Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Diplomová práce

Implementace FreeModbus na platformě STM32

Autor práce: Bc. Marek Neřold,
Vedoucí práce: Doc. Ing. Martin Poupa, Ph.D.

Plzeň 2014

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2013/2014**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Marek NEŘOLD**
Osobní číslo:	**E12N0052P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Implementace FreeModbus na platformě STM32**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

Seznamte se s architekturou mikrokontrolérů STM32F10x a vývojovými prostředky (MDK-ARM).

1. Prostudujte volně dostupnou implementaci protokolu Modbus - FreeModbus.
2. Prostudujte volně dostupnou implementaci operačního systému reálného času FreeRTOS.
3. Implementujte na platformě STM32F10x ukázkovou aplikaci realizující I/O modul s rozhraním Modbus běžící pod FreeRTOS.
4. Ukázková aplikace realizuje funkce sledování stavu vstupů, ovládání výstupů, měření teploty pomocí interních a externích teplotních senzorů, měření napětí, případně další funkce.
5. Realizaci v práci podrobně popište.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **http://www.freertos.org/**
2. **http://www.freemodbus.org/**
3. **http://www.modbus.org/**
4. **Další vhodnou literaturu si student vyhledá v dostupných pramenech.**

Vedoucí diplomové práce: **Doc. Ing. Martin Poupa, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce: **14. října 2013**

Termín odevzdání diplomové práce: **12. května 2014**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 14. října 2013

Abstrakt

Práce se zabývá implementací OpenSource knihovny FreeModbus na architektuře mikrokontrolérů rodiny STM32. Dále pak implementací a použitím operačního systému reálného času FreeRTOS.

Pro ověření funkčnosti je realizován I/O modul s rozhraním Modbus RTU založený na architektuře mikrokontroléru řady STM32. Pro navržený a realizovaný hardware I/O modulu je vytvořena ukázková SW aplikace, která umožňuje přes rozhraní Modbus RTU sledování / ovládání digitálních vstupů / výstupů, měření analogových vstupů a připojení digitálních čidel. Komunikace s modulem probíhá po sběrnici RS–485.

Klíčová slova

FreeModbus, FreeRTOS, STM32, RS485, I/O modul

Abstract

Neřold, Marek. *Implementation of FreeModbus on the STM32 platform* [*Implementace FreeModbus na platformě STM32*]. Pilsen, 2014. Master thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Applied Electronics and Telecommunications. Supervisor: Martin Poupa

---

The aim of this project is the implementation of OpenSource library FreeModbus on the architecture of the STM32 microcontroller series. Furthermore, the implementation and use of real-time operating system FreeRTOS.

To verify the functionality, the I/O module with Modbus RTU interface based on the architecture of the STM32 microcontroller series is built. For designed and implemented hardware of the I/O module the sample software application is created. The software application allows monitoring / control of digital inputs / outputs, measurement of analog inputs and connection of digital sensors. Communication with the module runs over RS–485 bus.

Keywords

FreeModbus, FreeRTOS, STM32, I/O module

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 12. května 2014

Bc. Marek Neřold,

.....

Podpis

Poděkování

Tato práce vznikla s podporou projektu SGS-2012-019: Moderní řešení elektronických řídících a informačních systémů.

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

TTL	Transistor Transistor Logic. Tranzistorově tranzistorová logika.
USB	Universal Serial Bus. Univerzální sériová sběrnice.
RTOS	Real Time Operating System. Operační systém reálného času.
NTC	Negative Temperature Coefficient. Záporný teplotní koeficient.
I/O	Input-Output. Vstupně-výstupní
PLC	Programmable Logical Controller. Programovatelný logický kontrolér.
ADU	Application Data Unit. Aplikační datová jednotka.
PDU	Protocol Data Unit. Protokolová datová jednotka.
CRC	Cyclic Redundance Check. Cyklický redundantní součet.
TCP	Transport Control Protocol. Transportní vrstva přenosového protokolu.
IP	Internet Protocol. Protocol síťové vrstvy.
LSB	Least Significant Byte. Nejméně významný byte.
MSB	Most Significant Byte. Nejvýznamnější byte.
MBAP	Modbus Application Protocol. Aplikační protokol Modbus.
RTU	Remote Terminal Unit. Jednotka vzdáleného terminálu.
ASCII	American Standard Code for Information Interchange. Americký základní kód pro výměnu informace.
CR	Carriage Return. Návrat kurzoru na začátek řádku.
LF	Line Feed. Posun o řádek.
LRC	Longitudinal Redundancy Check. Podélný redundantní součet.
RAM	Random Access Memory. Paměť s náhodným přístupem.
SRAM	Static Random Access Memory. Statická paměť s náhodným přístupem.
LSb	Least Significant Bit. Nejméně významný bit.
GPL	General Public License. Všeobecná veřejná licence.
CMSIS	Cortex Microcontroller Software Interface Standard. Standardní softwarové rozhraní Cortex mikrokontrolérů.
NVIC	Nested vectored interrupt controller. Vnořený vektorový řadič přerušení.

USART	Universal Synchronous/Asynchronous Receiver and Transmitter. Synchronní / asynchronní sériové rozhraní.
ISR	Interrupt Service Routine. Obsluha přerušení.
API	Application Programming Interface. Rozhraní pro programování aplikací.
FIFO	First In, First Out. Princip fronty.
DPS	Deska Plošných Spojů.
DC/DC	Direct Current to Direct Current. Měnič stejnosměrného proudu.
OTP	One Time Programmable. Nepřepisovatelná paměť.
ID	Identificator. Identifikátor.
GND	Ground. Uzemnění.
RTC	Real Time Clock. Hodiny reálného času.
MCU	Microcontroller Unit. Mikrokontrolér.
LED	Light Emitting Diode. Světlo emitující dioda.
SPI	Standard Peripheral Interface. Sériové periferní rozhraní.
CS	Chip Select. Signál pro volbu slave jednotky.
CD	Command / Data. Signál pro přepínání mezi příkazem a daty.

1

Úvod

Protokol Modbus je rozšířený komunikační protokol na úrovni aplikační vrstvy modelu ISO / OSI. Na trhu existuje jeho několik volně dostupných implementací. Mezi nejznámější patří knihovna FreeModbus. Jedná se o knihovnu podporující všechny hlavní funkce Modbus protokolu.

Knihovna FreeModbus je oficiálně implementována na několik platforem. Příklady těchto implementací jsou součástí knihovny. V oficiálních příkladech však chybí implementace na rozšířené 32-bitové mikrokontroléry řady STM32F10x s jádrem Cortex-M3. Z tohoto důvodu byla zvolena implementace FreeModbus na tuto platformu jako hlavní téma diplomové práce.

Hlavním úkolem je implementovat Modbus slave jednotku s komunikačním rozhraním Modbus RTU / ASCII. Ukázková SW aplikace bude realizovat funkce sledování / ovládání stavu vstupů / výstupů, měření teploty a vlhkosti, měření napětí a počítání reálného času pomocí RTC obvodu. Teplota bude měřena pomocí digitálních senzorů SHT10, DS18B20, LM92 a analogově pomocí NTC termistoru. Vzdušná vlhkost se bude měřit také senzorem SHT10.

Pro komunikaci se slave jednotkou bude použit počítačový program Modbus Poll, který slouží jako simulátor Modbus master jednotky. Komunikace s osobním počítačem bude probíhat pomocí sériové linky RS–232 s TTL napěťovými úrovněmi. Jelikož již v dnešní době osobní počítače neobsahují standardní sériový port, bude použit převodník RS–232 TTL / USB.

Dalším úkolem je zprovoznění aplikace pod volně dostupným operačním systémem reálného času FreeRTOS. Tento RTOS zaznamenává v poslední době obrovský rozvoj. Pro svou malou velikost a jednoduchost je přímo předurčen k použití v mikropočítačích. Použití RTOS přináší řadu výhod, které budou v práci popsány. V době návrhu zadání diplomové práce nebyly k dispozici oficiální příklady implementace na platformu STM32F10x. To bylo dalším důvodem, proč byla implementace FreeRTOS zvolena jako jeden z úkolů práce. Z mikrokontrolérů řady STM32F10x byl pro implementaci vybrán typ STM32F100RB. Tento mikrokontrolér je osazen na vývojovém kitu STM32 value line Discovery, což bylo jedním z hlavních důvodů jeho výběru. Vývojový kit má vyvedeny

všechny I/O piny a obsahuje všechny nezbytné externí součástky. Součástí kitu je také programátor ST-Link s podporou debugu. Napájení a programování je řešeno jednoduše přes USB port. Vývojář se tak nemusí zabývat návrhem hardwaru, ale může se plně soustředit pouze na vývoj softwarové aplikace.

Mezi další výhody patří cenová dostupnost mikrokontroléru. Vybraný typ patří mezi nejlevnější 32-bitové mikrokontroléry s Cortex-M3 architekturou a obsahuje všechny nezbytné periférie i dostatek programové a datové paměti.

Nakonec bude navržen a zkonstruován I/O modul, který bude řízen stejným MCU jako v případě vývojového kitu a bude realizovat podobné funkce. Doplněn bude o budič sběrnice RS–485 . Modul bude konstruován tak, aby se rozměrově vešel do elektroinstalační krabičky KU68L. Pro ovládání modulu bude použita SW aplikace vytvořená na vývojovém kitu.

2

Protokol Modbus

2.1 Úvod

Modbus je otevřený sériový komunikační protokol, který vyvinula v roce 1979 firma Modicon (dnes Schneider Electric) pro použití s PLC. Postupem času se stal de facto standardním komunikačním protokolem a je dnes běžně používaný pro komunikaci mezi průmyslovými elektronickými zařízeními [1].

Hlavní důvody jeho použití v průmyslovém prostředí jsou:

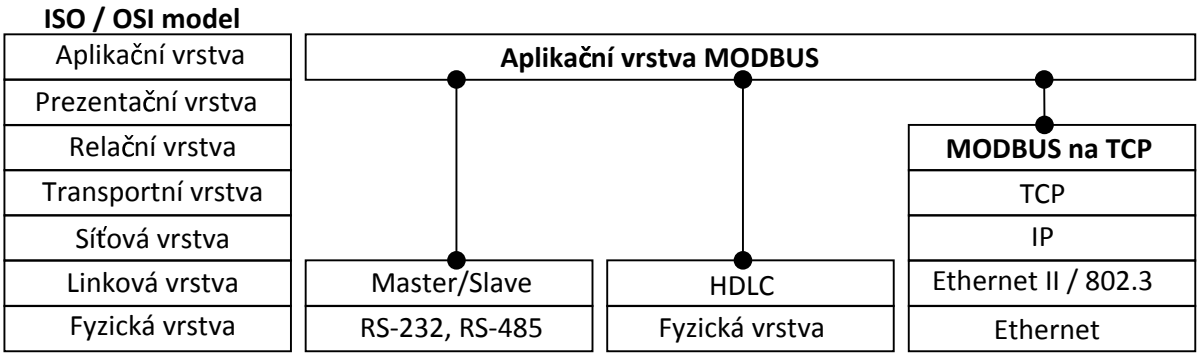
- byl vyvinut s ohledem na průmyslové aplikace
- volně šiřitelný, otevřený a bez poplatků
- jednoduchý na implementaci a správu
- přenáší jednotlivé bity nebo slova bez velikých omezení

Modbus je navržen jako nezávislý na fyzické vrstvě a pracuje na bázi výměny zpráv s definovaným formátem v síťovém prostředí — lze jej tedy považovat za protokol třetí až páté vrstvy modelu ISO/OSI (obr. 2.1). Podporována je celá řada komunikačních médií např. sériové linky RS–232, RS–422, RS–485, optické a rádiové sítě nebo síť Ethernet s využitím protokolu TCP/IP. Komunikace probíhá na principu předávání datových zpráv mezi klientem a serverem (master a slave) metodou požadavek – odpověď a požadovaná funkce je specifikována pomocí kódu funkce, jenž je součástí požadavku [3].

Od roku 2004 jsou vývoj a aktualizace Modbusu spravovány organizací Modbus, které předala práva firma Schneider Electric. Byl to jasný krok směrem k otevřenosti protokolu.

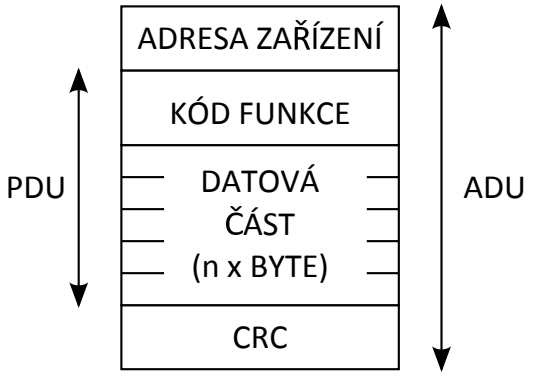
2.2 Struktura zprávy

Protokol Modbus definuje jednoduchou strukturu datové zprávy (PDU – Protocol Data Unit) nezávisle na typu nižší komunikační vrstvy. V závislosti na typu sítě nebo sběrnice,



Obr. 2.1: Příklady implementace (IP – Internet Protocol; TCP – Transport Control Protocol) |Převzato z [2]|

na které je protokol použit, je PDU rozšířena o další části a tvoří tak zprávu na aplikační úrovni (ADU – Application Data Unit). Struktura Modbus zprávy je znázorněna na obr. 2.2.



Obr. 2.2: Základní tvar Modbus zprávy (PDU – Protocol Data Unit; ADU – Application Data Unit; CRC – Cyclic Redundance Check)

Velikost Modbus PDU je limitována velikostí zděděné z první implementace Modbus na sériové lince RS–485, kde byla maximální velikost ADU 256 bytů [3].
Z toho tedy:

Max. velikost PDU na sériové lince: 256 - adresa (1 byte) - CRC (2 byty) = **253 bytů**

Odtud:

Velikost ADU na RS-485 = 253 bytů (PDU) + adresa (1 byte) + CRC (2 byty) = **256 bytů**

Velikost ADU na TCP/IP = 253 bytů (PDU) + MBAP = **260 bytů**

2.2.1 Adresa zařízení

Každému zařízení určenému ke komunikaci pomocí Modbus je přidělena jedinečná adresa. Při komunikaci pomocí sériové linky může iniciovat příkaz pouze uzel určený jako master. V síti Ethernet může poslat zprávu každé Modbus zařízení, obvykle je ale jako master zvoleno pouze jedno.

Modbus zpráva obsahuje adresu zařízení, pro které je zpráva určena. Jen takové zařízení bude na zprávu reagovat, i když ji mohou přijmout i ostatní zařízení. Výjimku tvoří speciální broadcast zprávy poslané na adresu 0, na které reagují všechna zařízení, ale reakce na tyto zprávy není od zařízení zpětně potvrzována. Všechny Modbus zprávy obsahují kontrolní součet k zajištění přijetí nepoškozené zprávy [8].

2.2.2 Kód funkce

Kód funkce je vyjádřen pomocí jednoho byte a udává, jaký druh operace má slave jednotka provést. Rozsah kódů je od 1 do 255, přičemž kódy od 128 do 255 jsou určeny pro signalizaci záporné odpovědi (chyby). Některé kódy funkcí obsahují i kód podfunkce, který upřesňuje požadovanou operaci [2].

2.2.3 Datová část

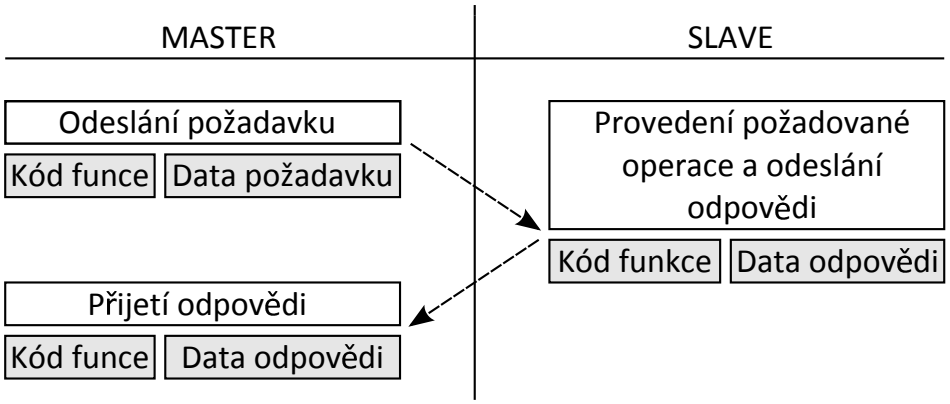
Obsah datové části zprávy poslané masterem slouží slave jednotce k provedení operace dané kódem funkce. Může obsahovat například adresu a počet vstupů, které má slave jednotka přečíst nebo hodnotu registrů, které má slave zapsat.

V některém případě nejsou pro provedení operace zapotřebí žádná další data. Datová část pak může ve zprávě zcela chybět (má nulovou délku) a kód funkce specifikuje požadovanou operaci sám o sobě [3].

2.3 Popis transakce

Protokol Modbus definuje tři základní typy zpráv (PDU):

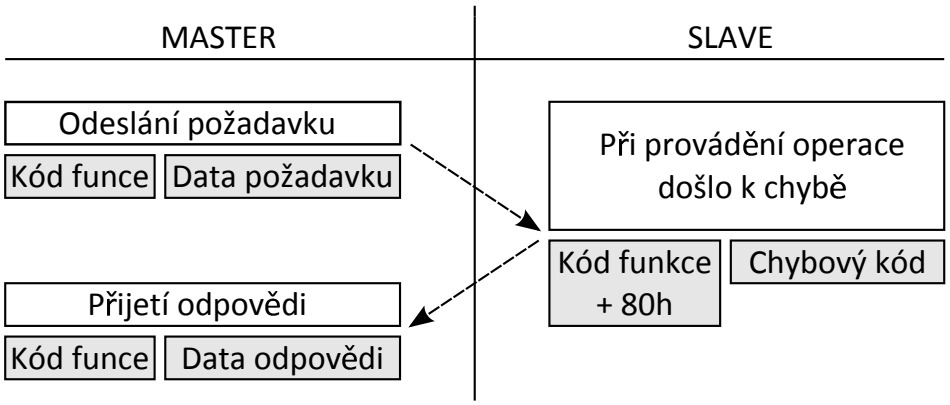
- Požadavek (Request PDU)
  - 1 Byte – Kód funkce
  - n Bytů – Datová část požadavku
- Odpověď (Response PDU)
  - 1 Byte – Kód funkce (kopie z požadavku)
  - m Bytů – Datová část odpovědi
- Chybová odpověď (Exception Response PDU)
  - 1 Byte Kód funkce + 80h (indikace neúspěchu)
  - 1 Byte Chybový kód (identifikace chyby)



Obr. 2.3: Modbus transakce s bezchybným provedením (error free) |Převzato z [2]|

Jestliže se při provádění požadované operace nevyskytne žádná chyba, bude odpověď od slave jednotky pro mastera obsahovat v poli `DATOVÁ ČÁST` požadovaná data.

Pakliže dojde při vykonávání požadované operace k chybě, bude pole obsahovat kód výjimky (exception code), která umožní serveru blíže specifikovat důvod chyby a zvolit další postup.



Obr. 2.4: Modbus transakce s chybovou odpovědí (exception response) |Převzato z [2]|

Slave jednotka používá v odpovědi pole `KÓD FUNKCE` k indikaci bezchybného vykonání požadavku. Při bezchybné odpovědi opakuje slave jednoduše původní kód funkce. V případě, že nastane chyba, vrátí slave kód funkce s nastaveným nejvyšším bitem [2].

Příklad Modbus transakce s bezchybným provedením je znázorněn na obr. 2.3 a transakce s chybou na obr. 2.4.

Pozn.: Na straně klienta je vhodné implementovat časový limit, aby klient při náhodné ztrátě odpovědi nebo požadavku nečekal na odpověď donekonečna.

2.3.1 Postup zpracování Modbus požadavku

Modbus transakci zahajuje vždy master, který posílá slave jednotce požadavek a očekává na něj odpověď. Při zpracování požadavku mohou nastat tři situace:

- zpracování požadavku slave jednotkou proběhne bezchybně

  - masteru je odeslána požadovaná odpověď

- slave jednotka požadavek nepřijme z důvodu chyby v komunikaci
  - masteru není odeslána odpověď
  - dojde k vypršení časového limitu pro příjem odpovědi (TIMEOUT ERROR)
- slave jednotka požadavek přijme, ale detekuje chybu (parita, CRC, ...)
  - masteru není odeslána odpověď
  - dojde k vypršení časového limitu pro příjem odpovědi (TIMEOUT ERROR)
- slave jednotka požadavek přijme, ale při jeho zpracování nastane chyba
  - masteru je odeslána záporná odpověď, která obsahuje v datové části kód chyby

V tab. 2.1 je uveden seznam kódů chyb, které mohou nastat [3].

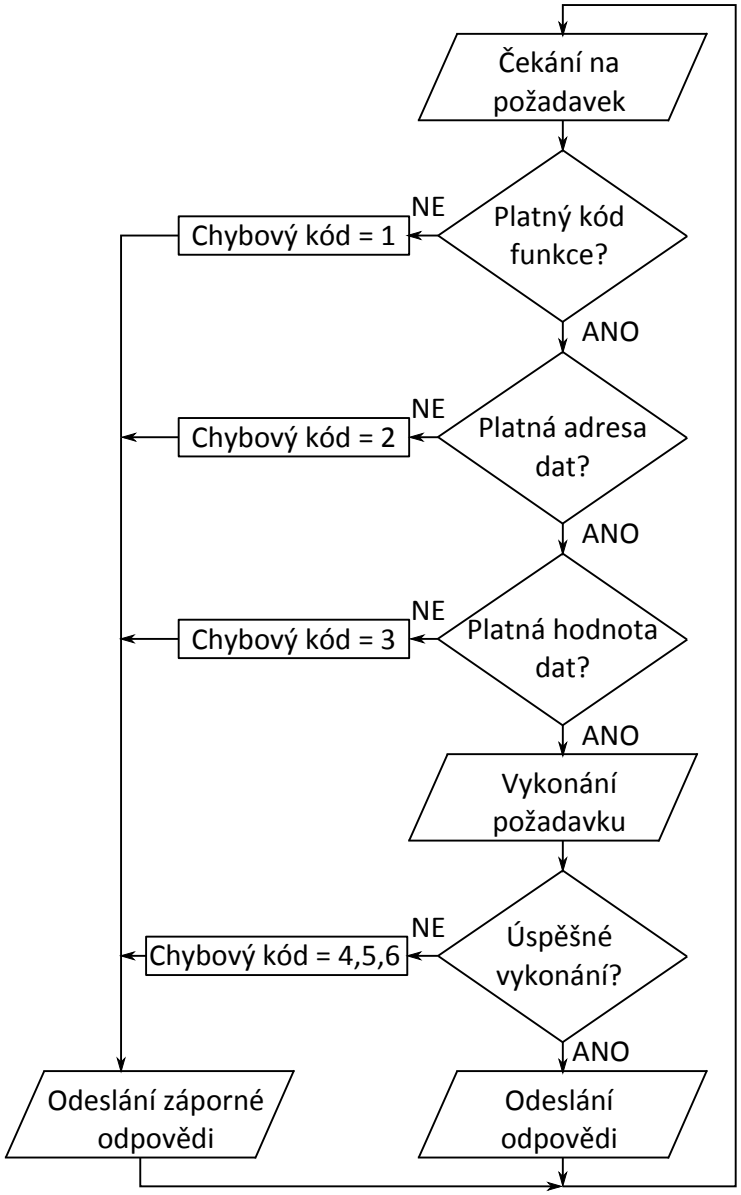
Kód	**Název**	**Význam**
01	Neplatná funkce	Požadovaná funkce není podporována
02	Neplatná adresa dat	Požadovaná adresa je mimo rozsah
03	Neplatná hodnota dat	Poslaná data jsou neplatná
04	Selhání zařízení	Při provádění požadavku došlo k neodstranitelné chybě
05	Potvrzení	Přijetí platného požadavku, jehož vykonání bude trvat delší dobu
06	Zařízení je zaneprázdněné	Slave jednotka je zaneprázdněná vykonáváním časově náročného příkazu
08	Chyba parity paměti	Kód určený k použití při práci se soubory. Slave jednotka zjistila chybu parity, při pokusu přečíst soubor

Tab. 2.1: Přehled chybových kódů protokolu Modbus

Postup při zpracování Modbus požadavku na straně slave jednotky je popsán pomocí stavového diagramu na obr. 2.5.

2.4 Kódování dat

Protokol Modbus používá pro kódování dat a adres tzv. "Big–endian" reprezentaci. To znamená, že pokud je posílána numerická hodnota větší než jeden byte, nejvyšší byte (MSB) je poslán jako první a nejnižší byte (LSB) jako poslední [2]. Příklad je uveden v tab. 2.2



Obr. 2.5: Postup zpracování Modbus požadavku ze strany slave jednotky |Převzato z [2]|

Velikost položky	Hodnota	1.poslaný byte	2.poslaný byte
16 bitů	0x1234	0x12	0x34

Tab. 2.2: Kódování dat „Big–endian“

2.5 Datový model

Datový model Modbusu je založen na několika tabulkách s charakteristickým významem. Čtyři základní tabulky jsou shrnuty v tab. 2.3.

Mapování tabulek v adresním prostoru je závislé na konkrétním zařízení. Každé tabulce může být přidělen její vlastní adresový prostor, nebo je vyhrazen společný adresový prostor pro všechny tabulky a jejich adresy se tak můžou částečně, či úplně překrývat. Každá tabulka může obsahovat až 65536 položek. Adresový prostor však bývá rozdělen

Základní tabulky	Velikost	Přístup	Komentář	Adresa
Diskrétní vstupy (Discrete Inputs)	1–bit	Pouze čtení	Data poskytována I/O systémem	10000÷19999
Cívky (Coils)	1–bit	Čtení/zápis	Data modifikovatelná aplik. programem	0÷9999
Vstupní registry (Input Registers)	16–ti bitové slovo	Pouze čtení	Data poskytována I/O systémem	30000÷39999
Uchovávací registry (Holding Registers)	16–ti bitové slovo	Čtení/zápis	Data modifikovatelná aplik. programem	40000÷49999

Tab. 2.3: Datový model Modbus protokolu

na bloky o velikosti 10000 položek tak, jak je uvedeno v tab. 2.3 [3].

2.6 Druhy funkcí

Protokol Modbus rozděluje funkce na tři základní skupiny:

Veřejné funkce

- dobře definované funkce
- funkce s garantovanou unikátností
- funkce schvalované MODBUS–IDA.org komunitou
- veřejně zdokumentované funkce
- funkce s dostupným testem shody
- zahrnují jak veřejné přiřazené funkce, tak funkce nepřiřazené, rezervované pro budoucí použití

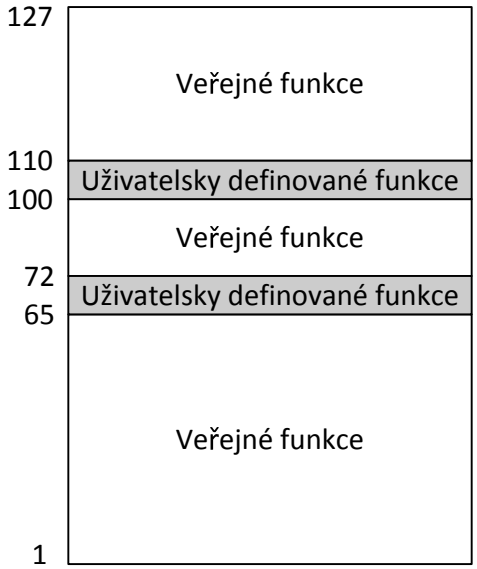
Uživatelsky definované funkce

- jsou to funkce v rozsahu od 65 do 75 a od 100 do 110
- uživatel může implementovat funkci, která není podporovaná danou specifikací
- není garantováno, že bude použití zvolené funkce unikátní
- funkci lze po schválení přesunout do veřejných funkcí

Rezervované kódy funkcí

- kódy používané v současnosti některou ze společností, které nejsou volně dostupné pro veřejné použití

Rozdělení Modbus funkcí je znázorněno na obr. 2.6 [2].



Obr. 2.6: Rozdělení Modbus funkcí |Převzato z [2]|

2.6.1 Kódy základních funkcí

Všechny podporované Modbus funkce jsou shrnuty na obr. 2.7.

Typ funkce			Jméno funkce	Kód funkce
Datový přístup	Bitový přístup	Fyzické diskrétní vstupy	Čti diskrétní vstupy	2
		Interní bity nebo fyzické cívky	Čti cívky	1
			Zapiš jednu cívku	5
			Zapiš více cívek	15
	16-ti bitový přístup	Fyzické vstupní registry	Čti vstupní registr	4
		Interní registry nebo fyzické výstupní registry	Čti uchovávací registr	3
			Zapiš jeden registr	6
			Zapiš více registrů	16
			Čti/zapiš více registrů	23
			Zapiš registr s maskováním	22
			Čti FIFO frontu	24
	Přístup k záznamům v souborech		Čti záznam ze souboru	20
			Zapiš záznam do souboru	21
Diagnostika			Čti stav	7
			Diagnostika	8
			Čti čítač kom. událostí	11
			Čti záznam kom. událostí	12
			Sděl ID slave jednotky	17
			Čti identifikaci zařízení	43
Jiné			Zapouzdřený přenos	43

Obr. 2.7: Shrnutí podporovaných funkcí |Převzato z [1]|

2.7 Modbus na sériové lince

Modbus na sériové lince je typu master / slave. Jen jeden master (ve stejný čas) je připojen na sběrnici a jedno nebo několik (nejvýše 247) slave jednotek je připojeno na tu samou sběrnici. Modbus komunikace je vždy zahájena masterem. Slave jednotky nepřenesou nikdy data, pokud o to nejsou masterem požádány, a nikdy nekomunikují navzájem mezi

sebou. Master nemůže iniciovat více než jednu komunikaci najednou.

Protokol Modbus definuje dva sériové vysílací režimy: Modbus RTU a Modbus ASCII. Režim definuje, jakým způsobem je informace ve zprávě přenášena a jak je dekódována. Vysílací režim (a parametry sériového portu) musí být stejný pro všechny jednotky na Modbus sériové lince. Všechny jednotky musí podporovat režim RTU. ASCII mód je volitelný [4].

2.7.1 Adresace

Master posílá požadavky slave jednotkám ve dvou režimech:

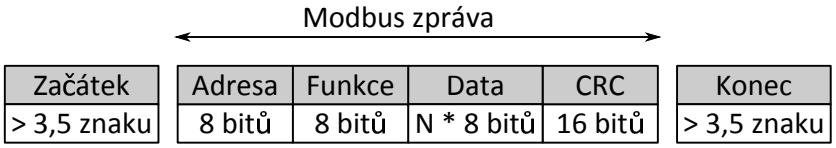
- unicast režim – master adresuje požadavek jedné konkrétní slave jednotce a ta pošle zpět odpověď (adresa 1 až 247)
- broadcast režim – master posílá požadavek všem slave jednotkám, žádná neodpoví (adresa 0)

Adresy od 248 do 255 jsou rezervovány a nejsou tedy používány.

Master nemá žádnou adresu. Pouze slave jednotky musejí mít adresu, která je na Modbus sběrnici jedinečná [4].

2.7.2 RTU mód

V režimu RTU obsahuje každý 8-bitový byte ve zprávě dva 4-bitové hexadecimální znaky. Hlavní výhodou tohoto režimu je, že jeho vyšší znaková hustota dovoluje přenést více dat při stejné přenosové rychlosti než režim ASCII. Vysílání zprávy musí být souvislé, mezery mezi znaky nesmějí být delší než 1,5 znaku. Začátek a konec zprávy je signalizován odmlkou na sběrnici delší než 3,5 znaku. Formát RTU rámce je znázorněn na obr. 2.8



Obr. 2.8: Znázornění Modbus RTU rámce zprávy |Převzato z [2]|

Ke kontrole slouží 16-bitové CRC pole s generujícím polynomem $x^{16} + x^{15} + x^2 + 1$. Formát každého bytu v režimu RTU je uveden na obr. 2.9 [4].

Start bit	Data	Parita	Stop bit
1 bit	8 bitů	1 bit	1 bit

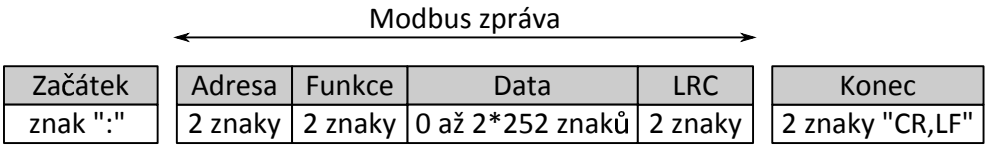
Obr. 2.9: Formát bytu v RTU rámci

Pozn. Sudá parita musí být podporována každou jednotkou. Může být použita i lichá parita. Pokud není parita použita, musí být nahrazena druhým stop bitem.

2.7.3 ASCII mód

V režimu ASCII je každý byte složen ze dvou ASCII znaků. Tento způsob je oproti RTU režimu pomalejší, ale umožňuje posílat znaky s mezerou dlouhou až 1 s. Je to dáno tím, že je začátek a konec zprávy určen odlišně než u RTU módu. Začátek zprávy je signalizován znakem „:“ a konec zprávy dvojicí znaků CR, LF.

Tímto je tato verze „lidsky čitelnější“, ale oproti verzi RTU méně využívaná.



Obr. 2.10: Znázornění Modbus ASCII rámce zprávy |Převzato z [2]|

K detekci chyb slouží 8 bitové LRC pole. Formát každého bytu v režimu ASCII je uveden na obr. 2.11 [4].

Start bit	Data	Parita	Stop bit
1 bit	7 bitů	1 bit	1 bit

Obr. 2.11: Formát bytu v ASCII rámci

Pozn. Stejně jako u RTU módu musí být každou jednotkou podporována sudá parita. Může být použita také lichá parita. Pokud není parita použita, musí být nahrazena druhým stop bitem.

3

Knihovna FreeModbus

3.1 Úvod

Knihovna FreeModbus představuje volnou implementaci Modbus protokolu, speciálně zaměřenou na embedded systémy. Jak již bylo v předchozí kapitole řečeno, komunikační protokol Modbus vyžaduje dvě vrstvy – aplikační protokol Modbus, který definuje datový model a funkce, a síťovou vrstvu. V současné verzi poskytuje FreeModbus implementaci *Aplikačního protokolu Modbus v1.1a* [3] a podporuje RTU/ASCII přenosové módy definované ve specifikaci *Modbus na sériové lince 1.0* [4].

Vše je licencované pod BSD licencí, která zakazuje používání v komerčním prostředí. FreeModbus knihovna podporuje následující Modbus funkce:

- Čti vstupní registr (0x04)
- Čti uchovávací registry (0x03)
- Zapiš jeden registr (0x06)
- Zapiš více registrů (0x10)
- Čti / zapiš více registrů (0x17)
- Čti cívky (0x01)
- Zapiš jednu cívku (0x05)
- Zapiš více cívek (0x0F)
- Čti diskrétní vstupy (0x02)
- Sděl ID slave jednotky (0x11)

Implementace je založená na nejnovějších standardech a měla by s nimi být plně kompatibilní. Přijímání a vysílání Modbus RTU/ASCII zpráv je implementováno jako stavové zařízení, které je řízené pomocí zpětného volání z hardwarové abstraktční vrstvy.

Díky tomu je portování na nové platformy poměrně snadné. Jakmile je sestavena zpráva, je předána Modbus aplikační vrstvě, kde je zpracován její obsah [5].

3.2 Hardwarové a softwarové požadavky

Hardwarové požadavky pro implementaci jsou minimální. Mikrokontrolér musí obsahovat sériové rozhraní a alespoň nějakou RAM paměť pro uchování Modbus zpráv.

Minimální požadavky:

- UART s podporou přerušení pro plný buffer přijímače a prázdný buffer vysílače
- Časovač schopný vytvořit časový limit o délce 3,5 znaku pro Modbus RTU

Pro softwarovou část je nezbytná jednoduchá fronta událostí. Pro tento účel je možné použít operační systém reálného času. Tak se ušetří čas strávený v úloze Modbus. Implementace FreeRTOS je blíže popsána v sekci 5.2. Menší mikrokontroléry budou spíše použity bez operačního systému. V takovém případě je vhodné použít jednoduché globální proměnné.

Skutečné požadavky na paměť závisejí na použitých modulech. V tab. 3.1 jsou zobrazeny paměťové nároky při použití všech dostupných Modbus funkcí [5].

Modul	**ARM kód [byte]**	**AVR kód [byte]**
Modbus RTU	1132	272
Modbus ASCII	1612	28
Modbus funkce(1)	1180	34
Modbus jádro	924	180
Implementační vrstva(2)	1756	16
Celkem	7304	530

Tab. 3.1: Datový model Modbus protokolu. (1) Skutečná velikost závisí na počtu použitých Modbus funkcí, které jsou konfigurovatelné v souboru *mbconfig.h* (2) Závisí na použitém hardware.

3.3 Modul Modbus

Tento modul definuje rozhraní pro použití. Obsahuje základní funkce potřebné pro použití Modbus protokolu. Při standardním použití se nejdříve musí zavolat funkce `eMBInit()`. Jakmile je zařízení připraveno odpovídat na požadavky ze sběrnice, zavolá se funkce `eMBEnable()`. V hlavní nekonečné smyčce je pak periodicky volána funkce `eMBPoll()`. Časový interval mezi dotazováním závisí na zvoleném časovém limitu. Při použití RTOS

je vhodné vytvořit oddělenou úlohu, která bude vždy volat funkci `eMBPoll()`. Příklad inicializace FreeModbus je předveden v následujícím úryvku kódu [5]:

```
// Inicializace protokolu v módu RTU pro slave jednotku s adresou 10 = 0x0A
eMBInit( MB_RTU, 0x0A, 38400, MB_PAR_EVEN );
// Povolení Modbus protokolu
eMBEnable(  );
for( ;; )
{
    // Zavolá hlavní dotazovací cyklus protokolu Modbus
    eMBPoll(  );
    ...
}
```

3.4 Popis hlavních funkcí knihovny FreeModbus

V následujícím textu jsou popsány základní funkce pro práci s protokolem [6].

3.4.1 Funkce eMBInit()

```
eMBErrorCode eMBInit (
    eMBMode     eMode,
    UCHAR       ucSlaveAddress,
    UCHAR       ucPort,
    ULONG       ulBaudRate,
    eMBParity   eParity
)
```

`eMBInit()` slouží k inicializaci protokolu. Tato funkce inicializuje ASCII nebo RTU režim a zavolá inicializační funkce portovací vrstvy pro přípravu hardware. Uživatel musí mít na paměti, že přijímání a zpracovávání zpráv je stále zakázáno, dokud není zavolána funkce `eMBEnable()`.

Parametry:

- *eMode* – Definuje, zda se má použít režim ASCII nebo RTU
- *ucSlaveAddress* – Adresa slave jednotky. Jsou zpracovávány jen zprávy poslané na tuto nebo na broadcast adresu
- *ucPort* – Vybraný port k použití (např. COM1 ve windows). Tato hodnota je závislá na použité platformě a nějaké implementace ji jednoduše ignorují

- *ulBaudRate* – Přenosová rychlost (např. 19200). Podporované přenosové rychlosti závisejí na implementační vrstvě
- *eParity* – Parita použitá pro sériový přenos

Návratová hodnota:

Jestliže nenastane při vykonávání žádná chyba, vrátí funkce hodnotu `eMBErrorCode::MB_ENOERR`. Protokol je potom v deaktivovaném stavu a čeká na aktivaci pomocí funkce `eMBEnable()`.

Při nesprávném provedení funkce mohou nastat následující chyby:

- `eMBErrorCode::MB_EINVAL` – Adresa slave jednotky není platná. Platné adresy slave jednotky jsou v rozsahu od 1 do 247
- `eMBErrorCode::MB_EPORTERR` – Jestliže vrátí portovací vrstva chybu.

3.4.2 Funkce eMBTCPInit()

```
eMBErrorCode eMBTCPInit ( USHORT usTCPPort )
```

Funkce `eMBTCPInit()` slouží k inicializaci Modbus TCP modulu. Zpracovávání zpráv je zakázáno, dokud není zavolána funkce `eMBEnable()`.

Parametry:

- *usTCPPort* – Zvolený TCP port

Návratová hodnota:

Jestliže byl protokol správně inicializován, vrátí funkce hodnotu `eMBErrorCode::MB_ENOERR`. V opačném případě jsou vráceny tyto chybové kódy:

- `eMBErrorCode::MB_EINVAL` – Adresa slave jednotky není platná. Platné adresy slave jednotky jsou v rozsahu od 1 do 247
- `eMBErrorCode::MB_EPORTERR` – Jestliže vrátí portovací vrstva chybu

3.4.3 Funkce eMBEnable()

```
eMBErrorCode eMBEnable ( void )
```

Funkce `eMBEnable()` povoluje zpracovávání Modbus zpráv. Povolení zpracovávání je možné jen tehdy, pokud bylo předtím v deaktivovaném stavu.

Návratová hodnota:

Jestliže je funkce provedena úspěšně a zpracovávání je nyní povoleno, vrátí hodnotu `eMBErrorCode::MB_ENOERR`. Jestliže nebylo zpracovávání v deaktivovaném stavu, vrátí hodnotu `eMBErrorCode::MB_EILLSTATE`.

3.4.4 Funkce eMBDisable()

```
eMBErrorCode eMBDisable ( void )
```

Funkce `eMBDisable()` zakazuje zpracovávání Modbus zpráv. Zakázání zpracovávání je možné jen tehdy, pokud bylo předtím v povoleném stavu.

Návratová hodnota:

Jestliže se funkce provede úspěšně a protokol je nyní v deaktivovaném stavu, vrátí hodnotu
`eMBErrorCode::MB_ENOERR`. Jestliže nebylo zpracovávání v povoleném stavu, vrátí hodnotu `eMBErrorCode::MB_EILLSTATE`.

3.4.5 Funkce eMBClose()

```
eMBErrorCode eMBClose ( void )
```

Funkce `eMBClose()` zakáže Modbus protokol a uvolní všechny hardwarové prostředky. Volání funkce je možné jen tehdy, jestliže je protokol v deaktivovaném stavu.

Návratová hodnota:

Jestliže se funkce provede úspěšně a všechny hardwarové prostředky jsou nyní uvolněny, vrátí hodnotu
`eMBErrorCode::MB_ENOERR`. Jestliže nebyl protokol při zavolání funkce v deaktivovaném stavu, vrátí funkce hodnotu `eMBErrorCode::MB_EILLSTATE`.

3.4.6 Funkce eMBPoll()

```
eMBErrorCode eMBPoll ( void )
```

Funkce `eMBPoll` tvoří hlavní dotazovací smyčku protokolu Modbus. Musí být volána periodicky. Požadovaný časový interval je dán časovým limitem slave jednotky, který je závislý na konkrétní aplikaci. Vnitřně se volá funkce `xMBPortEventGet()`, která čeká na událost od přijímače, nebo od vysílače.

Návratová hodnota:

Jestliže není protokol při zavolání funkce v povoleném stavu, vrací funkce
`eMBErrorCode::MB_EILLSTATE`, jinak vrací `eMBErrorCode::MB_ENOERR`.

3.4.7 Funkce eMBSetSlaveID()

```
eMBErrorCode eMBSetSlaveID (
    UCHAR           ucSlaveID,
    BOOL            xlsRunning,
    UCHAR const*    pucAdditional,
```

```
    USHORT          usAdditionalLen
)
```

Funkce `eMBSetSlaveID()` slouží ke konfiguraci ID slave jednotky. Používá se jestliže je povolena funkce *Report Slave ID* (definováním `MB_FUNC_OTHER_REP_SLAVEID_ENABLED` v knihovně *mbconfig.h*).

Parametry:

- *usSlaveID* – Hodnota vracená v *Slave ID* bytu, který je součástí *Report Slave ID* odpovědi
- *xlsRunning* – Při povolení (parametr `TRUE`) je *Run Indicator Status* byte nastaven na hodnotu 0xFF, jinak je jeho hodnota 0x00
- *pucAdditional* – Hodnoty, které mají být vráceny v *Additional* bytech *Report Slave ID* odpovědi
- *usAdditionalLen* – Délka bufferu *pucAdditonal*

Návratová hodnota:
Jestliže je statický buffer definovaný konstantou `MB_FUNC_OTHER_REP_SLAVEID_BUF` v knihovně *mbconfig.h* moc malý, vrací funkce `eMBErrorCode::MB_ENORES`. Jinak vrací hodnotu `eMBErrorCode::MB_ENOERR`.

3.4.8 Funkce eMBRegisterCB()

```
eMBErrorCode eMBRegisterCB (
    UCHAR                   ucFunctionCode,
    pxMBFunctionHandler     pxHandler,
)
```

Funkce `eMBRegisterCB()` slouží k vytvoření nové zpětně volané obslužné rutiny pro daný kód funkce. Vytvořená rutina je zodpovědná za interpretaci Modbus PDU a vytvoření vhodné odpovědi. V případě chyby vrací jednu z možných Modbus výjimek. Protokol poté vyšle odpovídající Modbus chybovou zprávu.

Parametry:

- *ucFunctionCode* – Kód funkce, pro kterou má být obslužná rutina vytvořena. Platné kódy funkcí jsou od 1 do 127
- *pxHandler* – Obslužná rutina, která je volána v případě, že je přijata zpráva se shodným kódem funkce Při dosazení parametru `NULL` je pro daný kód funkce odstraněna předešlá rutina

Návratová hodnota:

Jestliže byla obslužná rutina úspěšně vytvořena, vrátí funkce `eMBErrorCode::MB_ENOERR`. V případě, že nejsou k dispozici další prostředky, vrací funkce chybu `eMBErrorCode::MB_ENORES`. Pak je potřeba upravit hodnoty v knihovně *mbconfig.h*. Jestliže je zjištěn neplatný argument, je vráceno `eMBErrorCode::MB_EINVAL`.

3.5 Modbus registry

Protokol interně nepřiděluje registrům žádnou paměť. Tím je protokol velmi malý a použitelný i pro méně výkonná zařízení. Hodnoty navíc nemusí být uchovány v RAM paměti, ale můžou být uloženy například v programové paměti flash.

Kdykoliv požaduje protokol hodnotu, zavolá jednu z funkcí, která bude mít jako argument počáteční adresu registru a počet bytů k přečtení. Aplikace potom přečte aktuální hodnoty registrů (například hodnotu z A/D převodníku) a uloží výsledek do přiděleného bufferu. Jestliže protokol přijal funkci k zapsání do registru a chce aktualizovat jeho hodnotu, je funkci předán buffer s novými hodnotami. Funkce potom tyto hodnoty použije k aktualizování hodnot daných registrů.

V následujícím textu jsou shrnuty čtyři základní funkce používané pro práci s registry [6].

3.5.1 Funkce eMBRegDiscreteCB()

```
eMBErrorCode eMBRegDiscreteCB (
    UCHAR *     pucRegBuffer,
    USHORT      usAddress,
    USHORT      usNDiscrete
)
```

Funkce `eMBRegDiscreteCB()` je použita při čtení hodnot diskrétních vstupů.

Parametry:

- *pucRegBuffer* – Buffer, do kterého jsou zapisovány aktuální hodnoty diskrétních vstupů. První diskrétní vstup začínající na adrese *usAddress* musí být uložen v LSb prvního bytu bufferu. Jestliže není požadované číslo násobkem osmi, jsou zbylé bity nastaveny do nuly
- *usAddress* – Počáteční adresa prvního diskrétního vstupu
- *usNDiscrete* – Počet požadovaných hodnot diskrétních vstupů

Návratová hodnota:

- `eMBErrorCode::MB_ENOERR` – V případě, že nenastane žádná chyba. Je poslána standardní Modbus odpověď
- `eMBErrorCode::MB_ENOREG` – Pokud neexistují žádné takové diskrétní vstupy. V takovém případě je jako odpověď poslána chybová zpráva `ILLEGAL DATA ADDRESS`
- `eMBErrorCode::MB_ETIMEDOUT` – Jestliže právě není dostupný požadovaný blok registru a je překročen časový limit odpovědi pro danou aplikaci V takovém případě je jako odpověď poslána chybová zpráva `SLAVE DEVICE BUSY`
- `eMBErrorCode::MB_EIO` – Došlo k neodstranitelné chybě. V takovém případě je jako odpověď poslána chybová zpráva `SLAVE DEVICE FAILURE`

3.5.2 Funkce eMBRegCoilsCB()

```
eMBErrorCode eMBRegCoilsCB (
    UCHAR *          pucRegBuffer,
    USHORT           usAddress,
    USHORT           usNCoils,
eMBRegisterMode      eMode
)
```

Funkce `eMBRegCoilsCB()` se používá pro zapisování nebo čtení registru cívek.

Parametry:

Parametry funkce jsou analogicky stejné jako u funkce `eMBRegDiscreteCB()`, pouze se tentokrát místo s diskrétními vstupy pracuje s cívkami. Přibude však parametr *eMode*, který určuje, zdali-se má z cívek jejich hodnota číst, nebo do nich zapisovat.

- *eMode* – Jestliže `eMBRegisterMode::MB_REG_WRITE`, jsou hodnoty cívek aktualizovány hodnotami uloženými v bufferu *pucRegBuffer*
  Pokud `eMBRegisterMode::MB_REG_READ`, uloží aplikace stávající hodnoty cívek do bufferu *pucRegBuffer*

Návratová hodnota:

Stejně jako v případě parametrů funkce, i zde jsou návratové hodnoty analogicky stejné jako u funkce `eMBRegDiscreteCB()`.

3.5.3 Funkce eMBRegInputCB()

```
eMBErrorCode eMBRegInputCB (
    UCHAR *     pucRegBuffer,
    USHORT      usAddress,
    USHORT      usNRegs
)
```

Funkce `eMBRegInputCB()` se používá pro čtení hodnoty vstupního registru. Počáteční adresa registru je dána parametrem *usAddress* a adresa posledního hodnotou *usAddress + usNRegs - 1.*

Parametry:

- *pucRegBuffer* – Buffer, do kterého jsou zpětně volanou funkcí zapisovány aktuální hodnoty vstupních registrů
- *usAddress* – Počáteční adresa registru
- *usNRegs* – Počet registrů ke čtení

Návratová hodnota:

- `eMBErrorCode::MB_ENOERR` – V případě, že nenastane žádná chyba. Je poslána standardní Modbus odpověď
- `eMBErrorCode::MB_ENOREG` – Pokud nemůže aplikace dodávat hodnoty pro registry na zvoleném rozsahu adres. V takovém případě je jako odpověď poslána chybová zpráva `ILLEGAL DATA ADDRESS`
- `eMBErrorCode::MB_ETIMEDOUT` – Jestliže právě není dostupný požadovaný blok registru a je překročen časový limit odpovědi pro danou aplikaci V takovém případě je jako odpověď poslána chybová zpráva `SLAVE DEVICE BUSY`
- `eMBErrorCode::MB_EIO` – Došlo k neodstranitelné chybě. V takovém případě je jako odpověď poslána chybová zpráva `SLAVE DEVICE FAILURE`

3.5.4 Funkce eMBRegHoldingCB()

```
eMBErrorCode eMBRegHoldingCB (
    UCHAR *             pucRegBuffer,
    USHORT              usAddress,
    USHORT              usNRegs,
    eMBRegisterMode     eMode
)
```

Funkce `eMBRegHoldingCB()` se používá při zapisování nebo čtení uchovávacího registru. Počáteční adresa registru je dána parametrem *usAddress* a adresa posledního hodnotou *usAddress + usNRegs - 1*.

Parametry:

Parametry funkce jsou analogicky stejné jako u funkce `eMBRegInputCB()`, pouze se tentokrát místo se vstupními registry pracuje s uchovávacími registry. Přibude však parametr *eMode*, který určuje, zdali-se má z uchovávacích registrů číst, nebo do nich zapisovat.

- *eMode* – Jestliže `eMBRegisterMode::MB_REG_WRITE`, jsou hodnoty registru aktualizovány hodnotami z bufferu *pucRegBuffer* Pokud `eMBRegisterMode::MB_REG_READ`, zkopíruje aplikace aktuální hodnoty registrů do bufferu *pucRegBuffer*

Návratová hodnota:

Stejně jako v případě parametrů funkce, i zde jsou návratové hodnoty analogicky stejné jako u funkce `eMBRegInputCB()`.

4

Implementace FreeModbus

V následujícím textu budou dopodrobna popsány jednotlivé kroky pro správnou implementaci knihovny FreeModbus na platformě STM32. Konkrétně se bude jednat o vytvoření slave jednotky s moduly Modbus RTU a ASCII na mikrokontroléru STM32F100RB. Parametry mikrokontroléru jsou popsány v části 7.0.3.

Knihovna je dostupná na internetových stránkách tvůrce, viz [5]. V položce *Downloads* najdeme odkaz na nejnovější verzi ke stažení (aktuálně verze *freemodbus_v1.5.0*).

Po stažení souboru zjistíme, že obsahuje několik složek. Ve složce **demo** nalezneme několik příkladů implementace pro různé platformy (konkrétně 19 příkladů). Tyto ukázkové příklady jsou licencovány převážně pod licencí GPL a mohou být použity jako pomůcka pro portování na jednotlivé platformy. Jedná se například o příklady implementace pro mikrokontroléry s architekturou AVR a Cortex-M3 od firmy Atmel, MSP od firmy Texas Instruments nebo Coldfire od firmy Freescale. Nechybí ani příklady pro zařízení používající embedded Linux nebo Win32.

Složka **doc** obsahuje dokumentaci vytvořenou nástrojem Doxygen přímo ze zdrojového kódu.

Následuje složka **modbus**, která nás zajímá nejvíce. Zde jsou uloženy zdrojové kódy knihovny FreeModbus.

Poslední složka **tools** obsahuje spustitelný soubor Doxygen, který vygeneruje již zmiňovanou dokumentaci ze zdrojového kódu.

4.1 Vytvoření projektu

Při vytváření programu lze postupovat různými způsoby:

1. **Použití ukázkového projektu**

   Jestliže chci knihovnu implementovat na zařízení, pro které je již vytvořený ukázkový projekt, lze tento projekt použít a pouze upravit pro potřeby uživatele. Pokud pro zvolené zařízení ukázkový projekt neexistuje, ale existuje na zařízení, které má

podobné vlastnosti (například používá stejnou architekturu), mohu takový projekt také použít, ale jeho úpravy budou značně větší než v prvním případě.

2. **Vytvoření zcela nového projektu**

   V případě, že na zařízení neexistuje ukázkový projekt nebo ho uživatel z nějakého důvodu nechce použít. Tato možnost bude popsána v následujícím textu.

Program byl odladěn na vývojovém kitu STM32 value line Discovery, který obsahuje mikrokontrolér STM32F100RB. Tento kit se programuje a napájí pomocí mini USB. Obsahuje programátor a debugger ST-Link, uživatelské a resetovací tlačítko, dvě signalizační LED diody a má vyvedeny všechny I/O piny. Tyto vlastnosti dělají z kitu ideální nástroj pro vývoj aplikací.

Pro tvorbu programu bylo zvoleno vývojové prostředí MDK-ARM_V5 od společnosti Keil. Konkrétně pak jeho lite verze, jejíž používání je omezeno velikostí vygenerovaného kódu a dat na 32 Kbytů. Toto prostředí je určené pro zařízení využívající architekturu ARM Cortex. V základu obsahuje nástroje pro komunikaci s ST-Link debuggerem a pro programování flash paměti mikrokontroléru. Není tedy potřeba instalovat žádné další aplikace. Lite verze programu je zdarma dostupná ke stažení z internetových stránek výrobce, viz [20].

V tomto prostředí byl vytvořen zcela nový projekt a v něm složky **Modbus**, **ASCII** a **RTU**, kam byly podle funkce vloženy zdrojové kódy knihovny FreeModbus.

Pro komunikaci mezi jádrem procesoru a jeho periférie mi byla použita knihovna CMSIS. Tato knihovna je volně dostupná a je například součástí balíku `STM32F10x_StdPeriphLib_V3.5.0` od společnosti ST Microelectronic, který je volně dostupný ke stažení na internetových stránkách firmy, viz [17].

Knihovna CMSIS je abstraktní vrstva nezávislá na dodavateli hardwaru určená pro Cortex-M procesory a specifická ladící rozhraní. Vytváří konzistentní a jednoduché softwarové rozhraní mezi procesorem a jeho periférie mi. Knihovna zjednodušuje opětovné použití softwaru a snižuje učící křivku pro nové vývojáře aplikací.

Z knihovny byly použity soubory *core_cm3.c*, *system_stm32f10x.c* a zaváděcí soubor *startup_stm32f10x_md_vl.s* psaný v assembleru. Tyto soubory byly v projektu vloženy do složky **CMSIS**. Zároveň byla nastavena cesta k jejich hlavičkovým souborům.

Dále byla v projektu vytvořena složka **port**, ve které byly vytvořeny soubory *portother.c*, *portevent.c*, *portserial.c* a *porttimer.c*, sloužící ke správnému naportování FreeModbus na příslušnou platformu.

V souboru *portother.c* byly napsány jednoduché funkce sloužící k ovládání kritických částí programu. K tomu byly použity již hotové rutiny `taskENTER_CRITICAL()` a `taskEXIT_CRITICAL()` z operačního systému reálného času FreeRTOS, viz kap. 5. Funkce globálně nastavují a povolují přerušení v případě, kdy je potřeba provést časově kritické části kódu a program nesmí být „rušen“ obsluhami přerušení.

Hlavní soubor projektu *main.c*, který obsahuje funkci `main()`, ve které celý program začíná, byl v projektu vložen do složky **Main**. V této funkci je FreeModbus inicializován podobným způsobem, který byl předveden v kap. 3.3. V souboru jsou také vytvořeny funkce `eMBRegInputCB()`, `eMBRegHoldingCB()`, `eMBRegDiscreteCB()` a `eMBRegCoilsCB()`, popsané v kap. 3.5.

Dále nesmíme zapomenout nastavit konfigurační soubor *mbconfig.h*, ve kterém povolujeme nebo zakazujeme jednotlivé FreeModbus moduly a funkce a nastavujeme jejich vlastnosti. V mém případě bylo použito defaultní nastavení knihovny, kde jsou povoleny všechny funkce, režimy Modbus RTU, Modbus ASCII a zakázán modul Modbus TCP.

4.1.1 Práce s periferiemi

Práce s periferiemi mikrokontroléru (např. čtení hodnoty z A/D převodníku, komunikace po UART sériovém kanálu atd.) je obecně možná dvěma způsoby. Buďto si vývojář pilně prostuduje datový list daného mikrokontroléru a poté periférie ovládá pomocí přímého přístupu do registrů. Nevytváří univerzální funkce, ale zaměřuje se pouze na řešení konkrétního úkolu. V takovém případě je velikost zdrojového kódu obecně menší. Jsou však kladeny vysoké nároky na vývojáře a časová náročnost tohoto řešení je větší. Problém nastává tehdy, pokud je potřeba funkci programu upravit. V takovém případě mnohdy nestačí pouze změnit hodnotu konstanty, ale vývojář musí složitě přepracovat celý program.

Druhým způsobem je použití knihoven určených pro zjednodušení práce s periferiemi mikrokontroléru. Takových knihoven je k dispozici hned několik, od volně dostupných, až po komerční řešení. Knihovny slouží k usnadnění a zrychlení práce vývojáře. Ten nemusí dopodrobna studovat datové listy a nastavovat hodnoty jednotlivých registrů. Knihovny obsahují univerzální funkce, u kterých lze měnit jejich vlastnosti jednoduše změnou některého z parametrů.

V mém řešení byla použita knihovna `STM32F10x_StdPeriph_Driver`, která je součástí již zmiňovaného balíku `STM32F10x_StdPeriphLib_V3.5.0`. Knihovna obsahuje zdrojové kódy pro ovládání všech periférií mikrokontroléru. Kódy jsou podle funkce přehledně rozděleny do několika souborů. Potřebné soubory byly v projektu vloženy do složky **StdPeriph_Driver** a byla nastavena cesta k jejich hlavičkovým souborům.

Použité soubory musíme povolit odkomentováním příslušných řádek v souboru *stm32f10x_conf.h* a definováním symbolické konstanty `USE_STDPERIPH_DRIVER`.

Dále musíme upřesnit typ použitého mikrokontroléru definováním symbolické konstanty `STM32F10X_MD_VL` (Medium Density Value Line) nebo odkomentováním příslušné řádky v souboru *stm32f10x.h*. Symbolické konstanty můžeme definovat buď libovolně v preprocesoru kódu, nebo v nastavení projektu v záložce C/C++ v políčku *preprocessor symbols*.

4.1.1.1 Nastavení časovače

Jak již bylo řečeno dříve, je pro správnou implementaci Modbus RTU potřeba časovač schopný vytvořit časový limit o délce 3,5 znaku. Pro tuto funkci byl v mikrokontroléru zvolen 16-ti bitový Časovač 2 (TIM2).

V projektu byl ve složce **port** vytvořen soubor *porttimer.c* a v něm napsány jednotlivé funkce pro práci s časovačem.

Pro inicializaci časovače byla vytvořena funkce `xMBPortTimersInit()` s parametrem *usTim1Timerout50us*, který udává periodu časovače v násobcích 50 μs.

V této funkci byl pro TIM2 nejprve povolen hodinový signál z pomalé sběrnice APB1. Poté byl počet period časovače nastaven na hodnotu danou zmiňovaným parametrem *usTim1Timerout50us*. Aby byla jedna perioda časovače přesně 50 μs, musela být frekvence systémových hodin o hodnotě 24 MHz vydělena pomocí předděličky číslem 1199. Dále nastavíme hodnotu dělení hodin na nulu, směr čítání směrem nahoru a povolíme přednastavení časovače.

Jelikož je potřeba využívat přerušení generované časovačem, musí být pro TIM2 nastaven NVIC blok určený pro obsluhu přerušení. V použitém mikrokontroléru může řadič NVIC obsloužit až 56 externích přerušení (plus 16 přerušení přímo od jádra Cortex M3) s 16 úrovněmi priorit (4 bity), které můžou být dynamicky měněny. Je podporována i podpriorita, která ale musela být v mém řešení zakázána z důvodu použití FreeRTOS operačního systému popsaného v kap. 5.

Nastavení priority přerušení může být matoucí. Platí pravidlo, že čím nižší číslo, tím vyšší priorita. U obsluhy přerušení od čítače byla nastavena priorita na hodnotu 1.

Nakonec už jen vynulujeme příznakový bit přerušení od časovače a zatím zakážeme generování přerušení od časovače i samotný časovač. Tím je inicializace hotova.

Dále byly vytvořeny funkce `vMBPortTimersEnable()` pro povolení časovače, `vMBPortTimersDisable()` pro jeho zakázání a `vMBPortTimerClose()` pro zakázání obsluhy přerušení od časovače.

Nakonec byla vytvořena obsluha přerušení, která volá při každém přerušení od časovače funkci `pxMBPortCBTimerExpired()`. Tím dá protokolu vědět, že časový limit vypršel.

4.1.1.2 Nastavení sériového rozhraní

Nedílnou součástí implementace knihovny FreeModbus je správné nastavení sériového rozhraní USART.

K tomuto účelu byl ve složce **port** vytvořen soubor *portserial.c* a v něm funkce pro obsluhu USARTu.

Pro inicializaci USARTu slouží funkce `xMBPortSerialInit()` s parametry *ucPORT*, *ulBaudRate*, *ucDataBits* a *eParity*.

Parametr *ucPORT* určuje vybraný port k použití. Tato hodnota je závislá na použité platformě a některé implementace ji jednoduše ignorují. *ulBaudRate* slouží k definování

přenosové rychlosti, *ucDataBits* k určení počtu datových bitů a parametr *eParity* ke zvolení parity.

Ve funkci je nejprve povolen hodinový signál z rychlé sběrnice APB2 pro periférie USART1 a GPIOA. Potom se zvolí TX pin USARTu jako výstup v režimu push-pull a pin RX jako plovoucí vstup.

Dále se nastaví přenosová rychlost USARTu na hodnotu danou parametrem *ulBaudRate*. Podle parametru *eParity* je určena parita a upraví se konečná délka slova. Je zvolen jeden stop bit a žádná hardwarová kontrola přenosu. Inicializace je dokončena povolením režimu vysílání i příjmu.

Nesmí se zapomenout také na inicializaci NVIC řadiče, která je provedena analogicky jako v případě časovače.

Nakonec je povolen příjem i vysílání a generování přerušení od přijímače USARTu. Tím je inicializace dokončena.

Podobně jako u časovače jsou vytvořeny funkce `vMBPortSerialEnable()` s parametry *xRxEnable* a *xTxEnable*, pro povolení nebo zakázání přerušení od přijímače nebo vysílače. Dále pak funkce `xMBPortSerialPutByte()` pro poslání bytu, `xMBPortSerialGetByte()` pro přijmutí bytu a funkce `vMBPortSerialClose()` sloužící k deaktivaci USARTu.

Nakonec byly vytvořeny jednoduché obsluhy přerušení `prvvUARTRxISR()` a `prvvUARTTxReadyISR()` vyvolané přijímačem při přijetí bytu a vysílačem při poslání posledního bytu, které zpětně volají příslušné FreeModbus funkce.

4.2 Komunikace s Modbus masterem

Po úspěšném zkompilování programu a jeho nahrání do mikrokontroléru je možno začít s Modbus slave jednotkou komunikovat. K tomu však je potřeba nějaký Modbus master. K tomuto účelu byl vybrán počítačový program Modbus Poll, který na osobním počítači pod systémem Windows simuluje chování Modbus master jednotky.

Komunikace mezi STM32 a osobním počítačem probíhá po sériové lince s využitím protokolu RS–232 s TTL napěťovými úrovněmi. Moderní osobní počítače a notebooky již postrádají standardní sériový port (COMx), jelikož byly nahrazeny uživatelsky přívětivějším USB rozhraním. Proto byl pro převod komunikace mezi RS–232 TTL a USB použit převodník PL2303 od firmy Profilic. Převodník se ve Windows tváří jako virtuální komunikační port, ke kterému již lze jednoduše přistupovat jako k sériovému portu.

V programu Modbus Poll nejdříve zvolíme ID slave jednotky. Poté určíme typ požadované Modbus funkce. V ní nastavíme počáteční adresu Modbus registru a počet jeho prvků. Dále nastavíme typ připojení a jeho parametry. Zvolíme Modbus mód, definujeme časový limit pro odezvu od Modbus slave jednotky a prodlevu mezi Modbus dotazy. Po úspěšném připojení již můžeme v tabulce vidět hodnoty jednotlivých prvků registru, které se aktualizují s každým dotazovacím cyklem.

Jestliže chceme do registru slave jednotky hodnotu zapsat, napíšeme jí jednoduše do

daného políčka a při dalším dotazu se hodnota do Modbus registru zapíše. Zvolená funkce musí samozřejmě podporovat zápis.

Menší komplikace nastává, jestliže chceme poslat hodnotu větší než 16 bitů (např. float, long int, atd.). V takovém případě musíme data rozdělit do několika 16-ti bitových částí a ty poslat jednotlivě v každém dotazovacím cyklu. Modbus Poll je pak může jednoduchým nastavením zobrazit jako jednu hodnotu. Způsob rozdělení je předveden v následující ukázce kódu:

```
float       temp = 33,3;
uint16_t    *p_temp = (uint16_t*)&temp;

usRegInputBuf[1] = *p_temp;
usRegInputBuf[2] = *(++p_temp);
```

5

Operační systém reálného času FreeRTOS

5.1 Popis FreeRTOS

Mikrokontroléry s jádrem Cortex-M3 poskytují dostatek výkonu pro implementaci celé řady rozsáhlých funkcí. S vytvářením funkcí však roste složitost programu a zvyšuje se obtížnost jeho optimalizace. Použití jen jedné hlavní smyčky je v takovém případě problematické, hlavně v případě zpracování časově náročných úkolů.

Vhodným řešením je použití některého z operačních systémů reálného času (RTOS). Tyto systémy umožňují paralelní běh několika úloh, tedy nekonečných smyček. Zabudovaný plánovač pak zajišťuje vhodné přidělení času jednotlivým úlohám. Na trhu existuje celá řada RTOS. Některé jsou volně dostupné ke stažení a s volným kódem [12].

FreeRTOS je oblíbený operační systém reálného času volně dostupný pod licencí GPL. V současné době je oficiálně dostupný pro 33 architektur a za rok 2013 zaznamenal 107000 stažení. Jedná se o profesionálně vyvinutý robustní systém primárně určený pro embedded aplikace. Může být použit i v komerčních aplikacích bez nutnosti zveřejnění kódu [10].

5.1.1 Úlohy

Aplikace v reálném čase, která používá RTOS, je strukturovaná jako sada nezávislých úloh (anglicky task). Každá úloha se vykonává v rámci svého kontextu a je nezávislá na ostatních úlohách nebo RTOS plánovači. V daném čase může být vykonávána vždy jen jedna úloha. O tom, která to bude, rozhoduje RTOS plánovač. Ten při provádění aplikace opakovaně spouští a zastavuje jednotlivé úlohy (přepíná mezi jejich obsluhou). Jelikož nemá úloha žádné informace o aktivitě plánovače, je jeho povinností zajistit, aby byl kontext procesoru (hodnoty registrů, obsah zásobníku, atd.) při změně úlohy přesně stejný jako při jejím posledním vykonáváním. K tomuto účelu je každá úloha opatřena svým vlastním zásobníkem. Jakmile se plánovač chystá úlohu přepnout, je její kontext uložen do vlastního zásobníku. Po návratu k obsluze úlohy je její přesný kontext pomocí

zásobníku obnoven.

Úloha se může nacházet v jednom z následujících stavů:

1. **Běžící**

   V případě, že je úloha právě vykonávána a využívá procesor.

2. **Připravený**

   Připravené úlohy jsou schopné vykonání (nejsou zablokované ani zakázané), ale nejsou právě vykonávány, jelikož je vykonávána jiná úloha s větší nebo stejnou prioritou.

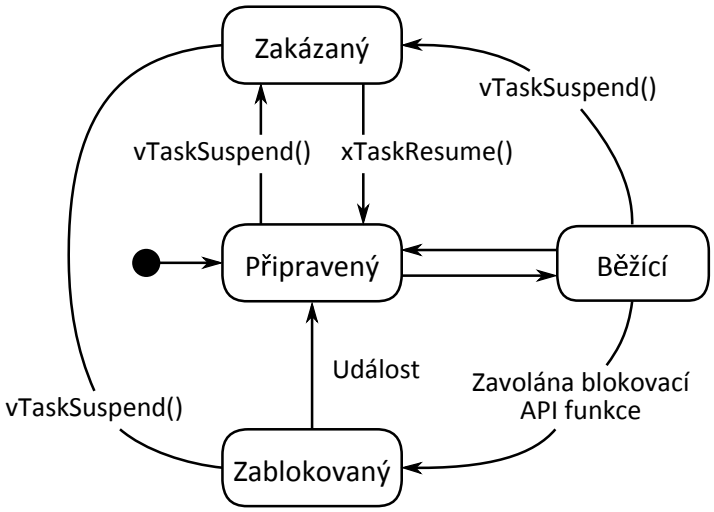
3. **Zablokovaný**

   Zablokovaná úloha čeká na nějakou časovou nebo externí událost. Takové úlohy nejsou k dispozici pro plánování.

4. **Zakázaný**

   Úlohy mohou vstoupit do a vystoupit ze zakázaného stavu pouze pomocí zavolání funkcí `vTaskSuspend()` and `xTaskResume()`. Stejně jako u zablokovaného stavu nejsou tyto úlohy k dispozici pro plánování [10].

Typy stavů a možnosti přechodů mezi nimi jsou znázorněny na obr. 5.1.



Obr. 5.1: Druhy stavů systému FreeRTOS a způsoby přechodů mezi nimi. |Převzato z [10]|

5.1.2 Plánovač

Plánovač (anglicky scheduler) je část jádra systému, která rozhoduje o tom, která úloha bude v daném čase vykonána. Konvenční procesor je schopný vykonávat v daném čase

pouze jednu úlohu. Plánovač vytváří iluzi multitaskingu tím, že rychle přepíná mezi jednotlivými úlohami. Obvyklá perioda přepínání je 1 ms. V průběhu životnosti úlohy jí jádro několikrát pozastaví a později opět obnoví.

Operační systémy reálného času jsou navrženy tak, aby poskytly potřebnou časovou odezvu pro vykonání úlohy. Události v reálném světě nejsou deterministické a plánovač musí zajistit, aby byly časově kritické úlohy vykonány v požadovaném čase. K tomu se používá tzv. politika plánování.

Politika plánování je algoritmus používaný plánovačem k rozhodnutí, která úloha má být v daném čase vykonána. Softwarový vývojář nejdříve přiřadí každé úloze odpovídající prioritu. Algoritmus poté jednoduše zajistí přidělení procesorového času dostupné úloze s nejvyšší prioritou. Úlohám, které jsou schopné vykonání ve stejný čas a mají stejnou prioritu, je přidělován procesorový čas rovnoměrně. Jestliže není k dispozici žádná úloha, je automaticky spuštěna nečinná úloha vytvořená FreeRTOSem, která má přidělenu nejnižší možnou prioritu [11].

5.1.3 Fronty

Fronty patří mezi hlavní způsob komunikace mezi úlohami. Mohou být použity k posílání zpráv mezi úlohami nebo mezi přerušeními a úlohami. Většinou jsou používány jako bezpečné FIFO vyrovnávací paměti s novými daty posílanými na začátek řady.

Narozdíl od obvyklých řešení, kdy je na data z fronty pouze odkazováno, jsou data do fronty fyzicky zkopírována. To přináší řadu výhod, jejichž popis přesahuje rámec tohoto textu.

Jestliže se úloha pokusí zapsat do plné fronty nebo číst z fronty prázdné, je automaticky umístěna do zablokovaného stavu. V případě, že je stejnou frontou zablokováno více úloh, je při odblokování uvolněna úloha s vyšší prioritou jako první [10].

5.1.4 Binární semafory

Semafory jsou využívány především pro synchronizační účely a vzájemné vyloučení. Na rozdíl od mutexů neobsahují mechanismus dědičné priority. To dělá ze semaforů lepší nástroj pro synchronizaci a z mutexů lepší volbu pro implementaci vzájemného vyloučení.

Funkce semaforu umožňují stanovit blokovací čas. Ten definuje maximální počet hodinových impulsů, za které vstoupí úloha do zablokovaného stavu, když se snaží „převzít“ semafor a ten není ihned k dispozici. Jestliže je jedním semaforem zablokováno více úloh, je v momentě dostupnosti semaforu odblokována úloha s nejvyšší prioritou jako první.

Semafor si můžeme nejsnáze představit jako frontu, která může uchovat maximálně jeden prvek. Taková fronta pak může být buď prázdná, nebo plná. Tento mechanismus může být využit například k synchronizaci úlohy s přerušením [10].

5.1.5 Mutexy

Mutexy jsou binární semafory, které obsahují mechanismus dědičné priority. Slovo mutex je odvozeno od anglického slova mutual exclusion, což lze do češtiny přeložit jako vzájemné vyloučení.

Pro použití pro vzájemné vyloučení se Mutex chová jako žeton (token), který střeží nějaký zdroj. Jestliže chce úloha k tomuto zdroji přistoupit, musí nejdříve žeton získat. Jestliže už zdroj nepotřebuje, musí dát žeton zpět k dispozici pro ostatní úlohy. Stejně jako semafory umožňují mutexy nastavit blokovací čas.

Narozdíl od semaforů používají mutexy dědičnou prioritu. V případě, že je úloha s vyšší prioritou zablokována, zatímco se snaží získat žeton, který má zrovna v držení úloha s nižší prioritou, je priorita úlohy držící žeton automaticky zvýšena na prioritu stejnou jako má zablokovaná úloha. Tento mechanismus zajišťuje, že je úloha s vyšší prioritou zablokována po nejkratší možný čas. Minimalizuje to tak vzniklou inverzi priority [15].

Dědičná priorita nezabraňuje vzniku inverzní priority, ale pouze v určitých situacích minimalizuje její vliv. Robustní aplikace reálného času by měly být navrhovány tak, aby inverzní priorita vůbec nevznikla [10].

5.2 Implementace FreeRTOS

V následujícím textu bude popsán postup pro správnou implementaci operačního systému reálného času FreeRTOS na mikrokontroléru STM32F100RB. Pod systémem FreeRTOS bude poté spuštěna implementace knihovny FreeModbus popsaná v kap. 4.

Nejdříve je potřeba stáhnout z internetových stránek výrobce, viz [10], knihovnu s nejnovější verzí FreeRTOS. V době vytváření projektu to byla verze `FreeRTOS_V7.6.0`.

Stažený soubor obsahuje složky FreeRTOS a FreeRTOS–Plus. Složka FreeRTOS obsahuje složky **Demo**, ve které jsou vytvořeny vzorové projekty pro různé platformy, **License**, ve které je uložený pouze licenční soubor a **Source**, která obsahuje zdrojové kódy systému FreeRTOS. Složku FreeRTOS–Plus budeme ignorovat.

5.2.1 Založení projektu

FreeRTOS je navržený tak, aby byl jednoduchý na implementaci. Pro správnou funkci jsou požadovány pouze tři zdrojové soubory, které jsou pro všechny platformy stejné, a jeden zdrojový specifický soubor pro daný mikrokontrolér.

Pro implementaci FreeRTOS musíme do projektu přidat tyto soubory:

- `FreeRTOS/Source/tasks.c`
- `FreeRTOS/Source/list.c`
- `FreeRTOS/Source/queue.c`

- `FreeRTOS/Source/portable/[kompilátor]/[architektura]/port.c`

Jestliže chceme používat softwarové časovače, potřebujeme ještě soubor `FreeRTOS/Source/timers.c` a pro používání co-rutin soubor `FreeRTOS/Source/croutine.c`.

Dále musíme přidat soubor pro správu paměti. Ze čtyř možností byl vybrán soubor `FreeRTOS/Source/portable/MemMang/heap_2.c`, ve kterém je vytvořen nejlepší algoritmus pro alokaci a uvolňování paměti.

V nastavení projektu se musí definovat cesty k odpovídajícím hlavičkovým souborům.

5.2.2 Knihovna FreeRTOSConfig.h

Knihovna *FreeRTOSConfig.h* obsahuje nastavení, které přizpůsobuje FreeRTOS dané aplikaci. Nastavuje se zde frekvence hodin procesoru, celková velikost alokované paměti, počet priorit úloh, povolení používání mutexů atd. Nastavení knihovny bylo s úpravami převzato ze vzorového projektu pro mikrokontroléry s architekturou Cortex-M3.

5.2.3 Vytvoření aplikace

Nyní je již vše správně nastaveno a může být vytvořen soubor *main.c*, který bude obsahovat jádro aplikace. V tomto souboru jsou ve funkci `main()` vytvořeny jednotlivé úlohy s odkazem na předvytvořené funkce a spuštěn plánovač. Vytvoření úlohy se provede pomocí funkce

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE                 pvTaskCode,
    const signed char *const    pcName,
    unsigned short              usStackDepth,
    void                        *pvParameters,
    unsigned portBASE_TYPE      uxPriority,
    xTaskHandle                 *pvCreatedTask
);
```

s parametry:

- *pvTaskCode* – Jméno vytvořené funkce obsahující nekonečnou smyčku
- *pcName* – Jméno úlohy sloužící pouze pro popis
- *usStackDepth* – Velikost vytvořeného zásobníku pro danou úlohu
- *pvParameters* – Parametr pro vytvořenou funkci
- *uxPriority* – Priorita vytvořené úlohy

- *pxCreatedTask* – Obsluha úlohy

Jakmile jsou vytvořeny všechny úlohy, je spuštěn časovač, který se již postará o obsluhu jednotlivých funkcí.

Pro časově kritické úseky v programu, kdy jsou kladeny extrémně vysoké nároky na dodržení časových intervalů (např. při inicializaci digitálního senzoru), je použita funkce `vTaskSuspendAll()`. Tato funkce pošle dočasně všechny ostatní úlohy do zakázaného stavu. Prováděná úloha tak není „vyrušována“ ostatními úlohami a může korektně provést požadovanou operaci. Jakmile je kritická část programu provedena, je zavolána funkce `xTaskResumeAll()` a zakázané úlohy jsou vráceny zpět do připraveného stavu.

Pro zakázání veškerých přerušení slouží funkce `taskENTER_CRITICAL()` a pro opětovné povolení funkce `taskEXIT_CRITICAL()`. Tyto a předešlé funkce by se měly používat opravdu jen v případě nutnosti a pro co nejkratší části kódu, jinak se omezuje využití RTOS a tím se značně snižuje kvalita výsledného řešení.

6

Použitý hardware

6.1 Senzor SHT1x

SHT1x je kombinovaný senzor teploty a relativní vlhkosti využívající technologii CMOSens®. Každý senzor je kalibrován výrobcem ve vlhkostní komoře a kalibrační koeficienty jsou uloženy do OTP paměti, která je součástí senzoru. Tyto koeficienty jsou použity interně při každém měření pro přepočítávání údajů ze senzorů. Kalibrace senzoru vlhkosti je poměrně složitá a v amatérských podmínkách prakticky nemožná. Proto je přesná kalibrace senzoru výrobcem obrovskou výhodou a podstatně usnadňuje práci s čidlem.

Senzor dále obsahuje analogový zesilovač sloužící k zesílení signálů od senzorů a 14-bitový A/D převodník. Pro komunikaci s čidlem je použita dvouvodičová sběrnice, která je podobná komunikaci po sběrnici I^2C. Pro aplikaci senzoru potřebujeme zapojit pouze blokovací kondenzátor mezi napájení a zem. Jestliže použitý mikropočítač obsahuje tzv. pull–up rezistory, připojují se definované vývody senzoru přímo na I/O piny. Senzor disponuje režimem snížené spotřeby, do kterého se uvádí automaticky po dokončení měření. Spotřeba se pak sníží přibližně tisíckrát (z 1 mA na 1 μA) [25].

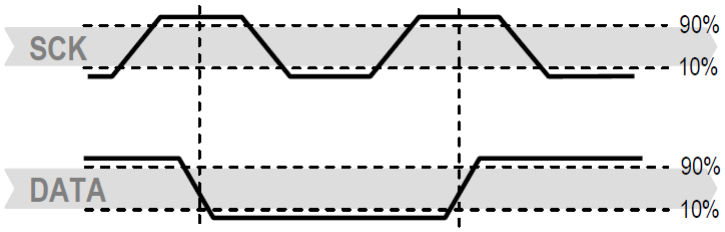
Pro komunikaci se senzorem, čtení dat a jejich přepočet na reálné hodnoty byla pod FreeRTOSem v souboru *mytask.c* vytvořena samostatná úloha `vTaskShtMeasure()`.

6.1.1 Komunikace se senzorem

Komunikace se senzorem probíhá pomocí datového a hodinového signálu. Použité rozhraní je podobné sběrnici I^2C, má ale pár odlišností, proto musela být komunikace s čidlem nově naprogramována. Výrobce čidla nabízí na svých stránkách již hotové vzorové rutiny ke stažení, viz [28]. Tyto rutiny byly v aplikaci s úpravami použity.

Komunikaci zahájíme vytvořením tzv. sekvence začátku přenosu. Graf sekvence je uveden na obr. 6.1.

Po startovní sekvenci již můžeme zapisovat jeden z příkazů. Příkaz začíná třemi adresními bity (je podporováno jen '000') a poté pěti příkazovými bity. Senzor potvrdí přijmutí



Obr. 6.1: Graf sekvence začátku přenosu |Převzato z [27]|

příkazu shozením datového vodiče (dále už jen DATA) do log. 0 (tzv. ACK bit). Poté začne samotné měření, které trvá v průměru 20/80/320 ms pro 8/12/14-bitové rozlišení měření.

Jakmile je měření hotovo, shodí senzor znovu DATA do log. 0. Mikrokontrolér musí na tento signál počkat, jinak nemůže data číst. Naměřená data jsou v senzoru uchována, dokud nejsou přečtena.

Čtou se dva byty (ve směru od MSB k LSB) obsahující naměřená data a poté jeden byte obsahující 8-bitový CRC kód. Každý byte je potvrzen mikrokontrolérem pomocí shození DATA do log. 0 (ACK).

Jestliže dojde při komunikaci s čidlem k chybě, je přerušena a nelze ji znovu navázat, je nutné připojení resetovat pomocí k tomu určené sekvence. Resetovací sekvence se provádí následovně: Nastavíme DATA do log. 1 a provedeme minimálně devět pulsů na SCK. Poté musí následovat provedení sekvence začátku přenosu. Tento reset ovlivní jen komunikaci s čidlem, která začne znovu od začátku [27].

SHT1x nabízí kromě příkazů pro měření teploty a vlhkosti ještě další příkazy. Všechny jsou shrnuty v tab. 6.1.

Příkaz	**Binární kód**
Vyhrazený	0000x
Měření teploty	00011
Měření vlhkosti	00101
Čtení ze status registru	00111
Zápis do status registru	00110
Vyhrazený	0101x – 1110x
Měkký reset	11110

Tab. 6.1: Dostupné příkazy senzoru SHT1x

6.1.2 Převod dat na reálné hodnoty

Pro kompenzaci nelineárního výstupu ze senzoru a tedy pro zpřesnění naměřených hodnot je vhodné použít následující rovnice. Pro zlinearizování výstupu z vlhkostního čidla použijeme rov. 6.1.

$$RH_{linear} = c1 + c2 \cdot SO_{RH} + c3 \cdot SO_{RH}^2 \qquad [\%RH] \tag{6.1}$$

kde $c1$ a $c2$ – koeficienty uvedené v tab. 6.2, SO_{RH} – výstup z vlhkostního čidla.

$\mathbf{SO}_{RH}$	**c1**	**c2**	**c3**
12bit	$-2,0468$	$0,0367$	$-1,5955E^{-6}$
8bit	$-2,0468$	$0,5872$	$-4,0845E^{-4}$

Tab. 6.2: Koeficienty pro linearizaci vlhkosti

Pro teploty, které se významněji liší od 25 °C, potřebuje naměřená vlhkosti ještě teplotně kompenzovat. K tomu slouží rov. 6.2.

$$RH_{real} = (T - 25)(t1 + t2 \cdot SO_{RH}) + RH_{linear} \qquad [\%RH] \tag{6.2}$$

kde T – naměřená teplota ve [°C], $t1$ a $t2$ – konstanty uvedené v tab. 6.3, SO_{RH} – výstup z vlhkostního čidla, RH_{linear} – lineárně kompenzovaný výstup z vlhkostního čidla v [%].

$\mathbf{SO}_{RH}$	**t1**	**t2**
12bit	0,01	0,00008
8bit	0,01	0,00128

Tab. 6.3: Koeficienty pro teplotní kompenzaci

6.1.3 Výpočet rosného bodu

Teplota rosného bodu není měřena senzorem přímo, ale je vypočítána z naměřené teploty a vlhkosti pomocí vzorce 6.3.

$$T_d(RH, T) = T_n \frac{ln(\frac{RH}{100}) + \frac{mT}{T_n+T}}{m - ln(\frac{RH}{100}) - \frac{mT}{T_n+T}} \qquad [°C] \tag{6.3}$$

kde RH – naměřená relativní vlhkost v [%], T – naměřená teplota ve [°C], T_n a m – konstanty uvedené v tab. 6.4.

Teplotní rozsah	T_n	**m**
0 – 50 °C	243,12	17,62
-40 – 0 °C	272,62	22,46

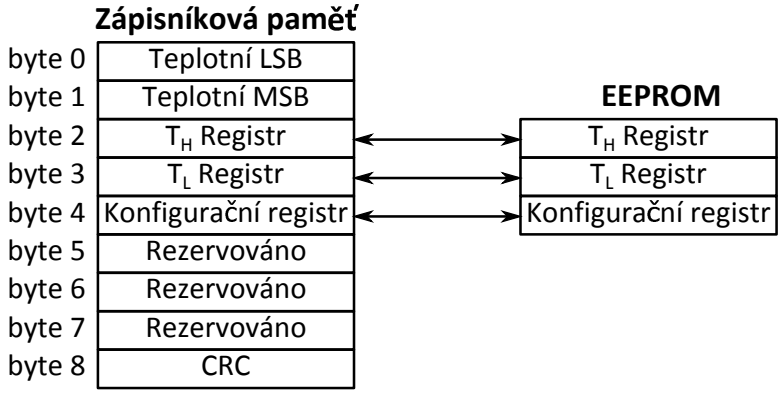
Tab. 6.4: Konstanty pro vypočítání rosného bodu.

6.2 DS18B20

DS18B20 je digitální čidlo od firmy Dallas s programovatelným rozlišením až 12 bitů. Komunikace probíhá po jednovodičové 1-wire sběrnici. Garantovaná přesnost senzoru v rozmezí od –55 °C do +125 °C je ±0,5 °C. Každé zařízení navíc obsahuje unikátní 64-bitový sériový kód, který umožňuje využívat 1-wire sběrnici více sensory.

Sensor obsahuje tzv. zápisníkovou RAM paměť o velikosti 9-bytů a paměť EEPROM, do které mohou být ze zápisníkové paměti data kopírována. Organizace paměti senzoru je znázorněna na obr. 6.2.

Pro správnou funkci 1-wire sběrnice je nutné připojit na datovou linku pull-up rezistor s hodnotou přibližně 5 $k\Omega$. V mém případě byl použit integrovaný pull-up rezistor dostupný pro pin mikrokontroléru [29].



Obr. 6.2: Paměťová mapa senzoru DS18B20 |Převzato z [29]|

6.2.1 Komunikace se senzorem

Pro komunikaci s DS18B20 je nutné dodržet následující kroky:

- **Inicializace**

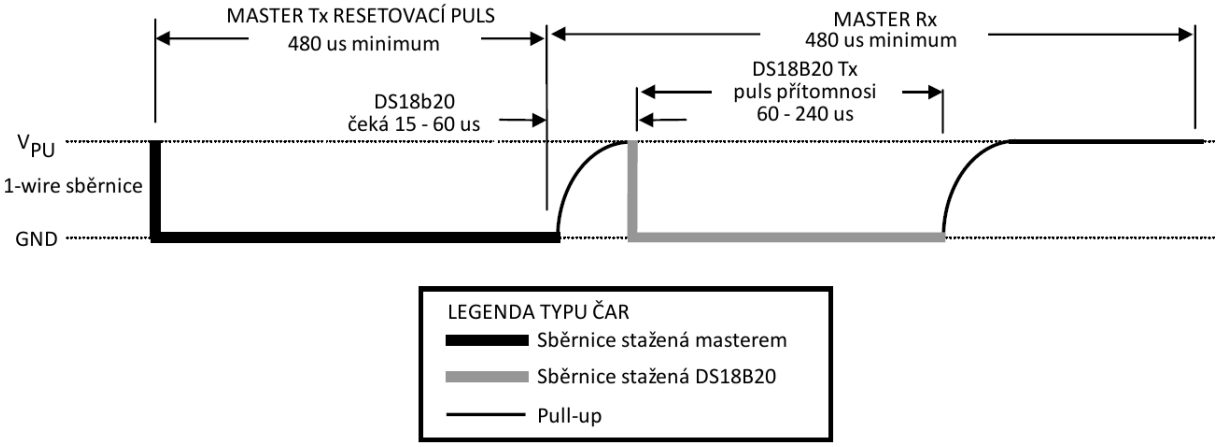
  Všechny transakce na 1-wire sběrnici začínají inicializační sekvencí. Inicializační sekvence se skládá z resetovacího pulsu od mastera následovaným pulsem přítomnosti od slave jednotky. Puls přítomnosti dává masterovi vědět, že jsou na sběrnici připravené jednotky. Časový diagram inicializace je znázorněn na obr. 6.3.

- **Příkaz ROM**

  Jakmile master detekoval puls přítomnosti, může vykonat příkaz ROM. Tyto příkazy používají 64-bitový unikátní ROM kód a umožňují masterovi vybrat konkrétní slave jednotku.

- **Příkaz funkce**

Tyto příkazy umožňují masterovi zápis/čtení do/ze zápisníkové paměti senzoru, zahajovat konverzi teploty a specifikovat režim napájení. Pro zahájení konverze teploty byl použit příkaz `CONVERT T`. Po dokončení konverze je vyvoláno čtení ze zápisníkové paměti pomocí příkazu `READ SCRATCHPAD` [29].



Obr. 6.3: Časový diagram inicializační sekvence senzoru DS18B20 (V_{PU} – pull-up napětí; GND – potenciál země) |Převzato z [29]|

6.2.2 Čtecí a zapisovací časové sloty

Master zapisuje data v průběhu zapisovacích slotů a čte data v průběhu čtecích slotů.

a) **Zapisovací časový slot**

Zapisovací časový slot je zahájen masterem stažením datové linky do nuly a trvá minimálně 60 μs. Pro zápis log. 1 uvolní master datovou linku po 15 μs. Pro zápis log. 0 ji musí držet staženou po dobu trvání zapisovacího slotu.

b) **Čtecí časový slot**

Čtecí časový slot trvá minimálně 60 μs a je zahájen masterem stažením datové linky do nuly po dobu minimálně 1 μs. Poté sensor přenese log. 0 stažením datové linky do nuly, nebo log. 1 uvolněním datové linky. Výstupní data ze senzoru jsou platná 15 μs po zahájení čtecího časového slotu [29].

Použitý mikrokontrolér neobsahuje řadič pro 1-wire komunikaci. Proto musela být komunikace naprogramována. Nejdříve byly vytvořeny funkce pro čtení a zápis bitu. Z nich pak funkce pro čtení a zápis bytu. V hlavní funkci byla poté provedena inicializace senzoru. Jelikož bylo použito pouze jedno čidlo DS18B20, byl přeskočen ROM příkaz pomocí příkazu `SKIP ROM`. Poté byl proveden příkaz `CONVERT T` pro přepočet teploty a po dokončení konverze příkaz `READ SCRATCHPAD` pro čtení zápisníkové paměti. Z paměti bylo vyčteno všech devět bytů, z nichž byly použity teplotní byty a CRC kód.

Pro komunikaci se senzorem byla podobně jako pro sensor SHT1X vytvořena úloha `vTaskReadTemp()`.

6.3 LM92

Senzor LM92 je digitální teplotní senzor a teplotní okénkový komparátor od firmy Texas Instruments. Komunikace s ním probíhá po I^2C sériové sběrnici (vývody SCA a SCL). Senzor dále obsahuje piny INT a T_CRIT_A. Vývod INT s otevřeným kolektorem začne být aktivní, jakmile teplota překročí hranice programovatelného teplotního okénka. Oddělený alarm kritické teploty T_CRIT_A je aktivní v případě, že teplota překročí programovatelný kritický limit.

Rozlišení senzoru je 12-bitů plus znaménko a přesnost v okolí 30 °C je ±0.33 °C [30].

6.3.1 Komunikace se senzorem

Komunikace se senzorem probíhá pomocí I^2C sběrnice (I^2C1) dostupné v použitém mikrokontroléru. Pro práci se sběrnicí byla použita knihovna *AN2824* od firmy ST Microelectronic. Tato knihovna obsahuje soubor *I2CRoutines.c*, ve kterém jsou napsány rutiny pro ovládání I^2C sběrnice, a soubor *stm32f10x_it.c*, ve kterém jsou vytvořeny obsluhy přerušení. Pro inicializaci I^2C sběrnice byla použita funkce `I2C_LowLevel_Init()`.

Pro nastavení adresy senzoru slouží dva vývody (A0, A1). Jejich připojením k napájení na nich nastavíme log. 1 a připojením k zemi log. 0. Jejich hodnota tvoří nejméně významné bity v 7-bitové adrese senzoru. V mém případě byly vývody připojeny k zemi (hodnota '00') a tím byla adresa slave jednotky nastavena na hexadecimální hodnotu 0x90.

Senzor obsahuje registr teploty, konfigurační registr, registr pro nastavení teplotních limitů a registr s výrobním ID. Zápisem do tzv. registru ukazatelů můžeme zvolit, který z registrů chceme číst nebo zapisovat. Defaultně je nastaven registr teploty, ze kterého lze jen číst. Má velikost 16-bitů a obsahuje znaménko (MSb), 12 bitů teploty a tři stavové bity [30].

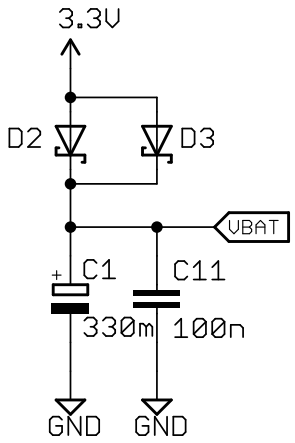
Pro čtení hodnoty registru teploty byla použita funkce `I2C_Master_BufferRead()`. Z registru byly vymaskovány teplotní bity a převedeny na reálnou hodnotu teploty vynásobením konstantou 0,0625. Ve FreeRTOS byla pro práci se senzorem vytvořena úloha `vTaskReadTemp_LM92()`.

6.4 RTC obvod

Pro měření reálného času byl použit obvod RTC, který je součástí mikrokontroléru. K taktování RTC byl použit externí hodinkový krystal s frekvencí 32,768 kHz.

U RTC obvodu je povoleno přerušení, které se vyvolá přesně každou sekundu. V obsluze tohoto přerušení se zvětší hodnota čítače o jedna. Uživatel si pak může hodnotu konstanty jednoduše převést na reálnou hodnotu času pomocí standardní knihovny *time.h* určené pro práci s časem. Hodnota čítače po resetu je 0, což podle knihovny *time.h* odpovídá datu 1.1.1970. Tuto hodnotu můžeme pomocí FreeModbus uchovávacího registru nastavovat i číst.

Hodnota času se automaticky ukládá do záložního registru, takže je dostupná i po resetu. Po odpojení napájení by se však uchované hodnoty z registru ztratily. To je vyřešeno připojením superkapacitoru o velikosti 330 mF, na pin mikrokontroléru VBAT. Tento pin slouží k napájení RTC v případě, že není k dispozici hlavní napájení. Schéma zapojení je znázorněno na obr. 6.4.



Obr. 6.4: Schéma zapojení superkapacitoru

Superkapacitor se nabíjí přes schotkyho diodu na hodnotu přibližně 3,1 V (úbytek napětí na diodě je přibližně 0,2 V). V případě, že je napájení odpojeno, slouží dioda k zabránění toku proudu zpět do zdroje.

6.5 A/D převodník

Pro převod převod analogového napětí na digitální byl použit integrovaný 12-bitový 16-kanálový A/D převodník. Pro práci s A/D převodníkem byla použita knihovna *stm32f10x_adc.h* z balíku `STM32F10x_StdPeriph_Driver` od firmy ST Microelectonic. A/D převodník byl nastaven do nezávislého módu, multikanálový převod byl zakázán a byla zvolena jednoduchá konverze. Spouštění převodu pomocí externí události nebylo použito a zarovnání dat bylo nastaveno směrem doprava. Nakonec bylo povoleno generování přerušení po skončení konverze a nastaven NVIC řadič.

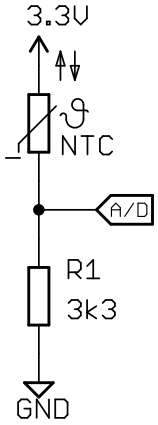
Pro měření teploty čipu byl povolen a zkalibrován integrovaný analogový senzor teploty. Pro ten je v převodníku vyhrazen kanál č. 16.

Pro práci s A/D převodníkem byla v programu vytvořena samostatná úloha. Pro obsluhu přerušení byl použit binární semafor. Na začátku úlohy je spuštěna konverze A/D

převodníku. Poté přejde úloha do zablokovaného stavu, protože potřebuje k pokračování „převzít“ semafor. Jakmile je konverze hotova, je v obsluze přerušení zavolána funkce `xSemaphoreGiveFromISR()`, úloha je ihned odblokována a může pokračovat zpracováním převedené hodnoty napětí.

6.5.1 NTC termistor

NTC termistor je použit jako analogové čidlo pro měření teploty. Způsob zapojení termistoru je znázorněn na obr. 6.5. V zapojení je použit rezistor s hodnotou 3300 Ω, která přibližně odpovídá hodnotě odporu termistoru při pokojové teplotě 25 °C. Pomocí A/D převodníku měříme hodnotu napětí na termistoru, kterou přepočítáme na odpor podle vzorce 6.4.



Obr. 6.5: Schéma zapojení NTC termistoru

$$R_{NTC} = \frac{R_1 \cdot V_{IN}}{V_{OUT}} - R_1[\Omega] \tag{6.4}$$

kde R_{NTC} – odpor termistoru v $[\Omega]$, R_1 – odpor rezistoru v $[\Omega]$, V_{IN} – vstupní napětí ve $[V]$, V_{OUT} – napětí měřené A/D převodníkem ve $[V]$.

Dále je potřeba převést vypočtený odpor na teplotu. Závislost odporu na termodynamické teplotě je dána vztahem 6.5.

$$R = R_\infty \cdot e^{B\frac{1}{T}}[\Omega] \tag{6.5}$$

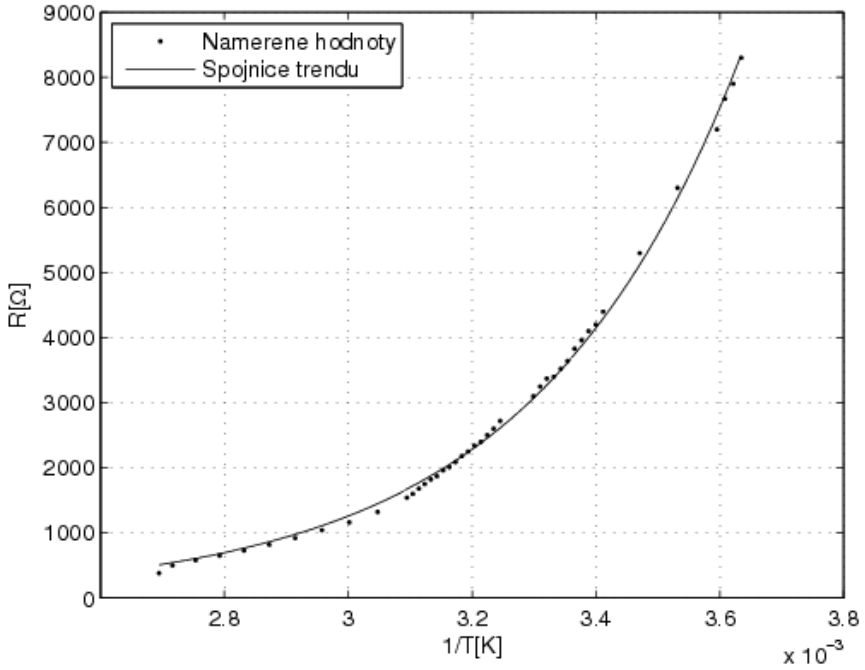
kde R_∞ – odpor termistoru při hypoteticky nekonečně velké teplotě v $[\Omega]$, B – bezrozměrná teplotní konstanta termistoru, T – termodynamická teplota v $[K]$.

Ze vzorce vidíme, že je k výpočtu teploty potřeba znát konstanty R_∞ a B. Konstanta B bývá uvedena v datovém listu termistoru. Jelikož není znám přesný typ použitého termistoru, byly konstanty určeny následujícím způsobem [32]:

1. Termistor byl připevněn k termočlánkovému teploměru a společně vložen do hrnce s ledovou tříští

2. Ledová tříšť byla na vařiči pomalu zahřívána a do připravené tabulky byla zaznamenávána její teplota a odpovídající odpor NTC termistoru. Tímto způsobem byl proměřen odpor termistoru od 0 °C do 100 °C s krokem 5 °C. Při teplotách blízkých pokojové teplotě byl krok zpřesněn na 1 °C

3. Z naměřených hodnot byl vytvořen graf závislosti odporu termistoru na $\frac{1}{T}$ a proložen exponenciální spojnicí trendu. Graf je znázorněn na obr. 6.6. Výsledná spojnice trendu je dána rovnicí 6.6. Z ní byly určeny konstanty $B = 2973$ a $T_\infty = 0{,}1691$



Obr. 6.6: Graf závislosti odporu termistoru na 1/T

$$R = 0,1691^{2973\frac{1}{T}}[\Omega] \tag{6.6}$$

6.5.2 Displej DOGS102

Displej DOGS102 je LCD display s rozměry 102 x 64 pixelů. Mezi jeho hlavní výhody patří nízký proudový odběr (250 μA), poměrně nízké napájecí napětí (od 2,5 V do 3,3 V) a vysoký kontrast. Pro ovládání displeje slouží integrovaný řadič UC1701, který podporuje komunikaci po SPI sběrnici. K displeji se dá připojit externí podsvícení, které se umisťuje přímo pod displej. Podsvícení potřebuje své vlastní napájecí napětí, které se může připojit k podsvětlovacím LED diodám paralelně, nebo sériově. Před připojením napájení je nutné pro omezení proudu připojit sériově externí rezistor. Hodnoty rezistorů se liší podle typu použitého podsvícení a jejich doporučené hodnoty lze najít v datovém listu výrobce, viz. [33].

Pro práci s displejem byla použita univerzální grafická knihovna `u8glib`, která je volně dostupná ke stažení na internetu, viz. [34]. Knihovna podporuje několik typů řadičů. Zdrojové soubory pro všechny podporované řadiče a typy displejů jsou uloženy ve složce **src**. Rutiny pro komunikaci s displejem DOGS102 jsou obsaženy v souboru *u8g_dev_uc1701_dogs102.c*.

Pro komunikaci knihovny s displejem musely být vytvořeny následující nízkoúrovňové rutiny:

- poslání bytu a sekvence bytů po SPI rozhraní
- vytvoření nastavitelného zpoždění s rozlišením 1 μs
- ovládání resetu displeje (pin $\overline{RST}$)
- ovládání volby slave jednotky (pin $\overline{CS0}$)
- přepínání mezi zápisem příkazu a dat (pin CD)

K tomuto účelu byl vytvořen soubor *u8g_arm.c* a v něm funkce realizující hardwarové rutiny, která je rámcově popsaná v následujícím úryvku kódu:

```
uint8_t
u8g_com_hw_spi_fn(u8g_t *u8g, uint8_t msg, uint8_t arg_val, void *arg_ptr)
{
  switch(msg)
  {
    case U8G_COM_MSG_STOP:
      // Přerušení komunikace s displejem
      break;
    case U8G_COM_MSG_INIT:
      // Inicializace hardwaru, čítačů, I/O pinů, atd.
      break;
    case U8G_COM_MSG_ADDRESS:
      // Přepínání mezi posíláním příkazů a dat hodnotou parametru arg_val
     break;
    case U8G_COM_MSG_RESET:
      // Ovládání resetu parametrem arg_val
      break;
    case U8G_COM_MSG_CHIP_SELECT:
      // Ovládání volby slave jednotky parametrem arg_val
      break;
    case U8G_COM_MSG_WRITE_BYTE:
      // Poslání bytu displeji
```

```
      break;
    case U8G_COM_MSG_WRITE_SEQ:
    case U8G_COM_MSG_WRITE_SEQ_P:
      // Poslání sekvence bytů displeji
      break;
  }
  return 1;
}
```

Aby knihovna `u8glib` věděla, že má ke komunikaci s displejem používat právě tuto funkci, je funkce zavolána následujícím způsobem:

```
u8g_t u8g;
u8g_InitComFn(&u8g, &u8g_dev_uc1701_dogs102_hw_spi, u8g_com_hw_spi_fn);
```

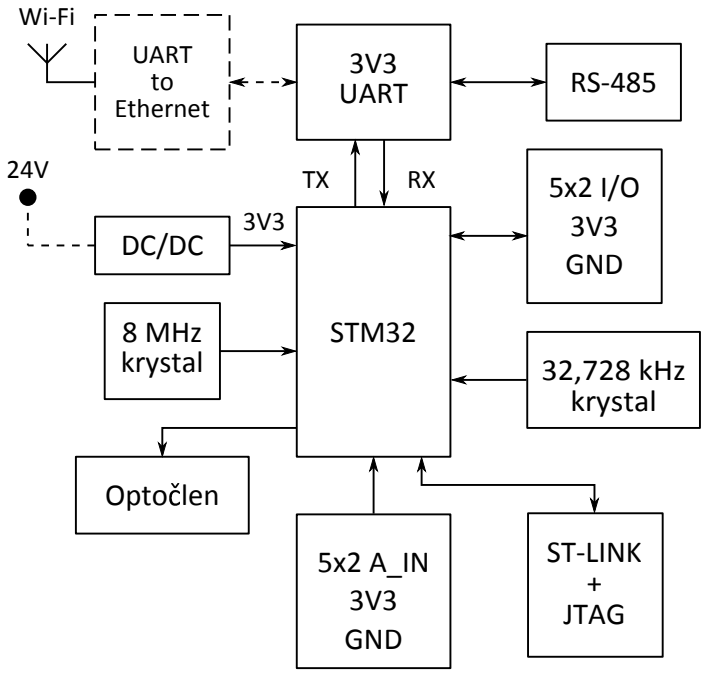
Funkce obsahuje jako třetí parametr vytvořenou hardwarovou rutinu. Nyní je možné používat funkce `u8glib` knihovny. Popis jednotlivých funkcí knihovny přesahuje rozsah této práce.

7

Návrh I/O modulu

Pro ověření funkčnosti implementace knihovny FreeModbus pod RTOS FreeRTOS byl realizován I/O modul. Pro navržený a realizovaný hardware I/O modulu byla vytvořena ukázková aplikace, která umožňuje pomocí Modbus RTU / ASCII sledování / ovládání digitálních vstupů / výstupů, měření analogových vstupů a připojení digitálních čidel. K tomu bylo na výstupní pinové lišty vyvedeno 10 digitálních a 10 analogových I/O pinů mikrokontroléru a vodiče pro sběrnice SPI, I^2C a USART (RX, TX). Dále pak JTAG a ST Link pro programování a debug mikrokontroléru.

Blokové schéma I/O modulu je znázorněno na obr. 7.1



Obr. 7.1: Blokové schéma I/O modulu

Pro spínání výkonových součástek (např. relé) byl použit optočlen, který zajišťuje galvanické oddělení mezi MCU a spínaným obvodem. Spínání optočlenu je ovládáno pinem mikrokontroléru.

Návrh byl realizován tak, aby se výsledná DPS vešla do elektroinstalační krabičky KU68L. Modul tak může být spolu s krabičkou vložen do elektroinstalace. Obvod byl realizován na dvouvrstvé DPS.

Pro převod mezi komunikací po sériové lince na komunikaci po sběrnici RS–485 byl použit obvod SN65HVD10 od firmy Texas Instruments. Tento obvod v sobě implementuje jak vysílač, tak přijímač. Směr přenosu se určuje pomocí vývodů $\overline{RE}$ a DE. Příjem se nastaví přivedením log. 0 na vývod $\overline{RE}$ (Receive Enable) a vysílání přivedením log. 1 na vývod DE (Driver Enable). Pomocí dvou propojek je na DPS možné k RS–485 sběrnici připojit safe bias zakončovací odpory.

Pro taktování mikrokontroléru byl použit krystal s frekvencí 8 MHz a pro RTC obvod hodinkový krystal s frekvencí 32,728 kHz.

7.0.3 Mikrokontrolér

Pro implementaci FreeModbus slave jednotky a řízení I/O modulu byl vybrán mikrokontrolér STM32F100RB, tedy stejný typ jako je k dispozici na discovery vývojovém kitu. Jedná se o jeden z nejlevnějších 32-bitových mikrokontrolérů s architekturou Cortex-M3 [18].

Základní parametry MCU:

1. maximální taktovací frekvence 24 MHz
2. 128 kB Flash
3. 8 kB SRAM
4. 32-bitový RTC
5. 16-kanálový A/D převodník
6. 51 I/O portů
7. SPI, I^2C, USART
8. 6*16-bitový časovač

7.0.4 Napájení

Pro napájení modulu je v elektroinstalaci k dispozici napětí 24 V. K napájení mikrokontroléru a použitých senzorů muselo být toto napětí vhodným způsobem sníženo na hodnotu okolo 3,3 V. K tomu byl použit spínaný DC/DC měnič IL2403 s galvanickým oddělením.

8

Závěr

Cílem práce bylo implementovat volně dostupnou knihovnu FreeModbus na platformě STM32F10x běžící pod operačním systémem reálného času FreeRTOS.

Byla vytvořena SW aplikace, která umožňuje přes rozhraní Modbus RTU a Modbus ASCII sledování / ovládání digitálních vstupů / výstupů. Aplikace byla vytvořena a odladěna na vývojovém kitu STM32 value line Discovery. Pomocí kombinovaného digitálního senzoru SHT10 je měřena teplota a relativní vzdušná vlhkost. Z těchto dvou hodnot je navíc vypočten rosný bod. Teplota je dále měřena pomocí digitálních čidel DS18B20 a LM92. Komunikace s čidly SHT10 a DS18B20 musela být naprogramována. Pro komunikaci s čidlem LM92 byla použita sběrnice I^2C, která je dostupná v použitém MCU. Analogově je teplota měřena pomocí NTC termistoru, u kterého byla proměřena závislost jeho odporu na teplotě. K měření napětí na NTC termistoru byl použit 12-bitový A/D převodník, který je součástí mikrokontroléru. Pomocí A/D převodníku je také měřena hodnota interního senzoru teploty, který je integrován v pouzdře mikrokontroléru a jeho výstup je přímo připojen k jednomu z kanálů A/D převodníku. Přesnost tohoto senzoru je však mizivá, jelikož je vlivem zátěže jádra ohříván.

Hodnoty senzorů jsou ukládány do Modbus registrů, ze kterých jsou poté čteny pomocí Modbus mastera. Modbus master je simulován pomocí programu Modbus Poll, který je spuštěn na osobním počítači. Komunikace mezi implementovanou Modbus slave jednotkou a osobním počítačem probíhá po sériové lince RS–232 s TTL úrovněmi napětí.

Dále je počítán reálný čas pomocí RTC obvodu, který je integrován v MCU. K taktování RTC čítače je použit externí 32,728 kHz krystal, který je dostupný na vývojovém kitu. Hodnota RTC čítače se dá pomocí Modbus registrů jak číst, tak zapisovat.

Měřené veličiny mohou být také zobrazovány na připojeném LCD displeji DOGS102 s úhlopříčkou 102 x 64 pixelů. Pro práci s displejem byla použita univerzální volně dostupná grafická knihovna u8glib. Komunikace s displejem probíhá pomocí sběrnice SPI v použitém mikrokontroléru.

Pro všechny popsané funkce byly v operačním systému reálného času FreeRTOS vytvořeny samostatné úlohy, jejichž obsluha je řízena plánovačem. Úlohy mezi sebou komunikují použitím binárních semaforů a mutexů.

Nakonec byl navržen, zkonstruován a oživen I/O měřící modul, pro jehož řízení byl použit mikrokontrolér STM32F100RB, tedy stejný jako je na vývojovém kitu. Modul obsahuje nezbytné externí součástky pro správnou funkci použitých integrovaných obvodů, 8 MHz a 32,728 kHz krystal, budič RS–485 sběrnice, optočlen pro spínání výkonových součástek, DC/DC 24V/3,3V měnič, superkapacitor pro záložní napájení RTC a má vyvedeno několik I/O pinů pro připojení senzorů a jiných externích součástek. Zkonstruovaný I/O modul používá stejnou SW aplikaci, která byla odladěna na vývojovém kitu.

Na vytvořeném hardwaru se zatím bohužel nepodařilo rozběhnout 32,728 kHz krystal. Byly použity dva typy krystalů, různé hodnoty předřadného odporu a kondenzátorů, ale ani v jednom případě se krystal nerozkmital. Důvodem nefunkčnosti by mohla být návrhová chyba. Podle návrhových pravidel by měl být krystal umístěn co nejblíže příslušným pinům MCU. Délka spoje od pinu k vývodu krystalu by neměla být delší než 1 cm, což je v mém případě nepatrně překročeno a krystal mohl být umístěn vhodněji. Pro taktování RTC obvodu je tedy použit interní LSI oscilátor.

Dále se s modulem nepodařilo úspěšně komunikovat po RS–485 sběrnici. Slave jednotka vysílá nesprávné byty a dotaz mastera skončí vždy CRC chybou. V tomto případě bude chyba nejspíše v nesprávném přepínání budiče mezi režimem příjmu a vysílání. Komunikace tedy zatím probíhá pouze po RS–232 sběrnici s použitím TTL úrovní napětí.

Literatura

[1] Modbus. In: *en.wikipedia.org* [online]. Poslední změna 9.4.2014 18:47 [cit. 11.4.2014]. Dostupné z:

`http://en.wikipedia.org/wiki/Modbus`

[2] RONEŠOVÁ, A. *Přehled protokolu MODBUS* [online]. květen 2005 [cit. 20.3.2014]. Dostupné z:

`http://home.zcu.cz/ ronesova/bastl/files/modbus.pdf`

[3] *MODBUS application protocol specification V1.1b* [online]. prosinec 2012 [cit. 5.4.2014]. Dostupné z:

`http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf`

[4] *MODBUS over serial line specification and implementation guide V1.02* [online]. 20.12.2012 [cit. 2.4.2014]. Dostupné z:

`http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf`

[5] WALTER, C. *FreeMODBUS.* A Modbus ASCII/RTU and TCP implementation [online]. 6.6.2010 [cit. 10.3.2014]. Dostupné z:

`http://www.freemodbus.org`

[6] *FreeModbus API V1.5* [online]. 6.6.2010 [cit 2.4.2014]. Dostupné z:

`http://www.freemodbus.org/api/index.html`

[7] *Modicon MODBUS Protocol Reference Guide* [online]. červen 1996 [cit. 3.4.2014]. Dostupné z:

`http://www.modbus.org/docs/PI_MBUS_300.pdf`

[8] LAMMERT, B. *Modbus interface tutorial* [online]. © 1997–2010 [cit. 4.4.2014]. Dostupné z:

`http://www.lammertbies.nl/comm/info/modbus.html`

[9] *Modbus* [online]. [Cit 22.3.2014]. Dostupné z:

`http://www.seriousintegrated.com/w/index.php?title=Modbus`

[10] *FreeRTOS.* Market leading RTOS for embedded systems with Internet of Things extensions [online]. leden 2014 [cit. 15.4.2014]. Dostupné z:

`http://www.freertos.org`

[11] FreeRTOS Tutorial. In: *socialledge.com* [online]. Poslední změna 29.3.2014 09:29 [cit. 5.4.2014]. Dostupné z:

`http://www.socialledge.com/sjsu/index.php?title=FreeRTOS_Tutorial`

[12] FreeRTOS on STM32. In: *scienceprog.com* [online]. 5.12.2011 [cit. 6.4.2014]. Dostupné z:

`http://www.scienceprog.com/freertos-on-stm32`

[13] Mutexes and Semaphores Demystified. In: *barrgroup.com* [online]. Poslední změna 19.8.2008 22:23 [cit. 15.4.2014]. Dostupné z: `http://www.barrgroup.com/Embedded-Systems/How-To/RTOS-Mutex-Semaphore`

[14] SVEC, C. *FreeRTOS* [online]. [cit. 15.4.2014]. Dostupné z:

`http://aosabook.org/en/freertos.html`

[15] Priority inheritance. In: *en.wikipedia.org* [online]. Poslední změna 20.4.2014 14:32 [cit. 22.4.2014]. Dostupné z:

`http://en.wikipedia.org/wiki/Priority_inheritance`

[16] Mutual exclusion. In: *en.wikipedia.org* [online]. Poslední změna 13.3.2014 12:47 [cit. 20.4.2014]. Dostupné z:

`http://en.wikipedia.org/wiki/Mutex`

[17] *STMicroelectronic* [online]. © 2014 [cit. 30.3.2014]. Dostupné z:

`http://www.st.com`

[18] Katalogový list výrobce ST Microelectronic. in: *st.com* [online]. 2012 [cit. 30.11.2013]. Dostupné z:

`http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00251732.pdf`

[19] Referenční manuál výrobce ST Microelectronic. in: *st.com* [online]. 2011 [cit. 30.11.2013]. Dostupné z:

`http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/CD00246267.pdf`

[20] *KEIL* [online]. © 2013 [cit. 20.10.2013]. Dostupné z:

`http://www.keil.com`

[21] Úvod do architektury Cortex-M3. In: *pendatron.cz* [online]. Poslední změna 10.2.2010 9:35 [cit. 19.1.2014]. Dostupné z:

`http://pandatron.cz/?1252&uvod_do_architektury_cortex-m3_-_dil._1`

[22] Cortex-M3 Processor. In: *arm.com* [online]. [cit. 15.1.2014]. Dostupné z:

`http://www.arm.com/products/processors/cortex-m/cortex-m3.php`

[23] RS–485. In: *en.wikipedia.org* [online]. Poslední změna 12.2.2014 20:19 [cit. 20.3.2014]. Dostupné z:

`http://en.wikipedia.org/wiki/RS485`

[24] USART. In: *en.wikipedia.org* [online]. Poslední změna 2.1.2014 20:35 [cit. 5.1.2014]. Dostupné z:

`http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter`

[25] Holain, M. Dálkově řízená meteostanice. *Praktická elektronika* 2010, **2**, 10-15. ISSN 1211-328X

[26] Václavík, R. Vlhkoměr a teploměr bez kalibrace. *Praktická elektronika* 2003, **2**, 8-12. ISSN 1211-328X

[27] Katalogový list výrobce Sensirion. in: *sensirion.com* [online]. květen 2010 [cit. 3.4.2014]. Dostupné z:

`http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT1x_Datasheet_V5.pdf`

[28] *Sensirion AG* [online]. © 2014 [cit. 10.2.2014]. Dostupné z:

`http://www.sensirion.com`

[29] Katalogový list výrobce Maxim Integrated. in: *maximintegrated.com* [online]. 22.4.2008 [cit. 23.10.2013]. Dostupné z:

`http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf`

[30] Katalogový list výrobce Texas Instruments. in: *ti.com* [online]. © 1995–2014 [cit. 10.3.2014]. Dostupné z:

`http://www.ti.com/lit/ds/symlink/lm92.pdf`

[31] Měření teplotní závislosti odporu termistoru NTC, návrh teploměru. In: *kof.zcu.cz* [online]. [cit. 30.4.2014]. Dostupné z:

`www.kof.zcu.cz/st/sm/prevodniky.doc`

[32] Špringl, V. Měření teploty - polovodičové odporové senzory teploty. in: *hw.cz* [online]. 19.8.2004 [cit. 15.4.2014]. Dostupné z:

`http://www.hw.cz/teorie-a-praxe/dokumentace/mereni-teploty-polovodicove-odporove-senzory-teploty.html`

[33] Katalogový list výrobce Electronic Assembly. in: *lcd-module.de* [online]. 27.3.2014 [cit. 20.4.2014]. Dostupné z:

`http://www.lcd-module.de/eng/pdf/grafik/dogs102-6e.pdf`

[34] *U8glib* [online]. 16.3.2014 [cit. 20.3.2014]. Dostupné z:

`https://code.google.com/p/u8glib`

[35] HRABOVSKÝ, M., JURÁNEK, A. *EAGLE pro začátečníky.* Praha: Nakladatelství BEN, 2005. ISBN 80-7300-177-2.

[36] PLÍVA, Z. *EAGLE prakticky* Praha: Nakladatelství BEN, 2010. ISBN 978-80-7300-252-7

[37] ZÁHLAVA, V. *Návrh a konstrukce desek plošných spojů* Praha: Nakladatelství BEN, 2010. ISBN 978-80-7300-266-4

[38] Herout, P. *Učebnice jazyka C* České Budějovice: Nakladatelství Kopp, 2007. ISBN 80-7232-220-6

[39] FIRSTOVÁ, Zdeňka. *Pravidla pro bibliografické odkazy a citace informačních zdrojů.* Plzeň: Univerzitní knihovna ZČU v Plzni, 2011. [cit. 2. 4. 2014]. Dostupné z: `http://www.iso690.zcu.cz`

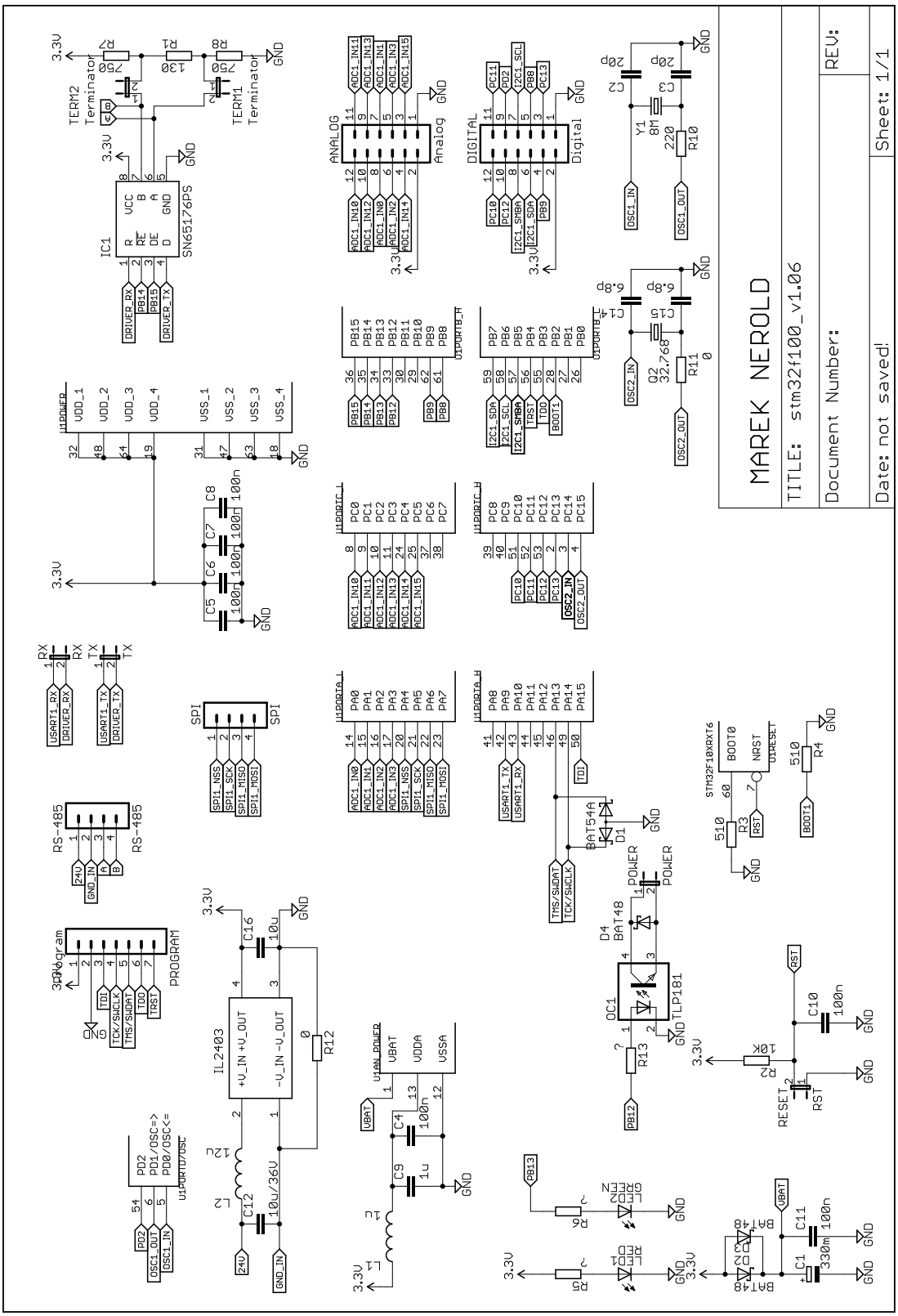
Příloha A

Výsledný návrh I/O modulu

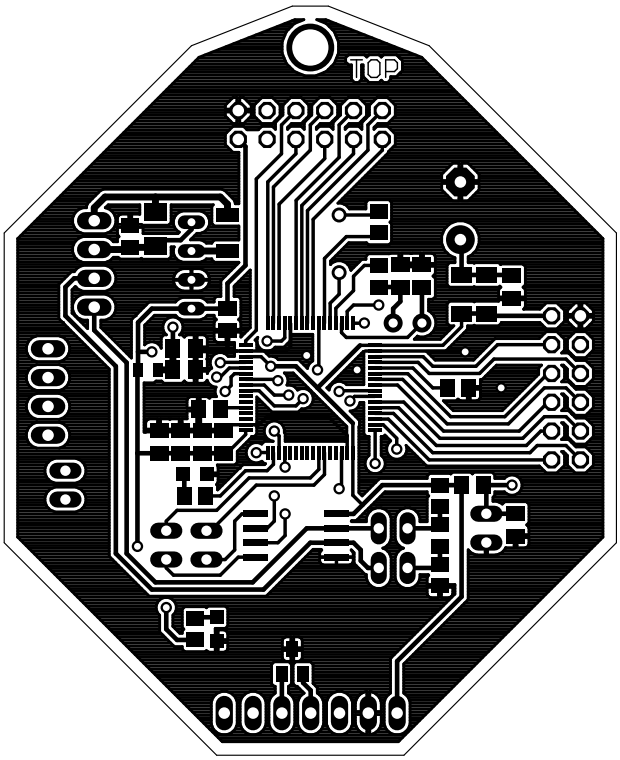
A.1 Schéma

A.2 DPS

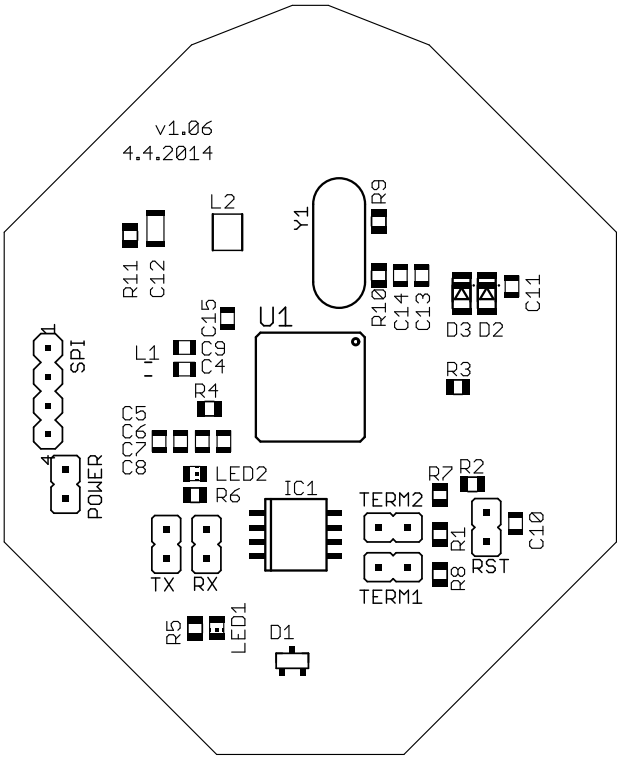
A.3 Seznam použitých součástek



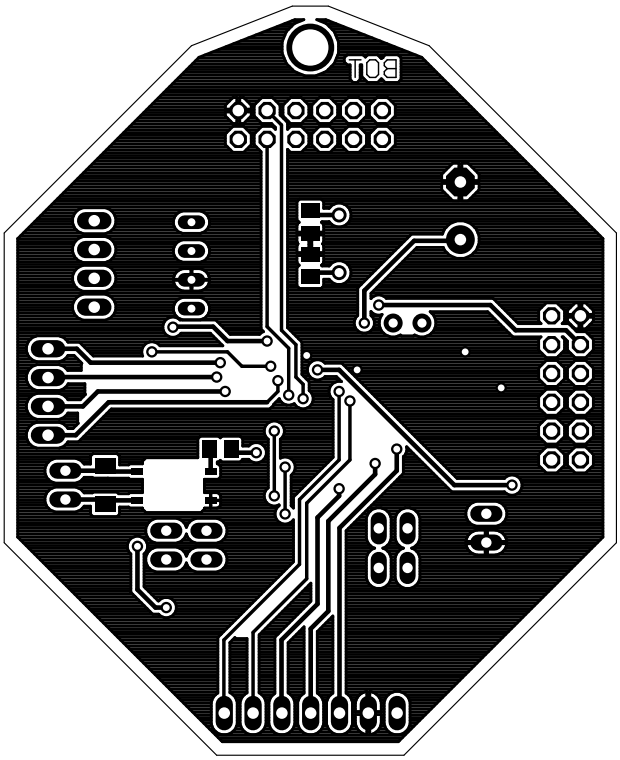
Obr. A.1: Schéma I/O modulu



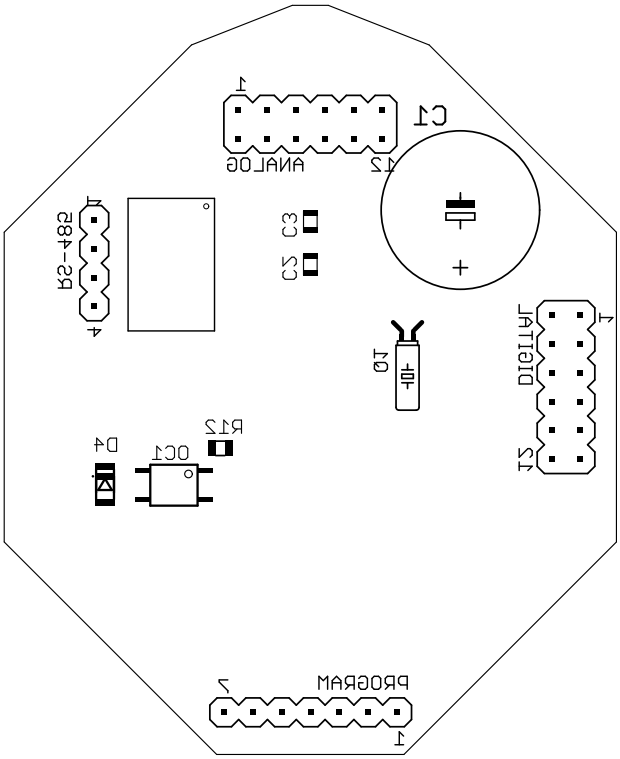
Obr. A.2: Deska plošných spojů I/O modulu (měřítko 1:1,5) – strana A



Obr. A.3: Osazovací plán I/O modulu (měřítko 1:1,5) – strana A



Obr. A.4: Deska plošných spojů I/O modulu (měřítko 1:1,5) – strana B



Obr. A.5: Osazovací plán I/O modulu (měřítko 1:1,5) – strana B

Tab. A.1

Označení	Hodnota	Pouzdro	Popis
Označení	**Hodnota**	**Pouzdro**	**Popis**
ANALOG	–	MA06-2	konektor
C1	330 mF	E5 – 13	superkapacitor
C2	20 pF	C0805	keramický kondenzátor
C3	20 pF	C0805	keramický kondenzátor
C4	100 nF	C0805	keramický kondenzátor
C5	100 nF	C0805	keramický kondenzátor
C6	100 nF	C0805	keramický kondenzátor
C7	100 nF	C0805	keramický kondenzátor
C8	100 nF	C0805	keramický kondenzátor
C9	1 μF	C0805	keramický kondenzátor
C10	100 nF	C0805	keramický kondenzátor
C11	100 nF	C0805	keramický kondenzátor
C12	1 μF	C1206	keramický kondenzátor
C13	6,8 pF	C0805	keramický kondenzátor
C14	6,8 pF	C0805	keramický kondenzátor
C15	10 μF	C0805	keramický kondenzátor
D1	BAT54A	SOT23	dioda
D2	BAT48	SOD80	dioda
D3	BAT48	SOD80	dioda
D4	BAT48	SOD80	dioda
DIGITAL	–	MA06-2	konektor
IC1	SN65HVD10	SOP–08	vysílač RS–485
IC2	IL2403S	SIP4	DC/DC měnič
L1	1 μH	L0805	tlumivka
L2	12 μH	L1210	tlumivka
LED1	RED	LED0805	LED dioda
LED2	GREEN	LED0805	LED dioda

Tabulka pokračuje na další straně . . .

Tab. A.1 – pokračování

Označení	**Hodnota**	**Pouzdro**	**Popis**
OC1	TLP185	SO-4	optočlen
POWER	–	JP1	jumper
PROGRAM	–	MA07-1	programovací konektor
Q1	32,768 kHz	TC26H	krystal
R1	130 Ω	R0805	rezistor
R2	10 kΩ	R0805	rezistor
R3	510 Ω	R0805	rezistor
R4	510 Ω	R0805	rezistor
R5	130	R0805	rezistor
R6	160	R0805	rezistor
R7	750 Ω	R0805	rezistor
R8	750 Ω	R0805	rezistor
R9	220 Ω	R0805	rezistor
R10	0 Ω	R0805	rezistor
R11	0 Ω	R0805	rezistor
R12	130	R0805	rezistor
RS–485	–	MA04-1	konektor
RST	–	JP1	jumper
RX	–	JP1	jumper
SPI	–	MA04-1	konektor
TERM1	–	JP1	jumper
TERM2	–	JP1	jumper
TX	–	JP1	jumper
U1	STM32F1	TQFP64	mikrokontrolér
Y1	8 MHz	HC49US	krystal

Tab. A.1: Seznam použitých součástek v I/O modulu