl, že zakazovala téměř vše (včetně částí, které s výkonem a paměťovou náročností neměly nic společného, například namespace). Členové výboru pro jazyk C++ na základě těchto tendencí začali zkoumat skutečné dopady na výkon a paměť [1]. Zde jsou jen některé:

- Namespace – Konstrukce vyhodnocená během překladu. Nemá žádný vliv na paměť ani výkon aplikace. Na druhé straně stojí fakt, že se dvě nestatické funkce v jazyce C v rámci jednoho projektu nesmí stejně jmenovat, což se v případě použití dvou nesouvisejících knihoven může stát.
- Třídy – Metody třídy definované jako static se z hlediska výkonu chovají stejně jako standardní funkce. Členské metody navíc potřebují ukazatel na kontext třídy – *this. V podobné situaci v jazyce C se předání kontextu stejným způsobem nevyhneme.
- Virtuální funkce a polymorfismus – Dopad na výkon může být znatelný, funkce je volána ukazatelem z tabulky virtuálních funkcí. Samotné volání ukazatelem bude znamenat o jednu instrukci navíc, ale funkci nelze vykonat jako inline a tedy maximálně optimalizovat. Problém se projeví při častém volání relativně krátkých funkcí. Druhá stránka věci je, že situace, kde se virtuální funkce použije, nelze v jazyce C vyřešit nijak efektivněji. Typická konstrukce switch case bude stát minimálně stejné prostředky.
- Inline funkce – C++ je poměrně typické in-linováním funkcí, čímž lze docílit i rychlejšího programu než v C. Dalším příkladem je přetěžování funkcí, kdy se již v době překladu rozhodne o použití funkce na základě typu argumentu. Využívá se toho například při formátovaném výstupu. Jazyk C (funkce printf) identifikuje typ parametru až za běhu programu.
- Šablony – Problém šablon je, že je pro každou instanci šablony vytvořena vlastní sada kódu, programová paměť tím samozřejmě narůstá. Měly by se používat s rozvahou, ale jsou bezpochyby velkým pomocníkem.
- Výjimky – Složitější je situace kolem výjimek. Obecně jsou považované spíše za nedeterministické a poměrně hodně náročné na prostředky. Otázkou zůstává, jestli mechanismus ošetření chyb bez výjimek lze dělat efektivněji. V této práci výjimky nebyly nepoužity z důvodů relativně velkých nároků na velikost stacku během vyvrhnutí výjimky.

Při větším projektu (případně projektu, kterému hrozí, že bude větší) použití C++ více než vhodné, alespoň kvůli namespace. V jazyce C je vývojář z důvodů absence jmenných prostorů nucen obalovat globální názvy funkcí, struktur i proměnných různými prefixy, například názvem knihovny. Celý kód se stává zbytečně „více upovídaný“ a hlavně nepřehledný. Navíc to stejně nezajistí, že ke kolizi nedojde.

Větší nároky na prostředky jsou diskutabilní. Existují konstrukce jazyka C++, které dokáží zajistit lepší výkon než obdobný kód v C. Navíc lze kdekoliv v C++ kódu použít i čisté C.

V každém případě je nutné držet se při zemi a nevytvářet příliš abstraktní a složité konstrukce. Problém lze snadno překombinovat a udělat složitější než je. Je potřeba zvolit tu správnou míru abstrakce, pak lze docílit velice přehledného a bezpečného kódu. Napomáhají tomu výhody jako je přetěžování funkcí a operátorů, lepší typová kontrola, přehlednější přístup k dynamické alokaci paměti a stejně tak i využití objektově orientovaného programování.

Použití výjimek je podle mě také v určitých případech v pořádku, z hlediska přehlednosti je to ideální technika a v mnoha situacích částečně nedeterministické chování nevadí. Určité úskalí je vyvrhnutí výjimky v konstruktoru objektu, kdy není jasné, jestli je objekt zkonstruován a má se volat destruktor nebo nikoliv. Jde spíše o speciální případy. Výjimky nebyly nepoužity pouze z výše uvedeného důvodu, tedy nedostatek paměti RAM. Chování během vyvrhnutí výjimky se nepovedlo úplně dobře analyzovat. Dochází k takzvanému odvíjení stacku (stack unwinding), během kterého se postupně volají destruktory objektů zanikajících opouštěním daného scope.

V každém případě velikost stacku musela být vždy větší než 2 kB, jinak vždy přetekl. Je pravděpodobné, že v reálné aplikaci by hranice byla ještě vyšší. Standardně si jednodušší úloha vystačí i s 512 B na stacku a proto výjimky použity nebyly. Hlavní procesor by měl paměti dostatek, ale menší procesory na měřicích modulech nikoliv. Kvůli zachování přenositelnosti kódu nebyly použity nikde. Bohužel s tím odpadla možnost použití větší části standardní knihovny C++ STL, která výjimky vyžaduje a požadované funkce musely být doplněny vlastní knihovnou.

2.2.4 Kódovací standard

Pro zápis programů byl zvolen kódovací standard „The Joint Strike Fighter Air Vehicle C++ Coding Standard“ (JSF AV C++). Ve zdrojových kódech není striktně dodržován, spíše je z něj převzatý formální zápis kódu a způsob pojmenovávání tříd, funkcí a proměnných.

2.3 Vývojové prostředí

2.3.1 IDE – vývojové prostředí

Zvolit vhodné vývojové prostředí byl podobný problém jako výběr procesoru. Hlavním požadavkem byla podpora procesorů ARM (konkrétně Cortex–M), programovací jazyk C/C++ a debug rozhraní. Mezi hlavní hráče na trhu, kteří nabízejí kom-

pletní IDE se jednoznačně řadí Keil a IAR. Oba mají vlastní C/C++ kompilátor. Mezi dalšími jsou například CrossWorks a Atollic, využívající GCC kompilátor. Alternativou jsou i prostředí vázané na výrobce MCU. Například LPCXpresso pro NXP.

Keil

- Vlastní kompilátor,
- RTOS,
- TCP/IP knihovna,
- USB Device a USB Host knihovny,
- GUI knihovna,
- analyzátor vykonávání programu, analýza výpočetního výkonu,
- verze do 32 kB programu zdarma,
- individuální ceny, řádově 5000 USD za základní IDE (bez knihoven a nástrojů).

IAR – IAR Embedded Workbench for ARM

- Vlastní kompilátor,
- kontrola MISRA C pravidel,
- RTOS,
- Nástroj pro optimalizaci spotřeby,
- verze do 32 kB programu zdarma,
- individuální ceny, řádově 5000 USD.

CrossWorkRowley Associates – CrossWorks for ARM

- GCC,
- Multi-tasking,
- TCP/IP,
- Mass Storage knihovna,
- cena komerční licence 2000 USD, možnost EDU verze i nekomerční licence.

Pro tuto práci byl zvolen nástroj CrossWorks kvůli ceně a standardnímu překladači.

2.3.2 Programátor a debugger

Požadavky na programátor a debugger jsou podpora procesorů ARM (konkrétně Cortex–M) a rozhraní JTAG a SWD. Existuje řada možností, včetně debuggeru s integrovaným GDB serverem, ke kterému lze přistupovat přímo přes LAN. Zajímavým nástrojem je debugger s podporou TRACE, který ukládá sekvenci vykonaných instrukcí před cílovým bodem v programu a lze je zpětně zrekonstruovat.

Pro tuto práci byl zvolen standardní J-Link od firmy SEGGER v provedení EDU.

SEGGER J-Link

- rozhraní JTAG a SWD,
- podpora ARM7/9/11, Cortex-A5/A8/A9, Cortex-M0/M1/M3/M4, Cortex-R4/R5,
- EDU verze.

3 Hardware

3.1 MCU

Stěžejní část celého projektu je mikroprocesor. Byla zvolena 32bitová architektura ARM Cortex M3/M4. Zejména z důvodu nezávislosti jádra na výrobci, jeho rozšířenosti a výkonu. Existuje velké množství procesorů od mnoha různých výrobců postavených na jádře ARM, ale vybrat konkrétní typ byl poměrně problém. Porovnáním možností, parametrů, podpory, cen a dostupnosti byla zvolena řada STM32 od firmy ST.

Systém je složen z více typů procesorů. Hardware je modulární a velmi rozmanitý, proto je k vývoji softwaru přistupováno obecně a abstraktně. Většina softwarových modulů, knihoven a ovladačů vytvořených v této práci byly vyvíjeny tak, aby mohly fungovat nezávisle (nebo s minimální úpravou) i na jiném procesoru. Přístup je výhodný svou kompatibilitou softwaru pro základní desku i moduly. Proto jsou i v následujícím textu jednotlivé prvky systému popisovány bez ohledu na tom, kde se v systému vyskytují. Obecně lze říci, že je popisován software všech vzniklých prvků současně. Výjimkou jsou části závislé na místě použití. Například Ethernet nebo microSD karta se týká pouze základní desky, naopak AD převodník používají pouze moduly.

3.1.1 MCU základní desky

MCU pro základní desku je SMT32F407.

SMT32F407

- Výkonný mikrokontrolér z řady STM32F4 s jádrem ARM Cortex M4.
- FPU – jednotka pro operace s plovoucí řádovou čárkou.
- ART Accelerator (adaptive real-time accelerator) – dovoluje vykonávání programu z flash paměti bez čekání.

- MPU – jednotka pro ochranu paměti.
- Taktovací frekvence 168 Mhz – 210 DMIPS. Až 1 MB paměti Flash a 192 kB RAM.
- DSP instrukce.
- Napájecí napětí 1,8 V to 3,6 V. Interní 16 Mhz oscilátor a 32kHz oscilátor pro RTC s kalibrací.
- RTC se zálohovanou částí paměti RAM.
- 3×12-bit AD převodník, až 24 kanálů, maximální rychlost 7,2 MSPS, 2×12-bit D/A převodník.
- 16 kanálový DMA řadič s podporou FIFO.
- Až 17 časovačů.
- Až 3× I2C rozhraní, až 4× USART, maximální rychlost 10,5 Mbit/s, až 3× SPI, maximální rychlost 42 Mbits/s, 2× CAN rozhraní.
- SDIO rozhraní.
- USB 2.0 high-speed/full-speed device/host/OTG.
- 10/100 Ethernet MAC.
- True random generátor náhodných čísel.

3.1.2 MCU měřicích modulů

MCU pro měřicí moduly jsou dva, STM32L152 a STM32F051. STM32L152 je osazen na teplotním modulu, na ostatních je STM32F051.

STM32L152

- Úsporný mikrokontrolér z řady STM32L1 s jádrem ARM Cortex M3.
- MPU – jednotka pro ochranu paměti.
- Taktovací frekvence 32 Mhz – 33,3 DMIPS. Až 128 kB paměti Flash a 16 kB RAM.
- Napájecí napětí 1,65 V to 3,6 V.
- Interní 16 Mhz oscilátor a 32kHz oscilátor pro RTC s kalibrací.

- RTC.
- 12-bit AD převodník, až 24 kanálů, maximální rychlost 1 MSPS, 12-bit D/A převodník.
- 7 kanálový DMA řadič.
- 10 časovačů.
- 2× I2C rozhraní, 3× USART, 2× SPI, maximální rychlost 16 Mbits/s.
- USB 2.0.

STM32F051

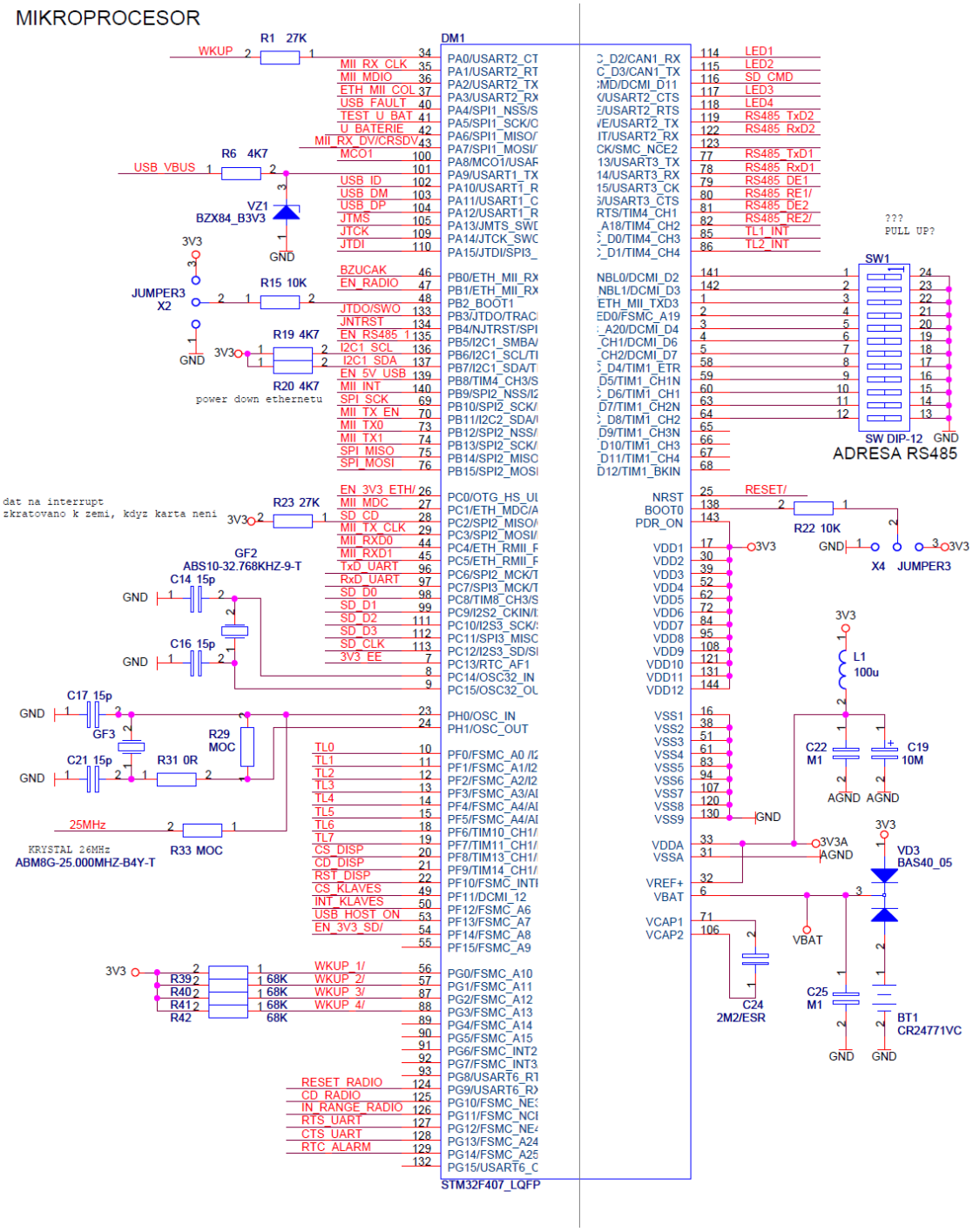
- Mikrokontrolér z řady STM32F0 s jádrem ARM Cortex M0.
- Taktovací frekvence 48 Mhz. Až 64 kB paměti Flash a 8 kB RAM.
- Napájecí napětí 2,0 V to 3,6 V.
- Interní 16 Mhz oscilátor a 32kHz oscilátor pro RTC s kalibrací.
- RTC.
- 12-bit AD převodník, až 24 kanálů, maximální rychlost 1 MSPS, 12-bit D/A převodník
- 5 kanálový DMA řadič.
- 11 časovačů.
- 2× I2C rozhraní, 2× USART, 2× SPI, maximální rychlost 18 Mbits/s.

Zapojení procesoru základní desky je na obrázku 3.1.2.

3.2 Ovladače periférií

Firma ST dodává knihovnu v jazyce C, která obstarává periférie procesoru. Výhodou je, že není nutné přímo přistupovat do registrů procesoru, ale vše lze obsluhovat pomocí API voláním funkcí. V průběhu řešení byla objevena celá řada nedostatků a v jednom případě dokonce i chyba v kódu. Základní operace knihovna zvládá, ale důslednější nastavení periférií vyžaduje datasheet a přístup k registrům.

V rámci práce prošly ovladače značným vývojem, první pokusy byly pouhý C++ wrapper na ovladači od ST. Postupem času se ukázalo, že bude z hlediska univerzálnosti, přenositelnosti, spolehlivosti a přehlednosti lepší napsat ovladače bez



Obrázek 3.1: Zapojení procesoru základní desky

software třetí strany. Část ovladačů je napsaná samostatně, používá přímý přístup do registrů. Méně důležité části zůstaly jako wrapper, případně jsou přímo používány ovladače od ST. Odlišný přístup byl zvolen u periferií jako SDIO nebo Ethernet. Jde o periférie, ke kterým výlučně přistupuje software třetí strany a spolupracuje s ovladačem od ST. V těchto případech byl ovladač použit, tak jak byl k dispozici.

V jazyce C++ lze řešit multiplatformní vývoj stejně jako v C (pomocí preprocesoru) nebo lépe pomocí šablon. Smyslem multiplatformního vývoje je stejné rozhraní pro periférie různých procesorů. Periférie se od sebe liší, na novějších MCU jsou složitější a modernější, ale zároveň by měly poskytovat stejnou základní funkcionalitu. Šablonový přístup byl použit na základní periférie jako je GPIO, USART a DMA, dále pro základní obsluhy procesoru, jako je kontrola přerušení (NVIC) nebo ovládání hodin (RCC). Další periférie by bylo vhodné doplnit, ale v rámci práce na to nezbyl čas.

V následující ukázce je naznačen koncept přístupu k hardware. Hierarchie dědičností definuje vlastnosti jednotlivých procesorů. Na základě těchto vlastností lze specializovat odvozené i nezávislé třídy a struktury.

```
1  class cpu_core_group_cortex_m0 {};
2
3  class cpu_core_cortex_m0 : public cpu_core_group_cortex_m0
4  {
5  public:
6      typedef cpu_core_group_cortex_m0 cpu_core_group;
7  };
8
9  class cpu_core_group_cortex_m3_m4 {};
10
11 class cpu_core_cortex_m3 : public cpu_core_group_cortex_m3_m4
12 {
13 public:
14     typedef cpu_core_group_cortex_m3_m4 cpu_core_group;
15 };
16
17 class cpu_core_cortex_m4 : public cpu_core_group_cortex_m3_m4
18 {
19 public:
20     typedef cpu_core_group_cortex_m3_m4 cpu_core_group;
21 };
```

Následující třídy dědí typ procesorového jádra a rozšiřují ho o další funkcionalitu. Ve třídách je mimo jiné soubor statických ukazatelů, které přiřazují třídy popisující

registry periférií ke konkrétním místům v paměťovém prostoru. V každém projektu je globálně viditelný typ *cpu*, pod kterým je v závislosti na procesoru dostupná odpovídající třída. Obecně lze k jednotlivým periferiím v globálním namespace přistupovat přes *cpu::* .

```
class stm32l1 : public cpu_core_cortex_m3
{
public:
    typedef stm32l1 _this;
    typedef cpu_core_cortex_m3 _parent;

    typedef _parent cpu_core;
}

class stm32f4 : public cpu_core_cortex_m4
{
public:
    typedef stm32f4 _this;
    typedef cpu_core_cortex_m4 _parent;

    typedef _parent cpu_core;

    //definice ukazatele na periferii usart1
    static Usart constexpr* usart1 = reinterpret_cast<Usart *>(
        apb2periph_base + 0x1000);
}
```

3.3 GPIO

GPIO je jednotka, která ovládá piny procesoru. Ovladač je rozdělen na dvě části. Jedna obstarává kontrolu nad GPIO porty a druhá definuje konkrétní pin (piny). Periférie je stejná pro všechny řady procesorů STM32.

Třída *GPIO* obsahuje jako členské proměnné datovou strukturu shodnou se strukturou GPIO portu v registrech. Samotná instance je pak ukazatel do paměťového prostoru registrů. Třída tím pádem nemůže obsahovat žádné vlastní členské proměnné, pouze statické proměnné a metody. Třída *Pin_in* umožňuje přímo ovládat konkrétní pin nebo sadu pinů na jednom portu. Přičemž třída *Pin_in* definuje pouze operace pro pin ve vstupním režimu a pin nedovolí nastavit do výstupního režimu. Od této třídy je odvozená třída *Pin_io*, která umožňuje pin používat i jako výstupní.

Řešení umožňuje zajistit, že pokud pin deklaruji jako vstupní, v následujícím kódu se nikdy nedostane do stavu výstupního a nezpůsobí případné hardwarové selhání.

Třída *Pin_in* a *Pin_out* má pro datové typy *bool* a *uint16_t* přetížené operátory přiřazení a přetypování. Přístup umožňuje velmi intuitivní přístup k pinům:

```
1  Pin_in pin1(cpu::gpioa, pin_00);
2  Pin_io pin2(cpu::gpioa, pin_01 | pin_02);
3
4  pin2.set_output();
5
6  if(pin1) // testuje pin na logickou hodnotu
7  {
8      // priradi logickou 1 vsem pinum zadanym v~konstruktoru
9      pin2 = true;
10
11     // priradi 16bitove cislo portu, metoda realne nastavi pouze ty
           piny, ktere jsou zadane v~konstruktoru
12     pin2 = pin_01;
13     pin2 = pin_01 | pin_02;
14 }
```

Dále obě třídy umožňují nastavení vlastností pinů pomocí metod.

3.4 Sběrnice RS485

RS-485 (stejně tak RS-422) se vyznačuje dvou–vodičovým propojením jednotek. Označují se písmeny A a B. Maximální délka sběrnice je až 1200 m, maximální počet vysílačů a přijímačů je typicky 32. Lze použít přijímače s větším vstupním odporem, které umožní použití až 256 jednotek. Maximální přenosová rychlost je 10 Mb/s, ale záleží na délce vedení. Sběrnice musí být zakončena terminátory. Vhodné je také použít velké odpory na připojení kladnějšího vodiče k napájecímu napětí (5V) a zápornějšího vodiče k zemi. Je to mnohdy důležitější než terminátory (nejsou na krátké vzdálenosti nutné), protože v situaci, kdy žádný z účastníků nevysílá, může být na sběrnici v podstatě cokoliv. Logické stavy jsou reprezentovány rozdílným napětím mezi oběma vodiči. Je to rozdíl oproti RS-232, kde se úrovně stavů vztahují k zemi. Rozdílným napětím mezi oběma vodiči je výhodný zejména kvůli eliminaci naindukovaného rušivého napětí. Logický stav 1 je reprezentován rozdílovým napětím mezi vodiči A a B menší než -300 mV, logický stav 0 rozdílovým napětím větším než +300 mV. Správný vysílač by měl na výstupu generovat napětí +2 V (příp. -2 V), správný přijímač by měl na vstupu rozlišit napětí ±200 mV.

- Dvouvodičová verze RS-485 – Přenos dat se uskutečňuje pomocí 7 nebo 8 bitových rámců se startbitem, jedním nebo více stopbity a případně i paritním bitem. Přenos je poloduplexní a proto se vyžaduje řízení přenosu dat. Výhodou je, že pomocí dvouvodičové linky RS-485 je možné vytvořit více prvkovou komunikační síť.
- Čtyřvodičová verze RS-485 – Umožňuje obousměrnou komunikaci. Ve spojení 1:1 není potřeba řízení směru. V praxi se používá spojení 1:N, kde se zpravidla master vysílá na jednom kanálu a všechny slave zařízení se při vysílání dělí o druhý kanál.

Fyzická vrstva je realizovaná budičem sběrnice připojeným k UART modulu v procesoru a dvěma GPIO signálům, které zapínají – vypínají vysílač – přijímač. Z pohledu procesoru se jedná o standardní sériovou komunikaci.

Na základní desce jsou dvě sběrnice RS485. Interní sběrnice je bez galvanického oddělení a je realizována budičem ADM3485. Externí je řešena budičem LTM2881 s galvanickým oddělením v jednom pouzdře.

3.4.1 UART

USART je univerzální synchronní a asynchronní přijímač a vysílač. UART je univerzální asynchronní přijímač a vysílač. Komunikace s budičem RS485 je asynchronní. Jednota USART v STM32F4 a STM32L1 je stejná, ale v STM32F0 je novější. V následující ukázce bude naznačena realizace ovladače.

Ovladač je definován jako třída *Usart*, která má jako parametr šablony verzi cpu. Třída je odvozená od třídy *___Usart*, která má jako parametry šablony verzi cpu a verzi USART modulu.

```
1 template<typename cpu> class Usart
2     : public ___Usart< cpu, typename ___Usart_version<cpu>::version >{};
```

Struktura *___Usart_version_l1_f4* identifikuje USART modul, který je v STM32F4 a STM32L1.

```
1 struct ___Usart_version_l1_f4 {};
```

Struktura *___Usart_version* a její specializace umožňuje identifikaci verze USART modulu na základě verze cpu.

```
template<typename cpu> struct __Usart_version {};
template<> struct __Usart_version<stm32l1>
{ typedef __Usart_version_l1_f4 version; };

template<> struct __Usart_version<stm32f4>
{ typedef __Usart_version_l1_f4 version; };
```

Třída *__Usart_base* definuje společné části pro všechny USART moduly.

```
template<typename cpu>
class __Usart_base
{
public:
    void rcc(bool b) { cpu::rcc->periph_enable(this, true); }
};
```

Třída *__Usart* je obecná třída, její specializace definuje metody určené pro odpovídající modul USART.

```
template<typename cpu, typename usart>
class __Usart {};
```

Třída *__Usart* specializovaná pro *__Usart_version_l1_f4*.

```
template<typename cpu>
class __Usart<cpu, __Usart_version_l1_f4>
    : public __Usart_base<cpu>, Regs_setters
{
public:
    __IO uint16_t SR;          /*!< USART Status register,
                                    Address offset: 0x00 */
    uint16_t      RESERVED0;   /*!< Reserved, 0x02
                                                        */
    __IO uint16_t DR;          /*!< USART Data register,
                                  Address offset: 0x04 */
    uint16_t      RESERVED1;   /*!< Reserved, 0x06
                                                        */
    __IO uint16_t BRR;         /*!< USART Baud rate register,
```

```
                    Address offset: 0x08 */
11      uint16_t      RESERVED2;  /*!< Reserved, 0x0A
                                                    */
12      __IO uint16_t CR1;        /*!< USART Control register 1,
                    Address offset: 0x0C */
13      uint16_t      RESERVED3;  /*!< Reserved, 0x0E
                                                    */
14      __IO uint16_t CR2;        /*!< USART Control register 2,
                    Address offset: 0x10 */
15      uint16_t      RESERVED4;  /*!< Reserved, 0x12
                                                    */
16      __IO uint16_t CR3;        /*!< USART Control register 3,
                    Address offset: 0x14 */
17      uint16_t      RESERVED5;  /*!< Reserved, 0x16
                                                    */
18      __IO uint16_t GTPR;       /*!< USART Guard time and prescaler
          register, Address offset: 0x18 */
19      uint16_t      RESERVED6;  /*!< Reserved, 0x1A
                                                    */
20
21      void send(uint16_t data) { DR = data; }
22      uint16_t recv() { return DR; }
23  }
```

Všechny členské proměnné, stejně jako v případě GPIO, popisují strukturu registrů modulu USART.

Od třídy *Usart* je odvozena třída *Usart_DMA*. Třída doplňuje standardní funkce příjmu a odesílání jedno byte, funkcemi pro příjem a odeslání celého bloku dat. Dále definuje virtuální metody *send_complete* a *recv_complete*, které lze dalším děděním této třídy uživatelsky definovat. Nejde o standardní virtuální metody, využívá se zde návrhového vzoru CRTP viz 5.6.2. Dále třída pracuje s přerušením. Přerušení od vysílacího DMA kanálu je voláno po ukončení vysílání a slouží například pro vypnutí vysílače u budiče sběrnice RS485. Přerušení od přijímacího DMA kanálu, znamená naplnění přijímacího bufferu a znamená, že buffer přetekl. Řádné ukončení příjmu signalizuje přerušení idle modulu USART, je vyvoláno pokud jsou na lince přenášeny data a následně nastane klid. Řešení ukončení příjmu je velmi univerzální, vyžaduje pouze poslat data bez časových mezer.

Dále je od třídy *Usart_DMA* odvozena třída *Bus_RS485*, která doplňuje přepínání režimu budiče sběrnice RS485.

3.5 Bezdrátové rozhraní

Pro bezdrátovou komunikaci byl zvolen komunikační protokol postavený na normě IEEE 802.15.4.

Norma IEEE 802.15.4 definuje fyzickou a linkovou vrstvu bezdrátové komunikace. Fyzická vrstva využívá radio–frekvenční komunikaci. Frekvence jsou rozděleny podle regionů:

- 868.0–868.6 MHz: Evropa, jeden kanál (2003), tři kanály (2006)
- 902–928 MHz: Severní Amerika, 10 kanálu (2003), 30 kanálů (2006)
- 2400–2483.5 MHz: Celosvětově, 16 kanálů (2003, 2006)

Originální verze standardu z roku 2003 specifikuje dvě fyzické vrstvy postavené na modulaci DSSS. Jedna pracuje na frekvenci 868/915 MHz s přenosovou rychlostí 20 and 40 kbit/s. Druhá v pásmu 2450 MHz s rychlostí 250 kbit/s.

Fyzická vrstva je realizována transreceiverem MRF24J40, ovládaným pomocí SPI rozhraní. Transreceiver zajišťuje i linkovou vrstvu, která je popsaná v části 5.3.5. Bezdrátová komunikace byla vyvíjena mimo hardware základní desky. Základní deska poskytuje univerzální rozhraní pro komunikační modul.

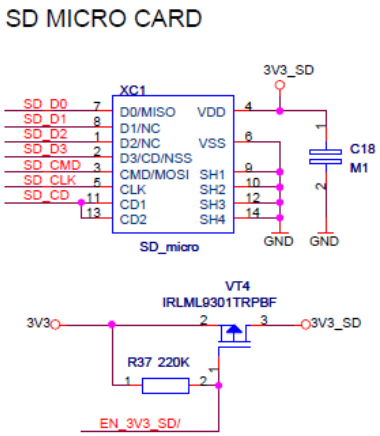
3.6 SDIO

Periférie SDIO je speciální varianta sběrnice SPI pro SD karty. Umožňuje přímé připojení SD karty. Lze použít DMA pro čtení a zápis větších bloků dat. Ovladač periférie SDIO od ST

Základní funkce ovladače jsou:

- *void SD_LowLevel_Init(void)* – Inicializace karty.
- *void SD_LowLevel_DeInit(void)* – Deinicializace karty.
- *void SD_LowLevel_DMA_TxConfig(uint32_t *BufferSRC, uint32_t BufferSize)* – Zápis dat do sektoru pomocí DMA.
- *void SD_LowLevel_DMA_RxConfig(uint32_t *BufferDST, uint32_t BufferSize)* – Čtení dat ze sektoru pomocí DMA.

Zapojení microSD karty je na obrázku B.1.



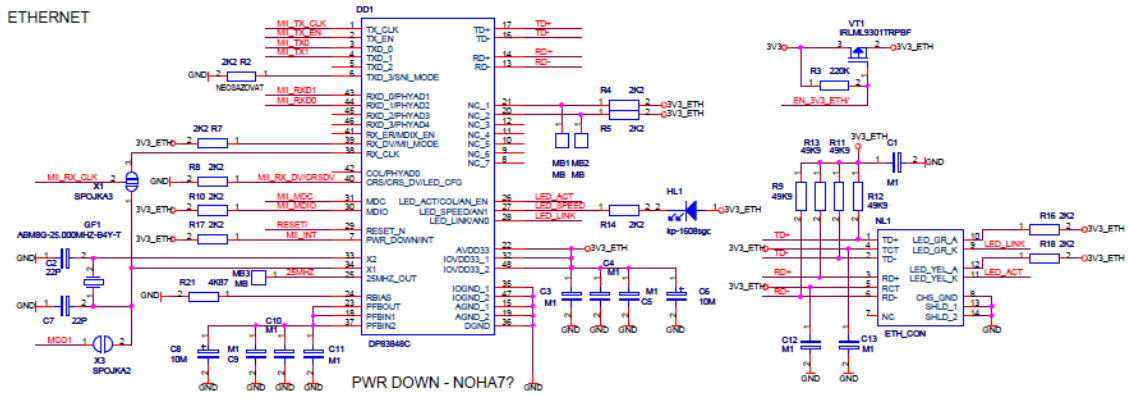
Obrázek 3.2: Zapojení microSD karty.

3.7 Ethernet

Ethernet definuje fyzickou a linkovou vrstvu komunikačního protokolu podle ISO/OSI modelu. Procesor STM32F4 obsahuje jako periférii linkovou vrstvu, fyzická se připojuje jako externí čip rozhraním MII nebo RMII. Konektoru RJ45 se k fyzické vrstvě připojuje přes oddělovací transformátor.

MII (Media Independent Interface) je standardní rozhraní pro propojení linkové a fyzické vrstvy Ethernetu. Jde o full–duplex, přičemž pro každý směr jsou 4 vodiče – posílá se 4bitové slovo. Hodinový signál pro data je 25 MHz (4x25 MHz – 100 Mb/s každým směrem). Celkový počet vodičů (datových a kontrolních) je 15. RMII (Reduced Media Independent Interface) je stejné rozhraní, ale redukuje počet vodičů. Datové vodiče pro jeden směr jsou jen dva. Hodinový signál je 50 MHz. Dále jsou vynechány některé kontrolní signály.

Byl použit ovladač linkové vrstvy od firmy ST, který stejně jako procesor umožňoval použít obě rozhraní. V rámci této práce byla zvolena varianta RMII. Z hlediska složitosti implementace kompletního TCP/IP, včetně hardware, jde o poměrně náročnou úlohu. Během řešení se vyskytly tři zásadní problémy. Pro realizaci fyzické vrstvy byl zvolen obvod od TI – DP83848C. První byl problém se zdrojem hodin pro fyzickou vrstvu, rozhraní MII dovoluje hodinový signál generovat přímo z procesoru, ale pro RMII se toto řešení ukázalo jako nespolehlivé. Generovaný signál s dvojnásobnou frekvencí byl méně přesný a docházelo k výpadkům spojení. Identifikovat problém nebylo snadné, v podstatě se zkoušely všechny možnosti. Doplněním 50 MHz oscilátoru k fyzické vrstvě se stabilita zlepšila. Bohužel výpadky přetrvávali. Další analýzou bylo zpozorováno, že pokud fyzická vrstva vyhodnotí spoj jako méně kvalitní, nastaví komunikační rychlost pouze na 10 Mb/s a spojení je zcela bez problémů. Pokud fyzická vrstva nastavila rychlost na 100 Mb/s, dříve nebo později se



Obrázek 3.3: Zapojení fyzické vrstvy Ethernetu.

spojení vždy rozpadlo a neobnovilo. Důvodem rozpadu rychlejšího spojení je pravděpodobně špatný layout na DPS (vodiče jsou dlouhé se signálem o frekvenci 50MHz, navíc je na DPS několik drátových propojů kvůli chybám). V režimu 100 Mb/s bylo spojení nefunkční, ale fyzická vrstva by měla přepnout zpět do pomalejšího režimu. Teprve nastudováním zdrojového kódu ovladače linkové vrstvy a datasheetu fyzické vrstvy byla objevena chyba v ovladači od ST, která způsobila, že první zvolená konfigurace se již nezměnila. Paradoxně tedy docházelo k problému v případě že se během inicializace nenastala chyba ve spojení. Opravou ovladače se podařilo docílit správného chování – pokud se spojení rozpadne, inicializace se opakovala.

Problémy byly značně nepříjemné, zejména proto, že se každá z nich projevovala stejným nebo velmi podobným chováním. Odstranění takové chyby se neprojeví změnou chování, což je mnohdy bezvýchodná situace. Velkým přínosem byl referenční návrh Ethernetového rozhraní na vývojovém prostředku a osciloskop s logickým analyzátorem.

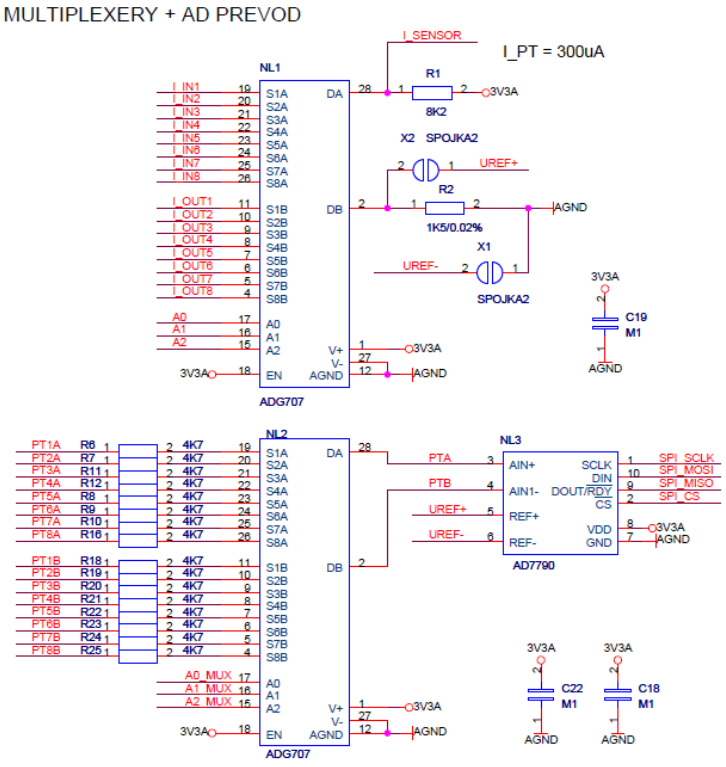
Původní zapojení je na obrázku 3.3. Stávající zapojení je upravené, nebylo zaneseno do schématu.

3.8 AD převodník

Byl použit 16bitový AD převodník AD7790. Je ovládaný rozhraním SPI. Zapojení je na obrázku 3.4.

3.9 Grafický displej

Byl zvolen černobílý grafický displej s podsvícením DOGXL160 od firmy Electronic Assembly. Displej má 160x104 obrazových bodů, úhlopříčku 3,3 palce a dokáže



Obrázek 3.4: Zapojení AD převodníku AD7790.

zobrazit 4 odstíny šedé barvy. Je ovládaný komunikačním rozhraním SPI.

3.10 Kapacitní klávesnice

Displej byl doplněn velkou kapacitní klávesnicí. Kapacitní snímání dotyku bylo zvoleno, protože lze klávesnici spolu s displejem překrýt sklem a snadněji docílit větší odolnosti proti vlhkosti. Dalším důvodem byly relativně nízké náklady na kusové množství v porovnání s výrobou fóliové klávesnice, přičemž kapacitní klávesnici je možné navíc podsvítit. Nevýhoda je nenulová spotřeba aktivní klávesnice.

Klávesnici představují snímací elektrody a vyhodnocovací obvod. Celkový počet kláves je 22. Byly použity dva obvody AT42QT1110 a jeden AT42QT1010. AT42QT1110 dokáže obsloužit až 11 kláves, mezi dva obvody je rozděleno 21 kláves. Obvod AT42QT1010 obsluhuje pouze jedno – zapínací tlačítko. Byl použit pro minimální spotřebu, která se pohybuje kolem 20 μA. Stiskem tohoto tlačítka se zapne zbytek klávesnice i displej.

Elektrody klávesnice byly vyrobeny jako DPS. Podsvícení bylo realizováno LED diodou pod DPS a dírou uprostřed elektrody. V rámci vývoje nejdříve vznikla menší 4–tlačítková klávesnice, která se chovala bezchybně. Velká klávesnice vykazovala

značné problémy. Část tlačítek byla málo citlivá, měla pomalou reakci detekce stisku a následně byl stisk detekovaný i 1 sekundu po uvolnění tlačítka. Problém se týkal zejména tlačítek na jednom z obvodů AT42QT1110, druhý obvod se choval lépe. Zapínací tlačítko na obvodu AT42QT1010 se chovalo bezchybně. Problém může být ve špatném layoutu DPS, relativně dlouhé vzdálenosti elektrod od obvodů a součinností více obvodů v těsné blízkosti. V rámci práce se nepovedlo klávesnici korektně odladit.

4 Operační systém

Pro tento projekt byl zvolen operační systém freeRTOS. Hlavní výhody a srovnání s CrossWorks Tasking Library jsou uvedeny v následujícím textu. Dalším důvodem je kompatibilita s lwIP stackem. Operační systém byl použit pro základní desku i měřicí moduly.

4.1 CTL

CrossWorks Tasking Library[10] je knihovna, která umožňuje spouštět několik vláken současně. Knihovna je součástí vývojového prostředí CrossWorks a je napsaná v jazyce C. Vlákna mají svou prioritu. Vlákna se stejnou prioritou se vzájemně nestřídají. Pro synchronizaci vláken je k dispozici základní mutex, semafor, fronta a událost. Součástí je thread-safe varianta funkcí pro dynamickou alokaci paměti malloc a free.

Použití je relativně jednoduché, vývojové prostředí knihovnu integruje bez složité konfigurace. Nevýhodou je závislost na daném prostředí a malá rozšířenost knihovny.

4.2 FreeRTOS

Sotisfikovanější řešení představuje operační systém freeRTOS [2] firmy Real Time Engineers Ltd. Licence freeRTOS je modifikovanou licencí GPL. Systém může být použitý v komerčních aplikacích bez zveřejnění proprietárního kódu.

- Jde o otevřený profesionální nástroj, používaný v řadě komerčních aplikací.
- Je napsán v jazyce C.
- Je zdarma bez nutnosti zveřejňovat proprietární zdrojový kód.
- Existuje komerční varianta OpenRTOS a SafeRTOS.
- OpenRTOS integruje podporu USB, souborového systému a TCP/IP stacku.

- SafeRTOS je varianta certifikovaná pro bezpečnostně kritické aplikace.
- Podporuje celou řadu architektur a vývojových nástrojů.
- Má minimální hardwarové nároky v porovnání s embedded Linux.
- Existuje port na STM32.
- Je podporován v lwIP stacku.

Pro použití freeRTOS je potřeba cca 5-10 kB paměti flash a časovač, v případě procesorů ARM je použit systémový časovač systick.

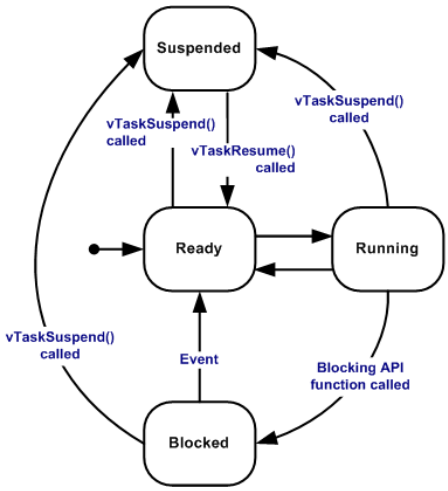
4.2.1 Dynamické přidělování paměti

Samotné freeRTOS poměrně hojně využívá dynamickou alokaci paměti (svoji paměť si alokuje každé vlákno i synchronizační primitiva). Jak již bylo uvedeno, real–time aplikace by měly k alokaci paměti přistupovat opatrně. Chování je časově nepředvídatelné a s omezenými paměťovými prostředky se může stát, že paměť dojde (například kvůli fragmentaci). Z hlediska více–vláknového systému je také potřeba řešit problém souběhu více vláken. Funkce alokující paměť musí být thread–safe.

FreeRTOS nabízí několik způsobů správy paměti. Obecně všechny jsou thread–safe. Liší se způsobem alokace, velikostí alokovaných bloků a způsobem uvolňování paměti.

- První způsob s uvolňováním vůbec nepočítá. Program na svém startu alokuje potřebnou paměť a pracuje s ní po celou dobu svého života. Jde o deterministický přístup, který může selhat jedině při startu aplikace, chyba (nedostatek paměti) se snadno odhalí, fragmentace nehrozí.
- Třetí způsob je pouhý wrapper nad standardní funkce malloc a free, zajišťuje thread–safe přístup.
- Druhý a čtvrtý způsob paměť alokují po větších blocích. Algoritmus je více deterministický než ve třetím způsobu a také umožňuje uvolnit paměť. Mezi sebou se liší nakládáním s uvolněnou pamětí. U druhého hrozí fragmentace. Je vhodný pro aplikace, kde se opakovaně alokují stejně velké bloky paměti. Čtvrtý způsob fragmentaci předchází spojováním sousedních volných bloků paměti. Oba jsou efektivnější (více deterministické) než standardní malloc.

V projektu byl použit čtvrtý způsob správy paměti.



Obrázek 4.1: Stavy a přechody úlohy freeRTOS

4.2.2 Úloha

Základním prvkem systému je úloha. Vytváří se voláním funkce xTaskCreate. Parametrem se předá ukazatel na funkci, která se začne vykonávat, parametr, který se funkci předá, velikost stacku, priorita úlohy a název pro potřeby diagnostiky. Úlohy se stejnou prioritou se pravidelně střídají. Pokud žádná není aktivní, systém skočí do idle task, kde je možné procesor uspávat. FreeRTOS podporuje i úsporný režim, ve kterém se zastaví systémový časovač. Na obrázku 4.1 jsou znázorněny stavy a přechody úlohy. Ve stavu blocked se čeká na událost a úloha je neaktivní. Ve stavu running je úloha aktivní.

Terminologie freeRTOS hovoří o úlohách, ale v zásadě jde o vlákna, protože společně sdílejí datovou paměť. Dále se v textu bude hovořit spíše o vláknech.

4.2.3 Fronta

Hlavním synchronizačním primitivem systému freeRTOS je obecná FIFO fronta. Fronta umožňuje standardně vkládat a odebírat objekty. První vložený objekt se vždy odebírá jako první. Navíc je možné vložit objekt na konec fronty a stávající objekty předběhnout a tím určovat prioritu. Dále je možné objekt rovnou zahodit. Přístupové funkce existují ve variantě, která lze volat v přerušení. Synchronizace vláken využívá dvou stavů fronty – prázdná a plná fronta. V obou případech – vlákno chce zapsat do plně obsazené fronty nebo vlákno chce odebrat objekt z prázdné fronty, způsobí blokování, dokud není možné operaci dokončit (z fronty bude objekt odebrán / přidán). Maximální dobu čekání lze omezit časem. Ideálním stavem je předávání všech dat mezi vlákny pomocí front a ne pomocí sdílené paměti.

Další synchronizační primitiva jsou interně realizovány vždy pomocí fronty, jsou

to:

- Mutex – vhodný pro vzájemné vyloučení, například jedinečný přístup k HW prostředkům, nabývá stavů zamčeno a odemčeno. Získáním mutexu jedním vláknem dojde k jeho zamčení až do doby, než se ho vlákno vzdá.
- Semafor – nabývá číselných hodnot, slouží například pro omezení počtu vláken, které můžou vstoupit do kritické sekce.

4.2.4 Co–routines

Zajímavou alternativou ke standardním úlohám jsou takzvané stack–less úlohy. Taková úloha nepotřebuje vlastní stack. Používá se jeden společný a při přepínání kontextu se jeho obsah zahazuje. Kontext vykonávané úlohy je potřeba ukládat explicitně. Příklad takové úlohy bude uveden v odstavci 4.3.

4.2.5 C++

V rámci práce byl pro více intuitivní použití freeRTOS vytvořen C++ wrapper. Politika pojmenování funkcí a struktur freeRTOS je zajímavá tím, že jméno každého datového typu začíná jednoznakovým prefixem, který typ identifikuje (například ukazatel začíná písmenem „x“). Analogicky jsou pojmenované funkce podle typu návratové hodnoty. Na přehlednosti kódu to ale nepřidá, spíše naopak.

Základní věc, kterou bylo nutné udělat, je globální přetížení operátorů pro dynamickou alokaci paměti a konstrukci objektů new a delete. Výchozí new a delete volají standardní funkce malloc a free, které nejsou thread–safe. Globálním přetížením zajistím, že se při volání new a delete v daném namespace, vždy zavolají funkce dynamické alokace paměti pro freeRTOS. Přetížit je potřeba také operátory new[] a delete[] a placement variantu všech těchto operátorů. Placement varianta slouží k pouhé konstrukci objektu do známého místa v paměti, new[] a delete[] slouží k alokaci a konstrukci polí objektů.

V následujícím náhledu je znázorněna deklarace základních globálních funkcí a třídy Thread wrapperu.

```
1 typedef portTickType Tick;
2 const Tick tick_max_delay = portMAX_DELAY;
3
4 void suspend(xTaskHandle handle = 0);
5 void resume(xTaskHandle handle);
6 void resume_isr(xTaskHandle handle);
7
```

```
8  void suspend_all();
9  bool resume_all();
10 void enter_critical();
11 void exit_critical();
12 void yield();
13
14 void delay(Tick delay);
15 void delay_until(Tick delay, Tick& prevtime);
16
17 Tick get_ticks();
18 Tick get_ticks_isr();
19
20 class Thread
21 {
22     xTaskHandle handle;
23     Delegate<> del;
24
25 public:
26     Thread(Delegate<> del, uint16_t priority, const char* name = 0,
           uint16_t stacksize = configMINIMAL_STACK_SIZE);
27     ~Thread();
28
29     static void run(void* del);
30
31     bool start();
32     void suspend();
33     void resume();
34 };
```

Funkce *suspend* a *resume* (*resume_isr*) lze volat bez parametru (provedou se nad aktuálně probíhajícím vlákně), případně s parametrem *handle*, který identifikuje freeRTOS úlohu. Analogické funkci mají stejnojmenné členské metody třídy *Thread*.

Funkce *suspend_all* (*resume_all*) a *enter_critical* (*exit_critical*) pozastavují běh všech úloh a systému. Vlákno, které zavolá funkce *yield* se dobrovolně vzdá procesorového času a předá ho dalšímu vláknu v pořadí. V následující smyčce vlákno standardně pokračuje v činnosti.

Funkce delay vlákno pozastaví na daný počet milisekund od doby jejího zavolání. Funkce *delay_until* dokáže odpočet času spustit před zahájením čekání. Rozdíl obou funkcí je patrný na příkladu vlákna, které běží nekonečnou smyčkou, ve které provádí výpočet po dobu n, čeká dobu m a celková doba jedné periody bude n+m. První funkce zajistí čekání po dobu m a celková doba periody je závislá na době výpočtu n.

Druhá funkce umožňuje zadat dobu čekání jako n+m, přičemž se měří doba výpočtu n, odečte se od požadované doby délka periody m+n a čeká se pouze po zbývající čas m. Lze tak docílit konstantní doby periody nekonečné smyčky.

Funkce *get_ticks* vrací aktuální hodnotu systémového časovače.

Třída *Thread* má dvě členské proměnné, handle – identifikátor freeRTOS úlohy a del typu *Delegate*, která definuje, jaká funkce (metoda) se má po startu vlákna začít vykonávat. Více k třídě *Delegate* v odstavci 5.6.1.

Konstruktor vytváří nové vlákno. Jeho parametry jsou *Delegate*, priorita, jméno úlohy a velikost stacku. Vlákno se startuje voláním metody start, jejíž návratová hodnota ověřuje úspěšnost vytvoření vlákna a následného startu. Destruktor vlákno automaticky likviduje.

Statická funkce run je volaná systémem freeRTOS během spouštění úlohy.

Třída Queue:

```
template <typename Type>
class Queue {
    xQueueHandle handle;

public:
    Queue(uint16_t len);
    ~Queue();

    operator bool () const;

    bool send_back(const Type &item, uint32_t time = tick_max_delay);
    bool send_front(const Type &item, uint32_t time = tick_max_delay);
    bool receive(Type &item, uint32_t time = tick_max_delay);
};
```

Příklad použití fronty;

```
Queue<int> fronta(10); //konstrukce fronty o~velikosti 10 prvku typu
    int

if(fronta)
{
    // konstrukce probehla uspesne

    fronta.send_back(10); // pridani prvku typu int do fronty
```

```
8
9       int a;
10      fronta.receive(a, 100); // odebrani prvku, ulozeni do promenne a,
            maximalni doba cekani na prvek je 100 ms.
11  }
```

Třída *Queue* kromě objektového obalu nad freeRTOS frontou využívá šablon. Funkce freeRTOS pro konstrukci a ovládání front definují jeden prvek pouze jeho velikostí a předávají ho pomocí generického ukazatele void*. Využití šablon umožňuje jednoznačné definování datového typu prvků fronty, tím docílit silné typové kontroly a minimalizovat špatné použití.

Konstruktor má jediný parametr, který určuje délku fronty (maximální počet prvků).

Metoda operator bool je přetížený operátor přetypování, který způsobí, že pokud budeme k instanci této třídy přistupovat jako k logické hodnotě, vrátí nám jeho stav. Logická hodnota true odpovídá úspěšně zkonstruované frontě. Důvodem pro neúspěšnost konstrukce vlákna nebo fronty může být nedostatek dynamicky alokované paměti.

Metody *send_back* a *send_front* vkládají objekty do fronty pomocí konstantní reference. Funkce receive přijatý objekt přiřadí objektu předaného referencí. Funkce jsou blokující, umožňují určit maximální dobu čekání a návratová hodnota určuje, jestli metoda skončila úspěšně nebo vypršela maximální doba. Metody mají i varianty pro volání z přerušení, ale pro stručnost nejsou uvedeny.

4.3 Protothreads

Protothreads je stackless multi–threading, jehož autorem je Adam Dunkels. Jde o multi vláknový systém, ve kterém všechny vlákna sdílejí jeden stack a během přepínání kontextu se jeho obsah přepisuje. Jeho využití je výhodné u systému s omezenou pamětí RAM a to zejména při potřebě existence velkého množství paralelních vláken. V této byl tento model použit pro realizaci obsluhy relativně velkého počtu senzorů – každý senzor představuje jedno vlákno.

Základní problém je absence stacku. Každé vlákno si musí potřebný kontext uložit samostatně a pak se dobrovolně vzdát svého procesorového času. Ve své podstatě si lze vlákno představit jako stavový automat. K přepnutí může vždy dojít až po uložení aktuálního stavu (pozice) a všech proměnných uložených na stacku, jejichž hodnotu budeme potřebovat při pokračování vlákna.

Nevýhodou řešení je ruční ukládání kontextu a nemožnost přerušit vlákno externě a kdykoliv, přepínání úloh je závislé na době přechodů mezi stavy automatu.

Elegantní myšlenkou Protothreads je využití preprocesoru jazyka C, který stavový automat vytvoří automaticky. Vlákno představuje funkce se strukturou jako parametrem. Všechny mezi–stavové proměnné i aktuální stav se ukládají do struktury. Pomocí maker se pak vytvoří standardní konstrukce switch–case. Na začátku funkce se podle uloženého stavu skočí do odpovídající větve programu. Zdrojový kód lze zapisovat jako pro standardní multi–tasking a je lépe čitelný než zápis stavového automatu.

Příklad kódu využívajícího Protothreads:

```
1  PT_THREAD(struct pt *pt)
2  {
3    PT_BEGIN(pt); // uvodni makro
4
5    while(1)
6    {
7      if(initiate_io())
8      {
9        timer_start(&timer);
10
11       PT_WAIT_UNTIL(pt, io_completed() || timer_expired(&timer)); //
             makro, ktere zajisti prepnuti stavu automatu na zaklade
             podminek
12
13       read_data();
14     }
15   }
16
17   PT_END(pt); // ukoncovaci makro
18 }
```

5 Software

5.1 TCP/IP protokol

TCP/IP je sada protokolů pro komunikaci v počítačové síti. Definuje síťovou, transportní a aplikační vrstvu. Základními protokoly jsou TCP a UDP.

5.1.1 TCP

TCP protokol (Transmission Control Protocol) je základní protokol Internetu. Jde o transportní vrstvu. Použitím TCP mohou aplikace na počítačích propojených do sítě vytvořit mezi sebou spojení, přes které mohou přenášet data. Protokol garantuje spolehlivé doručování a doručování ve správném pořadí.

5.1.2 UDP

UDP protokol je jednoduchý protokol, který nezaručuje doručení, ani správné pořadí. Je výhodný v aplikacích, kde opakování při nedoručení dat nemá smysl, například živé televizní vysílání. Je méně náročný na prostředky a složitost, než TCP, proto lze snadněji nasadit v embedded systémech.

5.1.3 Embedded

Podpora TCP/IP komunikace v embedded systémech je poměrně složitá úloha. TCP protokol není vůbec triviální záležitost z hlediska složitosti implementace, ale i v nárocích na paměť a výkon. Opakované odesílání a hlavně záruka pořadí doručených paketů zvyšuje nároky na paměť.

5.1.4 lwIP

V rámci byl použit kompletní TCP/IP stack lwIP, jehož autorem je Adam Dunkels. Licence lwIP je BSD – software lze libovolně šířit, jen je nutné uvádět autora. Stack je možné provozovat přímo na freeRTOS.

Podporované protokoly:

- IP, ICMP, UDP, TCP, IGMP, ARP, PPPoS, PPPoE DHCP klient, DNS klient, AutoIP/APIPA, SNMP agent
- HTTP server, SNTP klient, SMTP klient, ping, NetBIOS

Samotné lwIP si vystačí s 10 kB paměti RAM a 40 kB FLASH. Existují dvě možnosti sestavení knihovny, jedna je pro použití na systému s freeRTOS a druhá bez operačního systému. Druhá možnost vyžaduje pravidelné volání obslužné funkce. V práci byl použit první způsob. Stack definuje síťovou a transportní vrstvu protokolu, prostřednictvím ovladače přistupuje k libovolné linkové vrstvě, viz 3.7.

5.1.5 C++ wrapper

Pro snadné použití byl opět vytvořen wrapper.

Deklarace třídy *TCP*:

```
1  class TCP {
2      struct netconn *conn;
3      err_t err;
4
5  public:
6      TCP();
7      ~TCP();
8
9      void connect(const char *host, uint16_t port);
10     void disconnect();
11
12     operator bool();
13
14     void write(const uint8_t* ptr, size_t size);
15     size_t read(const uint8_t* ptr, size_t max_size);
16 };
```

5.2 Souborový systém FAT32

Pro ukládání dat byla zvolena microSD karta se souborovým systémem FAT32. SD karta byla připojena k procesoru SDIO rozhraním. Podpora FAT32 byla realizována externí knihovnou FatFs.

5.2.1 FatFs

Jedná se o kompletní FAT32 knihovnu, vztahuje se na ní BSD licence.

Je podporováno procházení a vytváření adresářové struktury, včetně dlouhých názvů souborů. Knihovna je nezávislá na architektuře. Pro implementaci je nutné pouze vytvořit těla funkcí pro spodní vrstvu, tzn. fyzický zápis do paměti. Potom lze používat API knihovny pro čtení, vytváření, zapisování i mazaní souborů a složek, které se velmi podobá standardním funkcím pro práci se souborem.

5.2.2 Implementace

Pro použití knihovny na konkrétní architektuře bylo nutné implementovat soubor *diskio.c*. Musí definovat tyto funkce:

- *DSTATUS disk_initialize(BYTE pdrv)* - Zajišťuje inicializaci záznamového zařízení – SD karty.
- *DSTATUS disk_status(BYTE pdrv)* - Vrací aktuální stav zařízení.
- *DRESULT disk_read(BYTE pdrv, BYTE *buff, DWORD sector, UINT count)* - Funkce přečte sektor. Adresa sektoru a ukazatel na cílové místo v paměti jsou předány parametrem.
- *DRESULT disk_write(BYTE pdrv, BYTE *buff, DWORD sector, UINT count)* – Funkce zapíše sektor. Cílová adresa sektoru a data jsou předány parametrem.
- *DWORD get_fattime(void)* – Funkce musí vrátit aktuální čas.

Implementace byla provedena pomocí funkcí souboru *stm324xg_eval_sdio_sd.c*. Jde o ovladač k rozhraní SDIO, které je popsané zde 3.6.

5.2.3 C++ wrapper

Pro zjednodušení zápisu do souboru vznikla knihovna, umožňující přístup k souboru na SD kartě ve stylu standardní C++ knihovny. Zapisovat a číst lze pomocí proudu

– třídy s přetíženými operátory « a ».

Ukázka deklarace třídy *F_stream*. Úmyslně je uvedena včetně definic metod. Funkce *f_open*, *f_close* a *f_write* je rozhraní knihovny FatFs.

```
1  class F_stream
2  {
3      FIL fil;
4      FRESULT res;
5
6  public:
7      struct beg {};
8      struct cur {};
9      struct end {};
10
11     //constructor
12     F_stream(const char * path, int par)     { res = f_open(&fil, path,
           par);}
13
14     operator bool()                          { return res == FR_OK; }
15
16     F_stream& operator<<(bool b)
17     { res = b ? f_write(&fil, "true", 4, 0) : f_write(&fil, "false", 5,
           0);  return *this; }
18
19     F_stream& operator<<(char c)
20     { res = f_write(&fil, &c, 1, 0); return *this; }
21
22     F_stream& operator<<(int a)
23     { char str[20]; int count = snprintf(str, 20, "\%02d", a); res =
           f_write(&fil, str, count, 0); return *this; }
24
25     F_stream& operator<<(const char * str)
26     { res = f_write(&fil, str, strlen(str), 0); return *this; }
27
28     F_stream& operator<<(F_stream& (*man)(F_stream&))
29     { man(*this); return *this; }
30
31     F_stream& seekp(size_t pos)              { res = f_lseek(&fil, pos);
           return *this; }
32     F_stream& seekp(int pos, beg)            { res = f_lseek(&fil, pos);
           return *this; }
```

```
33      F_stream& seekp(int pos, cur)           { res = f_lseek(&fil,
            f_tell(&fil) + pos); return *this; }
34      F_stream& seekp(int pos, end)           { res = f_lseek(&fil,
            f_size(&fil) - pos); return *this; }
35
36      //destructor
37      ~F_stream()                             { f_close(&fil); }
38  };
39
40  inline F_stream& endl(F_stream& d) { return d << '\n'; }
```

Cesta k souboru se definuje při konstrukci objektu, zavírá se zánikem objektu. Operátory « a » jsou přetížené pro základní datové typy, ale lze definovat další. Metoda *F_stream& operator«(F_stream& (*man)(F_stream&))* je rozhraní pro manipulátory. Příkladem manipulátoru bez parametru je funkce *endl*, která vloží konec řádku do proudu. Obecně manipulátory slouží pro nastavení parametrů proudu. Metoda *seekp* slouží k pohybu v proudu, data lze přeskočit, nebo se naopak vrátit na začátek.

5.3 Proprietární komunikační protokol

Nejnáročnější částí celé práce bylo vytvořit a napsat komunikační protokol. Primárně slouží k interní komunikaci mezi základní deskou a měřicími moduly.

5.3.1 Požadavky na komunikační protokol

- komunikace s měřicími moduly,
- identifikace typu modulu a měřených veličin,
- přenos naměřených hodnot,
- informace o nových hodnotách ze strany modulu – multi–master,
- možnost doplnění nových funkcí (kalibrace, vyčtení archivu),
- možnost rozšíření o nové typy měřicích modulů, měřených veličin a příkazů,
- přenosové medium – RS–485, bezdrátová komunikace a TCP/IP protokol,
- možnost směrování – jedna síť na více médiích a možnost mesh sítě.

Cílem nebylo všechny body realizovat v rámci této práce, ale postupovat tak, aby se funkce daly postupně doplnit.

5.3.2 Existující protokoly

Otázkou je, zda by nebylo snažší a rychlejší použít již hotový komunikační protokol. Obdobné protokoly existují, ale jejich použití je problematické z těchto důvodů:

- Neexistuje protokol, který by šel použít přímo. Aplikační vrstva protokolu bude vždy proprietární.
- Fungování na různých fyzických vrstvách umožňují zpravidla složité protokoly, například Ethernet.
- Možnost vytvoření mesh sítě umožňují spíše protokoly orientované pouze na bezdrátovou komunikaci.
- Implementovat existující (obecnější) protokol může být obtížnější, než implementovat protokol podle vlastních potřeb.
- Deklarace kompatibility se stávajícím protokolem zpravidla podléhá schvalovacímu procesu ze strany organizace spravující daný protokol. Složitost se tím značně zvyšuje a v této aplikaci je to zbytečné.

5.3.3 ISO/OSI model

Protokol částečně dodržuje ISO/OSI model. Počítá se s fyzickou, linkovou síťovou a aplikační vrstvou. Aplikační vrstva by neměla mít v tomto kontextu zásadní význam.

5.3.4 Fyzická vrstva

Definuje fyzickou komunikaci – elektrické a fyzikální vlastnosti zařízení, způsob reprezentace elektrických (obecně fyzikálních) jevů na digitální data. Fyzické vrstvy využívané v protokolu jsou popsané v částech 3.4 a 3.5.

5.3.5 Linková vrstva

Poskytuje spojení mezi dvěma sousedními prvky. Data z fyzické vrstvy uspořádává do rámců a formátuje rámce do fyzických dat. Obsahuje kontrolní součet. Součástí je i MAC vrstva – kontrola přístupu k médiu.

RS–485

Pro sběrnici RS–485 je linková vrstva navržena v rámci práce. Pro řízení přístupu k médiu je zvolen algoritmus CSMA/CD, který definuje detekci kolize kontrolou shody vysílaných dat (žádaný stav na lince) a dat současně přijímaných (reálný stav na lince). Zařízení se během vysílání chová následně:

- Čeká dokud není klid na lince (definovaný stav – logická 1).
- Náhodně dlouhou dobu čeká, pokud je klid na lince po celou dobu, zahájí vysílání. Jinak opakuje první krok.
- Zahájením vysílání současně naslouchá. Pokud vysílaná a přijímaná data nejsou stejná, došlo ke kolizi. Vrací se ke kroku jedna, náhodně dlouhá doba čekání ve druhém kroku se násobí počtem opakování.

Kolize (neshoda dat vysílaných a přijímaných) může být způsobena zkratem na lince. Scénář se bude opakovat dokud se nevyčerpá počet opakování. Druhý důvod může být vysílání dvou (a více) zařízení současně. V takovém případě by se měly obě zařízení zachovat stejně, u obou nebudou data souhlasit, obě přestanou vysílat a budou čekat náhodnou dobu než začnou znovu. Důležité je čekání opravdu náhodnou dobu. Je nutné použít alespoň pseudo–náhodné číslo, přičemž musí vycházet z čísla unikátního v síti. V rámci implementace je použito číslo adresy zařízení. Bylo by možné použít například výrobní jedinečné ID procesoru. Pokud by dvě zařízení čekala stejně dlouhou dobu, kolize by se vždy opakovala.

Linková adresa zařízení je definovaná jako 16bitové číslo. V rámci jednoho paketu je možné přenést 127 bajtů dat a jako kontrolní součet je použit CRC. Vlastnosti byly zvoleny s ohledem na linkovou vrstvu normy IEEE 802.15.4, která je použita podle specifikace pro RF přenos.

RF – IEEE 802.15.4

Linkovou vrstvu definuje norma IEEE 802.15.4 a zajišťuje ji transreceiver.

Řízení přístupu k médiu je realizováno algoritmem CSMA/CA, který se liší od výše popsaného tím, že nepočítá s možností současného vysílání a přijímání (transreceiver toho není schopný – rf část lze provozovat buď jako vysílač nebo přijímač, nikoliv obojí současně). Algoritmus se chová obdobně, jen kolizi nepozná během vysílání. Požaduje potvrzení od strany příjemce, pokud jí neobdrží opakuje vysílání stejně jako CSMA/CD.

Adresování je možné pomocí 16bitového nebo 64bitového čísla, dále je možná koexistence více sítí díky 16bitovému PAN-ID identifikátoru. Umožňuje přenášet zprávy o maximální velikost 127 bajtů. Kontrolním součtem je CRC. V rámci vyvíjeného protokolu je použita pouze 16bitová adresa.

Obě varianty linkové vrstvy byli v rámci práce implementovány.

5.3.6 Síťová vrstva

Síťová vrstva spojuje dvě zařízení, které spolu nemusí přímo sousedit. Zajišťuje směrování, hledání cest a propojení mezi různými typy linkových vrstev. Teoretický návrh síťové vrstvy vznikl v rámci této práce. Je volně inspirovaný protokoly ZigBee, DigiMESH a Atmel Lightweight Mesh.

Adresa zařízení v síťové vrstvě je totožná s adresou ve vrstvě linkové. Datové zprávy jsou nepotvrzované.

Routování

Síťová vrstva zajišťuje hledání cest mezi dvěma zařízeními. Byl zvolen relativně jednoduchý algoritmus pro dynamické hledání cesty v síti. V rámci algoritmu je důležité rozlišovat síťovou a linkovou adresu. Obě adresy má každé zařízení stejné. V rámci paketu je síťová adresa, adresa koncového zařízení k jehož docílení může být paket několikrát přeposlán. Linková adresa určuje přímého příjemce daného paketu.

Každý paket obsahuje cílovou a zdrojovou síťovou adresu a cílovou a zdrojovou linkovou adresu. V rámci přenosu jedné zprávy se síťové adresy nemění, ale linkové ano. Příkladem adres paketu je „nwk_dst: 3, nwk_src: 1, lnk_dst: 2, lnk_src: 1“

Směrování paketů určuje routovací tabulka. Záznam v routovací tabulce určuje linkovou adresu na základě síťové adresy. Například záznam „nwk_addr: 3, lnk_addr: 2“ říká, že pokud budu chtít odeslat zprávu pro zařízení se síťovou adresou 3, pošlu ji nejprve do zařízení s linkovou adresou 2 a to ji pošle dál.

- Zařízení 1 chce odeslat zprávu do zařízení 3, ke kterému nezná cestu.
- Zařízení 1 vyšle vše–směrovou zprávu s dotazem na zařízení se síťovou adresou 3. Dotaz bude mít adresy „nwk_dst: 3, nwk_src: 1, lnk_dst: X, lnk_src: 1“, kde X znamená broadcast.
- Dotaz obdrží zařízení 2, protože cestu k zařízení 3 nezná, dotaz pošle dál. Adresy dotazu budou „nwk_dst: 3, nwk_src: 1, lnk_dst: X, lnk_src: 2“. Zároveň si uloží do routovací tabulky záznam ”nwk_addr: 1, lnk_addr: 1”.
- Přeposlaný dotaz přijme zařízení 3. Potvrdí ho zpět odesílateli dotazu. Adresy potvrzení budou „nwk_dst: 1, nwk_src: 3, lnk_dst: 2, lnk_src: 3“. Do routovací tabulky si uloží ”nwk_addr: 1, lnk_addr: 2”.
- Zařízení 2 potvrzení předá dál podle tabulky (záznam ze třetího kroku). Adresy potvrzení budou „nwk_dst: 1, nwk_src: 3, lnk_dst: 1, lnk_src: 2“
- Zařízení 1 přijme potvrzení a do tabulky si uloží „nwk_addr: 3, lnk_addr: 2“.
- Pokud zařízení 1 bude opět odesílat zprávu do zařízení 3, musí ji poslat na linkovou adresu 2.

Ověřování platnosti cest se dělá sledováním komunikace mezi jednotlivými zařízeními. Každá přijatá zpráva aktualizuje záznam v routovací tabulce. Pokud selže přenos na úrovni linkové vrstvy, záznam se z tabulky maže a proces hledání cesty se opakuje.

Algoritmus umožňuje dynamické nahrazení neprůchozí cesty, cestou jinou. Poradí si s přemístěním nařízení. Do hledání lze zapojit i kvalitu spojení (počet přeskoků, kvalita signálu, vytížení cest) a vybírat nejoptimálnější cesty. Pro drátové sítě lze algoritmus optimalizovat.

Implementace

V rámci práce nebyl routovací algoritmus plně implementován, je ve fázi zkoušení a ladění. Ověřování správné funkčnosti je značně obtížné. Je potřeba většího množství prvků a ladících nástrojů, které musí mít rozestupy v řádech desítek metrů. Komunikace v rámci ústředny s měřicími moduly se omezila na jedinou sběrnici RS–485 a routování nebylo potřeba.

5.3.7 Transportní vrstva

Transportní vrstva mezi sebou propojuje jednotlivé služby daných zařízení. Propojení je definované pouze jako zpráva odeslaná danému zařízení na daný port a okamžitá odpověď na zprávu (případně potvrzení). Koncové body jsou rozděleny na dva typy. Přijímač je definovaný síťovou adresou a portem, na kterém naslouchá. Vysílač lze identifikovat pouze síťovou adresou – identifikace slouží pouze ke správnému směrování odpovědi. Vyslání zprávy probíhá následně:

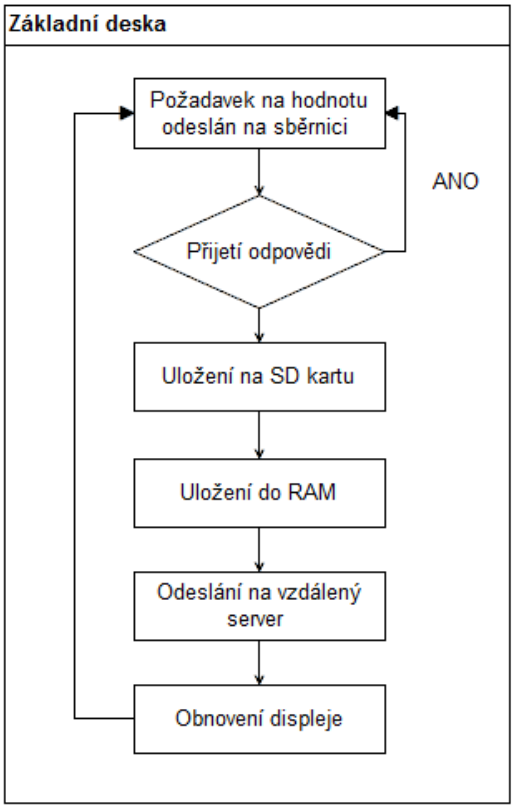
- Odesílatel vytvoří zprávu, ve které určí adresu a port přijímače, zvolí sekvenční číslo a připojí přenášená data z aplikační vrstvy.
- Transportní vrstva zprávu předá linkové vrstvě a uloží si id odesílatele, adresu, port a sekvenční číslo.
- Příjemce obdrží novou zprávu a předá ji aplikační vrstvě, která vytvoří odpověď.
- Transportní vrstva zachová stejné sekvenční číslo, port a zamění adresu příjemce a odesílatele. Zprávu odešle.
- Cílové zařízení přijme zprávu typu odpověď. Podle uložené adresy, portu a sekvenčního čísla identifikuje odesílatele.
- Odpověď předá odesílateli.
- V případě, že odpověď nedorazí před vypršením timeoutu, záznam se smaže a transportní vrstva informuje odesílatele.

V rámci práce byla vrstva implementována.

5.3.8 Aplikační vrstva

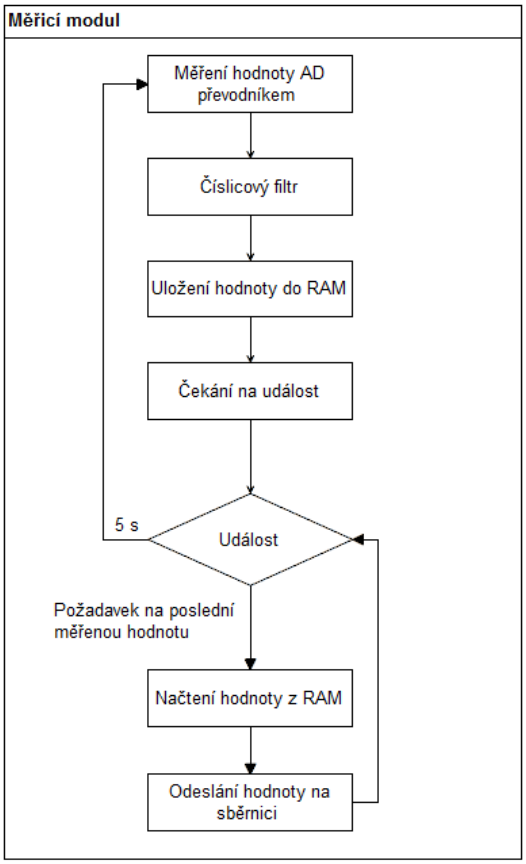
Aplikační vrstva je stále hodně otevřená. Obecně je možné použít libovolnou reprezentaci dat, pokud bude dodržen na straně odesílatele i příjemce. Pro přenos fyzikálních veličin byl zvolen jednoduchý formát, který 8bitovým identifikátorem definuje veličinu a jedním číslem ve formátu float (32 bitů) hodnotu.

Komunikace na úrovni aplikační vrstvy je definovaná jako příkaz a odpověď na něj. Například teplotní modul bude mít 8 senzorů, každý senzor bude jako služba na různých portech. Služba bude reagovat na příkaz „získat aktuální hodnotu“ a odpoví hodnotou teploty. Stejně tak bude nadřazený systém poskytovat na svých portech služby, kterými bude možné z měřicího modulu informovat o náhlé změně stavu. V takovém případě bude moci teplotní modul přímo odeslat teplotu do systému. Odpovědí bude pouze prázdná zpráva, která potvrdí úspěšný přenos.



Obrázek 5.1: Obsluha měření veličiny z pohledu základní desky.

Na diagramech 5.3.8 a 5.3.8 je znázorněn přístup k obsluze měření a komunikace. Diagram 5.3.8 popisuje vlákno (Protothreads 4.3), které se dotazuje na nové hodnoty od měřícího modulu. Diagram 5.3.8 popisuje vlákno, které na dotazy odpovídá a



Obrázek 5.2: Obsluha měření veličiny z pohledu měřicí ústředny.

zároveň zajišťuje měření fyzikální veličiny. Dvojice těchto vláken (jedno na základní desce a druhé na komunikačním modulu) je vytvořeno pro každou měřenou veličinu.

5.4 GUI knihovna

V rámci práce byla kompletně vyvinuta GUI knihovna. Je rozdělena na ovladač displeje a grafickou nadstavbu.

5.4.1 Ovladač

Ovladač displeje je odvozen od obecného rozhraní, se kterým pracuje grafická nadstavba. Poskytuje základní funkce kreslení na displej. Implementace funkcí se může lišit podle typu displeje.

- setColor – nastaví barvu, kterou se budou následující příkazy vykreslovat na displej,

- drawLine – nakreslí čáru, implementace počítá pouze s čárou svislou nebo vodorovnou,
- drawRect – nakreslí obdélník,
- drawBMP – vykreslí obrázek ve formátu BMP,
- writeLine – vykreslí text, font je předán parametrem,
- refresh – přenese všechny změny na displej

Implementace pro DOGXL160

Displej je ovládán rozhraním SPI. Jde o jednosměrnou komunikaci (signál MISO není zapojen). Kontrolním signálem se přepíná příkazový a datový režim. Komunikace je 8bitová. Při inicializaci je třeba displej smazat, nastavit parametry (kontrast, orientace, velikost) a nastavit souřadnice, na které se budou posílat obrazová data. Data se posílají v 8bitových slovech, přičemž 2 bity odpovídají barvě (4 odstíny šedé barvy) jednoho pixelu – současně se zapíšou 4 pixely. Následně se automaticky ukazatel o 4 pixely inkrementuje.

Hlavní nevýhodou tohoto řešení je, že se zapisují 4 pixely současně. Nelze například nakreslit čáru širokou jeden pixel, aniž by se nesmazal obraz v rámci 4 pixelového segmentu. Prakticky existují pouze dvě řešení. Displej rozdělit do segmentů, které se vždy vykreslí souběžně a nebudou se vzájemně ovlivňovat. Řešení je vhodné maximálně pro statické rozmístění prvků, ale i tak je značně omezující. Druhé řešení je veškeré vykreslování provádět v paměti RAM v CPU a sestavený obraz přenést jako celek na displej. Byl využit druhý způsob. Jediná funkce, která komunikuje s displejem je funkce *refresh*, všechny ostatní provádí kreslení pouze do RAM.

Ovladač umožňuje psát na displej text. Fonty jsou uloženy ve speciálním formátu ve struktuře jazyka C. Fonty jsou převedeny z fontů pro MS Windows pomocí programu FEdit.

Rozhraní je definováno jako soubor virtuálních metod, z hlediska výkonu a optimalizace kódu jde o špatné řešení. Nabízí se zde použití šablon. V dalším vývoji grafického ovladače by se měl způsob řešení přepracovat. Současné řešení by mělo smysl pouze v situaci, že by v době překladu nebylo zřejmé, jaký displej bude použit. Nicméně jde o velmi nepravděpodobný scénář v obdobném typu aplikací jako je tato.

5.4.2 GUI

Základní třídou GUI je *Object*. Deklaruje virtuální metody *draw* a *on_key*, dále nese informaci o relativní pozici a velikosti grafického prvku. Každá odvozená třída

implementuje funkce po svém.

Funkce *draw* je volána při vykreslování daného objektu. Funkce parametrem dostane referenci na objekt typu *DrawArea*. Objekt se tváří stejně jako rozhraní displeje, na který lze kreslit základními metodami, ale souřadnice vykreslovaných prvků se posunují na základě pozice grafického objektu. Tento přístup umožňuje nezávislost implementace vykreslování daného objektu na absolutní pozici na displeji. *DrawArea* zároveň kontroluje hranice oblasti, do které může objekt kreslit.

Funkce *on_key* je vyvolána na základě události. Parametrem funkce je popis události – například stisk tlačítka, tlačítko Enter.

Pro jednodušší práci se souřadnicemi byla vytvořena třída *point*. Jde o třídu, která má dvě členské proměnné – souřadnici X a Y. Třída má přetížené operátory sčítání a odčítání následujícím způsobem:

```
1  point& operator+=(const point& b)
2  {
3      this->x += b.x;
4      this->y += b.y;
5      return *this;
6  }
7
8  point operator+(const point& b)
9  {
10     return point(this->x + b.x, this->y + b.y);
11 }
12
13 point& operator-=(const point& b)
14 {
15     this->x = this->x > b.x ? this->x - b.x : 0;
16     this->y = this->y > b.y ? this->y - b.y : 0;
17     return *this;
18 }
19
20 point operator-(const point& b)
21 {
22     return point(this->x > b.x ? this->x - b.x : 0, this->y > b.y ?
           this->y - b.y : 0);
23 }
```

Přetížení umožňuje sčítání a odčítání souřadnic, přičemž jsou blokované záporné hodnoty (pokud by souřadnice měla mít zápornou hodnotu, nastaví se na nulu).

Dále je přetížen operátor operace modulo, který umožňuje libovolnou souřadnici omezit maximální hodnotou.

Třída *Layout* je odvozena od třídy *Object*. Třída představuje kontejner grafických objektů, umožňuje objekty přidávat a odebírat. Vykreslení třídy *Layout* vyvolá postupné vykreslování všech přidaných objektů. Od třídy *Object* jsou dále odvozeny třídy Label (objekt s textem), StackView (kontejner, který vykresluje pouze jeden ze svých objektů – realizace přepínání obrazovek), List (kontejner, který objekty automaticky řadí pod sebe) a analogická třída *HorizontalList*. Knihovna je připravena pro další rozšíření.

Během návrhu bylo vyzkoušeno několik přístupů. Reakce na externí událost byla původně řešena pomocí delegátů (viz 5.6.1), ale zbytečně tím narůstala paměťová náročnost, protože každý objekt musel nést typ *Delegate* i v případě, že ho nijak nevyužíval. Zpětným pohledem se jeví jako neefektivní (z pohledu využití paměti) i řešení pomocí kontejnerů. Výhodou je, že se pro vytvoření nové obrazovky nemusí implementovat žádná vykreslovací funkce. Nevýhodou je, že se všechny informace, které jsou potřeba pro vykreslení musí držet v paměti RAM. Implementací vykreslovacích funkcí na míru dané obrazovce se všechny konstantní informace nemusí do paměti RAM ukládat. V dalším vývoji knihovny by bylo vhodné implementovat tento přístup.

Na druhou stranu, pokud by se funkce implementovala a přímo by volala vykreslování dalších prvků, ušetří se ve většině objektů informace o pozici a velikosti.

5.5 Protokol Fiedler

Fiedler je binární protokol s pevným rámcem (úvodní a ukončovací znak) určený pro přenos po duplexních i poloduplexních komunikačních kanálech. Komunikaci vždy navazuje nadřízený systém na principu dotaz - odpověď. Stanice odpovídá na každý dotaz pro ni určený. Důležitým parametrem je doba označená jako klid na lince. Ta je podmínkou pro rozpoznání začátku bloku dat. Obvykle je nastavena na trojnásobek doby potřebné k odeslání jednoho bytu. Tato prodleva je nutná jak na straně SLAVE mezi přijetím dotazu a odesláním odpovědi, tak i na straně MASTER po přijetí odpovědi před vysláním dalšího dotazu. Druhým časovým parametrem je maximální doba mezi jednotlivými byty zprávy. Po jejím vypršení je zbytek zprávy ignorován a čeká se na příchod nové zprávy (SLAVE) nebo se pošle opakovaný dotaz (MASTER).

5.5.1 Implementace

Protokol Fiedler byl implementován pro externí sběrnici RS485 a zároveň pro komunikaci se vzdáleným databázovým serverem prostřednictvím TCP spojení. Na

sběrnici RS485 zařízení odpovídá na obecný dotaz sadou osmi reálných čísel. Při komunikaci po TCP může zařízení kdykoliv poslat zprávu ve stejném formátu jako je odpověď. Rozlišení jednotlivých měřicích modulů bylo vyřešeno přímou adresací modulů. Měřicí ústředna se zde tváří pouze jako router.

5.6 C++ algoritmy a techniky

Obecně byla celá práce inspirována standardní knihovnou STL, která bohužel nešla použít přímo. Jedním důvodem byla pouze částečná implementace pro požadovanou platformu a druhým bylo používání výjimek. V rámci práce vznikla knihovna, která potřebné funkce doplnila. Příkladem jsou kontejnery *Vector* a *Listf*, které jsou knihovnou STL inspirovány, včetně použití kopírovacích konstruktorů a move semantiky jazyka C++11.

Dále budou uvedeny jen nejzajímavější techniky.

5.6.1 Delegát

Delegát je stejně jako v ostatních jazycích obdobou ukazatele na funkci v jazyce C.

Pro snadnější vysvětlení bude předpokládáno následující:

- funkce je standardní konstrukce stejná jako v C,
- metoda je funkce, které je automaticky předán ukazatel na objekt, u kterého byla metoda volaná,
- nelze volat metodu bez objektu,
- typ ukazatele na metodu je závislý na typu třídy, ve které je metoda definovaná (nelze deklarovat obecný ukazatel na metodu, ale pouze ukazatel na metodu dané třídy),
- statická metoda se chová stejně jako standardní funkce, nepředává se ukazatel na objekt, ukazatel na statickou metodu lze přiřadit ukazateli na funkci.

Delegát umožňuje vyvolat libovolnou metodu libovolného objektu, která mu je přiřazena až za běhu programu. Standardně se řeší pomocí dynamické alokace paměti, případně by šlo řešit virtuálním voláním. Oboje je ale nevhodné.

Implementace, která vznikla v rámci práce byla inspirována článkem [11]. Jde o řešení, které se chová velice rychle (při maximální optimalizaci bylo volání dvakrát rychlejší, než volání virtuálních metod) a nepoužívá dynamickou alokaci paměti. Nápad spočívá ve vytvoření speciální statické šablonové funkce. Delegát nese pouze generický ukazatel na objekt a ukazatel na funkci (statickou metodu). Přiřazením

metody a objektu delegátovi se fakticky vytvoří specializace šablonové funkce (specializace na základě typu objektu a ukazatele na metodu odpovídající třídy), která se přiřadí ukazateli na funkci a adresa objektu se přiřadí generickému ukazateli. Při volání delegáta se volá statická metoda prostřednictvím ukazatele na funkci, přičemž se jí předá ukazatel na objekt jako parametr. Volaná funkce je specializovaná pro typ objektu a ukazatele na metodu odpovídající třídy, takže přetypuje generický ukazatel na ukazatel odpovídajícího typu a volá metodu ukazatelem daným specializací.

5.6.2 CRTP

Curiously recurring template pattern. Jde o návrhový vzor, který pomocí rekurentní šablony umožňuje virtuální volání časnou vazbou. Výhodou je lepší optimalizace a rychlost kódu, ale typ objektu, nad kterým se metoda volá musí být znám v době kompilace.

6 Závěr

Zadání práce bylo splněno s několika výjimkami.

Prostudování architektury procesorů ARM-Cortex M3/M4 bylo splněno v rámci implementace jednotlivých funkcí systému. Myslím, že jsem díky práci získal relativně dobrý přehled a zkušenosti s touto platformou, ale i obecně s vývojem hardware i software v embedded aplikacích a zejména s programováním v jazyce C++.

Komunikační interface mezi měřicími moduly a základní deskou byl realizován proprietárním protokolem. Tak jak byl navržen přesahuje potřeby dané aplikace. Protokol se povedlo implementovat jen částečně, například přeposílání paketů mezi moduly nebylo zcela dokončeno z časových důvodů. Nicméně implementace umožňuje komunikaci mezi moduly a základní deskou dle zadání a je napsaná tak, aby šla doplnit podle návrhu.

Uživatelské rozhraní bylo realizováno GUI knihovnou. Z časových důvodů vznikla pouze základní obrazovka. Nicméně knihovna umožňuje snadné vytváření grafických dialogů, což umožňuje operativně doplňovat grafickou podporu nových měřicích modulů a vlastností ústředny. Lokální ukládání dat bylo realizováno microSD kartou a souborovým systémem FAT32. Realizace umožňuje snadné ukládání a vyčítání libovolných dat do souborů. Pro ukládání dat byl zvolen jednoduchý formát CSV.

Protokol Fiedler byl implementován pro sběrnici RS485 i pro vzdálenou komunikaci prostřednictvím TCP spojení.

V reálném provozu byl software ověřen v laboratorních podmínkách a částečně i v bedřichovském tunelu. V laboratorních podmínkách byla ověřována funkce rozhraní Ethernet a TCP spojení se vzdáleným serverem, protože v té době nebylo k dispozici připojení k internetu přímo v tunelu. Po odstranění všech problémů s fyzickou vrstvou a ovladačem bylo spojení stabilní a spolehlivé. V tunelu bylo ověřeno měření reálných hodnot na teplotním modulu. Komunikace s nadřazeným systémem v tunelu selhala, chybu se povedlo odstranit až ve spolupráci s Ing. Milošem Hernychem v laboratorních podmínkách. Z časových důvodů nebylo měření v tunelu opakováno.

Pro nasazení zařízení jako celku do reálného provozu je třeba dokončit několik věcí.

- Doplnit aplikační vrstvu protokolu o automatické rozpoznání připojených mo-

dulů a odpovídající interakci základní desky.

- Odstranit problémy s kapacitní klávesnicí a doplnit grafickou podporu jednotlivých modulů.
- Ověřit reálnou spotřebu zařízení.
- Vytvořit bezdrátové rozhraní, kompatibilní se základní deskou.
- Výhodné by bylo zařízení rozšířit o webserver. Hardware i software je k tomu přizpůsoben, stačí vytvořit obsah.

Obecně bylo řešení celé práce velmi rozsáhlé. Dnes bych se například znovu nepustil do vlastního řešení Ethernetu. Podstatně jednodušší a rychlejší by bylo použití hotového modulu. Modul by byl sice relativně drahý, ale celkovou dobu vývoje by to zkrátilo o několik měsíců. Podobná situace je i u návrhu komunikačního protokolu, ten bych se dnes snažil vyřešit jednodušeji (co nejjednodušeji). Na druhou stranu se řešení podařilo a přineslo řadu zkušeností, díky kterým lze výsledek nasadit znovu v jiné aplikaci.

Literatura

[1] Fergus Bolger: Používání C++ v embedded systémech [online][cit. 5.4.2014] WWW: *http://www.programmingresearch.com/content/press-coverage/prqa-v%C3%BDvoj-may-2013.pdf*

[2] Manual freeRTOS [online][cit. 5.4.2014] WWW: *http://www.freertos.org*

[3] Datasheet STM32F4 [online][cit. 5.4.2014] WWW: *http://www.st.com/web/en/resource/technical/document/datasheet/DM00037051.pdf*

[4] Datasheet STM32F0 [online][cit. 5.4.2014] WWW: *http://www.st.com/web/en/resource/technical/document/datasheet/DM00039193.pdf*

[5] Datasheet STM32L1 [online][cit. 5.4.2014] WWW: *http://www.st.com/web/en/resource/technical/document/datasheet/CD00277537.pdf*

[6] Manual STM32F4 [online][cit. 5.4.2014] WWW: *http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf*

[7] Manual STM32F0 [online][cit. 5.4.2014] WWW: *http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031936.pdf*

[8] Manual STM32L1 [online][cit. 5.4.2014] WWW: *http://www.st.com/web/en/resource/technical/document/reference_manual/CD00240193.pdf*

[9] Datasheet MRF24J40 [online][cit. 5.4.2014] WWW: *http://ww1.microchip.com/downloads/en/DeviceDoc/39776C.pdf*

[10] CrossWorks Reference Manual [online][cit. 5.4.2014] WWW: *http://www.rowleydownload.co.uk/arm/documentation/index.htm*

[11] JaeWook Choi: Fast C++ Delegate [online][cit. 5.4.2014] WWW: *http://www.codeproject.com/Articles/13287/Fast-C-Delegate*

A Fotografie z tunelu

Obrázek A.1: Měření v bedřichovském tunelu.

Obrázek A.2: Bedřichovský tunel.

B Náhled databáze

Č. sen.	Čas	Hodnota		Přístroj	Veličina	Místo	Měření
0	2013-06-13 13:56:43.74	9,69	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik
0	2013-06-13 13:56:23.715	9,57	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik
0	2013-06-13 13:56:03.694	9,45	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik
0	2013-06-13 13:55:43.67	9,33	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik
0	2013-06-13 13:55:23.645	9,21	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik
0	2013-06-13 13:55:03.629	9,09	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik
0	2013-06-13 13:54:43.6	8,97	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik
0	2013-06-13 13:54:23.576	8,85	°C	GU100prototyp	Teplota	pokusny pro GU1	Test Lubos Slavik

Obrázek B.1: Náhled do vzdálené databáze s daty odeslanými TCP spojením.

C Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy následující dokumenty:

- Diplomová práce - tato práce v elektronické podobě
- Datasheety použitých obvodů
- Zdrojové kódy