

Programová knihovna pro podporu RS485 komunikace na vývojovém kitu M68EVB908GB60

Program library for RS485 communication support on M68EVB908GB60 development kit

Bc. Ondřej Hladiš



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Ondřej HLADIŠ**
Osobní číslo: **A09471**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Programová knihovna pro podporu RS485
komunikace na vývojovém kitu M68EVB908GB60**

Zásady pro vypracování:

1. Zpracujte literární rešerši na téma průmyslová komunikace.
2. Prostudujte hardwarové vlastnosti komunikačního rozhraní RS485 a jeho implementaci na vývojovém kitu M68EVB908GB60.
3. Vytvořte programovou knihovnu v jazyce symbolických adres mikropočítačů rodiny HCS08 podporující vybraný komunikační protokol.
4. Vypracujte ukázkový program používající vytvořenou komunikační knihovnu pro demonstraci její funkce.
5. Zpracujte prezentaci pro podporu výuky cvičení předmětu Mikropočítače pro seznámení studentů s funkcemi knihovny.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. Freescale Semiconductor. CPU08 Central Processor Unit Reference Manual., 2001. Dostupný z WWW: [www.freescale.com]
2. Freescale Semiconductor. HCS08 Family Reference Manual, Rev.1., 2003. Dostupný z WWW: [www.freescale.com]
3. Freescale Semiconductor. MC9S08GB/GT Data Sheet, Rev.2.3., 2004. Dostupný z WWW: [www.freescale.com]
4. GOFTON, Peter W. Sériová komunikace. 1. vyd. Praha : Grada, 1995. 234 s. ISBN 978807169131
5. HRUŠKA, František. Technické prostředky automatizace IV : snímače, převodníky, regulátory, průmyslová výpočetní technika, ovládací jednotky. Vyd. 3. Zlín : Univerzita Tomáše Bati ve Zlíně, 2005. 108 s. ISBN 8073182742.
6. Modbus Organization, Inc. Modbus Application Protocol Specification V1.1b., 2006. Dostupný z WWW: [www.modbus.org]
7. Modbus Organization, Inc. MODBUS over Serial Line Specification and Implementation Guide V1.02., 2006. Dostupný z WWW: [www.modbus.org]
8. VÁŇA, Vladimír. Začínáme s mikrokontroléry Motorola HC08 Nitron. Praha: BEN – technická literatura, 2003. 96 s. ISBN 80-7300-124-1

Vedoucí diplomové práce:

Ing. Petr Dostálek, Ph.D.

Ústav automatizace a řídicí techniky

Datum zadání diplomové práce:

24. února 2011

Termín odevzdání diplomové práce:

18. května 2011

Ve Zlíně dne 24. února 2011


prof. Ing. Vladimír Vašek, CSc.
děkan




doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Obsah práce je věnován především protokolu MODBUS a jeho implementaci na sériové sběrnici RS485. Cílem bylo vytvořit knihovnu v jazyce symbolických adres pro použití protokolu MODBUS ve vývojovém kitu Freescale M68EVB908GB60, ukázkový program a zpracovat výukovou prezentaci o knihovně pro účely výuky v předmětu „Mikropočítače“ na FAI UTB. Dále je také popsáno několik základních průmyslových sériových rozhraní a sběrnic.

Klíčová slova: MODBUS, RS485, M68EVB908GB60, CRC, Profibus, CAN, RS232, USB, Průmyslový Ethernet

ABSTRACT

The content of the thesis is mainly focused on the MODBUS protocol and its implementation on a RS485 serial bus. The aim was to create a library in an assembly language for using the MODBUS protocol in the Freescale M68EVB908GB60 development kit, compile a sample program and make a tutorial presentation about the library to support teaching in the course named „Microcomputers“ at FAI UTB. Several basic industrial serial interfaces and buses are also described.

Keywords: MODBUS, RS485, M68EVB908GB60, CRC, Profibus, CAN, RS232, USB, Industrial Ethernet

Tímto bych chtěl poděkovat vedoucímu své diplomové práce, Ing. Petru Dostálkovi, Ph.D. za ochotu, cenné rady a pomoc, kterou mi věnoval při odhalování téměř neřešitelných problémů, které při komunikaci dvou hardwarových zařízení nastávaly.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD	9
I TEORETICKÁ ČÁST	10
1 KOMUNIKAČNÍ SBĚRNICE A ROZHRANÍ	11
1.1 RS232	11
1.2 USB	12
1.3 RS485	13
1.4 AS-INTERFACE	14
1.5 CAN	15
1.6 PROFIBUS	16
1.7 PRŮMYSLVÝ ETHERNET	16
1.8 BLUETOOTH	17
2 KOMUNIKAČNÍ PROTOKOL MODBUS	19
2.1 OBECNÝ POPIS	19
2.1.1 Popis protokolu	19
2.1.2 Datový model	22
2.1.3 Adresovací model.....	23
2.1.4 Definice transakce protokolu MODBUS	24
2.2 KATEGORIE KÓDŮ FUNKCÍ.....	25
2.3 DEFINICE FUNKČNÍCH KÓDŮ.....	26
2.4 POPIS KÓDŮ ZÁKLADNÍCH FUNKCÍ.....	27
2.4.1 01 (01h) Čti cívky (Read Coils).....	27
2.4.2 02 (02h) Čti diskretní vstupy (Read Discrete Inputs)	28
2.4.3 03 (03h) Čti uchovávací registry (Read Holding Registers).....	29
2.4.4 04 (04h) Čti vstupní registry (Read Input Registers).....	30
2.4.5 05 (05h) Zapiš jednu cívku (Write Single Coil)	31
2.4.6 06 (06h) Zapiš jeden registr (Write Single Register).....	32
2.4.7 15 (0Fh) Zapiš více cívek (Write Multiple Coils).....	33
2.4.8 16 (10h) Zapiš více registrů (Write Multiple Registers).....	34
2.5 CHYBOVÉ ODPOVĚDI	35
2.6 IMPLEMENTACE PROTOKOLU MODBUS NA SÉRIOVÉ Lince	37
2.6.1 MODBUS RTU.....	38
2.6.2 MODBUS ASCII	39
2.6.3 Master/Slave stavové diagramy	39
2.6.4 CRC kód.....	41
3 VÝVOJOVÝ KIT M68EVB908GB60	44

3.1	ZÁKLADNÍ VLASTNOSTI.....	44
3.2	KONFIGURACE KITU PRO POUŽITÍ RS485.....	46
3.3	POPIS SÉRIOVÉHO ROZHRANÍ MIKROPOČÍTAČE MC9S08GB60	48
3.3.1	Formát datového rámce SCI.....	48
3.3.2	Nastavení SCI.....	49
II	PRAKTICKÁ ČÁST	53
4	KNIHOVNA MODBUS_GB60	54
4.1	POŽADAVKY NA MIKROPOČÍTAČ.....	54
4.2	DATOVÝ MODEL MODBUSU POUŽITÝ V KNIHOVNĚ.....	54
4.3	POPIS UŽIVATELSKÝCH FUNKCÍ	55
4.3.1	MODBUS_master_gb60	55
4.3.2	MODBUS_slave_gb60	58
4.4	PROMĚNNÉ A KONSTANTY	61
4.5	KOMUNIKAČNÍ (SCI) FUNKCE	63
4.6	FUNKCE PRO OBSLUHU PŘERUŠENÍ	64
4.7	DALŠÍ FUNKCE.....	65
4.7.1	Funkce pro práci s daty	65
4.7.2	Pomocné funkce	68
5	UKÁZKOVÝ PROGRAM	70
5.1	MASTER	70
5.1.1	Proměnné.....	70
5.1.2	Hlavní program	71
5.2	SLAVE	73
5.2.1	Proměnné.....	73
5.2.2	Hlavní program	73
	ZÁVĚR.....	75
	ZÁVĚR V ANGLIČTINĚ	76
	SEZNAM POUŽITÉ LITERATURY	77
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	79
	SEZNAM OBRÁZKŮ	81
	SEZNAM TABULEK.....	83
	SEZNAM PŘÍLOH.....	84
	PŘÍLOHA P I: ZDROJOVÉ KÓDY KNIHOVNY MODBUS_GB60.....	85
	PŘÍLOHA P II: VÝUKOVÁ PREZENTACE	86
	PŘÍLOHA P III: FONT MATRIX SCHEDULE.....	87
	PŘÍLOHA P IV: VÝPOČET CRC V EXCELU	88

ÚVOD

Zařízení, která komunikují v průmyslovém prostředí, musí splňovat určité přísnější požadavky pro dané prostředí, to se týká zejména prašnosti, vlhkosti, vibrací apod. Zvýšené nároky platí také na komunikační rozhraní a sběrnice, např. napěťové úrovně, odolnost proti rušení a šumu. S tím souvisí také vyšší požadavky na pevnost kabelů a konektorů.

Průmyslová komunikace může být využívána k řízení technologických procesů, měření a regulaci (zejména) technologických veličin. To jsou tři základní úlohy průmyslové automatizace. Tato problematika je velmi široká, a proto je obsah práce zaměřen pouze na stručný popis některých komunikačních rozhraní a sběrnic. Typickým zástupcem sériového rozhraní je RS232, které se i přes své stáří stále uplatňuje v oblasti průmyslové komunikace, byť v oblasti osobních počítačů bylo toto rozhraní již téměř definitivně vytlačeno sběrnicí USB. Ještě častěji je v průmyslu využívána sběrnice RS485. Na rozdíl od RS232 se již nejedná pouze o dvoubodový spoj, ale o sběrnici s možností připojení více zařízení, z nichž jedno zařízení řídí komunikaci na sběrnici. Standard RS485 definuje pouze způsob zapojení a související požadavky, ale neurčuje, jakým způsobem bude komunikace probíhat. K tomuto účelu slouží komunikační protokoly, což jsou programové prostředky, které určují způsob, postup a další aspekty přenosu. Nejčastěji používanými protokoly v praxi jsou Profibus, MODBUS, CAN, průmyslový Ethernet a další. Vzhledem k tomu, že dnešní systémy průmyslové automatizace a systémy informatiky jsou již velmi úzce propojeny, je stručně zmíněna i sběrnice USB, která má své místo v laboratorním prostředí. Z podobného důvodu je zmíněna i technologie Bluetooth jako zástupce bezdrátové komunikace.

Největší prostor je věnován protokolu MODBUS, který je nosným tématem celé práce. Jedná se o otevřený a dobře zdokumentovaný komunikační protokol, který slouží jako komunikační prostředek různých zařízení (PLC, mikropočítače ...). MODBUS může být použit jak na sériové sběrnici RS485, ale i na klasickém Ethernetu.

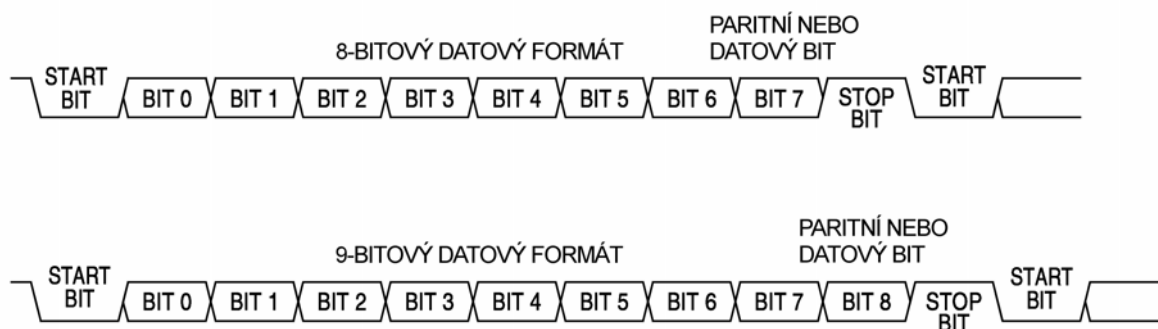
Cílem práce je především vytvořit programovou knihovnu v jazyce symbolických adres mikropočítačů rodiny HCS08 a v ukázkovém programu demonstrovat její funkce. S tím souvisí také vhodně napsaná dokumentace k této knihovně jak ve formě této práce, tak ve formě prezentace sloužící zejména pro výukové účely cvičení předmětu Mikropočítače.

I. TEORETICKÁ ČÁST

1 KOMUNIKAČNÍ SBĚRNICE A ROZHRANÍ

1.1 RS232

Asynchronní sériové komunikační rozhraní pro komunikaci dvou zařízení. Pro přenos signálu se používá standardní NRZ (Non Return to Zero) formát a je plně duplexní. V úplném zapojení má až 25 vodičů a používá konektor CANON25 s tímto počtem vývodů. Běžněji se používá varianta s 9 vodiči a konektorem CANON9, přičemž jsou často využity pouze 3 základní vodiče: TxD, RxD a GND. Standard udává maximální délku propojení 15 m. Rozhraní RS232 bylo dříve standardem pro komunikaci s PC, dnes již téměř vytlačeno sběrnici USB, nicméně stále zůstává průmyslovým standardem.



Obr. 1. Přenosový rámec rozhraní RS232

Logické úrovně datových signálů „1“ (high) a „0“ (low) jsou reprezentovány pomocí dvou bipolárních úrovní napětí. [6] [7] [14]

Tab. 1. Napěťové úrovně datových signálů

Datové signály		
Úroveň	Vysílač	Přijímač
Log. 0	+5 V to +15 V	+3 V to +25 V
Log. 1	-5 V to -15 V	-3 V to -25 V
Nedefinovaný	3 V to +3 V	

Tab. 2. Napěťové úrovně řídicích signálů

Řídicí signály		
Signál	Driver	Terminátor
"Off"	-5 V to -15 V	-3 V to -25 V
"On"	5 V to 15 V	3 V to 25 V

1.2 USB

USB (Universal Serial Bus) je univerzální sériová sběrnice. Nejčastěji se používá pro připojení periferních zařízení k počítači (tiskárna, fotoaparát, myš, klávesnice atd.). V tomto ohledu USB sběrnice nahradila starší způsoby připojení periferií (sériový a paralelní port, PS/2). Zařízení lze připojovat k počítači za chodu díky technologii Plug & Play.

USB sběrnice je založena na architektuře Master/Slave. Data se vysílají v krátkých (8 bytů) a delších (256 bytů) paketech. Přenos dat probíhá po rámcích (frame) trvajících přesně 1 ms. Pokud komunikuje s PC (master) více zařízení, zajišťuje jejich rozdělení hub.

USB sběrnice používá dva typy konektorů:

- Typ A – plochý konektor (PC konektor)
- Typ B – konektor pro periferie (tiskárna fotoaparát)

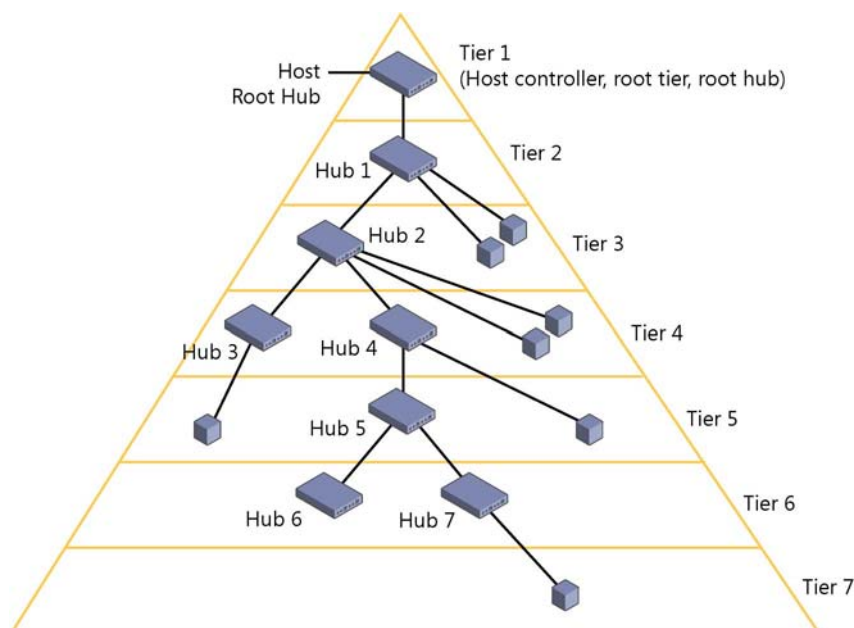
Dále existují konektory miniUSB a mikroUSB, které se používají např. pro mobilní telefon nebo mp3 přehrávač.

Tab. 3. Označení pinů USB propojení

Pin	Jméno	Barva	Popis
1	V_{BUS}	Červená	+5 VDC
2	D-	Bílá	Data -
3	D+	Zelená	Data +
4	GND	Černá	GND

Ke sběrnici lze připojovat huby (rozdělovače) nebo periferie (myš, klávesnice, tiskárna atd.). USB sběrnice používá víceúrovňovou hvězdicovou topologii. Hub je nejvyšším

bodem hvězdicového propojení. Každé zařízení má svoji adresu a může komunikovat s jedním nebo více koncovými uzly (nodes) v několika úrovních (tier).



Obr. 2. Topologie USB

Zařízení připojené na USB sběrnici může být napájeno přímo ze sběrnice napětím 5 V a odebírat proud až 100 mA (případně až 500 mA).

Přenosová rychlost je u verze USB 1.1 až 12 Mb/s, u verze USB 2.0 až 480 Mb/s a u nejnovější verze USB 3.0 je maximální přenosová rychlost až 5 Gb/s, přičemž tato verze má 8 vodičů namísto původních 4, ale je zpětně kompatibilní s verzí USB 2.0. Na sběrnici lze připojit až 127 zařízení, maximální délka kabelu mezi dvěma zařízeními je 5 m. [6] [7] [11]

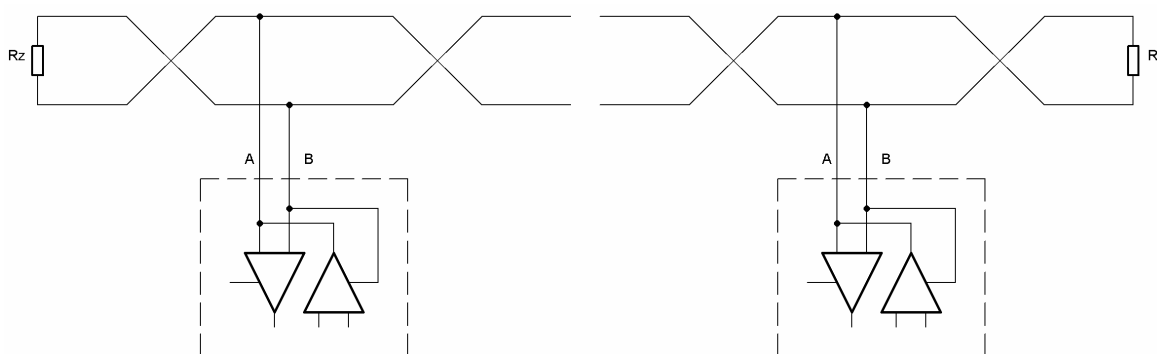
1.3 RS485

RS485 je sériová sběrnice a komunikační standard zejména v průmyslové oblasti. Na logické úrovni je podobný standardu RS232, ale používá odlišné napěťové úrovně a umožňuje připojit na linku více zařízení. Standard RS485 je navržen jako dvouvodičový poloduplexní sériový spoj, definuje pouze elektrické zapojení a technické požadavky ale ne způsob, jakým způsobem budou jednotlivá zařízení komunikovat (programové požadavky), to závisí na použitém komunikačním protokolu. Mezi nejznámější průmyslové protokoly patří Profibus nebo MODBUS.

Maximální délka sběrnice je až 1200 m, maximální počet zařízení připojených na sběrnici (bez použití opakovačů) je 32. Přenosová rychlost může dosáhnout až 10 Mb/s u krátkých spojů do 10 m, se vzrůstající délkou spoje přenosová rychlost značně klesá. Doporučeným kabelem je kroucená dvojlinka s charakteristickou impedancí 120Ω a na koncích kabelu jsou připojeny ukončovací rezistory (terminátory) 120Ω pro potlačení nežádoucích odrazů na konci linky. Nejběžněji se používá dvou vodičová poloduplexní varianta RS485, ale existuje čtyřvodičová plně duplexní varianta, která je někdy nesprávně zaměňována se sběrnici RS422. [6] [7]

Tab. 4. Napět'ové úrovně signálů

Napět'ové úrovně	
Mark(„1“)	kladné napětí ($B - A > +200 \text{ mV}$)
Space(„0“)	záporné napětí ($B - A < -200 \text{ mV}$)



Obr. 3. Princip propojení RS485 (standardní dvou vodičová varianta)

1.4 AS-Interface

Jedná se o sběrnicev'ý systém pro připojení senzorů (např. světelný senzor) a aktorů (např. pneumatický válec) pomocí jednoduchého dvou vodičového vedení. Na sběrnici ve verzi 1.0 může být připojeno až 31 jednotek, přičemž vzdálenost mezi nimi by neměla být větší než 100 m. Provoz na sběrnici řídí master, tak že cyklicky sbírá signály ze senzorů a distribuuje řídicí signály na aktory. Tento proces se nazývá *polling*.

Všechny komponenty jsou přizpůsobeny SIMATIC NET, to znamená, že je ASI připojitelný na Profibus nebo průmyslový Ethernet pomocí integrovaných rozhraní nebo komunikačních modulů. Provozní zařízení lze připojit ke sběrnici jednoduchým způsobem – prořezávací technika "Click&Go". [6] [7] [12]

1.5 CAN

Sběrnice CAN (Controller Area Network) vyvinuta firmou Bosch pro použití v průmyslové automatizace. Nejčastěji se CAN používá na propojení programovatelných řadičů, diskrétních prvků a inteligentních senzorů. V automobilech se CAN používá k ovládání světelných indikátorů a ukazatelů.

Jednotky připojené na sběrnici CAN jsou v rovnocenném postavení. Každá jednotka přijímá všechna data ze sběrnice, ale ukládá si jen ta data, která jí náleží. Data jsou identifikována pomocí funkčního označení (identifikátoru).

Pokud je sběrnice volná, může začít vysílat kterákoliv jednotka. Při současném vysílání více jednotek se upřednostňuje rámec s nejvyšší prioritou, ostatní jednotky ukončí vysílání a po uvolnění sběrnice se mohou opět pokusit o vysílání. Protokol rozlišuje 2 stavy pro přístup účastníka na sběrnici a to recesivní a dominantní. Je-li vyslán dominantní bit (nula), jsou recesivní bity vysílané ostatními jednotkami přepsány dominantními bity. Prioritu rámce udává 11-bitový identifikátor na začátku rámce. Menší číslo identifikátoru udává vyšší prioritu. Je-li detekována kolize se jednotky vyjma rámce s nejvyšší prioritou (nejvíce nul v identifikátoru převáží při logickém součinu proti jedničkám) odpojí. Tato metoda určení priority se nazývá metoda bitového arbitra sběrnice. Tato metoda je označována za nedestruktivní, protože kolize nezpůsobí přerušení rámce s nejvyšší prioritou. Protože se jednotky se rozhodují o přijetí rámce pouze na základě identifikátoru priority, je tento proces nazýván filtrace rámců.

Zabezpečení proti chybám, vzniklých při přenosu zprávy, zajišťuje 16-bitový CRC kód. Dále se kontrolují napět'ové úrovně signálů na sběrnici a průchodnost spojení (zda nedošlo k zablokování cesty). Průchodnost cesty se zkouší testovací posloupností 5 stejných a 1 opačného bitu. Chyba nastává, pokud příjemce detekuje více než 5 stejných bitů.

Vrstvový model CAN je odlišný od jiných sběrnic. Model má 4 vrstvy: fyzickou linkovou, objektovou a uživatelskou vrstvu. Fyzická vrstva je tvořena sériovou komunikační sběrnici, na kterou jsou připojeny jednotky. Spojovací vrstva zabezpečuje přístupovou metodu MAC (Medium Access Control) ke sdílenému přenosovému médiu, formát logických rámců, zabezpečení proti chybám řízení toku dat. Objektová vrstva zabezpečuje zprávy, určuje priority, filtraci rámců. Uživatelská vrstva interpretuje uživateli přenášené

zprávy. Přenosová rychlost sběrnice může dosahovat až 1 Mb/s při délce sběrnice 40 m. [12]

1.6 Profibus

Profibus je průmyslová sběrnice firmy Siemens a je určena pro široké uplatnění v automatizaci a to pro automatizaci výrobních linek, pro domovní či procesní automatizaci a pro řízení výroby a distribuce energie. Jde se dominantní průmyslovou sběrnici v Evropě.

Profibus podporuje 4 topologie propojení a to: sběrnice strom, hvězda a kruh, přičemž doporučenou topologií je sběrnice. Maximální délka sběrnice Profibus je (u varianty DP/FMS pro RS485) až 1200 m a maximální přenosová rychlost (pro délku sběrnice do 100 m) dosahuje až 12 Mb/s. Přenosová rychlost Profibus PA je konstantní 31,25 kb/s. Profibus je vhodný i pro úlohy běžící v reálném čase, protože přístup na sběrnici je deterministický.

Nejpoužívanější variantou sběrnice je Profibus DP (Decentralized Periphery), používá se zejména pro komunikaci s PLC a umožňuje využití více jednotek master, přičemž pro zajištění výlučného přístupu ke sběrnici se využívá metoda token passing nebo také token ring, tzn., že si jednotky master postupně předávají pověření k přístupu na sběrnici. Ke sběrnici lze připojit až 32 zařízení a při použití opakovačů i více. [12] [13]

1.7 Průmyslový Ethernet

Vzhledem k tomu, že systémy průmyslové automatizace a informatiky jsou stále složitější a komplexnější a často je nutné spojovat různé podsystémy řešící jiné úlohy, může být někdy výhodné použít jeden systém, který dokáže ovládat, jak dílčí úlohy automatizace (měření, řízení, regulace atd.), tak i komunikaci na vyšší úrovni (velín). Na druhou stranu existují osvědčená sériová rozhraní, která spolehlivě fungují i pro řešení úloh v reálném čase. Proč tedy zavádět Ethernet do průmyslu? Ethernet dokáže řešit problém distribuované inteligence, splňuje požadavky informačních technologií na automatizaci a komunikaci přes všechny úrovně systémů integrované automatizace a informatiky (SIA). Ethernet je otevřeným standardem pro komunikaci v globálním měřítku. Hlavní výhodou Ethernetu je vysoká přenosová rychlost (dnes až 10 Gb/s) a také komplexnost, tj. není potřeba spojovat různá sériová rozhraní (často nekompatibilní). Naopak překážkou pro

rozšíření průmyslového Ethernetu je fakt, že většina komunikace v průmyslu dnes stále zajišťují rozhraní RS485/RS422 a RS232. Dalším problémem je zajištění rychlé odezvy, což je nutné u úloh probíhajících v reálném čase, protože (kancelářský) Ethernet je v principu síť s náhodným přístupem pomocí metody CSMA/CD. Průmyslový Ethernet také klade větší požadavky na technické prostředky než kancelářská verze. V průmyslovém prostředí musí odolávat různým vlivům, takže vyžaduje lepší elektrické krytí (IP54 až IP68), odolnost vůči EMC, změnám teplot, vlhkosti, prašnosti, rázům a vibracím. Odlišné jsou také konektory a kabeláž.

Problém s řešením realtime přístupu je možné řešit cyklickým přístupem (EtherCat, PowerLink), vymyká se ale standardu. Další možností je doplnění mechanismu pro realtime do standardního (kancelářského) Ethernetu (Profinet).

Verze I-Ethe s implementovaným realtime mechanismem:

- PowerLink: Všechny stanice mají vlastní časování a striktní limitní přístup. V jeden okamžik může na síť přistupovat pouze jedna stanice, čímž je vyloučena kolize.
- Profinet: Vyvinut firmou Siemens, při realtime komunikaci se obchází 3. a 4. vrstva modelu OSI a zvyšuje se propustnost sítě zkrácením velikosti dat protokolu, toto je podporováno i hardwarově obvodem ASIC. Profinet neumožňuje použít hub.
- EtherCat: Cyklický přístup, data jsou přijímány pouze odpovídající stanicí a ne všemi jako u standardu, vstupní data jsou vložena do telegramu, který putuje přes všechny stanice až k cílové stanici. [7]

1.8 Bluetooth

Bluetooth je bezdrátová komunikační technologie vyvinutá firmou Ericsson, která k bezdrátovému propojení dvou nebo více elektronických zařízení, obvykle mobilní telefon, PDA, PC či náhlavní souprava.

Název Bluetooth byl inspirován jménem dánského krále Haralda Modrozuba (Harald Bluetooth), který díky svým diplomatickým schopnostem pomohl ukončit vzájemné rozepře válečných kmenů. Analogicky tedy technologie Bluetooth slouží k usnadnění vzájemné komunikace.

Bluetooth je definován standardem IEEE 802.15.1. a řadí se mezi osobní počítačové sítě tzv. PAN (Personal Area Network). Existuje několik vývojových verzí, dnes nejčastěji používána je verze 2.0 a je implementována (2010) ve většině podporovaných zařízení (mobilní telefony, notebooky atd.). Nejnovější je verze 4.0 (2011), která by měla mít dosah až 100 m a také podporu šifrování AES-128.

Rozdělení podle dosahu a výkonu (udávané hodnoty platí ve volném prostoru):

- Class 1 – až 100 metrů (100 mW)
- Class 2 – až 10 metrů (2,5 mW)
- Class 3 – až 1 metr (1 mW)

Překážky mezi komunikujícími zařízeními způsobí výrazný pokles dosahu. Obvykle pak dochází spíše k nárůstu počtu chybně přenesených paketů než ke skokové ztrátě spojení.

Rozdělení podle jednotlivých verzí:

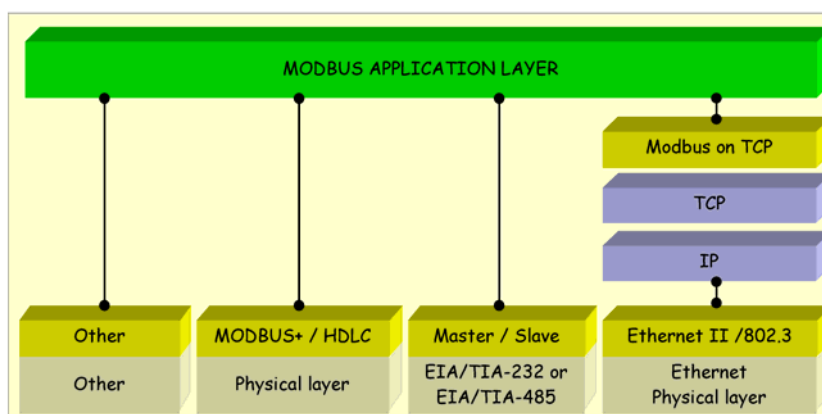
- Verze 1.2 1 Mbit/s
- Verze 2.0 + EDR 3 Mbit/s
- Verze 3.0 + HS 24 Mbit/s
- Verze 4.0 24 Mbit/s

Bluetooth pracuje stejně jako Wi-fi v pásmu 2,4 GHz. Přenos dat probíhá pomocí metody FHSS (Frequency-hopping spread spectrum), kdy je provedeno 1600 skoků (přeladění) mezi 79 frekvencemi (rozestup mezi frekvencemi je 1 MHz) za sekundu. Tímto způsobem se zlepšuje odolnost proti rušení na stejné frekvenci.

K identifikaci zařízení slouží adresa *BT_ADDR*, kterou má každé zařízení. Bluetooth umožňuje komunikaci dvou i více zařízení. Ve druhém případě jsou zařízení připojena do tzv. pikosítě, přičemž jedno zařízení je řídicí (master) a může obsluhovat až 7 podřízených (slave) zařízení. Zařízení uvnitř pikosítě jsou synchronizována s taktem řídicí stanice. Pikosítě lze dále spojovat a tím vzniká tzv. scatternet. [7]

2 KOMUNIKAČNÍ PROTOKOL MODBUS

MODBUS je otevřený komunikační protokol pracující na aplikační úrovni modelu OSI, založený na komunikační architektuře Klient/Server (Master/Slave) mezi zařízeními připojenými na různých typech sběrnic a sítí. Vznikl v roce 1979 jako otevřený průmyslový standard. Komunikace je založena na principu požadavek/odpověď, přičemž žádaná funkce specifikována kódem funkce je součástí požadavku. Pro komunikaci je podporováno poměrně velké množství různých sběrnic a sítí např. síť Ethernet pomocí protokolu TCP/IP, asynchronní sériový přenos (RS-232C, RS-422, RS-485, optické vlákno, radiový přenos) a vysokorychlostní síť MODBUS PLUS. MODBUS používá „Big-endian“ reprezentaci dat tzn., že zprávy delší než 1 byte jsou posílány postupně od nejvyššího bytu (MSB) po nejnižší (LSB).

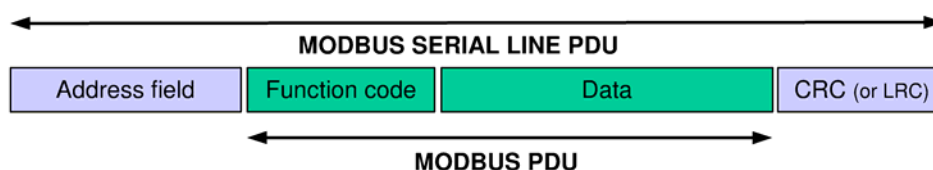


Obr. 4. Možnosti implementace protokolu MODBUS

2.1 Obecný popis

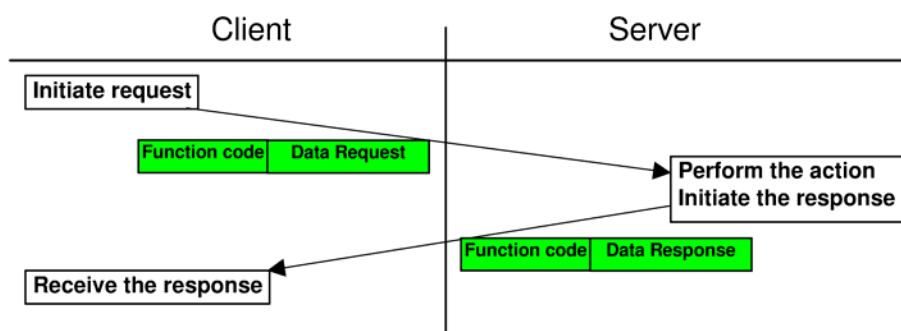
2.1.1 Popis protokolu

Protokol MODBUS definuje komunikační zprávu na úrovni protokolu – PDU paket (Protocol Data Unit), která je nezávislá na použité komunikační vrstvě. Ke komunikaci je nutné PDU rozšířit o další části v závislosti na použité síti nebo sběrnici a vytváří tak zprávu na aplikační úrovni – ADU rámeček (Application Data Unit).



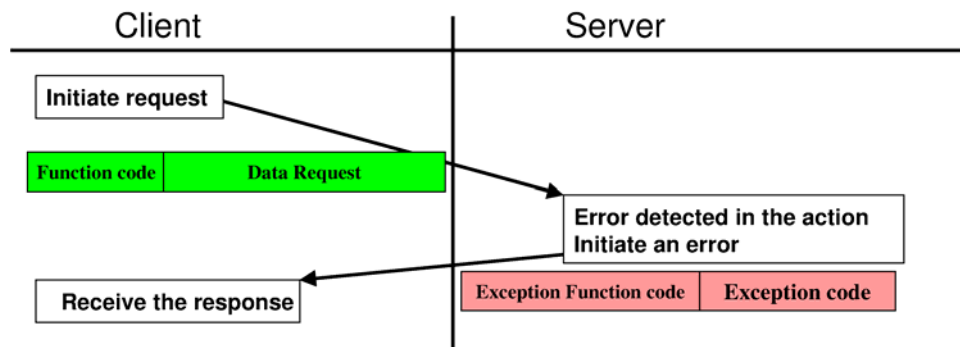
Obr. 5. Základní struktura zprávy protokolu MODBUS

ADU rámeček vytváří klient (též „master“) pro zahájení komunikace. Kód funkce udává serveru (též „slave“) jaký druh operace má provést. Kódy jsou v rozsahu 1 až 255 (kódy v rozsahu 128 až 255 jsou vyhrazeny pro oznámení chyby). Kódy některých funkcí obsahují navíc kód podfunkce, který definuje vícenásobné operace. Datová část posílaná klientem slouží serveru k vykonání operace určené kódem funkce a může obsahovat např. adresy registrů, do kterých má server zapisovat nebo vstupy ze kterých má server číst. Některé funkce nevyžadují datovou část, k vykonání operace stačí samotný kód funkce. Pokud je požadovaná funkce vykonána bez chyby, vrací server zprávu se stejným kódem funkce, který přijal od klienta, pro potvrzení úspěšného vykonání požadované funkce. Pokud klient požadoval nějaká data, jsou předána v datové části.



Obr. 6. MODBUS – bezchybná transakce

Nastane-li při vykonávání požadované funkce chyba, vrací server zprávu se stejným kódem funkce, který přijal od klienta, ale s nastaveným nejvyšším bitem indikujícím chybu (exception response). Datová část obsahuje chybový kód (exception code), který udává důvod chyby.



Obr. 7. MODBUS – chybná transakce

Pozn.: Na straně klienta je více než vhodné nastavit časový limit pro přijetí odpovědi, která z důvodu ztráty požadavku nebo odpovědi, nemusí nikdy přijít.

Maximální velikost PDU je přejata z první implementace protokolu MODBUS na sériové lince RS-485 (max. RS-485 ADU = 256 bytů).

Maximální velikost PDU na sériové lince = 256 – adresa serveru (1 byte) – kontrolní součet 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

Protokol MODBUS definuje 3 typy PDU zpráv:

- Požadavek (Request PDU)
 - [1 byte] Kód funkce
 - [n bytů] Datová část požadavku (adresa, proměnné, počet proměnných ...)
- Odpověď (Response PDU)
 - [1 byte] Kód funkce
 - [m bytů] Datová část odpovědi (přečtené vstupy, proměnné ...)
- Chybová odpověď (Exception Response PDU)
 - [1 byte] Kód funkce s nastaveným nejvyšším bitem (+ 80h)
 - [1 byte] Chybový kód (identifikace chyby)

2.1.2 Datový model

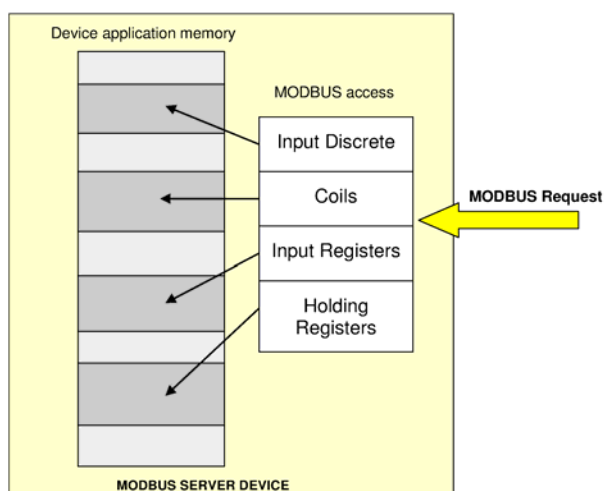
Datový model protokolu MODBUS je založen na sadě tabulek se specifickým významem. Čtyři základní tabulky jsou uvedeny v tabulce 1.

Tab. 5. Datový model protokolu MODBUS

Tabulka	Velikost položky	Přístup	Popis
Diskrétní vstupy (Discrete inputs)	1-bit	Pouze čtení	Data poskytovaná I/O systémem
Cívky (Coils)	1-bit	Čtení/Zápis	Data měnitelná aplikačním programem
Vstupní registry (Input Registers)	16-bitové slovo	Pouze čtení	Data poskytovaná I/O systémem
Uchovávací registry (Holding Registers)	16-bitové slovo	Čtení/Zápis	Data měnitelná aplikačním programem

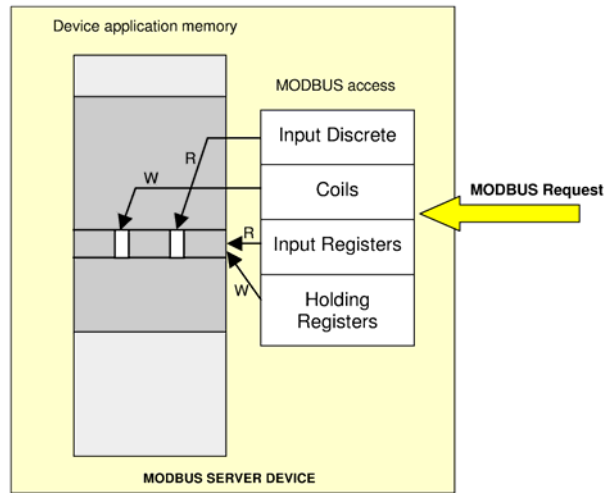
Tabulky mohou obsahovat až 65536 datových položek a mohou se vzájemně překrývat. Přístupovat lze k jednotlivým položkám i ke skupině položek, která je ovšem omezena maximální velikostí datové části zprávy.

Na Obr. 8 je znázorněno zařízení, kde každá tabulka má svůj oddělený prostor v aplikační paměti zařízení a mezi položkami jednotlivých tabulek není žádný vztah.



Obr. 8. Datový model protokolu MODBUS s oddělenými bloky

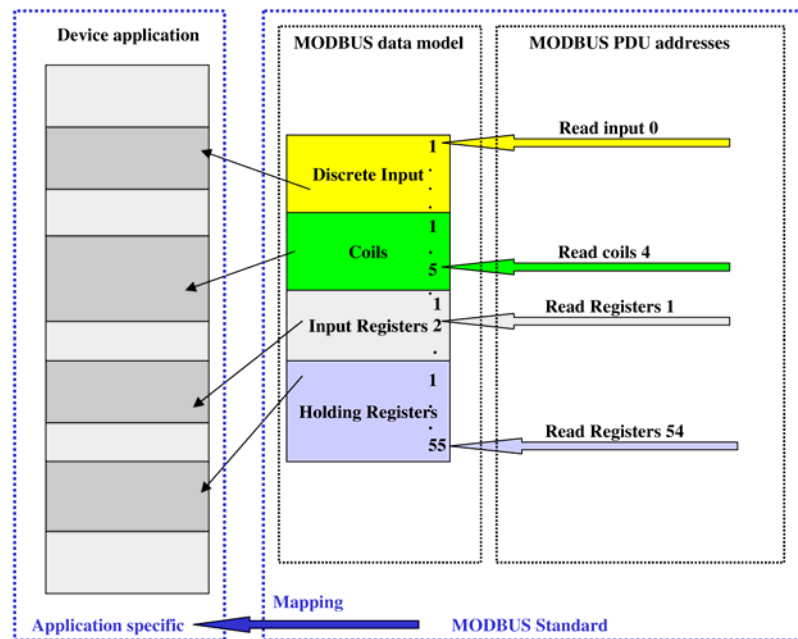
Obr. 9 znázorňuje zařízení s jedním společným datovým blokem a položky jsou přístupné pomocí různých funkcí protokolu MODBUS.



Obr. 9. Datový model protokolu MODBUS s jedním společným blokem

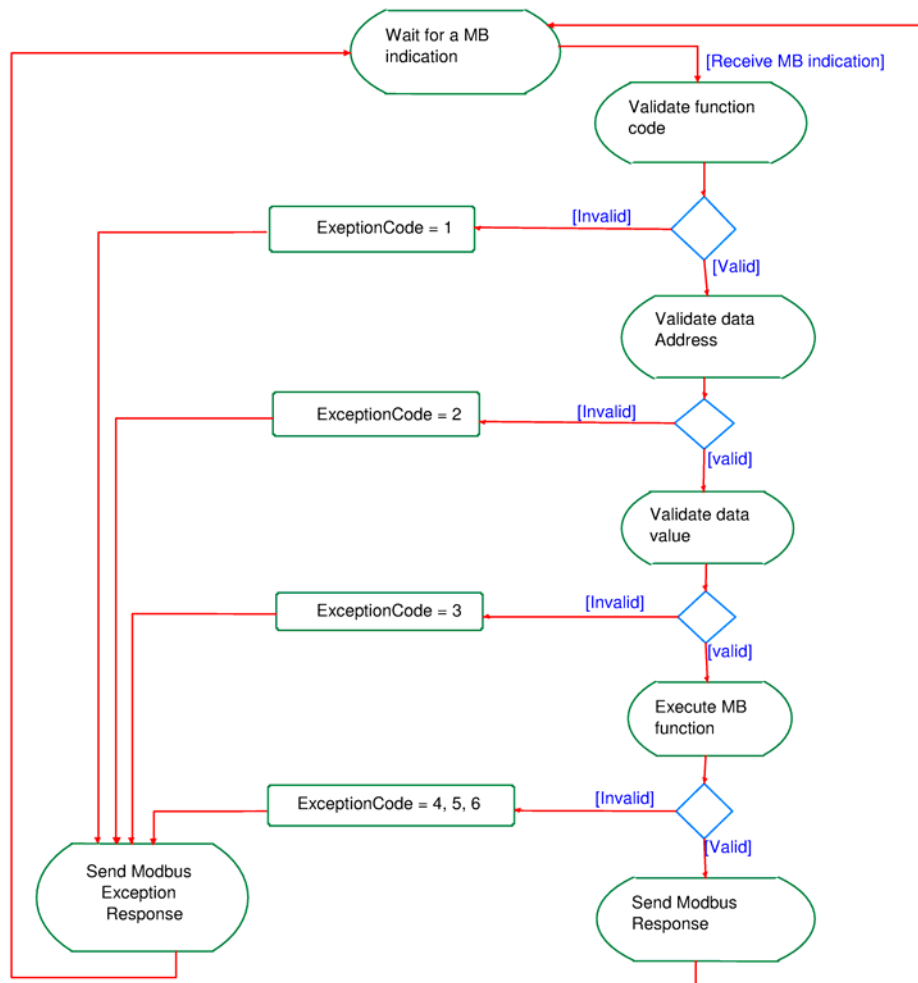
2.1.3 Adresovací model

Protokol MODBUS přesně specifikuje adresovací pravidla ve zprávách PDU. Data jsou adresována od 0 do 65535 a v rámci datového modelu složeného ze čtyř tabulek jsou položky v datových blocích číslovány od 1 do n.



Obr. 10. MODBUS – adresovací model

2.1.4 Definice transakce protokolu MODBUS



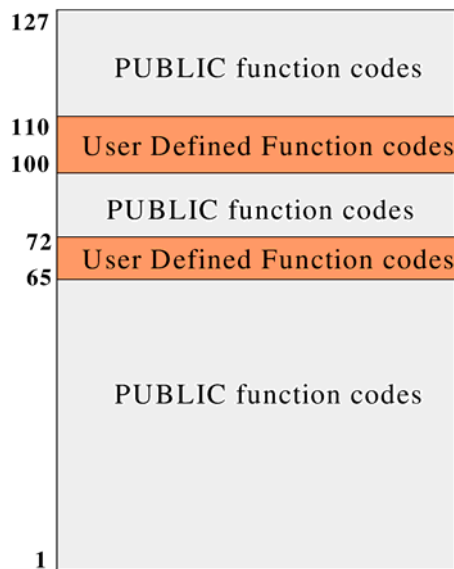
Obr. 11. Stavový diagram zpracování MODBUS požadavku

Po úspěšném zpracování požadavku pošle server odpověď s kódem vykonané funkce, v opačném případě pošle server odpověď s kódem vykonané funkce s nastaveným nejvyšším bitem a kódem chyby, udávajícím důvod neúspěchu.

2.2 Kategorie kódů funkcí

Protokol MODBUS definuje 3 skupiny kódů funkcí:

- **Veřejné kódy funkcí** jsou jasně definované unikátní kódy funkcí se zveřejněnou dokumentací. Jsou schvalovány organizací MODBUS-IDA.org a zahrnují veřejné přiřazené kódy funkcí i nepřiřazené kódy rezervované pro budoucí použití.
- **Uživatelsky definované kódy funkcí** slouží k vytváření funkcí definovaných uživatelem. Není garantována unikátnost klíčů a po schválení přesunout do veřejných kódů.
- **Rezervované kódy funkcí** jsou takové kódy funkcí, které jsou v současnosti používány některými firmami a které nejsou dostupné pro veřejné použití.



Obr. 12. Kategorie kódů funkcí

2.3 Definice funkčních kódů

Tab. 6. Definice funkčních kódů MODBUSU

				Kódy funkcí		
				Kód	Podfunkce	Hex
Přístup k datům	Bitový přístup	Fyzické diskretní vstupy	Čti diskretní vstupy	02		02
		Interní bity nebo fyzické cívky	Čti cívky	01		01
			Zapiš jednu cívku	05		05
			Zapiš více cívek	15		0F
	16-bitový přístup	Fyzické vstupní Registry	Čti vstupní registr			
		Interní registry nebo fyzické výstupní registry	Čti uchovávací registry	04		04
			Zapiš jeden registr	03		03
			Zapiš více registrů	06		06
			Čti/zapiš více registrů	16		10
			Zapiš registr s maskováním	23		17
	Čti FIFO frontu	22		16		
	Přístup k záznamům v souborech	Čti záznam ze souboru		24		18
		Zapiš záznam do souboru		20		14
	Diagnostika			Čti stav	21	
Diagnostika				07		07
Čti čítač kom. událostí				08	00-18,20	08
Čti záznam kom. událostí				11		0B
Sděl identifikaci				12		0C
Čti identifikaci zařízení				17		11
Ostatní			Zapouzdřený přenos	43	14	2B
			CANopen základní odkaz	43	13,14	2B

2.4 Popis kódů základních funkcí

2.4.1 01 (01h) Čti cívky (Read Coils)

Tato funkce umožňuje čtení stavu 1 až 2000 cívek. V požadavku je udána adresa první cívky a počet cívek. Odpověď obsahuje stavy cívek, jeden byte obsahuje stav celkem 8 cívek. První cívka se nachází v nejnižším bitu prvního bytu.

Požadavek

Kód funkce	1 byte	0x01
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet cívek	2 byty	1 až 2000 (0x7D0)

Odpověď

Kód funkce	1 byte	0x01
Počet bytů	1 byte	N
Stavy cívek	N bytů	

$N = \text{počet cívek} / 8$, je-li zbytek po dělení nenulový, $N = N + 1$

Chyba

Kód funkce	1 byte	0x81
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 01 0013 0025 0E84
11	Adresa slave (17 = 11 hex)
01	Kód funkce (čti cívky)
0013	Adresa (číslo) první cívky, která se bude číst (cívka 20 - 1 = 19 = 13 hex)
0025	Počet požadovaných cívek. (cívky od 20 do 56 = 37 = 25 hex)
0E84	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	11 01 05 CD6BB20E1B 45E6
11	Adresa slave (17 = 11 hex)
01	Kód funkce (čti cívky)
05	Počet bytů přečtených cívek (37 cívek / 8 bitů = 5 bytů)
CD	Cívky 27 - 20 (1100 1101)
6B	Cívky 35 - 28 (0110 1011)
B2	Cívky 43 - 36 (1011 0010)
0E	Cívky 51 - 44 (0000 1110)
1B	3 doplňující nuly & cívky 56 - 52 (0001 1011)
45E6	CRC (cyclic redundancy check).

2.4.2 02 (02h) Čti diskretní vstupy (Read Discrete Inputs)

Tato funkce umožňuje čtení stavu 1 až 2000 vstupů. V požadavku je udána adresa prvního vstupu a počet vstupů. Odpověď obsahuje stavy vstupů, jeden byte obsahuje stav celkem 8 vstupů. První vstup se nachází v nejnižším bitu prvního bytu.

Požadavek

Kód funkce	1 byte	0x02
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet vstupů	2 byty	1 až 2000 (0x7D0)

Odpověď

Kód funkce	1 byte	0x02
Počet bytů	1 byte	N
Stavy vstupů	N bytů	

$N = \text{počet vstupů} / 8$, je-li zbytek po dělení nenulový, $N = N + 1$

Chyba

Kód funkce	1 byte	0x82
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 02 00C4 0016 BAA9
11	Adresa slave (17 = 11 hex)
02	Kód funkce (čti diskretní vstupy)
00C4	Adresa prvního vstupu, který se bude číst. (10197 - 10001 = 196 = C4 hex)
0016	Počet požadovaných vstupů. (197 to 218 = 22 = 16 hex)

BAA9	CRC (cyclic redundancy check) pro kontrolu chyb.
Odověď	11 02 03 ACDB35 2018
11	Adresa slave (17 = 11 hex)
02	Kód funkce (čti diskretní vstupy)
03	Počet bytů přečtených vstupů (22 vstupů / 8 bitů = 3 byty)
AC	Diskretní vstupy 10204 -10197 (1010 1100)
DB	Diskretní vstupy 10212 - 10205 (1101 1011)
35	2 doplňující nuly & Diskretní vstupy 10218 - 10213 (0011 0101)
2018	CRC (cyclic redundancy check).

2.4.3 03 (03h) Čti uchovávací registry (Read Holding Registers)

Tato funkce umožňuje číst obsah až 125 uchovávacích registrů. V požadavku je udána adresa prvního registru a počet registrů. Odověď obsahuje hodnoty registrů, přičemž každému registru odpovídá dvojice bytů.

Požadavek

Kód funkce	1 byte	0x03
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet registrů	2 byty	1 až 125 (0x7D)

Odověď

Kód funkce	1 byte	0x03
Počet bytů	1 byte	2*N
Hodnoty registrů	2*N bytů	

N = počet registrů

Chyba

Kód funkce	1 byte	0x83
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 03 006B 0003 7687
11	Adresa slave (17 = 11 hex)
03	Kód funkce (čti uchovávací registry)
006B	Adresa prvního požadovaného registru. (40108-40001 = 107 = 6B hex)
0003	Počet požadovaných registrů. (číst 3 registry 40108 až 40110)
7687	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	11 03 06 AE41 5652 4340 49AD
11	Adresa slave (17 = 11 hex)
03	Kód funkce (čti uchovávací registry)
06	Počet bytů přečtených registrů (3 registry x 2 byty každý = 6 bytů)
AE41	Obsah registru 40108
5652	Obsah registru 40109
4340	Obsah registru 40110
49AD	CRC (cyclic redundancy check).

2.4.4 04 (04h) Čti vstupní registry (Read Input Registers)

Tato funkce umožňuje číst obsah až 125 vstupních registrů. V požadavku je udána adresa prvního registru a počet registrů. Odpověď obsahuje hodnoty registrů, přičemž každému registru odpovídá dvojice bytů.

Požadavek

Kód funkce	1 byte	0x04
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet registrů	2 byty	1 až 125 (0x7D)

Odpověď

Kód funkce	1 byte	0x04
Počet bytů	1 byte	2*N
Hodnoty registrů	2*N bytů	

N = počet registrů

Chyba

Kód funkce	1 byte	0x84
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 04 0008 0001 B298
11	Adresa slave (17 = 11 hex)
04	Kód funkce (čti vstupní registry)
0008	Adresa prvního požadovaného registru. (30009-30001 = 8)
0001	Počet požadovaných registrů. (číst 1 registr)
B298	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	11 04 02 000A F8F4
11	Adresa slave (17 = 11 hex)
04	Kód funkce (čti vstupní registry)
02	Počet bytů přečtených registrů (1 registr x 2 byty každý = 2 byty)
000A	Obsah registru 30009
F8F4	CRC (cyclic redundancy check).

2.4.5 05 (05h) Zapiš jednu cívku (Write Single Coil)

Tato funkce umožňuje zápis jedné cívky do stavu ON nebo OFF. V požadavku je udána adresa cívky, která se má nastavit a požadovaná hodnota cívky. 0x0000 = OFF, 0xFF00 = ON. Bezchybná odpověď je kopií požadavku.

Požadavek

Kód funkce	1 byte	0x05
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Hodnota cívky	2 byty	0x0000 nebo 0xFF00

Odpověď

Kód funkce	1 byte	0x05
Počet bytů	2 byte	0x0000 až 0xFFFF
Hodnota cívky	2 byty	0x0000 nebo 0xFF00

Chyba

Kód funkce	1 byte	0x85
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 05 00AC FF00 4E8B
11	Adresa slave (17 = 11 hex)
05	Kód funkce (zapiš jednu cívku)
00AC	Adresa (číslo) cívky, která se má zapsat. (cívka 173 - 1 = 172 = AC hex)
FF00	Požadovaná hodnota cívky (FF00 = ON, 0000 = OFF)
4E8B	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	11 05 00AC FF00 4E8B
11	Adresa slave (17 = 11 hex)
05	Kód funkce (zapiš jednu cívku)
00AC	Adresa (číslo) zapsané cívky. (cívka 173 - 1 = 172 = AC hex)
FF00	Zapsaná hodnota cívky (FF00 = ON, 0000 = OFF)
4E8B	CRC (cyclic redundancy check).

2.4.6 06 (06h) Zapiš jeden registr (Write Single Register)

Tato funkce umožňuje zápis jednoho uchovávacího registru. V požadavku je udána adresa registru, který se má zapsat a požadovaná hodnota registru. Bezchybná odpověď je kopií požadavku.

Požadavek

Kód funkce	1 byte	0x06
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Hodnota registru	2 byty	0x0000 až 0xFFFF

Odpověď

Kód funkce	1 byte	0x06
Počet bytů	2 byte	0x0000 až 0xFFFF
Hodnota registru	2 byty	0x0000 až 0xFFFF

Chyba

Kód funkce	1 byte	0x86
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 06 0001 0003 9A9B
11	Adresa slave (17 = 11 hex)
06	Kód funkce (zapiš jeden registr)
0001	Adresa registru. (40002 - 40001 = 1)
0003	Požadovaná hodnota registru
9A9B	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	11 06 0001 0003 9A9B
11	Adresa slave (17 = 11 hex)
06	Kód funkce (zapiš jeden registr)
0001	Adresa registru. (40002 - 40001 = 1)
0003	Zapsaná hodnota registru
9A9B	CRC (cyclic redundancy check).

2.4.7 15 (0Fh) Zapiš více cívek (Write Multiple Coils)

Tato funkce umožňuje zápis až 1968 cívek do stavu ON nebo OFF. V požadavku je udána adresa první cívky, které se mají nastavit a požadované hodnoty cívek. Bezchybná odpověď obsahuje počáteční adresu a počet nastavených cívek.

Požadavek

Kód funkce	1 byte	0x0F
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet cívek	2 byty	1 až 1968 (0x7B0)
Počet bytů	1 byte	N
Hodnoty cívek	N bytů	

$N = \text{počet výstupů} / 8$, je-li zbytek po dělení nenulový, $N = N + 1$

Odpověď

Kód funkce	1 byte	0x0F
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet cívek	2 byty	1 až 1968 (0x7B0)

Chyba

Kód funkce	1 byte	0x8F
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 0F 0013 000A 02 CD01 BF0B
11	Adresa slave (17 = 11 hex)
0F	Kód funkce (zapiš více cívek, 15 = 0F hex)
0013	Adresa (číslo) první cívky, která se má zapsat. (cívka 20 - 1 = 19 = 13 hex)
000A	Počet cívek k zápisu (10 = 0A hex)
02	Počet bytů cívek (10 cívek / 8 bitů = 2 byty)
CD	Cívky 27 - 20 (1100 1101)
01	6 doplňujících nul & Cívky 29 - 28 (0000 0001)
BF0B	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	11 0F 0013 000A 2699
11	Adresa slave (17 = 11 hex)
0F	Kód funkce (zapiš více cívek, 15 = 0F hex)
0013	Adresa (číslo) první zapsané cívky. (cívka 20 - 1 = 19 = 13 hex)
000A	Počet zapsaných cívek (10 = 0A hex)
2699	CRC (cyclic redundancy check).

2.4.8 16 (10h) Zapiš více registrů (Write Multiple Registers)

Tato funkce umožňuje zápis až 120 registrů. V požadavku je specifikována adresa prvního registru, který se má zapsat, počet registrů a požadované hodnoty registrů. Bezchybná odpověď obsahuje počáteční adresu a počet zapsaných registrů.

Požadavek

Kód funkce	1 byte	0x10
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet registrů	2 byty	1 až 120 (0x78)
Počet bytů	1 byte	2*N
Hodnoty registrů	2*N bytů	

N = počet registrů

Odpověď

Kód funkce	1 byte	0x10
Počáteční adresa	2 byty	0x0000 až 0xFFFF
Počet registrů	2 byty	1 až 120 (0x78)

Chyba

Kód funkce	1 byte	0x90
Chybový kód	1 byte	01, 02, 03 nebo 04

Příklad: [5]

Požadavek	11 10 0001 0002 04 000A 0102 C6F0
11	Adresa slave (17 = 11 hex)
10	Kód funkce (zapiš více registrů 16 = 10 hex)
0001	Adresa prvního registru, který se má zapsat. (40002 - 40001 = 1)
0002	Počet registrů k zápisu
04	Počet bytů registrů (2 registry x 2 byty každý = 4 byty)
000A	Hodnota registru 40002
0102	Hodnota registru 40003
C6F0	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	11 10 0001 0002 1298
11	Adresa slave (17 = 11 hex)
10	Kód funkce (zapiš více registrů 16 = 10 hex)
0001	Adresa prvního zapsaného registru. (40002 - 40001 = 1)
0002	Počet zapsaných bytů.
1298	CRC (cyclic redundancy check).

2.5 Chybové odpovědi

Klient očekává od serveru odpověď na požadavek, který zaslal. Mohou nastat čtyři situace:

- Pokud server bezchybně přijme a zpracuje požadavek, vrátí klientovy normální (bezchybnou) odpověď.
- Pokud server požadavek nepřijme z důvodu komunikační chyby, nevrací žádnou odpověď. Na straně klienta dojde k vypršení časového limitu pro příjem odpovědi.
- Pokud server požadavek bezchybně přijme, ale zjistí chybu při komunikaci (parita, CRC...), žádnou odpověď nevrací. Na straně klienta je tato situace vyřešena vypršením časového limitu pro příjem odpovědi.
- Pokud server požadavek bezchybně přijme, ale nastane chyba při zpracování, vrátí klientovi chybovou odpověď s udáním důvodu neúspěchu.

Normální a chybová odpověď odlišuje nejvyšší bit kódu funkce. Při normální odpovědi je bit nulový a při chybové odpovědi je bit nastavený. Chybová odpověď pak v datové části obsahuje kód chyby. V následující tabulce je seznam možných chybových kódů. [1] [3]

Tab. 7. MODBUS chybové kódy

MODBUS chybové kódy		
Kód	Jméno	Význam
01	Ilegální funkce	Neplatný kód funkce (není podporována serverem).
02	Ilegální adresa dat	Neplatná adresa (je mimo serverem podporovaný rozsah).
03	Ilegální hodnota dat	Předávaná data jsou neplatná.
04	Selhání zařízení	Při provádění požadavku nastala neodstranitelná chyba.
05	Potvrzení	Kód určený k použití při programování. Server (nebo slave) přijme a zpracuje požadavek v pořádku, ale jeho vykonání bude trvat delší dobu.
06	Zařízení je zaneprázdněné	Kód určený k použití při programování. Server (nebo slave) je zaneprázdněn vykonáváním dlouho trvajícího příkazu.
08	Chyba parity paměti	Kód určený k použití při práci se soubory. Server (nebo slave) při pokusu přečíst soubor detekoval chybu parity.
0A	Brána – přenosová cesta nedostupná	Kód určený k práci s bránou (gateway). Brána není schopná přidělit interní přenosovou cestu od vstupního portu k výstupnímu. Obvykle to znamená, že je přetížená nebo nesprávně nastavená.
0B	Brána – cílové zařízení neodpovídá	Kód určený k práci s bránou (gateway). Cílové zařízení neodpovídá, obvykle to znamená, že zařízení není přítomno.

Příklad: [5]

Požadavek	0A 01 04A1 0001 AC63
0A	Adresa slave (10 = 0A hex)
01	Kód funkce (čti cívky)
04A1	Adresa první cívky, která se bude číst. (cívky 1186 - 1 = 1185 = 04A1 hex)
0001	Počet požadovaných cívek.
AC63	CRC (cyclic redundancy check) pro kontrolu chyb.

Odpověď	0A 81 02 B053
0A	Adresa slave (10 = 0A hex)
81	Kód funkce (čti cívky) s nastaveným nejvyšším bitem
02	Chybový kód
B053	CRC (cyclic redundancy check).

2.6 Implementace protokolu MODBUS na sériové lince

MODBUS je protokol založený na architektuře Master/Slave a je definován na 2. vrstvě modelu OSI. Na fyzické úrovni modelu OSI mohou být použity různá sériová rozhraní a sběrnice, např. RS232 nebo RS485/RS422 aj.

Princip protokolu

V jeden okamžik může být na sběrnici připojen pouze jeden master, ale jedno nebo více (maximálně 247) slave zařízení. Komunikaci zahajuje vždy master, teprve poté reaguje příslušné slave zařízení. Komunikace vždy probíhá mezi masterem a slavem, nikdy mezi dvěma zařízeními slave.

Master může posílat požadavky slave zařízením ve dvou režimech:

- **unicast režim** – master posílá požadavek jednomu konkrétnímu slave zařízení, které pošle odpověď
- **broadcast režim** – master posílá požadavek všem slave zařízením, žádné zařízení neodpovídá.

Adresovací pravidla

Tab. 8. Adresní prostor složený z 256 různých adres

0	1 až 247	248 až 255
Broadcast adresa	Adresy slave zařízení	Rezervováno

Master nemá žádnou adresu, pouze slave zařízení musí mít svou adresu unikátní v celé sběrnici.

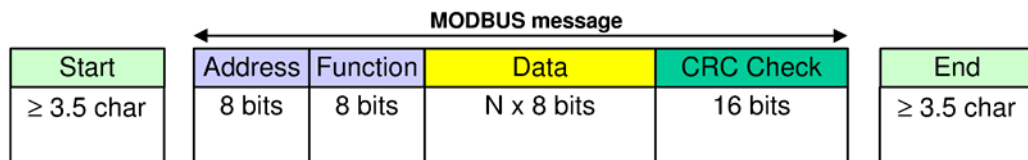
Vysílací režimy

MODBUS protokol definuje dva sériové vysílací režimy, MODBUS RTU a MODBUS ASCII. V obou režimech se liší formát vysílaných dat a jejich dekódování. MODBUS

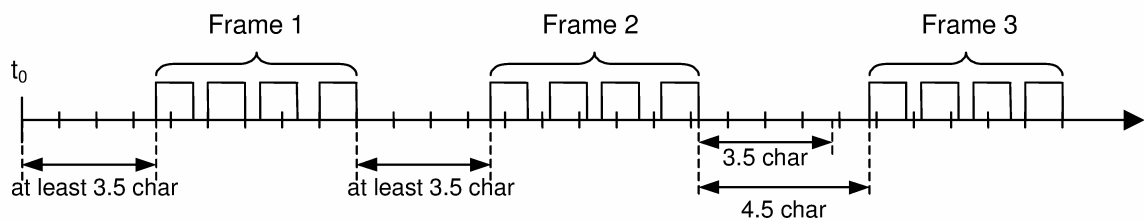
vyžaduje podporu režimu RTU, režim ASCII není povinný. Všechny zařízení na sběrnici musí pracovat ve stejném vysílacím režimu.

2.6.1 MODBUS RTU

V tomto režimu se každý datový byte (8 bitů) zprávy skládá ze dvou 4-bitových hexadecimálních znaků. Hlavní výhodou tohoto režimu je větší hustota posílaných znaků a tedy lepší propustnost při stejné přenosové rychlosti. Vysílání zprávy musí být souvislé, mezery mezi znaky nesmějí být delší než 1.5 znaku. Začátek a konec zprávy je určen mezerou (pomlčkou) na sběrnici delší než 3.5 znaku.



Obr. 13. Rámec MODBUS zprávy v režimu RTU



Obr. 14. Časování zpráv v RTU režimu

K detekci chyb slouží 16-bitový CRC (Cyclic Redundancy Check) kód s generujícím polynomem $x^{16} + x^{15} + x^2 + 1$.

Formát přenosového „bytu“ (11 bitů):

- 1 start bit
- 8 datových bitů (nejméně významný bit je poslán nejdříve)
- 1 bit parita
- 1 stop bit

Vyžaduje se podpora sudé parity. Může být ale použita i lichá nebo žádná parita. Pokud není použita parita, je nahrazena druhým stop bitem.

2.6.2 MODBUS ASCII

V tomto režimu je každý datový byte (8 bitů) posílán jako dvojice ASCII znaků. Je tedy sice pomalejší než režim RTU, ale vysílání nemusí být souvislé, mezera mezi znaky může být až 1 s. Začátek a konec zprávy je totiž určen jiným způsobem než u RTU módu. Začátek zprávy je určen znakem „:“ a konec zprávy dvojicí řídicích znaků CR (Carriage Return), LF (Line Feed).

Start	Address	Function	Data	LRC	End
1 char :	2 chars	2 chars	0 up to 2x252 char(s)	2 chars	2 chars CR,LF

Obr. 15. Rámec MODBUS zprávy v režimu ASCII

K detekci chyb slouží 8 bitový LRC (Longitudinal redundancy check) kód s generujícím polynomem $x^8 + 1$

Formát přenosového „bytu“ (10 bitů):

- 1 start bit
- 7 datových bitů
- 1 bit parita
- 1 stop bit

Vyžaduje se podpora sudé parity. Může být ale použita i lichá nebo žádná parita. Pokud není použita parita, je nahrazena druhým stop bitem. [2] [3]

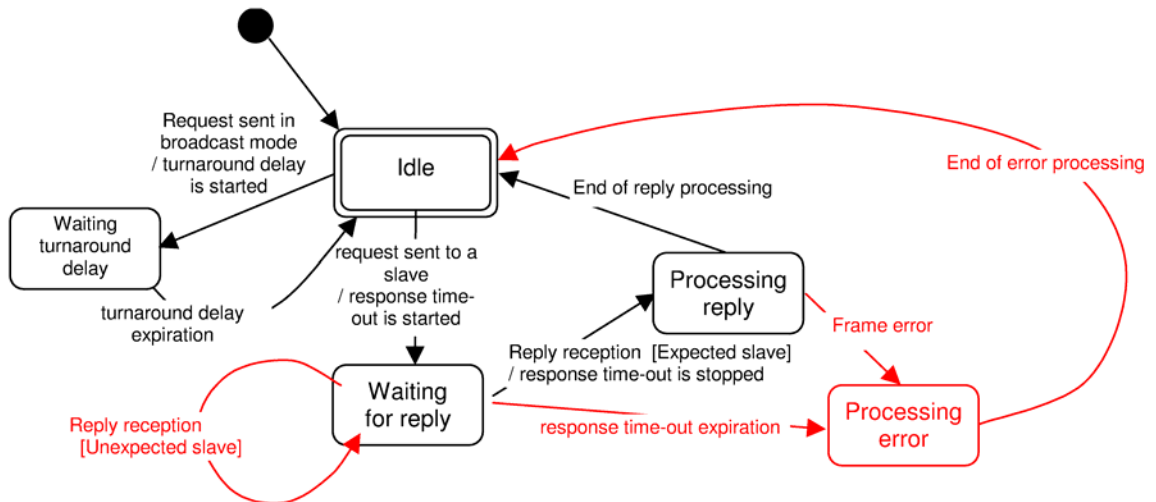
2.6.3 Master/Slave stavové diagramy

Linková vrstva MODBUSu se dělí na 2 podvrstvy:

- Master/slave protokol
- Přenosový režim (RTU a ASCII režim)

Stavový diagram pro master

Následující diagram popisuje chování masteru.



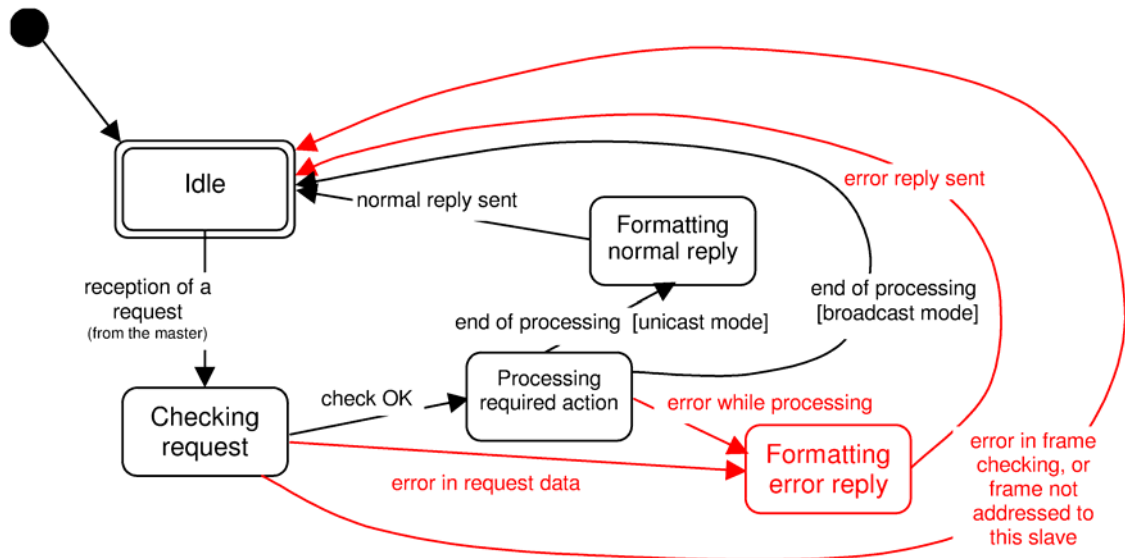
Obr. 16. Stavový diagram pro master

Popis jednotlivých stavů:

- **Idle** (Nečinnost) – výchozí stav po zapnutí zařízení, pouze v tomto stavu lze posílat požadavek, tím master tento stav opouští a už nelze zasílat další požadavek.
- **Waiting for reply** (Čekání na odpověď) – v tomto stavu čeká master pouze omezenou dobu (timeout) na odpověď slave zařízení, aby nedošlo k uvíznutí aplikace v neurčitěm bodě.
- **Processing reply** (Zpracování odpovědi) – master zkontroluje bezchybnost odpovědi, chyba může být indikována např. při přijetí odpovědi z nesprávného slave zařízení (v tom případě timeout stále běží) nebo přímo v přijatém rámci (v tom případě může být poslání odpovědi zopakováno).
- **Processing error** (Zpracování chyby) – nastává při neobdržení odpovědi v daném timeoutu, master přechází do stavu *Idle* a tím umožňuje zopakování požadavku.
- **Waiting turnaround delay** (Čekání na zpracování požadavku) – nastává po broadcast požadavku, slave zařízení na tento požadavek neodpovídají, přesto nastává toto zpoždění, aby jednotlivá slave zařízení mohla zpracovat požadavek předtím než master pošle nový.

Stavový diagram pro slave

Následující diagram popisuje chování slave.



Obr. 17. Stavový diagram pro slave

Popis jednotlivých stavů:

- **Idle** (Nečinnost) – výchozí stav po zapnutí zařízení.
- **Checking request** (Kontrola požadavku) – po přijetí požadavku slave kontroluje možné chyby: nesprávná struktura žádosti, neplatná operace a další. V případě chyby v PDU paketu musí slave poslat masteru odpověď s informací o chybě, v případě chyby v ADU rámci (nesprávná adresa slave, chybný CRC) se odpověď neposílá.
- **Formatting normal reply** (Formátování bezchybné odpovědi) – slave musí poslat odpověď masteru při bezchybné žádosti.
- **Formatting error reply** (Formátování chybné odpovědi) – pokud slave detekuje chybu v PDU paketu, posílá chybovou odpověď. [2]

2.6.4 CRC kód

V režimu RTU je na konci každé zprávy zahrnuto pole s CRC (Cyclic Redundancy Check) pro kontrolu chyb ve zprávě. CRC je 16-bitová hodnota implementovaná jako dva 8-bitové znaky tak, že nejprve je ke zprávě umístěn méně významný byte (LSB) a za ním více významný byte (MSB) CRC.

CRC spočítá vždy odesílající zařízení a pak jej připojí na konec zprávy. Přijímací zařízení přepočítá CRC přijaté zprávy a porovná s přijatým CRC, pokud nejsou shodné, nastala chyba.

Výpočet CRC začíná načtením hodnoty FFFFh (samé jedničky) do 16-bitového registru (CRC registr). Pak se postupně procházejí všechny 8-bitové znaky, pro generování CRC je použito vždy jen 8 bitů každého znaku. Start, stop a paritní bity se pro generování CRC nepoužívají.

Během generování CRC je každý 8-bitový znak je exkluzivně sčítán (XOR) s obsahem CRC registru. Výsledek je pak posunován ve směru nejméně významného bitu (*lsb*) a doplňován nulami na pozici nejvíce významného bitu (*msb*). Pokud byl *lsb* roven 1, pak je provedena operace XOR mezi CRC registrem a generujícím polynomem, pokud byl *lsb* roven 0, není XOR operace provedena.

Takto je postupně CRC registr posunován 8 krát, po posledním posunu je načten další 8-bitový znak a opět proveden XOR s aktuálním obsahem CRC registru, dále se postupuje stejně jako u předchozího znaku. Výsledné CRC vzniká po dokončení tohoto postupu u všech znaků zprávy. Generující polynom je u 16-bitového CRC roven: $x^{16} + x^{15} + x^2 + 1$.

Postup generování CRC:

1. Načtení hodnoty FFFFh do 16-bitového registru (CRC registr).
2. XOR nižšího bytu CRC registru s prvním 8-bitovým znakem zprávy.
3. Posunout CRC registr doprava (směrem k *lsb*), zleva je *msb* naplněn 0.
4. Pokud *lsb* byl 0, pak pokračovat bodem 3 (další posun).
Pokud *lsb* byl 1, pak je proveden XOR CRC registru s generujícím polynomem (A001h).
5. Opakovat body 3 a 4 až bude provedeno 8 posunů, tím je znak zpracován.
6. Opakovat body 2 až 5 pro další znaky až do konce zprávy.
7. Konečná hodnota CRC registru odpovídá hodnotě CRC.
8. Při připojení CRC do zprávy je nutné prohodit vyšší a nižší byte CRC. [2]

Příklad:

Obsah zprávy: **110A220Bh**

byte#	Hex	Počáteční CRC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
1	11	0000000000010001	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	
		xor předchozích 2 řádků	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	
		posunout xor 1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	
		posunout xor 2	1	0	0	1	1	1	1	1	1	1	1	1	0	1	0	0	
		posunout xor 3	0	1	0	0	1	1	1	1	1	1	1	1	1	0	1	0	
		posunout xor 4	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	
		posunout xor 5	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	0	
		posunout xor 6	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	
		posunout xor 7	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	
		posunout xor 8	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	
																		7F4C	
2	0A	0000000000001010	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	
		xor předchozích 2 řádků	0	1	0	0	1	1	0	0	0	1	1	1	0	1	0	1	1
		posunout xor 1	1	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1	
		posunout xor 2	1	1	1	0	0	0	1	1	0	0	0	1	1	1	0	0	
		posunout xor 3	0	1	1	1	0	0	0	1	1	0	0	0	1	1	1	0	
		posunout xor 4	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1	1	
		posunout xor 5	1	0	1	1	1	1	0	0	0	1	1	0	0	0	1	0	
		posunout xor 6	0	1	0	1	1	1	1	0	0	0	1	1	0	0	0	1	
		posunout xor 7	1	0	0	0	1	1	1	1	0	0	0	1	1	0	0	1	
		posunout xor 8	1	1	1	0	0	1	1	1	1	0	0	0	1	1	0	1	
																		8DE7	
3	22	0000000000100010	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
		xor předchozích 2 řádků	1	1	1	0	0	1	1	1	1	0	1	0	1	1	1	1	1
		posunout xor 1	1	1	0	1	0	0	1	1	1	1	0	1	0	1	1	0	
		posunout xor 2	0	1	1	0	1	0	0	1	1	1	1	0	1	0	1	1	
		posunout xor 3	1	0	0	1	0	1	0	0	1	1	1	1	0	1	0	0	
		posunout xor 4	0	1	0	0	1	0	1	0	0	1	1	1	1	0	1	0	
		posunout xor 5	0	0	1	0	0	1	0	1	0	0	1	1	1	1	0	1	
		posunout xor 6	1	0	1	1	0	0	1	0	1	0	0	1	1	1	1	1	
		posunout xor 7	1	1	1	1	1	0	0	1	0	1	0	0	1	1	1	0	
		posunout xor 8	0	1	1	1	1	1	0	0	1	0	1	0	0	1	1	1	
																		A77C	
4	0B	0000000000001011	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1
		xor předchozích 2 řádků	0	1	1	1	1	1	0	0	1	0	1	0	1	1	0	0	0
		posunout xor 1	0	0	1	1	1	1	1	0	0	1	0	1	0	1	1	0	
		posunout xor 2	0	0	0	1	1	1	1	1	0	0	1	0	1	0	1	1	
		posunout xor 3	1	0	1	0	1	1	1	1	1	0	0	1	0	1	0	0	
		posunout xor 4	0	1	0	1	0	1	1	1	1	1	0	0	1	0	1	0	
		posunout xor 5	0	0	1	0	1	0	1	1	1	1	1	0	0	1	0	1	
		posunout xor 6	1	0	1	1	0	1	0	1	1	1	1	1	0	0	1	1	
		posunout xor 7	1	1	1	1	1	0	1	0	1	1	1	1	1	0	0	0	
		posunout xor 8	0	1	1	1	1	1	0	1	0	1	1	1	1	1	0	0	
																		7C7D	

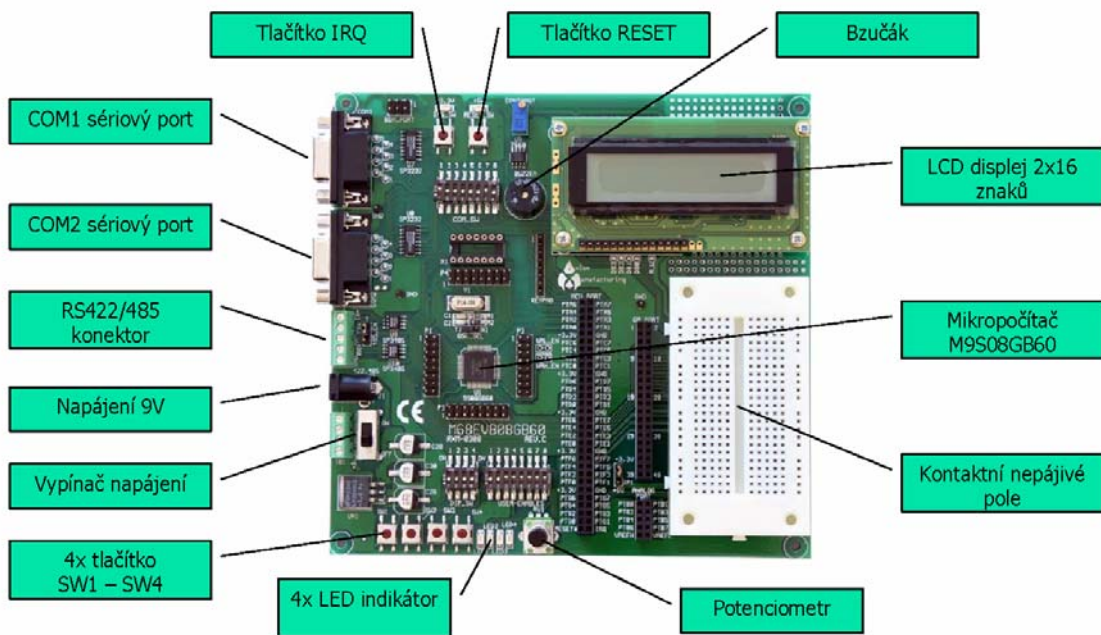
CRC: **7C7Dh**

Výsledná zpráva: **110A220B 7D7Ch**

[5] [16]

3 VÝVOJOVÝ KIT M68EVB908GB60

Vývojový kit Freescale M68EVB908GB60 je určen především pro výukové účely, umožňuje efektivní vývoj a ladění aplikací. Obsahuje mikropočítač HCS08 a k němu připojené periferie (LCD displej, tlačítka, LED diody atd.).



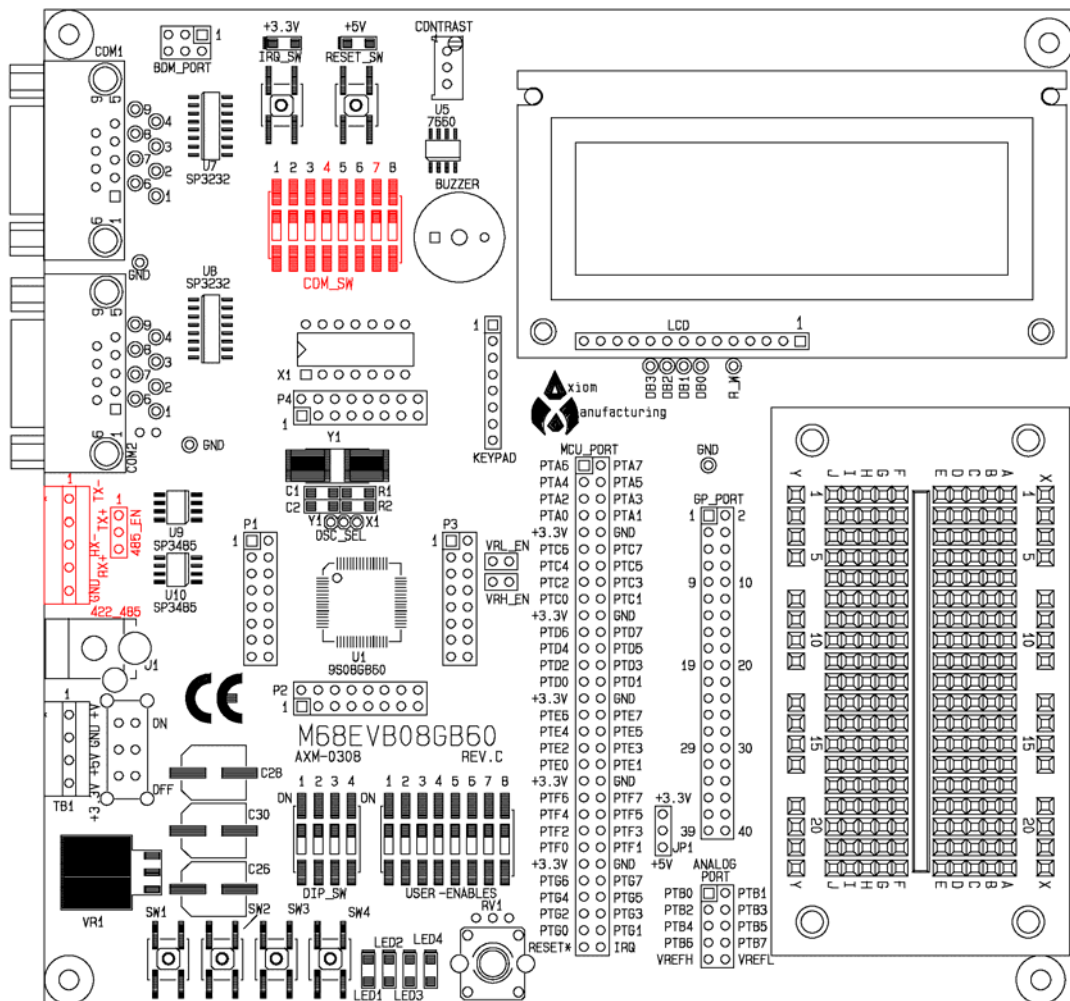
Obr. 18. Vývojový kit M68EVB908GB60 [15]

3.1 Základní vlastnosti

- Mikropočítač M9S08GB60
 - 8 bitová centrální procesní jednotka HCS08, taktovací frekvence max. 40MHz
 - 60KB FLASH paměti
 - 4KB RAM paměti
 - 56 vstupně/výstupních linek na 7 portech (porty A-G)
 - 5 kanálový TPM2 časovač
 - 3 kanálový TPM1 časovač
 - 1x Synchronní sériové periferní rozhraní (SPI)

- 1x I2C rozhraní
- 2x Asynchronní sériové komunikační rozhraní (SCI)
- Interní generátor hodinového kmitočtu s FLL obvodem (32KHz – 20MHz frekvence sběrnice)
- 8 kanálový, 10 bitový A/D převodník
- COP watchdog systém s nastavitelnou časovou prodlevou na 218 nebo 213 cyklů sběrnice
- Systém kontroly napájecího napětí umožňující detekovat pokles napětí pod stanovenou mez
- BDM rozhraní pro pokročilé ladění a programování aplikací přímo v aplikaci
- 32KHz nebo 4MHz krystalový oscilátor
- Nastavitelný oscilátor
- Stabilizovaný zdroj 3,3V a 5V
- Sériový port COM1 (rozhraní RS232)
- Sériový port COM2 (rozhraní RS232 nebo RS422/485)
- Vypínač ON/OFF se signalizací
- Uživatelské periferie
 - LED diody (port PTF0-PTF3)
 - DIP přepínač (port PTB4-PTB7)
 - 4 tlačítka (port PTA4-PTA7)
 - Dvouřádkový 16 znakový LCD displej (porty PTG3-PTG7 a PTE6-PTE7)
 - Buzzer (port PTD0)
- MCU konektor, který zpřístupňuje všechny digitální I/O
- Kontaktní nepájivé pole

3.2 Konfigurace kitu pro použití RS485



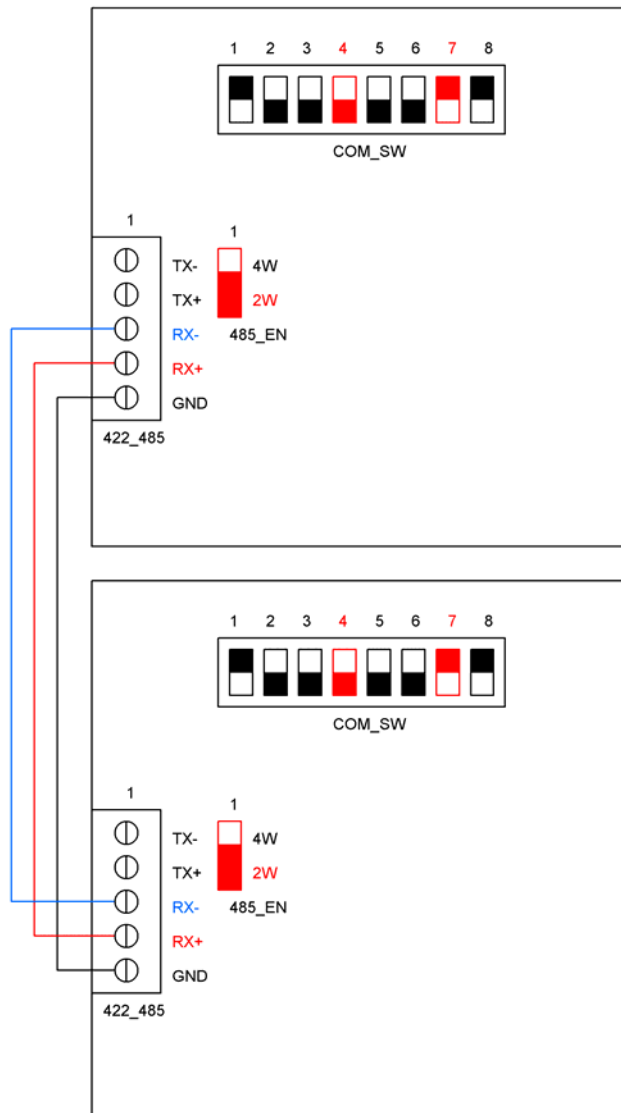
Obr. 19. Vývojový kit M68EVB908GB60 – rozmístění jednotlivých součástí

Tab. 9. Konektor 422_485

1	4W TX+	
2	4W TX-	
3	4W RX-	2W TX- / RX-
4	4W RX+	2W TX+ / RX+
5	GND	GND

Pro standardní dvou vodičovou variantu (2W) RS485 je nutné pomocí propojky spojit piny 1 a 2 na **485_EN**. Pro čtyřvodičovou variantu (4W) RS485 spojíme piny 2 a 3. U

dvouvodičové varianty je nezbytné nastavit směr komunikace. Ten se nastaví pomocí pinu **PTF7** (PTF7 = 1 – vysílací režim, PTF7 = 0 – přijímací režim). Přepínač 7 na **COM_SW** přepneme do stavu ON (povoluje 422_485 RXD IN), v tomto případě nesmí být přepínač 4 ve stavu ON (povoluje COM2 RXD IN), protože signál RXD IN je sdílen rozhraním SCI2 a RS485/RS422 a nemohou tedy pracovat současně. [4]



Obr. 20. Nastavení a propojení kitů pro podporu sběrnice RS485

3.3 Popis sériového rozhraní mikropočítače MC9S08GB60

Rozhraní SCI je sériový asynchronní komunikační systém typu UART, který umožňuje komunikovat na s dalšími mikropočítači nebo jinými systémy vybavenými rozhraním RS-232 (PC, průmyslové počítače).

Vlastnosti SCI u HCS08:

- Plně duplexní provoz, standardní NRZ formát
- Nastavitelná přenosová rychlost (13-bitová dělička)
- Řízení rozhraní pomocí přerušení nebo pomocí dotazování se na dokončení operace čtením příznaků (polling):
 - Vysílací datový registr – jeho vyprázdnění značí odeslání znaku
 - Přijímací datový registr – jeho naplnění značí příjem znaku
 - Detekce chyb (šum na lince, chyba parity, chybný datový rámec)
 - Detekce nečinnosti přijímače
- Nastavitelná délka datového rámce 8 nebo 9 bitů.
- Zcela oddělený přijímač a vysílač.
- Hardwarové generování a kontrola parity.
- Dvojitá vyrovnávací paměť přijímače i vysílače.

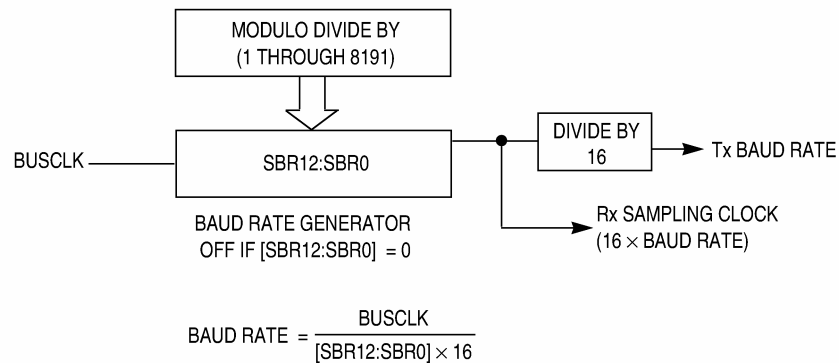
3.3.1 Formát datového rámce SCI

Datový rámec obsahuje 8 (9) datových bitů, které jsou uzavřeny mezi tzv. Start a Stop bit, sestupná hrana Start bitu synchronizuje přijímač s vysílačem. V klidovém stavu je signál v úrovni logické 1. Přenos začíná Start bitem – překlopením signálu do logické úrovně 0. Pak jsou vysílána data o délce 8 bitů (1 B) a případně parita. Přenosová rychlost je po celou dobu konstantní se udává v baudech (počet změn úrovní signálu za jednotku času).

3.3.2 Nastavení SCI

SCI má osm 8-bitových registrů pro nastavení přenosové rychlosti, nastavení SCI, hlášení stavu SCI a pro vysílání a příjem dat.

Přijímač a vysílač SCI pracují na sobě nezávisle, ačkoli používají stejný zdroj přenosové rychlosti (baud rate). Tato rychlost je odvozena od vnitřní frekvence sběrnice.



Obr. 21. Zdroj přenosové rychlosti SCI

Pro nastavení přenosové rychlosti se používají registry **SCIxBDH** a **SCIxBDL**, které nastavují hodnotu 13-bitové děličky přenosové rychlosti [SBR12:SBR0]. Po nastavení registru **SCIxBDL** na nenulovou hodnotu zbývá už jen povolit vysílač a přijímač nastavením bitů **RE** a **TE** v registru **SCIxC2** na hodnotu 1. Práce s SCI rozhraním spočívá v nastavení řídicích registrů **SCIxC1**, **SCIxC2**, čtení příznakových registrů **SCIxS1**, **SCIxS2**, **SCIxS3** pro získání informací o stavu SCI a zápisem datového registru **SCIxD** při vysílání nebo jeho čtením při příjmu dat. [9]

Registr **SCIxC1** – základní nastavení SCI

- **LOOPS** – nastavuje, zda přenos probíhá normálně (RxD1 a TxD1 odděleny) nebo zda je výstup vysílače spojen se vstupem přijímače – vnitřní smyčka.
- **SCISWAI** – umožňuje zastavit SCI, pokud je CPU v režimu „Wait“
- **RSRC** – má efekt pouze pokud **LOOPS** = 1, umožňuje propojit vnitřní smyčku s výstupem vysílače
- **M** – výběr 8-bitového nebo 9-bitového režimu (0 = 8-bitový režim)
- **WAKE** – nastavení metody „probuzení“ přijímače.

- ILT – nastavení typu detekce nečinnosti linky
- PE – povolení parity (1 = parita povolena, 0 = parita zakázána)
- PT – typ parity (0 = sudá parita, 1 = lichá parita)

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
Write:								
Reset:	0	0	0	0	0	0	0	0

Obr. 22. Registr SCIxC1

Registr SCIxC2 – základní nastavení SCI

- TIE – povolení přerušení při vysílání (1 = přerušení povoleno)
- TCIE – povolení přerušení při dokončení přenosu (1 = přerušení povoleno)
- RIE – povolení přerušení od přijímače
- ILIE – povolení přerušení při nečinnosti linky
- TE – povolení vysílače (1 = vysílač povolen)
- RE – povolení přijímače (1 = přijímač povolen)
- RWU – nastavuje povolení režimu standby
- SBK – kontrolní bit použitý k posílání ukončovacích znaků

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Write:								
Reset:	0	0	0	0	0	0	0	0


Obr. 23. Registr SCIxC2

Registr SCIxS1 – registr příznaků SCI (pouze ke čtení)

- TDRE – příznak vyprázdnění vysílacího datového registru – předání znaku do vysílacího posuvného registru (1 = přenosový datový registr je prázdný)
- TC – příznak dokončení přenosu (1 = přenos je dokončen)

- RDRF – příznak naplnění přijímacího datového registru – příjem dat (1 = vysílací datový registr je naplněn)
- IDLE – příznak nečinnosti linky (1 = linka je v nečinnosti)
- OR – příznak přetečení přijímače (1 = příjem nových dat před zpracováním předchozích dat)
- NF – příznak detekce šumu (1 = detekován šum v přijatém znaku)
- FE – příznak chyby rámce (1 = detekována chyba rámce)
- PE – příznak chyby parity (1 = detekce chyby parity)

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write:								
Reset:	1	1	0	0	0	0	0	0


 = Unimplemented or Reserved

Obr. 24. Registr SCIxS1

Registr SCIxS2 – registr příznaků SCI (pouze ke čtení)

- RAF – příznak aktivity přijímače

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	0	0	RAF
Write:								
Reset:	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved


Obr. 25. Registr SCIxS2

Registr SCIxS3 – registr příznaků SCI (pouze ke čtení)

- R8 – devátý bit pro přijímač (při 9-bitovém přenosu)
- T8 – devátý bit pro vysílač (při 9-bitovém přenosu)
- TXDIR – nastavení směru TxD1 v jednovodičovém režimu
- ORIE – povolení přerušení při přetečení (povoluje příznak OR)

- NEIE – povolení přerušení při detekci šumu
- FEIE – povolení přerušení při detekci chyby rámce
- PEIE – povolení přerušení při detekci chyby parity [9]

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	R8		TXDIR	0	ORIE	NEIE	FEIE	PEIE
Write:		T8						
Reset:	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Obr. 26. Registr SCLxS3

II. PRAKTICKÁ ČÁST

4 KNIHOVNA MODBUS_GB60

Vytvořená knihovna pro obsluhu protokolu MODBUS byla kvůli úspoře paměti vytvořena v jazyce symbolických adres. Obsahuje základní funkce pro čtení a zápis čtyř typů dat: bitové hodnoty – cívky a vstupy (pouze čtení), vstupní (pouze čtení) a uchovávací registry. Je implementován vysílací režim MODBUS RTU a naopak není v knihovně obsažena implementace režimu MODBUS ASCII, který je ale nepovinný a režimu broadcast. Knihovna také podporuje 4 základní chybové kódy.

4.1 Požadavky na mikropočítač

Základními požadavky jsou přítomnost časovače a sériového rozhraní SCI, které reaguje přerušením na příchod znaku, zařízení pak musí podporovat komunikace po sběrnici RS485. Knihovna je určena speciálně pro mikropočítač Freescale MC9S08GB60 a podobné mikropočítače z rodiny HC(S)08 se stejnou instrukční sadou. Některé tyto mikropočítače mají odlišně označené registry, porty atd.. Mimo základní registry A a H:X, je práce s hardwarovými registry soustředěna zejména do přerušení a funkcí pracujícím se sériovým rozhraním SCI2. Při migraci na jiný mikropočítač je nutné tyto názvy registrů upravit.

4.2 Datový model MODBUSu použitý v knihovně

Knihovna využívá fyzicky datový model s jedním společným blokem. Z uživatelského pohledu jednotlivé tabulky spolu však nesouvisí. Na obrázku níže je vidět příklad alokované paměti. Je alokováno 16 cívek a 16 vstupů, což jsou bitové hodnoty a tedy každá tabulka zabírá pouze 2 byty. Nežřetelné může být adresování cívek a vstupů zprava doleva, což je z toho důvodu, že hodnoty cívek s nižším číslem jsou při přenosu umístěny v nižších bitech – první cívka je umístěna v bitu 0. Z uživatelského hlediska však toto není vůbec podstatné, uživatel nemusí znát přesné umístění, pouze číslo cívky (vstupu). Registry jsou adresovány standardně zleva doprava. Jelikož jsou registry 16-bitové hodnoty, zabírá tabulka o 5 vstupních registrech 10 bytů a tabulka uchovávacích registrů 20 bytů.

16					Cívky (bity)						1
16					Vstupy (bity)						1
1	Vstupní registry (16-bitová slova)										5
1	Uchovávací registry (16-bitová slova)										10

Obr. 27. Datový model MODBUSu knihovny Modbus_gb60

4.3 Popis uživatelských funkcí

Uživatelské jsou ty funkce knihovny, které jsou uživateli přístupné a jsou odlišné na straně masteru a na straně slave. Výjimkou je funkce *sci_init*, která je společná a musí být v programu vždy použita.

char sci2_init (char)	
<i>Parametr</i>	kód přenosové rychlosti SCI2 (4800BD, 9600BD, 19200BD, 38400BD, 57600BD, 115200BD)
<i>Návratová hodnota</i>	1 – OK, 0 – Inicializace neúspěšná (nesprávně zadaná přenosová rychlost)
<i>Příklad</i>	sci2_init(9600BD)
Inicializuje sériové rozhraní SCI2 a nastavuje přenosovou rychlost. Nastavuje časovač a globálně povoluje přerušení.	

4.3.1 MODBUS_master_gb60

Obecně všechny funkce masteru posílají nějaký požadavek na slave, který vrací odpověď. Každá funkce uloží svůj požadavek do proměnné *frame*, vypočítá kontrolní kód CRC a odešle pomocí funkce *sci2_str_out*. Následně čeká na odpověď, ta obsahuje požadovaná data, příp. potvrzení o zápisu dat, v případě chyby obsahuje chybový kód, a pokud odpověď do určité doby vůbec nedorazí, dojde k vypršení timeoutu a funkce skončí

s chybovým kódem. V případě bezchybné odpovědi vrací funkce 0, neplatný kód funkce značí 1, neplatné adresování 2, chyba v datech (CRC) 3 a vypršení timeoutu 4.

char modbus_read_coils (char, int, int, char*)	
<i>Parametry</i>	adresa slave, číslo první cívk, počet cívek, adresa proměnné, kam se uloží hodnoty přečtených cívek
<i>Příklad</i>	modbus_read_coils(1,1,7,civky) (čtení cívek 1-7 ze slave s adresou 1, cívky budou uloženy do proměnné <i>civky</i> – pole bytů)
Posílá požadavek na čtení cívek (bitů).	

char modbus_read_inputs (char, int, int, char*)	
<i>Parametry</i>	adresa slave, číslo prvního vstupu, počet vstupů, adresa proměnné, kam se uloží hodnoty přečtených vstupů
<i>Příklad</i>	modbus_read_inputs(1,5,10,vstupy) (čtení vstupů 5-14 ze slave s adresou 1, vstupy budou uloženy do proměnné <i>vstupy</i> – pole bytů)
Posílá požadavek na čtení diskretních vstupů (bitů).	

char modbus_read_hold_regs (char, int, int, char*)	
<i>Parametry</i>	adresa slave, číslo prvního registru, počet registrů, adresa proměnné, kam se uloží hodnoty přečtených registrů
<i>Příklad</i>	modbus_read_hold_regs(1,2,5,registry) (čtení registrů 2-6 ze slave s adresou 1, registry budou uloženy do proměnné <i>registry</i> – pole bytů)
Posílá požadavek na čtení uchovávacích registrů (16-bitových).	

char modbus_read_in_regs (char, int, int, char*)	
<i>Parametry</i>	adresa slave, číslo prvního registru, počet registrů, adresa proměnné, kam se uloží hodnoty přečtených registrů
<i>Příklad</i>	modbus_read_in_regs(1, 3, 1, registry) (čtení registru 3 ze slave s adresou 1, registr bude uložen do proměnné <i>registry</i>)
Posílá požadavek na čtení vstupních registrů (16-bitových).	

char modbus_write_coil (char, int, char)	
<i>Parametry</i>	adresa slave, číslo cívky, požadovaná hodnota cívky
<i>Příklad</i>	modbus_write_coil(1, 1, 0) (zápis hodnoty 0 do cívky 1 ve slave s adresou 1)
Posílá požadavek na zápis jedné cívky.	

char modbus_write_hold_reg (char, int, int)	
<i>Parametry</i>	adresa slave, číslo registru, požadovaná hodnota registru
<i>Příklad</i>	modbus_write_hold_reg(1, 1, 0xFFFF) (zápis hodnoty FFFFh do uchovacího registru 1 ve slave s adresou 1)
Posílá požadavek na zápis jednoho uchovacího registru.	

char modbus_write_coils (char, int, int, char*)	
<i>Parametry</i>	adresa slave, číslo první cívky, počet cívek, požadované hodnoty cívek
<i>Příklad</i>	modbus_write_coils(1, 1, 15, cívky) (zápis cívek 1-15 hodnotami z proměnné <i>cívky</i> ve slave s adresou 1)
Posílá požadavek na zápis více cívek.	

<code>char modbus_write_hold_regs(char, int, int, char*)</code>	
<i>Parametry</i>	adresa slave, číslo registru, požadovaná hodnota registru
<i>Příklad</i>	<code>modbus_write_hold_regs(1,3,2,regs)</code> (zápis registrů 3-4 hodnotami z proměnné <i>regs</i> ve slave s adresou 1)
Posílá požadavek na zápis jednoho uchovávacího registru.	

4.3.2 MODBUS_slave_gb60

Uživatelské funkce slave slouží k práci s paměťovým prostorem (proměnná *data*). Master funkce čtou a zapisují data z/do paměti automaticky bez vědomí uživatele, proto musí existovat funkce, které uložené data přečtou anebo zapíší. Matoucí mohou být funkce, které zapisují vstupy nebo vstupní registry. Je nutné si uvědomit, že tyto funkce nezapisují vstupy, ale obrazy vstupů a ukládají je do paměti, tak aby s nimi mohl master pracovat. Tyto funkce pracují obdobným způsobem, jako funkce pro práci s daty popsané v kapitole 4.7.1 pouze jiným způsobem předávají parametry, proto zde následuje pouze jednoduchý uživatelský popis funkcí.

<code>void modbus_slave_init(char, int, int, char, char)</code>	
<i>Parametr</i>	adresa slave, počet cívek, počet vstupů, počet vstupních a uchovávacích registrů
<i>Návratová hodnota</i>	1 – OK, 0 – Inicializace neúspěšná (neplatná alokace paměti)
<i>Příklad</i>	<code>modbus_slave_init(1,32,64,5,10)</code> (adresa slave = 1, alokuje 32 cívek, 64 vstupů, 5 vstupních a 10 uchovávacích registrů)
Alokuje paměťový prostor pro cívky, vstupy, vstupní a uchovávací registry. Kontroluje, zda není překročen maximální počet cívek, vstupů nebo registrů, ale i to, jestli celkový počet cívek, vstupů a registrů nepřekračuje velikost paměťového prostoru.	

char read_coil (int, char*)	
<i>Parametry</i>	číslo cívky, proměnná, do které se hodnota cívky uloží
<i>Příklad</i>	read_coil(1, &civka) (čtení cívky 1 a její uložení do proměnné <i>civka</i>)
Načte hodnotu cívky z paměti a uloží do proměnné.	

char read_coils (int, char, char*)	
<i>Parametry</i>	číslo první cívky, počet cívek, proměnná, do které se hodnoty cívek uloží
<i>Příklad</i>	read_coils(1, 3, &civky) (čtení cívek 1-3 a jejich uložení do proměnné <i>civky</i>)
Načte hodnotu cívek z paměti a uloží do proměnné.	

char read_input (int, char*)	
<i>Parametry</i>	číslo vstupu, proměnná, do které se hodnota vstupu uloží
<i>Příklad</i>	read_input(2, &vstup) (čtení vstupu 2 a její uložení do proměnné <i>vstup</i>)
Načte hodnotu vstupu z paměti a uloží do proměnné.	

char read_inputs (int, char, char*)	
<i>Parametry</i>	číslo prvního vstupu, počet vstupů, proměnná, do které se hodnoty vstupů uloží
<i>Příklad</i>	read_inputs(2, 3, &vstupy) (čtení vstupů 2-4 a jejich uložení do proměnné <i>vstupy</i>)
Načte hodnotu vstupů z paměti a uloží do proměnné.	

char write_img_input (int, char)	
<i>Parametry</i>	číslo vstupu, požadovaná hodnota vstupu
<i>Příklad</i>	write_img_input(5,1) (zápis hodnoty 1 na vstup 5)
Zapiše obraz vstupu do paměti.	

char write_img_inputs (int, char, char)	
<i>Parametry</i>	číslo prvního vstupu, počet vstupů, požadované hodnoty vstupů
<i>Příklad</i>	write_img_inputs(1,5,0xF0) (zápis hodnot F0h na vstupy 1-5 – zapiše pouze 5 hodnot vstupů, zbytek se ignoruje)
Zapiše obraz vstupů do paměti.	

char write_coil (int, char)	
<i>Parametry</i>	číslo cívky, požadovaná hodnota cívky
<i>Příklad</i>	write_coil(20,1) (zápis hodnoty 1 do cívky 20)
Zapiše cívku do paměti.	

char write_coils (int, char, char)	
<i>Parametry</i>	číslo první cívky, počet cívek, požadované hodnoty cívek
<i>Příklad</i>	write_coils(1,8,0x10) (zápis hodnot 10h na vstupy 1-8)
Zapiše cívky do paměti.	

char write_hold_reg (char, int)	
<i>Parametry</i>	číslo registru, požadovaná hodnota registru
<i>Příklad</i>	write_hold_reg(13,0x1020) (zápis hodnoty 1020h do registru 13)
Zapiše uchovávací registr do paměti.	

char write_img_in_reg (char, int)	
<i>Parametry</i>	číslo registru, požadovaná hodnota registru
<i>Příklad</i>	write_img_in_reg(16,0x1122) (zápis hodnoty 1122h do registru 16)
Zapiše obraz vstupního registru do paměti.	

char read_hold_reg (char, int*)	
<i>Parametry</i>	číslo registru, proměnná, do které se hodnota registru uloží
<i>Příklad</i>	read_hold_reg(31,&registr) (čtení registru 31 a jeho uložení do proměnné <i>registr</i>)
Načte hodnotu uchovávacího registru z paměti a uloží do proměnné.	

char read_in_reg (char, int*)	
<i>Parametry</i>	číslo registru, proměnná, do které se hodnota registru uloží
<i>Příklad</i>	read_hold_reg(7,&registr) (čtení registru 7 a jeho uložení do proměnné <i>registr</i>)
Načte hodnotu vstupního registru z paměti a uloží do proměnné.	

4.4 Proměnné a konstanty

Následující tabulka obsahuje výčet proměnných a konstant. Konstanty jsou vždy označeny velkými písmeny. Většina proměnných a konstant je využita na straně master i slave, v opačném případě je to v tabulce uvedeno. V hlavním programu nejsou tyto proměnné

vůbec potřeba, v ukázkovém programu jsou použity proměnné *frame* pro výpis rámce a *status_reg* pro detekci příjmu nového rámce.

FRAME_SIZE	konstanta omezující maximální velikost rámce (frame)
DATA_SIZE	konstanta omezující maximální velikost alokované paměti (pouze slave)
frame	slouží pro ukládání rámce při vysílání zprávy i při přijetí odpovědi
data	slouží jako paměťový prostor pro ukládání cívek, diskrétních vstupů, vstupních a uchovávacích registrů
counter_r	čítá počet přerušení a umožňuje detekovat konec rámce při čtení (prodleva 3,5 znaku)
counter_w	čítá počet přerušení a umožňuje nastavit konec rámce při vysílání (prodleva 3,5 znaku)
timeout	čítá dobu, po kterou nepřichází odpověď (pouze master)
TIMEOUT	konstanta značící maximální dobu čekání na odpověď (pouze master)
idle35	značí, kolik přerušení odpovídá prodlevě 3,5 znaku
data_adress	slouží k uložení adresy uvnitř rámce
adress	slouží k uložení adresy uvnitř rámce – používá se pouze v přerušení pro zajištění výlučného přístupu
dev_adress	obsahuje adresu slave (pouze slave, master nemá adresu)
coils	počet alokovaných cívek (pouze slave)
coilsB	počet bytů alokovaných cívek (pouze slave)
inputs	počet alokovaných vstupů (pouze slave)
inputsB	počet bytů alokovaných vstupů (pouze slave)
in_regs	počet bytů alokovaných vstupních registrů (pouze slave)
hold_regs	počet bytů alokovaných uchovávacích registrů (pouze slave)
status_reg	příznakový registr (příznaky níže)
EFW	příznak End Frame Written – rámec byl celý zapsán (pouze master)
EFR	příznak End Frame Read – rámec byl celý přečten
WRF	příznak Writing Frame – probíhá zápis (pouze master)

REF	příznak Reading Frame – probíhá čtení
WT	příznak Waiting – čekání pokud nepřichází odpověď (pouze master)
TOUT	příznak time-out – vypršela maximální doba čekání na odpověď – TIMEOUT (pouze master)
TOIE	TOF Interrupt Enable – povolení přerušení od časovače (konstanta obsahující hodnotu bitu, který přerušení povoluje)
GEN_POLY_MSB	konstanta – generující polynom pro CRC (vyšší byte)
GEN_POLY_LSB	konstanta – generující polynom pro CRC (nižší byte)

4.5 Komunikační (SCI) funkce

Tyto funkce slouží k odesílání dat přes rozhraní SCI2.

sci2_out

Odesílá znak pomocí rozhraní SCI2. Vysílaný znak se zapíše do datového registru *SCI2D* a pak se čte příznak *TDRE* z registru *SCI2SI*. Pokud je tento příznak nastaven, znamená to, že datový registr je vyprázdněn a tedy znak úspěšně předán do vysílacího posuvného registru, v opačném případě se periodicky kontroluje stav příznaku *TDRE*.

sci2_str_out

Odesílá pole znaků pomocí rozhraní SCI2. Neposílá se řetězec zakončený znakem NULL, ale binární data, je tedy nutné předat funkci počet odesílaných znaků. Funkce *sci2_out* se v cyklu zavolá tolikrát, kolik je zadaný počet znaků. Funkce *sci2_out* nejprve zjišťuje, zda je vysílací registr prázdný a teprve poté do něj zapíše nový znak. To je korektní postup, ale potom se může stát, že poslední dva znaky nemusí být vyslány (jeden zůstane v datovém registru *SCI2D* a druhý ve vysílacím posuvném registru). Proto je zde ještě testován příznak *TC* registru *SCI2SI*, který značí úspěšné odeslání znaku.

end_frame_delay

Nastavuje pomlku na sběrnici trvající 3,5 znaku. Je volána ihned po funkci *sci2_str_out* (pouze master).

4.6 Funkce pro obsluhu přerušení

Veškerá komunikace mezi masterem a slavem je řízena pomocí přerušení od přetečení časovače a čtení znaků pomocí sériového rozhraní je řízeno přerušením při příjmu znaku na SCI2.

Přerušení při přetečení (TOF) časovače TPM1

Na straně masteru je výchozí stav čekání na vysílání požadavku. Přerušení je povoleno funkcí *end_frame_delay* a periodicky nastává přerušení od časovače – *writing* (perioda přerušení je závislá na přenosové rychlosti SCI2). Počet přerušení načítá proměnná *counter_w* a jakmile se jeho hodnota rovná hodnotě proměnné *idle35* (opět závislá na přenosové rychlosti SCI2) je rámec kompletně odeslán včetně ukončovací pomlky. Toto je implementováno pouze na straně masteru. Slave nemusí zakončovat odeslaný rámec pomlkou 3,5 znaku, protože odesílá pouze jeden rámec (odpověď) a ne souvislá data, pomlka na sběrnici tedy vznikne automaticky vždy.

Slave ve výchozím nastavení čeká na příjem rámce (master pokud čeká na odpověď). V případě příchodu znaku je vyvoláno přerušení rozhraním SCI2, znak je zapsán do rámce *frame* a je povoleno přerušení od časovače a nastaven příznak *REF* značící probíhající čtení (*reading*). Tento příznak má význam hlavně u masteru, na straně slave spíše slouží v ukázkovém programu pro výpis na displej. V přerušení od časovače je čítán počet těchto přerušení proměnnou *counter_r* a analogicky pokud dosáhne hodnoty *idle35* je rámec kompletně načten. Důvod pro to, že perioda přerušení od časovače není rovna délce odeslání 3,5 znaku je ten, že přestože je při příchodu prvního znaku přerušení od časovače ještě zakázáno, jeho čítač stále běží a v okamžiku povolení přerušení může být v čítači libovolná hodnota. Doba zbývající do přerušení je tedy velmi nejistá, snížením periody přerušení a několikanásobným zopakováním přerušení se tento problém do velké míry eliminuje.

Pokud master čeká na odpověď, ale znaky nepřichází je spuštěno čekání (*waiting*). Postupně se načítá hodnota proměnné *timeout* a pokud dosáhne hodnoty *TIMEOUT* než přijde znak, je čtení předčasně ukončeno s chybovým kódem. Toto je implementováno pouze na straně masteru.

Přerušení SCI2 od příjmu znaku

Toto přerušení zajišťuje příjem znaku a jeho uložení na správné místo v rámci *frame*, nastavuje příznak *REF* (probíhající čtení), nuluje proměnnou *counter_r* a také povoluje přerušování od časovače, který zajišťuje detekci konce rámce.

4.7 Další funkce

4.7.1 Funkce pro práci s daty

Tyto funkce jsou využity ve většině na straně slave (pokud není uvedeno jinak) a jelikož nejsou pro uživatele vůbec viditelné, bude následující popis zaměřen spíše na popis jejich funkce.

frame2data

Tato funkce slouží pro překopírování dat na straně slave mezi rámcem *frame* a paměťovým prostorem *data* a to oběma směry, přestože název naznačuje pouze zápis požadavku jedním směrem. Obecně dokončuje operace požadované masterem – čtení i zápis. V případě úspěchu operace vrátí 0, v opačném případě kód chyby 1 (neplatný kód funkce), 2 (neplatné adresování) nebo 3 (chyba dat – nesprávný CRC) a pomocí funkce *exception* se posílá chybová odpověď.

Funkce *frame2data* načte kód funkce z přijatého rámce a zavolá příslušnou funkci, která obslouží tento požadavek. Následuje popis těchto funkcí.

Společným znakem je postup při zpracování přijatého rámce, lišící se pouze povahou dat. Nejprve je zavolána funkce *CRC_check* (vyjma funkcí *wcoils* a *whold_regs*), která porovná CRC přijatého rámce (zprávy) s vypočítaným CRC rámce. Nerovnost těchto hodnot indikuje chybu uvnitř rámce a funkce končí s chybovým kódem 3. V opačném případě si na zásobník uloží potřebné údaje (číslo cívky, počet registrů atd.) a zavolá funkci, která se postará o zápis (načtení) dat (cívek, vstupů nebo registrů) do (z) proměnné *data*. Následuje výpočet a zápis CRC do rámce (funkce *add_CRC*) a odeslání rámce (odpovědi) masteru pomocí funkce *sci2_str_out*. V případě úspěchu funkce vrátí 0, jinak vrátí chybový kód 2 (neplatné adresování) nebo již zmíněný kód chyby 3. Vzhledem

k podobnosti funkcí budou zmíněny především odlišnosti dané růzností dat (bitové a šestnáctibitové hodnoty).

rcoils

Načten požadavek na čtení cívek (bitových hodnot). Uloží na zásobník číslo první cívky, počet cívek, počet bytů cívek vypočítaný pomocí funkce *bits2bytes* a adresu uvnitř rámce, kam se uloží cívky, které master požaduje. Zavolá se funkce *r_coils* (popis dále), která načte cívky z proměnné *data* a uloží na příslušné místo do rámce *frame*. Tato funkce může vracet chybový kód 2 (neplatné adresování cívek), v tom případě funkce *rcoils* skončí s tímto chybovým kódem, jinak je vypočítán a uložen do rámce CRC kód a celý rámeček je odeslán masteru.

rinputs

Načten požadavek na čtení vstupů. Funkce je analogická k *rcoils* s tím rozdílem, že *rinputs* načítá vstupy a místo funkce *r_coils* se analogicky volá *r_inputs*.

rhold_regs

Načten požadavek na čtení uchovávacích registrů (16-bitových slov). Na zásobník se ukládá číslo prvního registru, počet registrů a adresa uvnitř rámce, kam se registry uloží. Samotné načtení registrů z *data* a jejich zápis do *frame* provádí funkce *r_hold_regs*.

rin_regs

Načten požadavek na čtení vstupních registrů. Analogie k *rhold_regs* pro čtení registrů je funkce *r_in_regs*.

wcoil

Načten požadavek na zápis jedné cívky. Rozdíl oproti funkci čtení cívek je uložení požadované hodnoty cívky do *data* pomocí funkce *wcoil*.

whold_reg

Načten požadavek na zápis jednoho registru. Zápis registru do *data* provádí funkce *w_hold_regs*.

wcoils

Načten požadavek na zápis více cívek. Zde je odlišností nevyužití kontrolní funkce *CRC_check*, ale je kontrolován CRC přímo ve funkci *wcoils*. Důvodem je to, že délka požadavku na čtení cívek není předem známá, proto je nejprve potřeba ve smyčce *loop1* posunout adresu uvnitř rámce o počet bytů cívek + 1 až na adresu, kde se nachází CRC.

Nyní je CRC zkontrolován obdobným způsobem jako u funkce *CRC_check* (popis dále). Cívky jsou zapsány funkcí *w_coils*.

w_hold_regs

Načten požadavek na zápis více uchovávacích registrů. Zde se řeší stejný problém jako u *w_coils*. Registry jsou do *data* zapsány pomocí funkce *w_hold_regs*.

Další popis bude zaměřen na funkce, které s předchozími velmi úzce souvisí, již byly výše zmiňovány a slouží přímo ke čtení nebo zápisu dat. Společným rysem těchto funkcí je kontrola adresování, tedy není-li např. zadaná cívka mimo rozsah, v takovém případě je vrácen kód chyby 2, v případě úspěchu je vrácena 0.

w_coil

Zapíše cívku do *data*. Nejprve zkontroluje, zda číslo zadané cívky není mimo povolený rozsah cívek, poté je číslo cívky poděleno 8 pomocí funkce *bits2bytes*, výsledkem je číslo bytu ve kterém se cívka nachází a konkrétní pozice cívky je udána zbytkem po dělení. Dále je načtena adresa bytu, kam se bude zapisovat hodnota cívky pomocí funkce *get_adress1* a cívka (bit) je zapsána funkcí *set_bit*.

w_coils

Zapíše více cívek do *data*. Na rozdíl od *w_coil* je zde nutné zkontrolovat, zda není mimo rozsah i poslední adresovaná cívka. Pokud je číslo první cívky dělitelné 8 a počet cívek je alespoň 8, probíhá zápis cívek po celých bytech pomocí funkce *write_bytes*, což je mnohem rychlejší, jinak probíhá zápis v cyklu po jednotlivých cívkách jako u *w_coil*.

r_coils

Čte jednu nebo více cívek z *data*. Princip je obdobný jako u *w_coils* s tím rozdílem, že ke čtení cívky je použita funkce *get_bit*. Pokud je možné čtení celých bytů cívek, je využita funkce *read_cbytes*.

r_inputs

Čte jeden nebo více vstupů z *data*. Princip opět stejný jako u *r_coils*. Čtení celých bytů může probíhat pomocí funkce *read_ibytes*.

w_hold_reg

Zapíše jeden uchovávací registr do *data*. K adresování registrů se používá funkce *get_adress16*. K posunu na tabulku (uchovávacích) registrů slouží funkce *table_offset*.

w_hold_regs

Zapisuje jeden nebo více uchovávacích registrů do *data*. Zapisování probíhá po jednotlivých registrech jakou předchozí funkce postupně v cyklu.

r_hold_regs

Čte jeden nebo více uchovávacích registrů z *data*. Registry se čtou z *data* a zapisují do rámce *frame*.

r_in_regs

Čte jeden nebo více vstupních registrů z *data* obdobně jako předchozí funkce.

4.7.2 Pomocné funkce

Poslední kategorií jsou pomocné funkce, které slouží k adresování, čtení a zápisu bitů apod.

set_bit

Zapisuje jeden bit (cívku). Pomocí funkce *mask_bit* se vytvoří maska příslušného bitu. V závislosti na požadované hodnotě cívky je příslušný byte maskován funkcí OR (zápis stavu ON) nebo AND (zápis stavu OFF).

get_bit

Čte jeden bit (cívku nebo vstup). Stav bitu se načte pouze maskováním funkcí AND.

get_adress16

Adresuje registry (16-bitové hodnoty). Sčítá dvě 16-bitové hodnoty a jednu 8-bitovou hodnotu – adresa počátku rámce + posun na tabulku registrů + číslo registru (v rámci tabulky registrů). Nejprve jsou sečteny nižší byty čísel s přenosem a poté jsou sečteny vyšší byty s přičteným přenosem.

get_adress1

Adresuje cívky a vstupy (bitové hodnoty). Cívky (vstupy) jsou ukládány od konce tabulky cívek (vstupů)

table_offset

Vypočítá posun (offset) uvnitř paměti *data*. Sčítá až tři 8-bitové hodnoty s případným přenosem.

add16

Sčítá dvě 16-bitové čísla. Slouží funkci *modbus_slave_init* pro kontrolu, zda celkový počet zadaných cívek, vstupů a registrů není větší než maximální povolený paměťový prostor (určený konstantou *DATA_SIZE*).

bits2bytes

Dělí zadané číslo 8. Vrací výsledek dělení a zbytek po dělení.

clear_data

Při inicializaci *modbus_slave_init* vynuluje alokovaný paměťový prostor *data*.

CRC

Vypočítá CRC přesně podle algoritmu popsaného v kapitole 2.6.4 s tím rozdílem, že každá operace probíhá zvlášť pro nižší a vyšší byte výpočtu. Je využíván příznak *Carry* při přenosu bitu z nižšího bytu do vyššího. Před výstupem je prohozen nižší a vyšší byte CRC, protože v tomto pořadí se připojuje ke zprávě.

add_CRC

Připojuje CRC ke zprávě (rámcí). Využívá funkci *CRC* a výsledek uloží na konec rámce.

ICG_init

Nastavuje vnitřní frekvenci sběrnice na 20 MHz. Tato funkce nesouvisí přímo s protokolem MODBUS, ale řeší problém, kdy po resetu jednoho zařízení se změní jeho frekvence sběrnice a tím se přeruší komunikace, což byl problém i při ladění a testování aplikace, i proto zde zůstala tato funkce ponechána, byť mohla být napsána až v hlavním programu.

5 UKÁZKOVÝ PROGRAM

Pro názornější zobrazení možností knihovny byl vytvořen jednoduchý ukázkový program v jazyce C, který demonstruje použití základních funkcí protokolu MODBUS.

5.1 Master

Na straně masteru jsou v programu vloženy zdrojové kódy knihovny MODBUSu *modbus_master_gb60.asm* a displeje *disp_gb60.asm*. Knihovny jsou připojeny pomocí hlavičkových souborů *modbus_master_gb60.h*, *disp_gb60.h* příkazem *#include*. Kvůli využití funkce *sprintf* bylo potřeba připojit také hlavičkový soubor *<stdio.h>*.

5.1.1 Proměnné

```
extern char frame [ ]
```

Externí proměnná z knihovny *modbus_master_gb60.asm*. Slouží pouze ke čtení a umožňuje vypsát na displej obsah zprávy (rámce).

```
int count
```

Tato proměnná slouží jako čítač odeslaných rámců.

```
char status
```

Udává, zda byla úspěšně přijata odpověď či nikoliv.

```
char buff[16] = {0}
```

Pomocná proměnná, slouží k výpisu textu na displej.

```
char odpoved[16] = {0}
```

Čtecí funkce masteru používají tuto proměnnou pro uložení přijatých dat.

```
char civky[ ] = {0x0A, 0x20}
```

Proměnná obsahující 2 byty = 16 cívek.

```
int registry[ ] = {0x000A, 0x000B}
```

Proměnná obsahující 4 byty = 2 registry.

5.1.2 Hlavní program

Po deklaraci proměnných je na začátku programu zakázáno přerušení, poté je volána funkce `ICG_init()`, která zajistí nastavení vnitřní frekvence sběrnice na 20 MHz, tak aby nemohla být změněna např. po resetu vývojového kitu. Inicializuje se port F, na kterém jsou připojeny LED diody, jsou odpojeny pull-up rezistory a nastaven výstupní režim. Je vhodné pracovat pouze s bity PTFD0-3, v žádném případě nesmí být ovlivněn bit PTFD7, který určuje směr komunikace sběrnice, přepsáním tohoto bitu nesprávnou hodnotou dojde ke zhroucení komunikace! Port D s tlačítky je nastaven na vstupní režim a pull-up rezistory jsou připojeny. Proběhne inicializace displeje a rozhraní SCI2 (nastavuje i globální povolení přerušení, ne však časovače). Na displej je vypsán text „MODBUS MASTER“.

Uvnitř hlavní smyčky čeká podprogram na stisk tlačítka, jejich funkce jsou následující:

- SW1 – posílá požadavek na slave s adresou 1 na zápis jedné (první) cívky na hodnotu 1. Po vyslání požadavku se na displeji objeví následující text.

```
M: WC(1,1,1) 01  
01050001FF00DDFA
```

Obr. 28. Výpis na displeji (zápis cívky)

M značí, že se jedná o master, *WC* značí zápis cívky (Write Coil) a *01* je počítadlo vyslaných požadavků. Protože však master funkce pošlou požadavek a následně přijmou odpověď, nelze vypsát na displej (z hlavního programu) poslanou žádost, ale pouze přijatou odpověď, jejíž obsah je vypsán na druhém řádku displeje. Formát odpovědi je: 01 – adresa slave, 05 – kód funkce zápisu cívky, 0001 – číslo cívky, FF00 – zapsaná hodnota cívky, DDFA – CRC. V tomto případě je odpověď totožná s žádostí. Zápis cívky způsobí rozsvícení nebo zhasnutí LED1 (slave).

- SW2 – posílá požadavek na slave s adresou 1 na čtení čtyř vstupů 5-8. Slouží ke čtení stavu tlačítek, jejich stisknutí (slave) při poslání požadavku, způsobí rozsvícení příslušných LED (master). Pokud není stisknuto ani jedno tlačítko, všechny LED zhasnou. Po vyslání požadavku se na displeji objeví následující text.

```
M: RI(1,5,4) 02  
0102010FE18C
```

Obr. 29. Výpis na displeji (čtení vstupů)

RI značí čtení vstupů (Read Inputs). Formát odpovědi je: 01 – adresa slave, 02 – kód funkce čtení vstupů, 01 – počet bytů přečtených vstupů, 0F – hodnoty vstupů – port D (0F znamená, že žádné tlačítko nebylo stisknuto), E18C – CRC.

- SW3 – posílá požadavek na slave s adresou 1 na zápis 1 (prvního) uchovávacího registru. Tato funkce není na kitu vizualizovaná, pouze je vypsána odpověď, která je v tomto případě opět totožná s požadavkem.

```
M: WHR(1, 1, 1) 03
010600011020D412
```

Obr. 30. Výpis na displeji (zápis registru)

WHR značí zápis uchovávacího registru (Write Hold Register). Formát odpovědi je: 01 – adresa slave, 06 – kód funkce zápisu registru, 0001 – adresa registru, 1020 – zapsaná hodnota registru, D412 – CRC.

- SW4 – posílá požadavek na slave s adresou 1 na čtení vstupního registru. Čte polohu potenciometru (slave) převedenou na 8-bitové číslo.

```
M: RIR(1, 1, 1) 04
01040200FFF970
```

Obr. 31. Výpis na displeji (čtení registru)

RIR značí čtení vstupního registru (Read Input Register). Formát odpovědi je: 01 – adresa slave, 04 – kód funkce čtení vstupního registru, 02 – počet bytů přečtených registrů, 00FF – přečtená hodnota registru (potenciometr je v krajní poloze vlevo), F970 – CRC.

Po stisknutí libovolného tlačítka se tedy vypíše příslušný text, který se pomocí funkce *sprintf()* uloží do proměnné *buff*. K výpisu odpovědi na displej slouží funkce *Vypis()*. Protože jednotlivé rámce jsou různě dlouhé a nejsou zakončeny žádným znakem, je vypsáno napevno 8 znaků (maximální délka odpovědi výše uvedených funkcí). Funkci *Vypis()* je předána délka rámce, takže pokud je rámeček kratší než 8 znaků (bytů), pak je na příslušné místo proměnné *buff* (pole znaků) vložena ukončovací nula a tím se stává řetězcem a je vypsán voláním funkce *dtext()*. Pokud master funkce skončila s chybou je na konci 2. řádku vypsán kód chyby, chybový kód je kratší než standardní odpověď, chybový kód tedy nic nepřepíše. V případě pokud nebyla odpověď vůbec přijata (timeout) bude vypsán pouze kód chyby 04.

5.2 Slave

Na straně slave jsou v programu vloženy zdrojové kódy knihovny MODBUSu *modbus_slave_gb60.asm* a displeje *disp_gb60.asm*. Kvůli využití funkce *sprintf* bylo opět potřeba připojit také hlavičkový soubor *<stdio.h>*.

5.2.1 Proměnné

```
extern char frame[], int count, char buff[16] = {0}
```

Tyto proměnné mají stejný význam jako u masteru.

```
extern char status_reg
```

Externí proměnná z knihovny *modbus_slave_gb60.asm* – registr příznaků, slouží ke zjištění okamžiku, kdy byla odeslána odpověď.

```
char civka
```

Slouží k uložení hodnoty cívky zapsané masterem.

```
char vstup
```

Slouží k uložení hodnoty vstupu zapsané masterem.

5.2.2 Hlavní program

Úvod programu je podobný jako u masteru. Je zde navíc inicializace MODBUSu – přiřazení adresy slave a alokace proměnných pro cívky, vstupy, vstupní a uchovávací registry. Konkrétně má slave přiřazenu adresu 1, je alokováno 32 cívek, 32 vstupů, 5 vstupních a 5 uchovávacích registrů. Na displej je vypsán text „MODBUS SLAVE“. Je také inicializován A/D převodník a nastaven příznak *REF* (REading Frame) proměnné *status_reg*, který značí probíhající čtení rámce (k tomuto přesně slouží u masteru, zde spíše znamená čekání na čtení rámce). Pokud byl přijat požadavek a odeslána odpověď (obojí probíhá v přerušení, v hlavním programu lze tedy vypsát na displej pouze odpověď) je příznak *REF* vynulován a program může vypisovat na displej (jinak by se muselo vypisovat neustále periodicky). Ještě předtím jsou ale načteny stavy tlačítek (funkce *write_img_inputs()*) a zapsány do paměti, stejně tak je načtena a zapsána do paměti poloha potenciometru převedená na číslo pomocí A/D převodníku (funkce *write_img_in_reg()*). Pokud je přijatá žádost na zápis cívky, je tato cívka načtena z paměti pomocí funkce *read_coil()* a pomocí funkce XOR je použita ke změně stavu LED1.

```
S:          01
01050001FF00DDFA
```

Obr. 32. Výpis na displeji (zápis cívký) – slave

S značí, že se jedná o slave, 01 je počítadlo a na druhém řádku je již popsána odpověď. Na straně slave je tedy nutné se pouze pomocí funkcí postarat o zápis (čtení) údajů do (z) paměti a zbytek obslouží knihovna.

ZÁVĚR

Cílem diplomové práce bylo vytvořit programovou knihovnu MODBUS v jazyce symbolických adres pro mikropočítač Freescale z rodiny HCS08. Konkrétně byl použit typ MC9S08GB60, který je součástí vývojového kitu Freescale M68EVB908GB60. Použitý mikropočítač bez problémů splňuje požadavky, které jsou z hlediska implementace protokolu MODBUS potřeba a vývojový kit podporuje komunikační sběrnici RS485. Konfigurace kitu pro použití sběrnice RS485 je popsána a na obrázku také přehledně znázorněna.

Výstupem práce je knihovna MODBUS, která je implementovaná pro použití na sériovém rozhraní RS485. Knihovna podporuje nejčastější komunikační režim MODBUS RTU, který je charakteristický oddělováním jednotlivých rámců mezi sebou pomocí krátké pomlky na sběrnici a datovým modelem s jedním společným blokem. Knihovna podporuje 4 typy dat definované MODBUSem, kterými jsou cívky, diskrétní vstupy, což jsou bitové hodnoty a vstupní a uchovávací registry, což jsou 16-bitové hodnoty a základní funkce, které s těmito daty pracují. V teoretické části práce jsou funkce podrobně popsány včetně příkladů použití. K zabezpečení integrity dat byl implementován zabezpečovací cyklický kód CRC. Není využita podpora parity, která je doporučována, ale není v knihovně implementována její kontrola.

Pro snadnější pochopení práce s knihovnou byl vytvořen ukázkový program, který vizualizuje možnosti knihovny a také ukazuje práci s daty a jejich využití, protože jejich účel nemusí být na první pohled zřejmý. To platí zejména pro cívky, jedná o běžné bitové hodnoty, které mohou ovládat např. relé a v tomto případě slouží pro rozsvícení nebo zhasnutí LED diody, dále jsou využity vstupy pro čtení stavu tlačítek a vstupní registr pro zjištění polohy potenciometru.

K práci je také přiložena podpůrná prezentace, která slouží pro potřeby výuky ve cvičení předmětu Mikropočítače pro seznámení studentů s funkcemi knihovny. Knihovna by tedy měla sloužit především pro výuku na Fakultě aplikované informatiky a následně při použití v průmyslové praxi.

ZÁVĚR V ANGLIČTINĚ

The aim of my master thesis was to create a MODBUS program library in an assembly language for Freescale microcontrollers of the HCS08 family. Specifically, the type MC9S08GB60 was used, which is placed on the Freescale M68EVB908GB60 development kit. This microcontroller meets all requirements for the MODBUS protocol implementation and the development kit supports RS485 interface. Configuration enabling the RS485 bus on the kit is described and also illustrated in a figure.

The output of this work is a MODBUS library implemented for the RS485 serial interface usage. It supports the most common communication mode – MODBUS RTU, for which is characteristic separating individual frames with a silent interval on the bus and a data model with only one block. The library supports four data types defined by MODBUS, namely coils, discrete inputs, which are bit values, input registers and hold registers, which are 16-bit values. Basic functions for working with the data were also implemented. The functions are described in detail in the theoretical part, including examples. The CRC code was added to ensure data integrity. Parity checking is not implemented in the library, despite it is recommended.

A sample program was created for easier understanding how to work with the library. It visualizes options of the library and demonstrates the data manipulation and usage, because that may not be obvious at first look. This is especially significant for coils, common bit values that can control relays and in this case used to turn LEDs on and off. Inputs are used in order to detect switches states and input register is read to determine the potentiometer position.

A tutorial presentation is also enclosed and will be used for teaching the Microcomputers course to familiarize students with the library functions. Therefore the library should be used mainly for educational purposes of the Faculty of Applied Informatics and within the industrial application.

SEZNAM POUŽITÉ LITERATURY

- [1] Modbus Organization, Inc. *Modbus Application Protocol Specification V1.1b.*, 2006. Dostupný z WWW: <www.modbus.org/>.
- [2] Modbus Organization, Inc. *MODBUS over Serial Line Specification and Implementation Guide V1.02.*, 2006. Dostupný z WWW: <www.modbus.org/>.
- [3] RONEŠOVÁ, Andrea. *Přehled protokolu MODBUS*, 2005. 19 s. Dostupné z WWW: <<http://home.zcu.cz/~ronesova/bastl/files/modbus.pdf>>.
- [4] Freescale Semiconductor. *Development Board for Freescale MC9S08GB60*, 2005. Dostupný z WWW: <www.freescale.com>.
- [5] *Simply Modbus* [online]. 2008 [cit. 2011-04-30]. Dostupné z WWW: <<http://simplymodbus.ca/>>.
- [6] HRUŠKA, František. *Technické prostředky informatiky a automatizace : (úvod, popis funkce, konstrukce a aplikace)*. Vyd. 1. Ve Zlíně : Univerzita Tomáše Bati, 2007. 193 s. ISBN 978-80-7318-535-0.
- [7] HRUŠKA, František. *Projektování řídicích a informačních systémů : P8: Přenos dat a komunikace*. Vyd. 1. Ve Zlíně : Univerzita Tomáše Bati, 2010. 176 s. ISBN 978-80-7318-979-2.
- [8] Freescale Semiconductor. *HCS08 Family Reference Manual, Rev.1.*, 2003. Dostupný z WWW: <www.freescale.com>.
- [9] Freescale Semiconductor. *MC9S08GB/GT Data Sheet, Rev.2.3.*, 2004. Dostupný z WWW: <www.freescale.com.com>.
- [10] VÁŇA, Vladimír. *Začínáme s mikrokontroléry Motorola HC08 Nitron*. Praha: BEN – technická literatura, 2003. 96 s. ISBN 80-7300-124-1.
- [11] KAINKA, Burkhard. *USB - měření, řízení a regulace pomocí sběrnice USB*. 1. české vyd. Praha : BEN - technická literatura, 2002. 247 s. ISBN 8073000733.
- [12] HÄBERLE, Heinz O; HANDLÍŘ, Jiří. *Průmyslová elektronika a informační technologie*. Vyd. 1. Praha : Europa-Sobotáles, 2003. 719 s. ISBN 8086706044.
- [13] Hw.cz [online]. 2004 [cit. 2011-04-29]. *Průmyslová sběrnice Profibus*. Dostupné z WWW: <<http://hw.cz/Rozhrani/ART1028-Prumyslova-sbernice-Profibus.htm>>.

- [14] OLMR, Vít. HW server [online]. 2005-12-12 [cit. 2011-02-19]. *HW server představuje - Sériová linka RS-232*. Dostupné z WWW: <<http://hw.cz/rs-232>>.
- [15] *Výuka na FAI UTB ve Zlíně: Kurz Mikropočítače* [online]. 2011 [cit. 2011-04-30]. Dostupné z WWW: <<http://www.vyuka.fai.utb.cz/>>.
- [16] Lammertbies.nl [online]. 1997 [cit. 2011-04-30]. *On-line CRC calculation and free library*. Dostupné z WWW: <<http://www.lammertbies.nl/comm/info/crc-calculation.html>>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

422_485	Konektor pro připojení sběrnice RS485
485_EN	Zkratovací propojka nastavující dvou vodičový čtyřvodičový nebo režim RS485
ADU	Aplikační rámec
ASI	Sběrnice AS-Interface
ASIC	Integrovaný obvod pro specifické použití
CAN	Sběrnice Controller Area Network
COM	Sériový port
COM_SW	Konfigurační DIP přepínač
COP	Watchdog
CR	Ukončovací znak řádku
CRC	Zabezpečovací cyklický kód (kontrolní součet)
CSMA/CD	Přístupová metoda sítí typu Ethernet
EMC	Elektromagnetická kompatibilita
FHSS	Metoda přenosu v rozprostřeném spektru (Frequency-hopping spread spectrum)
GND	GrouND (nulový potenciál)
HCS08	Řada mikropočítačů Freescale
IEEE	Institut pro elektrotechnické a elektronické inženýrství
I-Ethe	Průmyslový Ethernet
LF	Znak přechodu na nový řádek
LRC	Kontrola podélnou paritou
LSB	Nejméně významný byte
lsb	Nejméně významný bit
MCU	Mikropočítač (mikrokontrolér)

MODBUS ASCII	Vysílací režim ASCII
MODBUS RTU	Vysílací režim RTU
MSB	Nejvíce významný byte
msb	Nejvíce významný bit
NRZ	Kódování NRZ (bez návratu k nule)
OSI	Propojení definované normami ISO pro výměnu dat pomocí sedmi vrstev
PAN	Osobní počítačová síť (Personal Area Network)
PDA	Kapesní počítač
PDU	Paket na úrovni protokolu
PLC	Programovatelný automat
PTF7	Bit 7 portu F – nastavuje směr komunikace RS485
RxD	Příjem dat
SCI	Sériové rozhraní mikropočítače
SPI	Synchronní sériové periferní rozhraní
TCP/IP	Řídící přenosový protokol / protokol Internetu
TPM	Časovač
TxD	Vysílání dat
USB	Univerzální sériová sběrnice

SEZNAM OBRÁZKŮ

Obr. 1. Přenosový rámeček rozhraní RS232	11
Obr. 2. Topologie USB	13
Obr. 3. Princip propojení RS485 (standardní dvou vodičová varianta)	14
Obr. 4. Možnosti implementace protokolu MODBUS	19
Obr. 5. Základní struktura zprávy protokolu MODBUS	20
Obr. 6. MODBUS – bezchybná transakce	20
Obr. 7. MODBUS – chybná transakce	21
Obr. 8. Datový model protokolu MODBUS s oddělenými bloky	22
Obr. 9. Datový model protokolu MODBUS s jedním společným blokem	23
Obr. 10. MODBUS – adresovací model	23
Obr. 11. Stavový diagram zpracování MODBUS požadavku	24
Obr. 12. Kategorie kódů funkcí	25
Obr. 13. Rámeček MODBUS zprávy v režimu RTU	38
Obr. 14. Časování zpráv v RTU režimu	38
Obr. 15. Rámeček MODBUS zprávy v režimu ASCII	39
Obr. 16. Stavový diagram pro master	40
Obr. 17. Stavový diagram pro slave	41
Obr. 18. Vývojový kit M68EVB908GB60 [15]	44
Obr. 19. Vývojový kit M68EVB908GB60 – rozmístění jednotlivých součástí	46
Obr. 20. Nastavení a propojení kitů pro podporu sběrnice RS485	47
Obr. 21. Zdroj přenosové rychlosti SCI	49
Obr. 22. Registr SCIxC1	50
Obr. 23. Registr SCIxC2	50
Obr. 24. Registr SCIxS1	51
Obr. 25. Registr SCIxS2	51
Obr. 26. Registr SCIxS3	52
Obr. 27. Datový model MODBUSu knihovny Modbus_gb60	55
Obr. 28. Výpis na displeji (zápis cívky)	71
Obr. 29. Výpis na displeji (čtení vstupů)	71
Obr. 30. Výpis na displeji (zápis registru)	72
Obr. 31. Výpis na displeji (čtení registru)	72

Obr. 32. Výpis na displeji (zápis cívky) – slave 74

SEZNAM TABULEK

Tab. 1. Napěťové úrovně datových signálů.....	11
Tab. 2. Napěťové úrovně řídicích signálů	12
Tab. 3. Označení pinů USB propojení.....	12
Tab. 4. Napěťové úrovně signálů.....	14
Tab. 5. Datový model protokolu MODBUS.....	22
Tab. 6. Definice funkčních kódů MODBUSU	26
Tab. 7. MODBUS chybové kódy.....	36
Tab. 8. Adresní prostor složený z 256 různých adres.....	37
Tab. 9. Konektor 422_485	46

SEZNAM PŘÍLOH

Příloha P I: Zdrojové kódy knihovny MODBUS_GB60

Příloha P II: Výuková prezentace

Příloha P III: Font Matrix Schedule

Příloha P IV: Výpočet CRC v Excelu

PŘÍLOHA P I: ZDROJOVÉ KÓDY KNIHOVNY MODBUS_GB60

Master:

Soubory: *modbus_master_gb60.asm*

modbus_master_gb60.h

Umístění na CD: *\Knihovna MODBUS_GB60\Master*

Slave:

Soubory: *modbus_slave_gb60.asm*

modbus_slave_gb60.h

Umístění na CD: *\Knihovna MODBUS_GB60\Slave*

PŘÍLOHA P II: VÝUKOVÁ PREZENTACE

Soubor: *Knihovna MODBUS_GB60 pro komunikaci na RS485.ppt*

Umístění na CD: *Přílohy*

Prezentace stručně popisuje základy protokolu MODBUS, nastavení vývojového kitu pro využití rozhraní RS485 a základy práce s knihovnou MODBUS_GB60 včetně popisu ukázkového programu.

PŘÍLOHA P III: FONT MATRIX SCHEDULE

Soubor: *matrix_schedule.ttf*

Umístění na CD: *Přílohy*

Jedná se o volně šiřitelný font písma, který zobrazuje znaky v matici 5x8 a imituje tak písmo LCD displeje. Je potřeba pro korektní zobrazení znaků v Obr. 27 až Obr. 32.

PŘÍLOHA P IV: VÝPOČET CRC V EXCELU

Soubor: *crc.xls*

Umístění na CD: *Přílohy*

Soubor, který velmi názorně demonstruje způsob výpočtu CRC kódu pro MODBUS krok za krokem.