VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

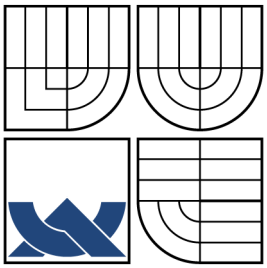
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

KOMUNIKAČNÍ ADAPTÉR PRO SYSTÉM SBĚRU DAT — HARDWAROVÁ I SOFTWAROVÁ REALIZACE

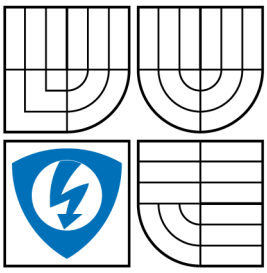
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR KAŠPAR
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

KOMUNIKAČNÍ ADAPTÉR PRO SYSTÉM SBĚRU DAT — HARDWAROVÁ I SOFTWAROVÁ REALIZACE

COMMUNICATION ADAPTOR FOR DATA ACQUISITION SYSTEM
HARDWARE AND SOFTWARE DESIGN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR KAŠPAR
AUTHOR

VEDOUCÍ PRÁCE — Ing.IVANA JAKUBOVÁ
SUPERVISOR

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Bakalářská práce

bakalářský studijní obor
Elektronika a sdělovací technika

Student: Petr Kašpar ***ID:*** 78563
Ročník: 3 ***Akademický rok:*** 2008/2009

NÁZEV TÉMATU:

Komunikační adaptér pro systém sběru dat – hardwarová i softwarová realizace

POKYNY PRO VYPRACOVÁNÍ:

Komunikační mikrokontrolérem řízený adaptér má zprostředkovat přenos dat z ultrazvukového tloušťkoměru do nadřazeného systému k dalšímu zpracování tak, že zajistí potřebnou změnu přenosových parametrů a formátu přenášených dat. Dle informací zadavatele se seznamte s konkrétními parametry vstupního a výstupního zařízení. Ideově navrhněte blokovou strukturu adaptéru a sestavte vývojový diagram obslužného programu. Dle požadovaných vlastností aplikace vyberte vhodný typ mikrokontroléru.

Na základě závěrů předchozí etapy navrhněte schéma zapojení s vybraným typem mikrokontroléru a realizujte je na zkušební desce. Navrhněte a odlaďte program v ASM a ověřte funkci zařízení. Navrhněte desku plošných spojů, zařízení realizujte a pro typické soubory vstupních dat ověřte jeho spolehlivou funkčnost.

DOPORUČENÁ LITERATURA:

[1] Technická dokumentace výrobců Microchip, Aurel.
[2] HRBÁČEK, J. Komunikace mikrokontroléru s okolím. Praha: BEN – technická literatura, 2002.
[3] Microchip MPASM user guide.
[4] GOFTON, P. W. Sériová komunikace. Praha: Grada, 1995.

Termín zadání: 9.2.2009 ***Termín odevzdání:*** 5.6.2009

Vedoucí práce: Ing. Ivana Jakubová

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

ABSTRAKT

Cílem této bakalářské práce je tvorba adaptéru pro systém sběru dat řízeného mikrokontrolérem. Funkce zařízení spočívá v sériovém asynchronním příjmu ASCII řetězce odeslaného ultrazvukovým tloušťkoměrem přenosovou rychlostí 2400Bd. Přenos je realizován dle standardu RS-232. Část řetězce obsahuje číselnou informaci o hodnotě změřené tloušťkoměrem. V mikrokontroléru je tato číselná informace vynásobena korekční konstantou uloženou v paměti EEPROM a umístěním desetinné čárky je určen řád čísla. Poté je řetězec zařízením asynchronně odeslán přenosovou rychlostí 9600Bd do nadřazeného systému. V práci je popsán obslužný program mikrokontroléru v jazyce Assembler a hardwarová konstrukce zařízení.

KLÍČOVÁ SLOVA

Mikrokontrolér, PIC, Assembler, RS-232, Sériová komunikace.

ABSTRACT

The aim of this bachelor thesis is to create adaptor with microcontroller for data acgusition system. Function of the device is to receive serial asynchronous string of ASCII characters which was send from ultrasonic thickness gauge. Baud rate of transmittion is 2400Bd. Transmittion is realized according to RS-232 standard. Part of the string contains numeric information about value which was measured by thickness gauge. This numeric information is in microcontroller multiplied with corrective constant and decimal point in result is placed to right position. Corrective constant is stored in EEPROM memory. After that adaptor transsmits data to superior system in format of serial asynchronous string of ASCII with baud rate 9600Bd. Thesis describes service code for microcontroller in Assembler language and hardware construction of device.

KEYWORDS

Microcontroller, PIC, Assembler, RS-232, Serial Communication.

KAŠPAR, P. *Komunikační adaptér pro systém sběru dat – hardwarová i softwarová realizace*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2009. 79 s. Vedoucí práce Ing.Ivana Jakubová.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Komunikační adaptér pro systém sběru dat — hardwarová i softwarová realizace“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucí práce Ing.Ivaně Jakubové a odbornému konzultantovi Ing.Jiřímu Jakubovi za osobní přístup, organizační vedení, odborné konzultace a velmi cenné rady při zpracovávání bakalářské práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Bakalářská práce se zabývá návrhem komunikačního adaptéru pro systém sběru dat. Funkce zařízení spočívá v zajištění přenosu dat mezi ultrazvukovým tloušťkoměrem a nadřazeným systémem zajištěním změny informační části datové informace, přenosových parametrů a formátu přenášených dat. Data vstupují do zařízení ve formě řetězce ASCII (americký normalizovaný kód pro výměnu informací – American Standard Code for Information Interchange) znaků se zadanou přenosovou rychlostí. Tento řetězec je zařízením přijat, zpracován a opět ve formě ASCII řetězce se zadanou přenosovou rychlostí odeslán do nadřazeného systému.

Zadání nabádá k výběru vhodného typu mikrokontroléru PIC od výrobce Microchip Technology Inc. Dále k vytvoření obslužného programu v jazyce symbolických adres Assembler. Zadáním je požadována i hardwarová realizace zařízení s ověřením jeho funkčnosti na typických vstupních datech.

1 SPECIFIKACE ZADANÍ

1.1 Vstupní část

Z důvodu omezení počtu vodičů nutných pro spojení je využito simplexní asynchronní komunikace dle standardu RS-232-C. Délka slova při přenosu je 8 bitů, zakončení je provedeno jedním stop bitem. Není využívána kontrola paritním bitem. Vstupní přenosová rychlost je 2400 Bd. Měřením získaná vstupní data mají formu řetězce v otevřeném ASCII tvaru. Řetězec se skládá z informačních a režijních znaků. Informační znaky nesou informaci o hodnotě změřené tloušťkoměrem v mikrometrech. Část informačních znaků je zakončena ASCII znaky CR (návrat vozíku – Carriage Return) a LF (posun o řádek – Line Feed), které slouží k oddělení jednotlivých měřených hodnot v nadřazeném systému. Režijní znaky se na vstupu systému filtrují a dále se nepřenáší a nezpracovávají. Typický tvar vstupního řetězce je uveden na Obr.1.1. Řetězec je zařízením přijímán cca 3x za sekundu.

Hexadecimální vyjádření:	54	31	32	33	34	35	0D	0A	20	0C
ASCII vyjádření:	T	1	2	3	4	5	CR	LF	SPACE	FF

■ Režijní znaky □ Informační znaky

Obr. 1.1: Typický tvar vstupního řetězce.

1.2 Zpracování řetězce

Po příjmu řetězce a nutném odfiltrování režijních znaků je informace o hodnotě změřené tloušťkoměrem převedena z mikrometrů na milimetry umístěním desetinné čárky na příslušnou pozici a vynásobena volitelnou kalibrační konstantou v rozsahu 0,001 až 65,535. Takto zpracovaný řetězec spolu se znaky CR a LF na jeho konci je zařízením odeslán.

1.3 Výstupní část

Požadavky na parametry výstupní části zařízení jsou odvozeny od vstupních parametrů nadřazeného systému. Jedná se taktéž o simplexní asynchronní přenos dle standardu RS-232-C. Délka slova je 8 bitů a slovo je zakončeno jedním stop bitem. Kontrola vysíláním paritního bitu není vyžadována. Výstupní přenosová rychlost je 9600 Bd. Typický tvar výstupního řetězce je uveden na Obr.1.2.

S využitím protokolu RS-232 je počítáno pouze pro testovací prototyp. V reálném systému bude výstupem zařízení bezdrátový vysílací modul.

Hexadecimální vyjádření:	31	32	2C	33	34	35	0D	0A
ASCII vyjádření:	1	2	,	3	4	5	CR	LF

Obr. 1.2: Typický tvar výstupního řetězce.

1.4 Hardware

Zadání požaduje použití mikrokontroléru PIC. Adaptér je napájen nestabilizovaným stejnosměrným napětím 7,5V. Zadáním je požadován návrh DPS (deska plošných spojů) ve formě vhodné pro SMT (technologie povrchové montáže – Surface Mount Technology). Této technologii je podřízen výběr vhodných SMD (součástka pro povrchovou montáž – Surface Mount Device) součástek zařízení.

2 TEORETICKÝ ZÁKLAD PRO ŘEŠENÍ PROBLEMATIKY

2.1 Asynchronní sériový přenos

Datová informace je přenášena v sériovém bitovém toku ve formě časově proměnného signálu. Každý bit je přenášen pevně stanovenou dobu, známou jako bitovou periodu příp. baud periodu. Úrovně při komunikaci mohou nabývat dvou stavů. Binární 1 nazývaný MARK a stav nazývaný SPACE vyjadřující binární 0 [10].

Pří asynchronním sériovém přenosu jsou jednotlivé datové bity přenášený v oddělených skupinách 7 - 10 bitů. Tyto skupiny se nazývají znaky. Využívá se proces zvaný vytváření rámců. Před znak je zařazen start bit. Za znak je zařazen stop bit. Stop bit může mít délku až dvou bitových period.

V případě, že po sériové lince neprobíhá přenos, je tato linka ve stavu MARK. Start bit na začátku přenosu způsobí změnu ze stavu MARK do stavu SPACE a tím upozorňuje přijímač na přicházející data. V tomto okamžiku se přijímač zasynchronizuje. Stop bit, který ukončuje rámec, změní stav přenosové linky zpět na MARK.

Při zjišťování hodnot jednotlivých datových bitů přijímač nepracuje asynchronně, nýbrž s každou bitovou periodou zjišťuje stav přenosové linky. Lze tedy zjednodušeně říci, že přijímání start bitu je prováděno asynchronně, příjem datových bitů a stop bitu je však již synchronní s přenosovou rychlostí.

Přenosová rychlost vyjadřuje počet diskrétních signálů za sekundu. Je vyjádřena v Bd (Baud). U jednoduchého asynchronního přenosu je hodnota v baudech shodná s rychlostí v bit/s (bity za sekundu), protože se jedná pouze o dvoustavovou modulaci [1].

2.2 Standard sériové komunikace RS-232-C

Jedná se o normu publikovanou v roce 1969 pod hlavičkou EIA (sdružení pro elektronický průmysl – Electronic Industries Association). Byla vyvinuta pro zajištění vzájemné kompatibility při sériové komunikaci zařízení různých výrobců, původně zejména modemů a terminálů. Z tohoto základu vyplývá v popisu standardu užívané značení komunikujících zařízení. Standard určuje elektrické vlastnosti obvodů mezi dvěma zařízeními, definuje názvy a čísla jednotlivých vodičů.

V dnešní době v oblasti spotřební elektroniky je standard RS-232-C nahrazován novějším USB (univerzální sériová sběrnice – Universal Serial Bus). Díky jednoduchosti implementace však RS-232 stále nachází uplatnění zejména ve specializovaných systémech.

Elektrické signály použité pro přímé sériové spojení mají normou určené hodnoty napětí. Stav MARK reprezentující binární 1 má normou povolené napěťové úrovně - 5V až - 15V na výstupu vysílače a -3V až -15V na vstupu přijímače. Povolené napěťové úrovně pro stav SPACE jsou +5V až +15V na výstupu vysílače a +3V až +15V na vstupu přijímače. Rozdíl mezi normou definovanými hodnotami napětí pro výstup vysílače a vstup přijímače slouží ke kompenzaci jeho poklesu ve spojovacím vodiči vlivem útlumu. Dle literatury [1] se za maximální vzdálenost přenosového vodiče při obvyklých přenosových rychlostech považuje hodnota 15m. Při hodnotě vyšší může klesnout úroveň napětí mimo povolené meze. Kvalitu signálu mimo délky spojovacího vodiče ovlivňuje také jeho kapacita.

Při bitovém přenosu znaku je přenášen jako první bit s nejnižším významem neboli LSb (Nejméně významný bit – Least Significant bit) a jako poslední bit MSb (Bit s největším významem – Most Significant bit).

Pro simplexní přenos z DCE (ukončující zařízení datového okruhu – Data Communication Equipment) do DTE (koncové zařízení přenosu dat – Data Terminal Equipment), využitý v této práci, je dle literatury [1] plně dostačující užití dvojice vodičů TXD (vysílaná data – Transmitted Data) a SG (signálová země – Signal Ground). Vodič SG slouží jako napěťová reference pro určení polarity a napětí vodiče TXD. Vodič TXD slouží k přenosu sériového signálu. Vodič pro přenos v opačném směru se nazývá RXD (přijímaná data – Received Data).

2.3 Mikrokontroléry PIC

Jsou jednočipové počítače založené na modifikované Harvardské RISC (Počítač s redukovanou instrukční sadou – Reduced Instruction Set Computer) architektuře. Základ Harvardské architektury spočívá v oddělení paměti programu a dat (viz. Obr.2.1). Oddělením je umožněno využití samostatných sběrnic. Redukcí instrukční sady pouze na základní instrukce je zvýšena rychlost jejich vykonávání. Konstantní délka instrukcí umožňuje jejich vykonávání v přesně stanoveném čase a to buď v jednom nebo dvou hodinových cyklech.

Nabídka společnosti Microchip je velmi široká, čítá stovky mikrokontrolérů. Každý mikrokontrolér však obsahuje:

- paměť RAM pro data, paměť EPROM nebo Flash pro program,
- digitální vstupně výstupní porty,
- časovač s 8-bitovým „předděličem“,
- časovač Watchdog, rozhraní pro připojení externího oscilátoru,

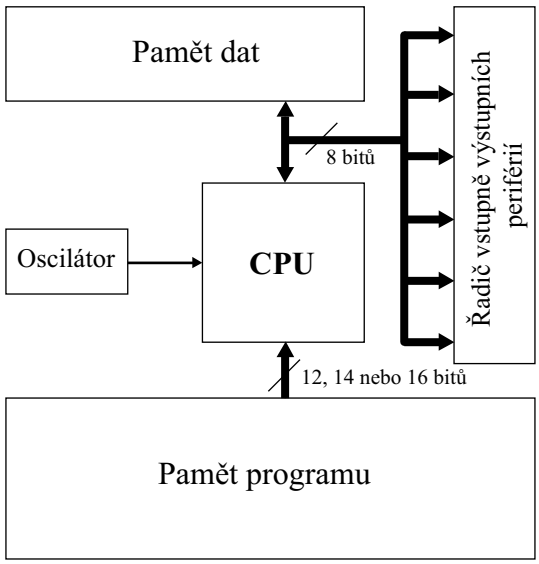
- přímé a nepřímé adresování,
- mód s nižší spotřebou tzv. SLEEP, Power-on reset.

Mikrokontrolér může obsahovat další z volitelných příslušenství, například:

- analogové komparátory, větší množství čítačů, datovou pamět EEPROM,
- přerušení od vnitřních či vnějších zdrojů, vlastní oscilátor,
- PWM výstup, Sériové komunikační rozhraní SCI - USART, a mnoho dalších.

Je možné vybírat z 8,16 nebo 32-bitových mikrokontrolérů. Základní dělení je do produktových řad tzv. rodin (z anglického: Family).

Řada 8-bitových mikrokontrolérů zastupuje mikrokontroléry s nízkým výpočetním výkonem do 16 MIPS (Milion instrukcí za sekundu – Million Instructions Per Second), s 12, 14 či 16-bitovou délkou slova instrukce, s taktovacím kmitočtem do 32MHz, pamětí programu do 128Kb, datovou pamětí do 4Kb. Mohou obsahovat (z hlediska kompletní nabídky) jednodušší příslušenství např. rozhraní pro komunikaci na sběrnici CAN (Controller-Area Network), řadiče LCD, podporu USB a další. Výkon a příslušenství předurčuje 8-bitové mikrokontroléry hlavně k řídícím funkcím a pro výkonově nenáročné aplikace.



Obr. 2.1: Blokové schéma struktury 8-bitové mikrokontroléru.

Řada 16-bitových mikrokontroléru zastupuje součástky s výpočetním výkonem od 16 do 40 MIPS, délkou slova instrukce až 24-bitů, taktovacím kmitočtem do

80MHz, pamětí programu do 512Kb a pamětí dat do 32Kb. Jsou určeny pro náročnější aplikace jako signálové zpracování, řízení grafických displejů, síťové řídící prvky apod. Základna možných příslušenství je oproti 8-bitovým mikrokontrolérům významně rozšířena.

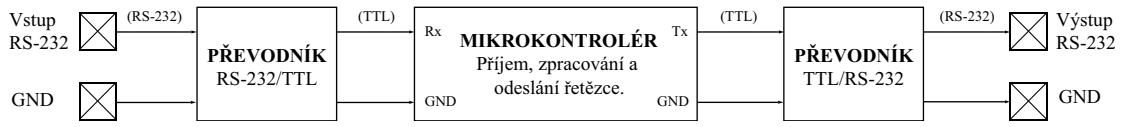
Vrcholovou řadu z hlediska výpočetního výkonu i technologie představují 32-bitové mikrokontroléry. Výpočetní výkon 1,5 DMIPS/MHz (Dhrystone MIPS/MHz), taktovací kmitočet 80MHz.

Velkou konkurenční výhodu obecně všech produktů společnosti Microchip tvoří jejich „odborná podpora“. Na webu výrobce je možné najít obsáhlou knihovnu návodu jak řešit určitý technický problém tzv. aplikační poznámky (Z anglického: Application Note). Pro každý produkt je k dispozici několik specifikačních dokumentů tzv. datasheetů (Z anglického: Datasheet). V neposlední řadě také možnost volného stažení propracovaného aplikačního prostředí MPLAB pro vývoj programů pro všechny distribuované mikrokontroléry.

Kromě vývojového prostředí je pro práci s mikrokontroléry nutný programátor. Jedná se o zařízení, které program vytvořený na počítači nahraje do cílové součástky. Pro práci na praktické části bakalářské práce byl využit programátor PICQUICK od české společnosti ASIX s.r.o. Zejména kvůli nižším pořizovacím nákladům a podpoře širokého spektra mikrokontrolérů. Programátor pracuje s ovládacím softwarem UP rovněž od společnosti ASIX s.r.o.

3 MOŽNOSTI ŘEŠENÍ

Pro snadnější orientaci byla práce rozdělena na část hardwarovou, zaměřující se na výběr vhodných součástek a konstrukční provedení adaptéru a část softwarovou popisující obslužný program v jazyce symbolických adres. Cílem bylo vybrat pro každou z částí optimální řešení. Výchozím bodem řešení bylo vytvoření blokové struktury adaptéru (Obr.3.1) respektující požadavky zadání.



Obr. 3.1: Bloková struktura navrhovaného adaptéru.

3.1 Softwarová část

Funkci obslužného programu zařízení lze rozdělit na dvě dílčí části:

- příjem a vysílání,
- zpracování přijatých dat.

3.1.1 Přijímání a vysílání

Pro usnadnění sériově komunikace může být hardware mikrokontroléru vybaven obvodem USART (Univerzální synchronní asynchronní přijímač vysílač – Universal Synchronous Asynchronous Receiver Transmitter). V případě, že tomu tak není, je možné programově obvod USART emulovat. V některých případech je obvod USART označován obecně jako sériové komunikační rozhraní SCI (Serial Communication Interface). Asynchronní komunikaci dle zadání lze při užití mikrokontroléru realizovat čtyřmi způsoby.

Vstup:software - výstup:software

Plnou aplikační emulací asynchronního komunikačního rozhraní je umožněno využití hardwarově jednoduššího mikrokontroléru za cenu vyšší složitosti obslužného programu. Příjem či odesílání dat je podmíněno řízením všech částí vykonávaných procesů, snižuje se tak výpočetní kapacita dostupná pro další části obslužného programu. Návod k vytvoření nutných procedur je možné nalézt např. v literatuře [10]

nebo v aplikační poznámce [7]. Pin mikrokontroléru určený pro vstup asynchronního signálu je označován *RX*, pin pro výstup je označen *TX*.

Při emulaci přijímače je nutné zajistit následující úkony:

- nastavení časovače (např. *TMR0*) na bitovou periodu přijímaného signálu a nastavit přerušení při jeho přetečení,
- nastavení přerušení při změně úrovně na vstupním pinu *RX* pro detekci start bitu,
- vytvořit proceduru pro spuštění čítače *TMR0* při přerušení pinu *RX*,
- vytvořit proceduru pro vzorkování a správné ukládání hodnoty vstupního pinu do proměnné tzv. bufferu vždy, když nastane přerušení čítače TMR0.

Pro softwarem emulovaný asynchronní vysílač je nutno zajistit:

- nastavení časovače např. *TMR0* na bitovou periodu vysílaného signálu a nastavit přerušení při jeho přetečení,
- vytvořit proceduru která spustí čítač *TMR0*, odešle start bit a dále bude znovu nastavovat hodnotu *TMR0* a odesílat jednotlivé bity bufferu na výstupní pin *TX* vždy při přerušení přetečením čítače TMR0. Po ukončení přenosu dat z bufferu musí procedura vyslat stop bit.

Správné nastavení časovače, je základní předpoklad přijímání či vysílání s nulovou chybovostí. Pro závislost bitové periody a přenosové rychlosti platí vztah:

$$T_b = \frac{1}{f_b}$$

, kde T_b označuje bitovou periodu a f_b přenosovou rychlost. Hodnota bitové periody při přenosové rychlosti 9600 Bd je:

$$T_b = \frac{1}{f_b} = \frac{1}{9600} = 104,16\mu s.$$

Například při uvažování mikrokontroléru řady PIC16F87X je časovač s nepředřazeným „předděličem“ dle literatury [9] [12] inkrementován s frekvencí $f_{OSC}/4$, kde f_{OSC} značí taktovací kmitočet. Množina hodnot času, kterou je čítač schopen indikovat z tohoto důvodu není spojitá, ale nabývá pouze diskrétních hodnot. Aby bylo dosaženo co možná nejpřesnějšího nastavení bitové periody, je nutná aproximace na nejbližší diskrétní hodnotu. Při taktovacího kmitočtu 4 MHz je čítač inkrementován s periodou:

$$T_c = \frac{1}{\frac{f_{OSC}}{4}} = \frac{1}{\frac{4*10^6}{4}} = 1\mu s.$$

Skutečné bitová perioda $T_b = 104,16\mu s$ je aproximována na hodnotu $T'_b = 104\mu s$. Přenosová rychlost se aproximací změní z 9600 Bd na:

$$f'_b = \frac{1}{T'_b} = \frac{1}{104 * 10^{-6}} = 9615,38Bd,$$

což představuje relativní chybu přenosové rychlosti způsobenou aproximací:

$$\delta_a = \left(\frac{f'_b - f_b}{f_b}\right) * 100 = \left(\frac{9615,38 - 9600}{9600}\right) * 100 = 0,16\%.$$

Nulové hodnoty relativní chyby lze dosáhnout pouze při taktovacím kmitočtu odpovídajícímu celočíselnému násobku přenosové rychlosti. Pro přenosovou rychlost 9600 Bd se jedná např. o taktovací kmitočet 3.6864MHz. Za mezní hodnotu nepřesnosti nastavení přenosové rychlosti je považována relativní chyba 5%.

Při vytváření procedur je však nutné vzít v potaz nejen zmíněnou bitovou periodu, ale také časovou náročnost provádění jednotlivých instrukcí, které zabezpečují softwarovou emulaci a provést příslušnou korekci hodnoty časovače.

Výhodou této varianty je úplná kontrola přenosového protokolu. Jednoduše je možné změnit počet přijímaných či vysílaných bitů v jednom znaku, pokud je to nutné.

Vstup:USART - výstup:USART

Využití hardwarové podpory sériové asynchronní komunikace je umožněno obvodem USART. Blokové schéma přijímače USART je uvedeno na Obr.3.2.

Data vstupují do přijímače z pinu *RX/DT*, který je v případě nastavení bitu *SPEN* spojen s blokem obnovy dat. Blok obnovy dat zahájí přijímání znaku se sestupnou hranou prvního bitu (start bitu). Je odpočítána polovina bitové periody a v tomto čase je opětovným vzorkováním ověřeno, že se skutečně jedná o start bit. Pokud navzorkovaná hodnota neodpovídá logické úrovni start bitu, je přijímání znaku zrušeno a blok obnovy dat přejde do stavu kontroly vstupního signálu na výskyt sestupné hrany. Po úspěšném ověření start bitu je blokem obnovy dat odpočítán čas jedné bitové periody a v tomto čase je signál vzorkován. Výsledek vzorkování (logická 1 nebo 0) je přesunut registru *RSR* (Receive Shift Register). Tento proces je opakován do chvíle, kdy jsou do registru *RSR* přesunuty všechny bity přijímaného znaku. Následuje ověření konzistence rámce vzorkováním stop bitu. Nesprávná úroveň stop bitu indikuje chybu rámce a je nastaven bit *FERR*. Dokončením příjmu stop bitu jsou datové bity znaku přesunuty z registru *RSR* do dvouúrovňového registru typu FIFO (First-In First-Out) označovaného *RCREG*. Přenos je indikován nastavením bitu *RCIF*, který v případě současného nastavení povolovacího bitu *RCIE* zahájí přerušení. Čtením registru *RCREG* je vyzvednut vždy nejdříve přijatý znak. Při nevyzvednutí dvojice znaků z registru, dojde po příchodu znaku třetího k jeho přetečení a v důsledku toho k zablokování přijímače.

Pro správnou funkci přijímacího bloku USART jako asynchronního přijímače je nutné nastavit:

- řídící přijímací registr *RCSTA*,
- bitovou periodu přijímaného signálu (registry *SPBRG*, *TXSTA*).

Nastavení periody generátoru bitové periody se provádí pomocí konstanty vložené do registru *SPBRG* a nastavení dvojice řídících bitů *SYNC* a *BRGH* v registru *TXSTA*. Hodnotu konstanty pro vložení do registru *SPBRG* lze vypočítat dle Tab.3.1, kde f_b značí přenosovou rychlost a f_{osc} značí taktovací kmitočet mikrokontroléru.

Konfigurační bity		**Hodnota pro nastavení do SPBRG**
SYNC	BRGH	
0	0	[(f_{osc} / f_b)/64] - 1
0	1	[(f_{osc} / f_b)/16] - 1

Tab. 3.1: Vztahy pro výpočet hodnoty konstanty pro nastavení *SPBRG* [13].

Například pro taktovací kmitočet 4MHz, *BRGH*=1 a přenosovou rychlost 9600Bd lze dle vztahu uvedeného v Tab.3.1 vypočíst:

$$SPBRG = \frac{f_{osc}}{\frac{f_b}{16}} - 1 = \frac{4 * 10^6}{\frac{9600}{16}} - 1 = 25,041.$$

Do registru *SPBRG* je možné nastavit pouze celá čísla, je tedy nutné zaokrouhlit vypočtenou hodnotu konstanty na 25 celých. Při nastavení této hodnoty do *SPBRG* registru je skutečně nastavená přenosová rychlost [13]:

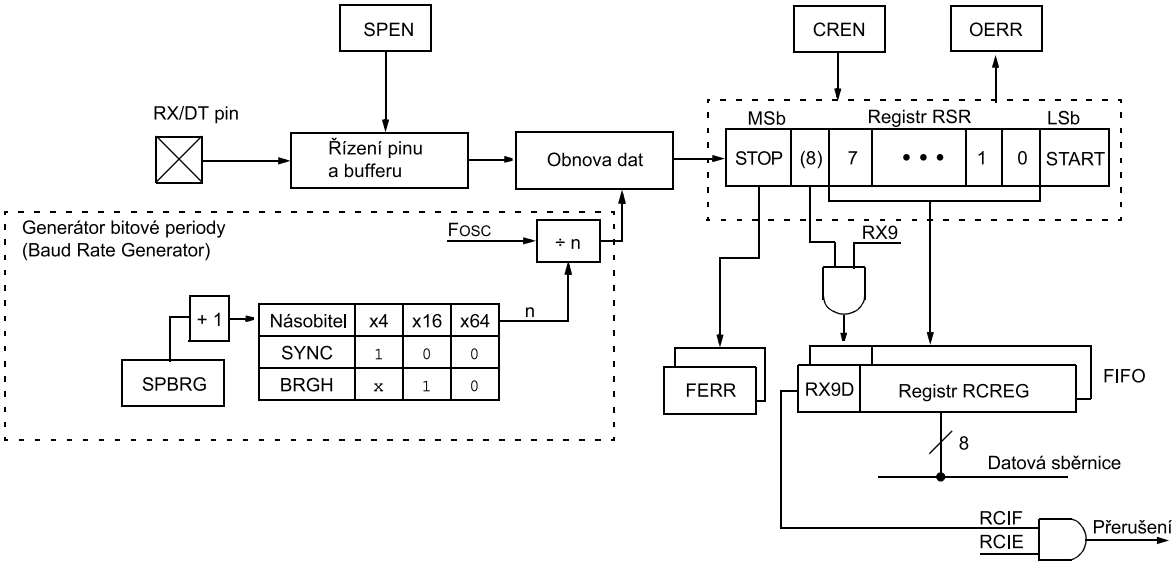
$$f'_b = \frac{f_{osc}}{16 * (SPBRG + 1)} = \frac{4 * 10^6}{16 * (25 + 1)} = 9615,38Bd.$$

Rozdíl požadované přenosové rychlosti a skutečně nastavené lze vyjádřit relativní chybou:

$$\delta_a = \left(\frac{f'_b - f_b}{f_b}\right) * 100 = \left(\frac{9615,38 - 9600}{9600}\right) * 100 = 0,16\%.$$

Dokumentace k jednotlivým typům mikrokontrolérů obsahuje tabulky s hodnotami konstant vhodných pro nastavení do registru *SPBRG* pro typické hodnoty taktovacích kmitočtů a přenosových rychlostí. Vypočtená hodnota se shoduje s uvedeným údajem v literatuře [13].

Funkce vysílače obvodu USART je patrná z blokového diagramu na Obr.3.3. Znak, který má být odeslán, je vložen do vysílacího bufferu - registru *TXREG*.



Obr. 3.2: Blokové schéma přijímače USART mikrokontroléru PIC16F13 [13].

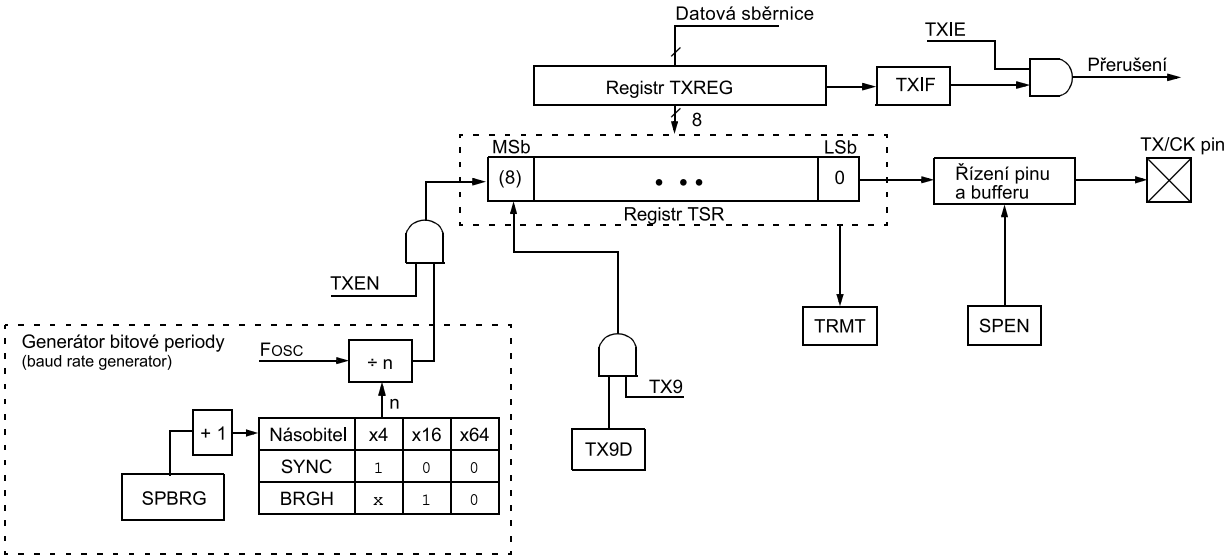
Pokud právě neprobíhá vysílání, je znak ihned přesunut do posuvného vysílacího registru *TSR*. Přesun indikuje nastavení bitu *TXIF* registru *PIR1*. V případě, že probíhá vysílání, čeká datová informace v podobě znaku na vyprázdnění *TSR* a až poté je do něho přesunuta. Po vložení znaku do registru *TSR* je vyslán start bit a za ním následují přenášená data (znak). Vysílání je ukončeno stop bitem. Vysílač USART s přijímačem sdílí registr *SPBRG* a bity *SYNC* a *BRGH* pro nastavení generátoru bitové periody. Z toho vyplývá nemožnost vysílat a přijímat data s rozdílnou přenosovou rychlostí současně.

Vstup:USART - výstup:software

Tato varianta je kombinací hardwarového přijímače obvodu USART a softwarem emulovaného vysílače. Po plném využití obvodu USART jde o variantu, která nejméně výkonově zatěžuje mikrokontrolér. Výhodou oproti přijímání a vysílaní pomocí obvodu USART (shodně s první variantou) je možnost nastavit počet vysílaných bitů v jednom znaku.

Vstup:software - výstup:USART

Kombinace softwarového přijímání a vysílání pomocí obvodu USART. Z hlediska řešeného úkolu nemá tato varianta v porovnání s ostatními žádné výhody.



Obr. 3.3: Blokové schéma vysílače USART mikrokontroléru PIC16F13 [13].

Výběr vhodné varianty

Z hlediska aplikačního využití je nejjednodušší realizovat zařízení s mikrokontrolérem obsahující hardwarovou podporu sériového protokolu. Pro realizaci v adaptéru byla zvolena varianta přijímání i vysílání pomocí obvodu UART. Jedná se bezesporu o nejspolehlivější variantu, která je jednoduše přenositelná na jiný typ mikrokontroléru. Případná změna přenosových rychlostí se provede jednoduchým přenastavením.

V případě, že by bylo nutné výstupní data z adaptéru kódovat dle určitého kódovacího schématu, bylo by vhodnější využít variantu hardwarového přijímaní a softwarového vysílání.

3.1.2 Zpracování řetězce

Zpracování informační části řetězce, která obsahuje číselnou informaci o změřené hodnotě vyžaduje převod jednotlivých znaků z formátu ASCII na tvar, který je pro zpracování v mikrokontroléru vhodnější. Řetězec je výhodné převést do jednoho ze dvou formátů:

- BCD (Binárně kódované desítkové číslo – Binary Coded Decimal) formát,
- binární formát.

Pro řešení byl zvolen formát čísel s pevnou desetinnou čárkou, který je podstatně jednodušší pro implementaci, nevyžaduje příliš velký výpočetní výkon a lépe odpovídá požadavkům zadaní. Nejdříve je tedy provedena operace násobení korekční

konstantou a až následně je (v kroku před odesláním řetězce) vhodně umístěna desetinná čárka.

Algoritmus založený na přímém zpracování BCD

Princip algoritmu je znázorněn na vývojovém diagramu (viz.Obr.3.4). Číselná informace v řetězci je převedena z formátu ASCII do formátu BCD odečtením hexadecimálního čísla 30. Číslo 30 tvoří rozdíl mezi hexadecimální hodnotou čísla v ASCII a BCD (viz Tab.C.1). Každé BCD číslo je uloženo v samostatném registru, jedná se tedy o tzv. Unpacked formát. Čísla jsou označena:

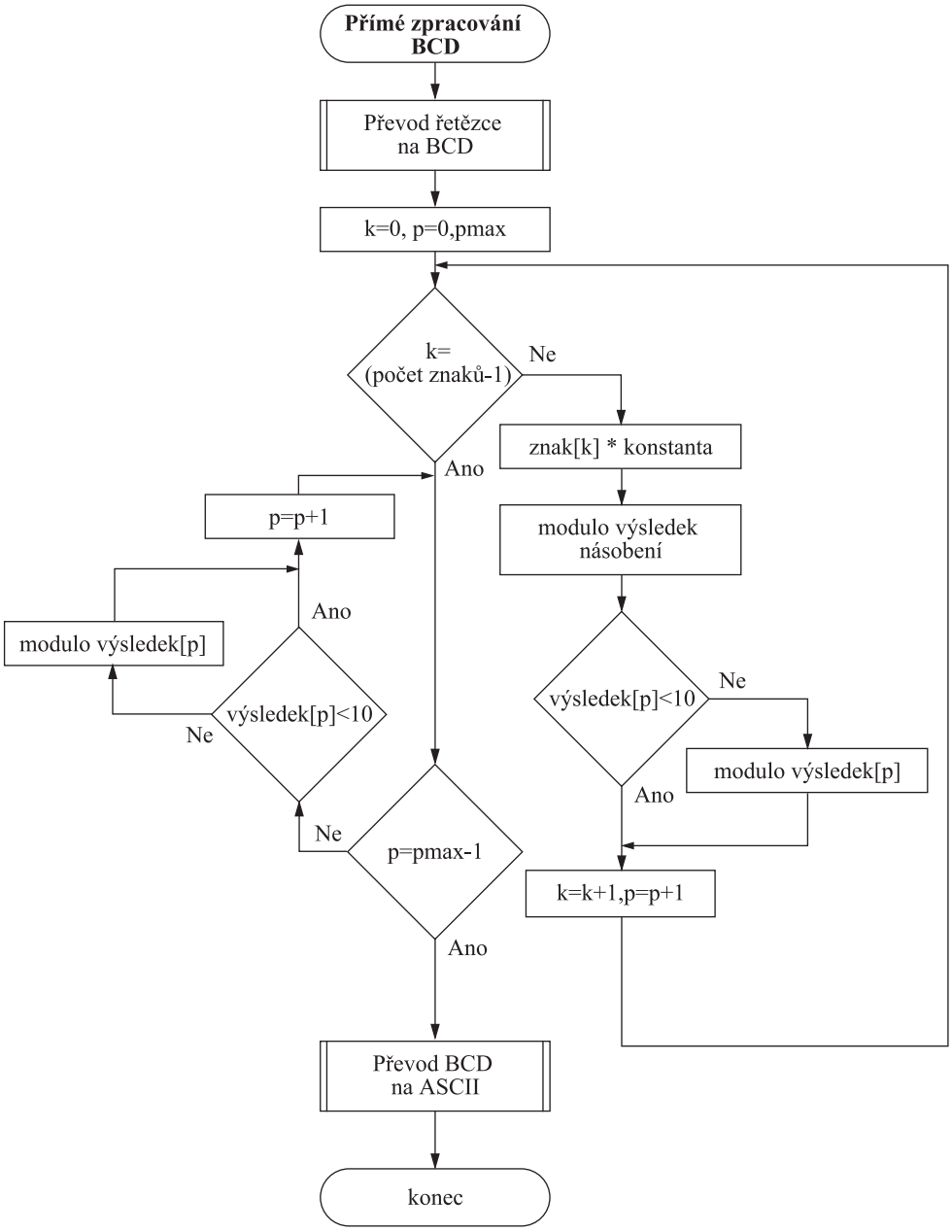
$$znak[0], znak[1], znak[2], ..., znak[k],$$

kde parametr k určuje pořadí přijatých čísel resp. dekadický řád čísla. Například $znak[0]$ označuje naposledy přijaté číslo resp. číslo s řádem 0 [1]. Znak[0] (například číslo 4) je vynásoben konstantou (například číslo 32), výsledek (číslo 128) je dočasně uložen v paměti. Proces pokračuje spuštěním procedury modulo, která výsledek násobení převede na jednotlivé BCD čísla a přičte je do příslušných paměťových míst. Maximální hodnota výsledku je $9*konstanta$. Výsledek (číslo 128) tedy bude rozdělen na čísla 1, 2 a 8. Tyto čísla jsou přičtena do registrů $vysledek[2]$, $vysledek[1]$ a $vysledek[0]$. Obecně do registru $vysledek[p]$, kde parametr p vyjadřuje dekadický řád rozkládaného čísla. Následuje kontrola, zda *výsledek[p]* zpracovávaného řádu (v tomto případě nultý) není větší než-li 9, v případě nesplnění podmínky je výsledek podroben znovu funkci modulo a jeho vyšší řád je přičten k registru *výsledek[p+1]*. V případě splnění podmínky jsou inkrementovány pomocné proměnné k a p. Celý proces se opakuje do splnění podmínky k = *(počet znaků-1)*, kde *počet znaků* označuje počet přijatých ASCII čísel. Po splnění podmínky se již pouze vyrovnávají koeficienty vyšších řádů tak, aby žádný z nich neměl hodnotu vyšší než-li 9. Proces je zakončen splněním podmínky p = *pmax-1*, kde *pmax* označuje maximální možný řád dosažitelný při násobení. Například při násobení dekadického čísla o 5 řádech a dekadické konstanty o 3 řádech by měla proměnná hodnotu *pmax = 5 + 3 = 8*. Výsledkem algoritmu je řada BCD čísel

$$vysledek[pmax-1], vysledek[pmax-2], ..., vysledek[2], vysledek[1], vysledek[0]$$

uložených v samostatných registrech. V závěru je nutné tyto čísla převést zpět do formátu ASCII. Před vysláním je v řetězci na pozici desetinné čárky umístěno hexadecimální číslo 2C (viz.Tab.C.1).

[1] Z hlediska skutečné hodnoty přijatého čísla se nejedná řád 0 (10^0), nýbrž o řád 10^{-6}.



Obr. 3.4: Vývojový diagram algoritmu založeném na přímém zpracování BCD.

Algoritmus založený na převodu do binárního formátu

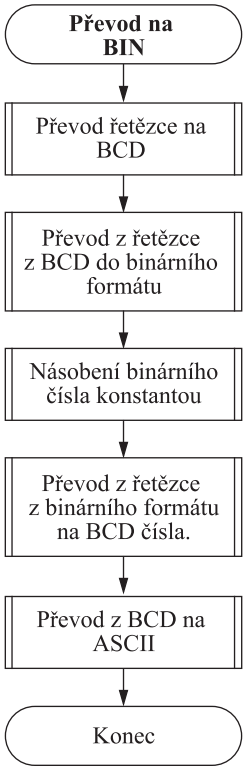
Vývojový diagram algoritmu je uveden na Obr.3.5. Číselná informace v přijatém řetězci je nejdříve převedena z ASCII formátu na formát BCD. Čísla jsou uložena v samostatných registrech a označena:

$$znak[0], znak[1], znak[2], ..., znak[k],$$

kde parametr k určuje pořadí přijatých čísel resp. dekadický řád čísla, například *znak*[0] označuje naposledy přijaté číslo resp. číslo s řádem 0 [2].

Pro výsledek převodu do binárního tvaru je vyhrazen dostatečný prostor. Pro jednoduchost označován jako akumulační registr. Proces začíná přičtením čísla s nejnižším řádem (znak[0]) do doposud nulového akumulačního registru. Následně je registr vynásoben dekadickým číslem deset. Což lze jednoduše realizovat bitovým posunem a sčítáním. Do akumulačního registru je poté přičteno číslo s druhým nejnižším řádem (znak[1]), následně je opět akumulační registr vynásoben deseti. Celý proces se cyklicky opakuje do chvíle, kdy jsou ke akumulačnímu registru přičteny všechny číselné znaky. Výsledkem tohoto kroku je binární číslo vyjadřující přijatou číselnou hodnotu.

Dalším krok tvoří vynásobení binárního čísla korekční konstantou. Jeho výsledkem je také binární číslo vyjadřující výsledek po násobení. Po násobení následuje



Obr. 3.5: Vývojový diagram algoritmu založeném na převodu do binárního formátu.

převod z binárního tvaru do BCD. Princip převodu je založen na závislosti desítkové a hexadecimální řady vyjadřující totožné číslo [8]. Libovolné číslo vyjádřené čtyřmi hexadecimálními číslicemi lze zapsat jako:

$$H = h\_3 * 16^3 + h\_2 * 16^2 + h\_1 * 16^1 + h\_0 * 16^0,$$

[2] Z hlediska skutečné hodnoty přijatého čísla se nejedná řád 0 (10^0), nýbrž o řád 10^{-6}.

kde H označuje hodnotu hexadecimálního čísla a prvky h_3 až h_0 označují hexadecimální číslice vyjadřovaného čísla. Totožné hexadecimální číslo lze po převodu do desítkové soustavy vyjádřit jako:

$$D = d\_4 * 10^4 + d\_3 * 10^3 + d\_2 * 10^2 + d\_1 * 10^1 + d\_0 * 10^0,$$

kde D označuje hodnotu dekadického čísla a prvky d_4 až d_0 označují desítkové číslice vyjadřovaného čísla. Jednotlivé složky hexadecimálního čísla lze roznásobit jako:

$$h\_0 * 16^0 = h\_0 * 1 = h\_0 * 10^0,$$

$$h\_1 * 16^1 = h\_1 * 16 = h\_1 * 10^1 + 6 * h\_1 * 10^0,$$

$$h\_2 * 16^2 = h\_2 * 256 = 2 * h\_2 * 10^2 + 5 * h\_2 * 10^1 + 6 * h\_2 * 10^0,$$

$$h\_3 * 16^3 = h\_3 * 4096 = 4 * h\_3 * 10^3 + 9 * h\_3 * 10^1 + 6 * h\_3 * 10^0.$$

Pokud číslice vyjadřující příslušné dekadické řády sečteme, dostáváme vztahy:

$$d\_0' = h\_0 + 6 * h\_1 + 6 * h\_2 + 6 * h\_3,$$

$$d\_1' = h\_1 + 5 * h\_2 + 9 * h\_3,$$

$$d\_2' = 2 * h\_2,$$

$$d\_3' = 4 * h\_3,$$

$$d\_4' = 0.$$

Takto vyjádřená dekadické čísla mohou nabývat hodnot vyšších než 9. Pro úpravu na čísla která budou menší nebo rovny 9, je nutné každé z nich podrobit dělením modulo 10 a následně zbytek přičíst číslu o řád vyššímu číslu. Tento postup znázorňují následující rovnice:

$$d\_0 = d\_0' mod 10,$$

$$p\_0 = (d\_0' - d\_0' mod 10)/10$$

$$d\_1 = (d\_1' + p\_0) mod 10,$$

$$p\_1 = [(d\_1' + p\_0) - (d\_1' + p\_0) mod 10]/10$$

$$d\_2 = (d\_2' + p\_1) mod 10,$$

$$p\_2 = [(d\_2' + p\_1) - (d\_2' + p\_1) mod 10]/10$$

$$d\_3 = (d\_3' + p\_2) mod 10,$$

$$p\_3 = [(d\_3' + p\_2) - (d\_3' + p\_2) mod 10]/10$$

$$d\_4 = d\_4' + p\_3,$$

kde d_4 až d_0 vyjadřují desítkové číslice, p_4 až p_0 vyjadřují zbytky po dělení modulo 10 dělené deseti. V závěru je nutné realizovat převod čísel d_4 až d_0 z BCD formátu na formát ASCII. Posledním krokem je správné umístění desetinné čárky v podobě hexadecimálního čísla 2C.

Popsaný princip převodu z binárního do BCD formátu je co se týče rozsahu převáděného čísla zjednodušen.

Výběr vhodné varianty

V rámci zpracování práce byly realizovány obě dvě zmíněné varianty. Při porovnaní času zpracovaní byly algoritmy téměř srovnatelné a vyhovovaly zadaní. Výhodou algoritmu s převodem do binárního tvaru je snadnější upravitelnost v případě, že bude nutné výstupní informaci adaptéru před odesláním kódovat, nebo jiným způsobem upravit. Pro využití v adaptéru byla proto zvolena tato varianta.

3.2 Hardwarová část

3.2.1 Vhodné typy mikrokontrolérů PIC

Z hlediska výpočetního výkonu jsou pro řešení zcela dostačující 8-bitové mikrokontroléry. Tato řada se dále dělí na tři podskupiny: Baseline, Mid-Range, High-Performance.

Řada Baseline zastupuje nejjednodušší mikrokontroléry s taktovacím kmitočtem do 4MHz a velmi omezeným příslušenstvím. Instrukční sada obsahuje 33 instrukcí a jejich šířka je 12 bitů. Předností je nízká cena a fyzické rozměry. Řada Mid-Range představuje výkonový střed 8-bitových mikrokontrolérů s komunikačním rozhraním v podobě SCI či UART a dalším příslušenstvím. Instrukční sada obsahuje 35 instrukcí a jejich šířka je 14 bitů. Výkonově nejvyšší je řada High-Performance. Instrukční sada obsahuje 75 až 83 instrukcí a jejich šířka je 16 bitů. Mikrokontroléry této řady obsahují rozšiřující příslušenství např. podporu USB.

Pro aplikaci ve vytvářeném adaptéru je nejvhodnější volit mikrokontroléry z řady Mid-Range. Výběr z této řady byl proveden na základě následujících požadavku:

- paměť programu typu Flash o kapacitě min. 1024 slov,
- paměť dat min. 128 bytů,

- USART,
- paměť EEPROM.

Požadavek na velikost paměti programu a dat je vztažen k velikosti obslužného programu a počtu používaných proměnných. Rozhraní USART bude využito pro sériovou komunikaci. Paměť EEPROM bude využita pro uložení kalibrační konstanty a řídících konstant.

Pro výběr mikrokontrolérů dle specifikovaných požadavků byla využita webová aplikace společnosti Microchip zvaná MAPS (Microchip Advanced Part Selector). Specifikovaným požadavkům odpovídá 33 mikrokontrolérů. Stručný přehled je uveden v Tab.3.2.

PIC16F6XX a PIC16F6XXA: PIC16F627A, PIC16F628A, PIC16F687, PIC16F688, PIC16F648A, PIC16F690, PIC16F689.
PIC16F7XX a PIC16F7X: PIC16F727, PIC16F777, PIC16F77, PIC16F726, PIC16F767, PIC16F76, PIC16F747, PIC16F724, PIC16F74, PIC16F737, PIC16F723, PIC16F73, PIC16F722.
PIC16F87XA a PIC16F8XX a PIC16F8X: PIC16F887, PIC16F886, PIC16F884, PIC16F883, PIC16F87, PIC16F88, PIC16F882, PICF873A.
PIC16F9XX: PIC16F946, PIC16F917, PIC16F916, PIC16F914, PIC16F13.

Tab. 3.2: Mikrokontroléry vhodné pro použití v adaptéru.

Pro použití v adaptéru byl zvolen mikrokontrolér PIC16F13. Základní parametry tohoto mikrokontroléru jsou:

- paměť programu typu Flash o kapacitě 4096 slov,
- paměť dat 256 bytů,
- USART,
- paměť EEPROM 256 bytů.

Jednou z doplňkových funkcí je tzv. Fail-Safe clock monitor. Tato funkce umožňuje monitorování funkce externího taktovacího oscilátoru. V případě, že je jeho chod vyhodnocen jeho nepřesný, zastoupí ho interní přesně kalibrovaný oscilátor. Zatímco externí oscilátor je možno resetovat.

3.2.2 Podpůrné obvody mikrokontroléru

Oscilátor

Modul oscilátoru zvoleného mikrokontroléru lze nastavit do jednoho z osmi podporovaných stavů [13]:

1. *EC* - připojení externího zdroje taktovacího kmitočtu na vstupu OSC1/CLKIN s dostupností na výstupu OSC2/CLKOUT,
2. *LP* - připojení vnějšího krystalového nebo keramického oscilátoru do 32kHz,
3. *XT* - připojení vnějšího krystalového nebo keramického oscilátoru do 4MHz,
4. *HC* - připojení vnějšího krystalového nebo keramického oscilátoru do 20MHz,
5. *RC* - připojení externího RC oscilátoru na vstup OSC1/CLKIN s výstupem $f_{osc}/4$ na OSC2/CLKOUT,
6. *RCIO* - připojení externího RC oscilátoru na vstup OSC1/CLKIN a bez vnějšího výstupu,
7. *INTOSC* - interní oscilátor s $f_{osc}/4$ na výstupu OSC2/CLKOUT,
8. *INTOSCIO* - interní oscilátor bez vnějšího výstupu.

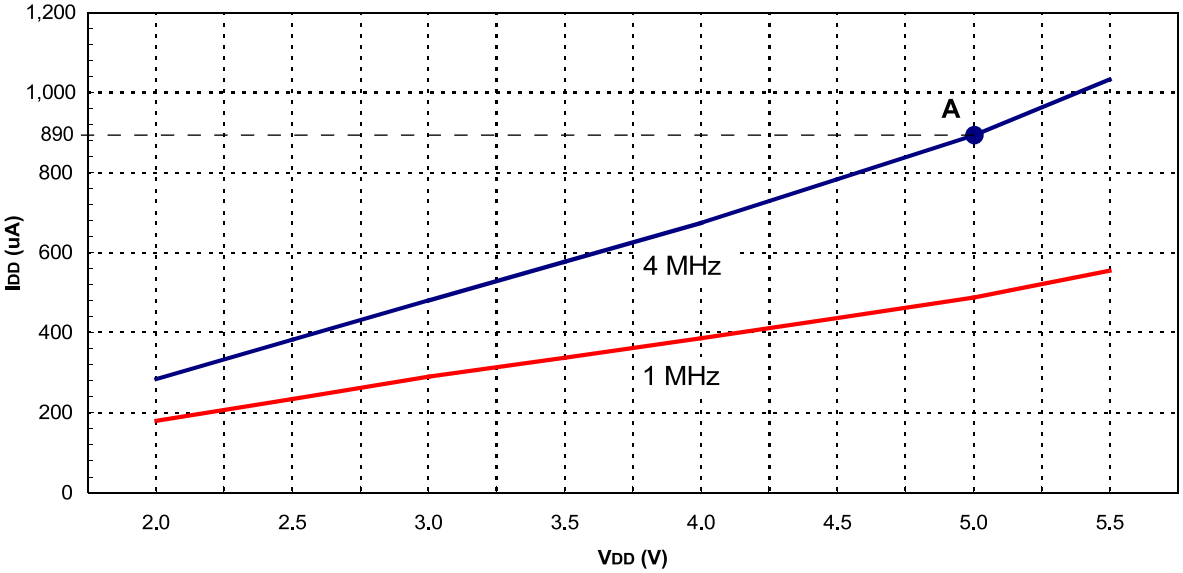
RC oscilátory se obecně vyznačují nízkou stabilitou, nepřesností nastavení taktovacího kmitočtu a relativně vysokým proudovým odběrem [27]. Pro použití v adaptéru byl proto zvolen externí krystalový oscilátor.

Oscilační kmitočet přímo ovlivňuje proudový odběr mikrokontroléru (viz. Obr.3.6). Byl proto zvolen kmitočet 4MHz (stav XT), který je kompromisem mezi výpočetním výkonem mikrokontroléru a jeho proudovým odběrem. Předpokládaný teoretický proudový odběr mikrokontroléru, který činí 890μA je vyznačen na Obr.3.6.

Převodníky TTL/RS232

Mikrokontroléry při komunikaci pracují s napěťovými úrovněmi TTL (5V). Komunikace dle standardu RS-232 však probíhá na úrovních odlišných a tedy nekompatibilních. Pro zajištění kompatibility je nutné využít převodník TTL/RS232. Vhodným řešením je převodník ve formě integrovaného obvodu.

Na dnešním trhu lze nalézt integrované převodníky od společností Intersit, STMicroelectronics, Maxim IC, Microchip a dalších. Sestávají se z jednoho nebo několika převodníků TTL na RS232 (ovladač) a RS232 na TTL (přijímač). Běžně se vyrábějí součástky se dvěma ovladači a dvěma přijímači. Pro konstruované zařízení je tento



Obr. 3.6: Závislost odebíraného proudu na napájecím napětí při taktovacích kmitočtech 1 a 4 MHz ve stavu XT [13].

počet víc než dostačující, bude využita pouze polovina. Na trhu je možné nalézt součástky pouze s jedním ovladačem a jedním přijímačem. Cena těchto integrovaných obvodu je však paradoxně vyšší.

Vlastnosti jednotlivých integrovaných převodníků se od sebe příliš neliší. Jejich přehled uveden v Tab.3.3. Z důvodu dostupnosti a nižší spotřeby byl pro použití v zařízení vybrán obvod MAX232 od výrobce Maxim IC.

Při srovnání bylo čerpáno z literatury [14] [22] [23] [24] [25] [26].

Název	**Výrobce**	**Napájecí napětí**	**Napájecí proud (typ.)**	**Ovladač**	**Přijímač**
HIN232	Intersit	5V	5mA	2x	2x
ICL232	Intersit	5V	5mA	2x	2x
ST232	STMicroelectronics	5V	5mA	2x	2x
MAX232	Maxim IC	5V	4mA	2x	2x
TC232	Microchip	5V	5mA	2x	2x
MAX232	Texas instrument	5V	8mA	2x	2x

Tab. 3.3: Srovnání použitelných typů převodníků TTL/RS232.

Stabilizátory napájecího napětí

Většina mikrokontroléru PIC včetně zvoleného využívá napájecí napětí 5V a nižší. Jelikož bude konstruované zařízení napájeno napětím 7,5V, je nutné využít stabilizátor. Na trhu s pozitivními stabilizátory nízkého napětí dnes dominuje STMicroelectronics, proto je výběr proveden z nabídky této společnosti.

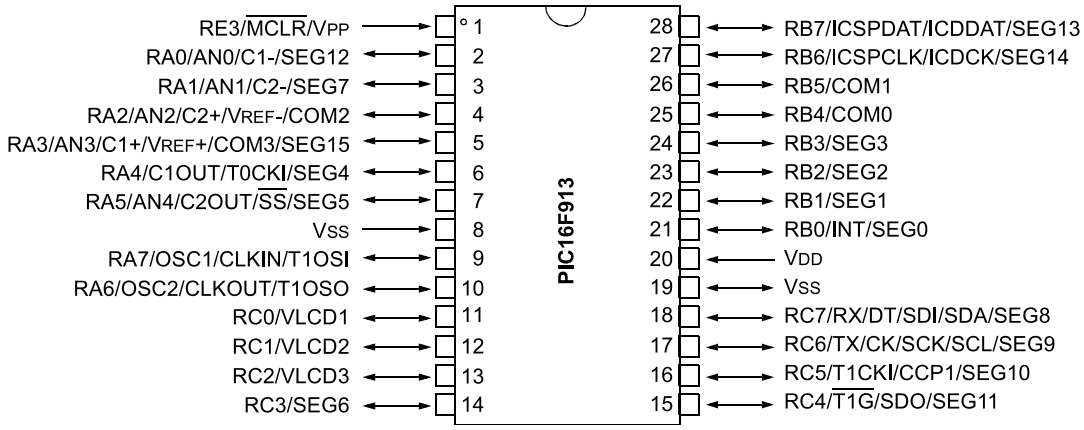
V České republice je běžně dostupný např. 5V stabilizátor 78L05 nebo jeho inovovaná a několikanásobně dražší varianta LE50. Tyto součástky jsou schopny (pří použití chladiče) do zařízení dodávat proud až 100mA. Což plně dostačuje zamýšlenému využití. Mikrokontrolér i převodník MAX232 mají malý proudový odběr v řádu jednotek mA. Stabilizátor LE50 má výrobcem udávanou přesnost nastavení napětí ±2% [19]. U 78L05 výrobce uvádí ±4% [18]. Oba integrované obvody jsou pro konstrukci použitelné. S přihlédnutím na dostupnost (SMD varianty) a nižší cenu byl pro konstrukci adaptéru vybrán stabilizátor 78L05 v pouzdře SO-8.

4 KONSTRUKCE HARDWAROVÉ ČÁSTI

4.1 Použité integrované obvody

4.1.1 Mikrokontrolér PIC16F913

Zapojení vývodu zvoleného mikrokontroléru je uvedeno na Obr.4.1. Na vývod *VDD* je nutno přivést napájecí napětí v rozmezí 2V až 5,5V. Vývod *VSS* je nutno uzemnit. Na obvodový reset - vývod $\overline{MCLR}$ - je nutno přivést napětí odpovídající logické 1. Z hlediska konstrukce budou využity vývody *TX* a *RX* pro sériovou komunikaci. Dále vývody *ICDDAT*, *ICDCK*, *VPP*, které slouží k ICSP (Sériové programování v cílovém obvodu – In Circuit Serial Programming) programování. Na vývody *OSC1* a *OSC2* bude připojen krystalový oscilátor.



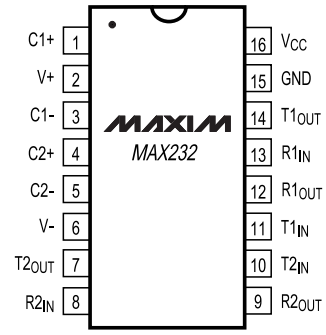
Obr. 4.1: PIC16F913 - zapojení vývodů pouzdra SOIC-28 [13].

4.1.2 Převodník MAX232

Zapojení vývodů převodníku TTL/RS232 je uvedeno na Obr.4.2 A). Pro správnou funkci vyžaduje integrovaný obvod doplnění externích kondenzátorů viz. Obr.4.2 C). Výrobcem doporučené hodnoty jsou uvedeny na Obr.4.2 B). Pro konstrukci bude využit jeden ovladač - vývody *T1IN*, *T1OUT* a jeden přijímač - vývody *R1IN* a *R1OUT*.

4.1.3 Stabilizátor napájecího napětí 78L05

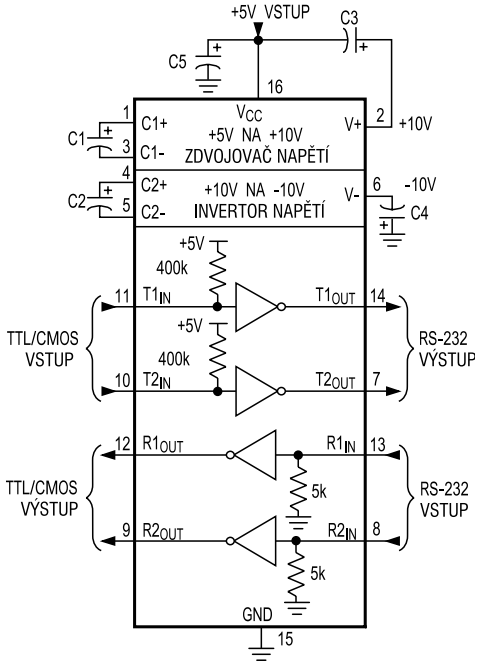
Zapojení vývodů stabilizátoru je uvedeno na Obr.4.3. Zdroj napájecího napětí je připojen na vstup *VIN*. Stabilizované napětí 5V pro napájení adaptéru je k dispozici na výstupu *VOUT*.



A)

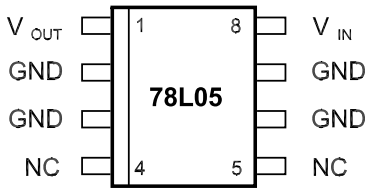
KAPACITY KONDENZÁTORŮ (μ F)					
	C1	C2	C3	C4	C5
MAX232	1.0	1.0	1.0	1.0	1.0

B)



C)

Obr. 4.2: MAX232 - A) Zapojení vývodů pouzdra, B) Doporučené hodnoty kapacit, C) Funkční schéma. [14].

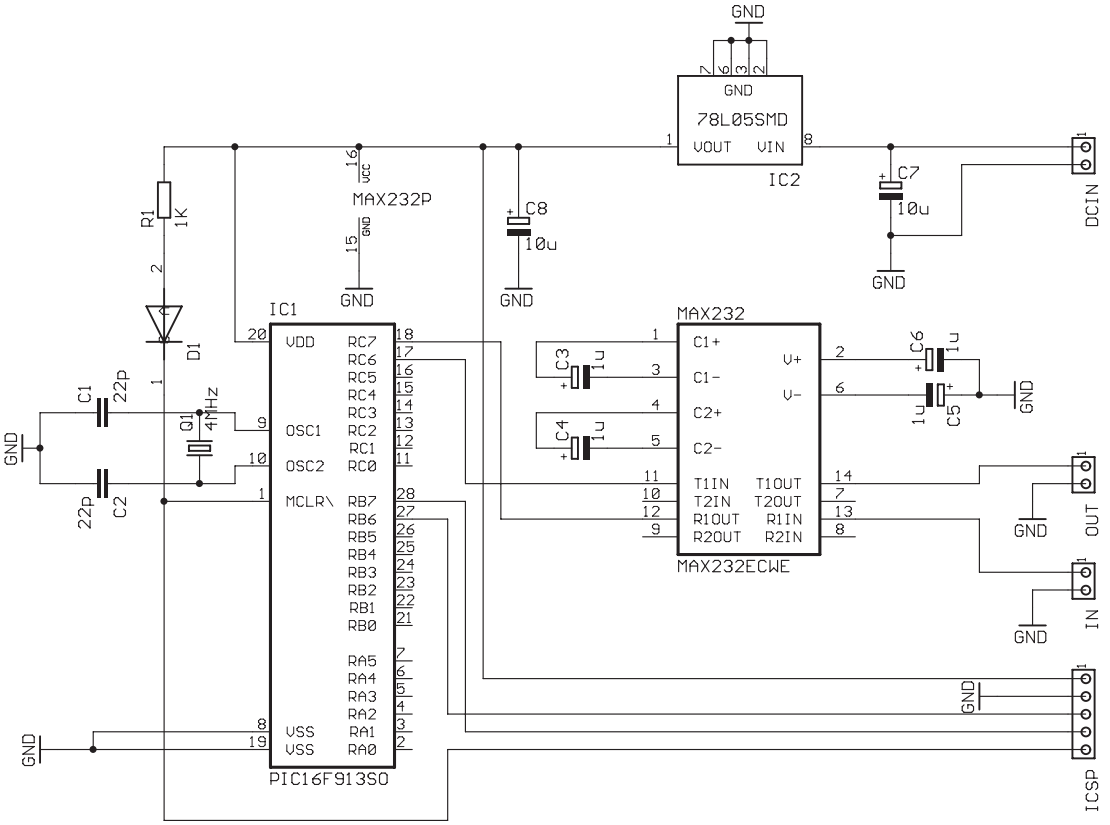


Obr. 4.3: 78L05 - Zapojení vývodů pouzdra S0-8 [18].

4.2 Schéma zapojení adaptéru

Schéma zapojení (Obr.4.4) bylo vytvořeno v programu EAGLE. Obsahuje výše zmíněné integrované obvody. Před a za stabilizátor byly zapojeny tantalové blokovací kondenzátory. Jejich použití v programu EAGLE vyžadovalo vytvoření pouzdra typu A dle rozměrů udávaných výrobcem [27]. Použitý mikrokontrolér rovněž nebyl obsažen v knihovně návrhového programu. Bylo proto nutné vytvořit součástku novou. Dle doporučení výrobce je mezi vývody $\overline{MCLR}$ a *VDD* zařazen rezistor 1kΩ. Při sériovém programování je na vývodu $\overline{VPP}$ napětí až 12V, vývod napájecího napětí mikrokontroléru a stabilizátor je vůči tomuto napětí chráněn diodou 1N4148. K vývodům *OSC1* a *OSC2* mikrokontroléru je dle doporučení výrobce zapojen krys-

talový oscilátor spolu s dvojicí keramických kondenzátorů hodnoty 22pF. Vývody *RX* a *TX* jsou spojeny s obvodem MAX232. MAX232 je zapojen s doporučovanými kondenzátory hodnoty 1μF (viz. Obr.4.2 B)). Pro napájecí napětí, vstupní a výstupní datové vodiče byly zvoleny patice konektorů typu PSH02 se dvěma vývody. Pro sériové programování byla zvolena patice stejného typu s pěti vývody.

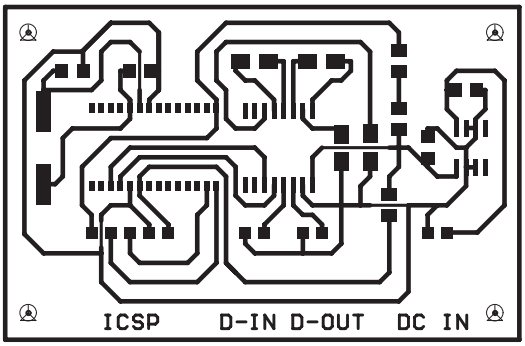


Obr. 4.4: Schéma zapojení adaptéru.

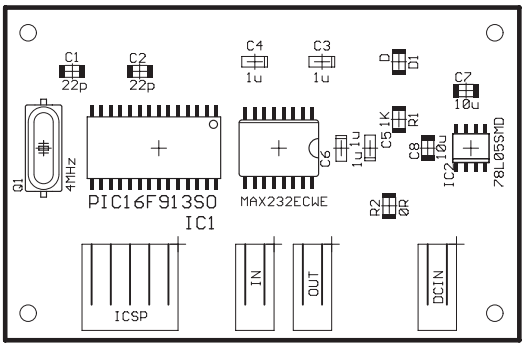
Ze schématu byla vytvořena deska plošných spojů (Obr.4.5). Šířka běžných spojů byla zvolena 0,6mm. Pouze spoje mezi vývody pasivních součástek mají šířku 0,3mm. Pro přemostění jednoho křížení byl využit nulový rezistor. Z důvodu předpokládané ruční montáže jsou rozestupy mezi součástkami záměrně větší. Deska plošných spojů je z jednostranně plátované mědi. Má velikost 44x68mm. Pro vyloučení mikroskopických přerušení byly všechny spoje ohmicky proměřeny.

Umístění jednotlivých součástek je uvedeno na osazovacím schématu na Obr.4.6. Veškeré zvolené součástky jsou typu SMD. Pro zvýšení mechanické pevnosti kontaktů byly patice konektorů umístěny na spodní stranu DPS. Pájení bylo provedeno na běžné hrotové pájecí stanici. Seznam součástek je uveden v Tab.4.1.

Na Obr.4.7 je zobrazena finální podoba realizovaného adaptéru. Při napájecím napětí 7,5V zařízení odebírá ze zdroje proud 5,25mA.



Obr. 4.5: Návrh desky plošných spojů. Pohled z hora.

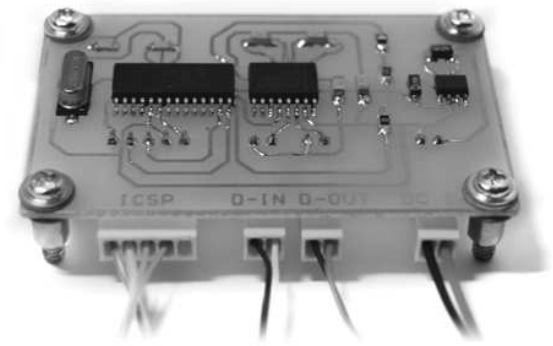


Obr. 4.6: Osazovací schéma. Pohled z hora.

4.3 Kompatibilita zapojení s jinými typy PIC

Jelikož ceny mikrokontrolérů plně podléhají zákonu trhu, tedy nabídky a poptávky. Cena konkrétního typu se může v průběhu času zvyšovat zatímco cena jiného typu může naopak klesat. Při návrhu zařízení je proto výhodné definovat mikrokontroléry, které mohou v zapojení konkrétně použitý typ zastoupit a umožnit tak případné snížení výrobních nákladů. Navržená DPS je kompatibilní například s mikrokontroléry uvedenými v Tab.4.2. Pro uvedené mikrokontroléry je však vždy nutné provést kontrolu a případnou úpravu obslužného programu dle konkrétní specifikace typu, který je plánován využít jako zástupný.

Při vypracování Tab.4.2 bylo využito literatury [12] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37].



Obr. 4.7: Realizované zařízení.

Označení ve schématu	Název součástky	Popis	Pozdro	Výrobce
C8, C7	CTS 10M/16V	Tantalový kondenzátor $10\mu F$	A	Shjinpei
C3, C4, C5, C6	CTS 1M/16V	Tantalový kondenzátor $1\mu F$	A	Shjinpei
Q1	Q 4MHZ-SMD	Krystal 4Mhz	HC49US	Auris
IC2	78L05 SMD	Stabilizátor 5V	SO-8	STM
MAX232	MAX232CWE	Převodník RS232/TTL	SO-16	Maxim IC
IC1	PIC16F913	Mikrokontrolér	SO-28	Microchip
D1	1N4148 SMD	Dioda	1206	TS
C1, C2	CK1206 22P/50V NPO	Keramický kondenzátor 22pF	1206	Yageo
DCIN, IN, OUT	PSH02-02WG	Konektorová patice - 2 piny	–	–
ICSP	PSH02-05WG	Konektorová patice - 5 pinů	–	–
R1	R1206 1K	Rezistor 1kΩ	1206	–
R0	R1206 0R	Nulový rezistor	1206	–

Tab. 4.1: Seznam všech součástek použitých pro konstrukci.

PIC16F7XX a PIC16F7X: PIC16F727, PIC16F726, PIC16F767, PIC16F76, PIC16F737, PIC16F723, PIC16F73, PIC16F722.
PIC16F8XX a PIC16F8X PIC16F886, PIC16F883, PIC16F882, PICF873A.
PIC16F9XX PIC16F916, PIC16F13.

Tab. 4.2: Pinově kompatibilní mikrokontroléry.

4.4 Úprava zapojení pro přípojení bezdrátového modulu

V provozu bude zařízení připojeno na bezdrátový modul. Z hlediska technologie lze vybírat mezi moduly WI-FI, ZigBee, Bluetooth nebo RF.

Naprostá většina modulů není kompatibilní s úrovněmi standardu RS-232 – nepoužívá záporné napětí. Podporované vstupní úrovně sériového signálu jsou do 3,3V.

Z tohoto důvodu by v případě, že by byl připojen bezdrátový modul, bylo nutné odstranit připojení výstupního pinu Tx mikrokontroléru na převodník MAX232. A připojit ho přímo na vstup bezdrátového modulu. Dále by bylo nutné snížit úrovně výstupního signálu mikrokontroléru na úrovně vyhovující bezdrátovému modulu. Což lze realizovat snížením napájecího napětí mikrokontroléru z 5V na 3,3V. Například použitím druhého stabilizátoru v napájecí větvi mikrokontroléru.

5 REALIZACE SOFTWAROVÉ ČÁSTI

Vývojový diagram hlavní části programu je uveden na Obr.5.2. Hlavní část představuje nekonečný cyklus, ve kterém se program nachází.

Program začíná spuštěním procedury *nacteni_konstant*, která slouží k prvotnímu vyčtením konstant z datové paměti EEPROM a jejich uložení do pomocných proměnných. K tomu je využita posloupnost příkazů uvedená v [13]. Mezi jednotlivými operacemi čtení bylo nutné zařadit čekací smyčky, bez kterých funkce nefungovala. Zdrojový kód procedury je uveden v příloze B.1. Datová paměť EEPROM obsahuje 16-bitovou korekční konstantu o dvou složkách (nižší a vyšší) a konstantu maximálního počtu přijatých číselných znaků.

Po načtení konstant je provedena inicializace. Což představuje uložení ukazatelů do pomocných proměnných a nastavení obvodu UART do asynchronního módu s přenosovou rychlostí 2400Bd.Následně je spuštěna procedura pro smazání proměnných.

Dále se program pohybuje v cyklu obsahujícím tyto hlavní procedury:

- *data_na_vstupu* – procedura pro příjem řetězce,
- *vstupni_uprava* – procedura pro vstupní úpravu řetězce,
- *prevod_BCD_bin* – procedura pro převod BCD řetězce na binární číslo,
- *mul_32x16* – procedura pro násobení korekční konstantou,
- *prevod_bin_BCD* – procedura pro převod binárního čísla na BCD řetězec,
- *vystupni_uprava* – procedura pro výstupní úpravu řetězce,
- *odeslani_retezce* – procedura pro odeslani retezce.

A dále obsahuje příkaz pro nulování čítače Watchdog, proceduru pro kontrolu stability externího oscilátoru, skok na inicializační část programu (tzv. restart programu).

Čítač Watchdog není nutné nastavovat, jelikož je na zvoleném typu mikrokontroléru při jeho povolení v konfiguračním slově *__CONFIG* automaticky nastaven na periodu cca 16ms [13].

Kromě procedur čtení a kontroly stability externího krystalu jsou všechny ostatní řízeny tzv. řídícími registry *PROCES_CONTROL* a *PROCES_CONTROL2*, jejich struktura je uvedena na Obr.5.1.

Registr	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PROCES_CONTROL	SC1	SC2	SC3	SC4	SC5	SC6	SC7	SC8
PROCES_CONTROL2	SC9	SC10	SC11	SC12	---	---	---	---

Obr. 5.1: Struktura řídících registrů.

Nastavení jednotlivých bitů (logická 1) indikuje následující události:

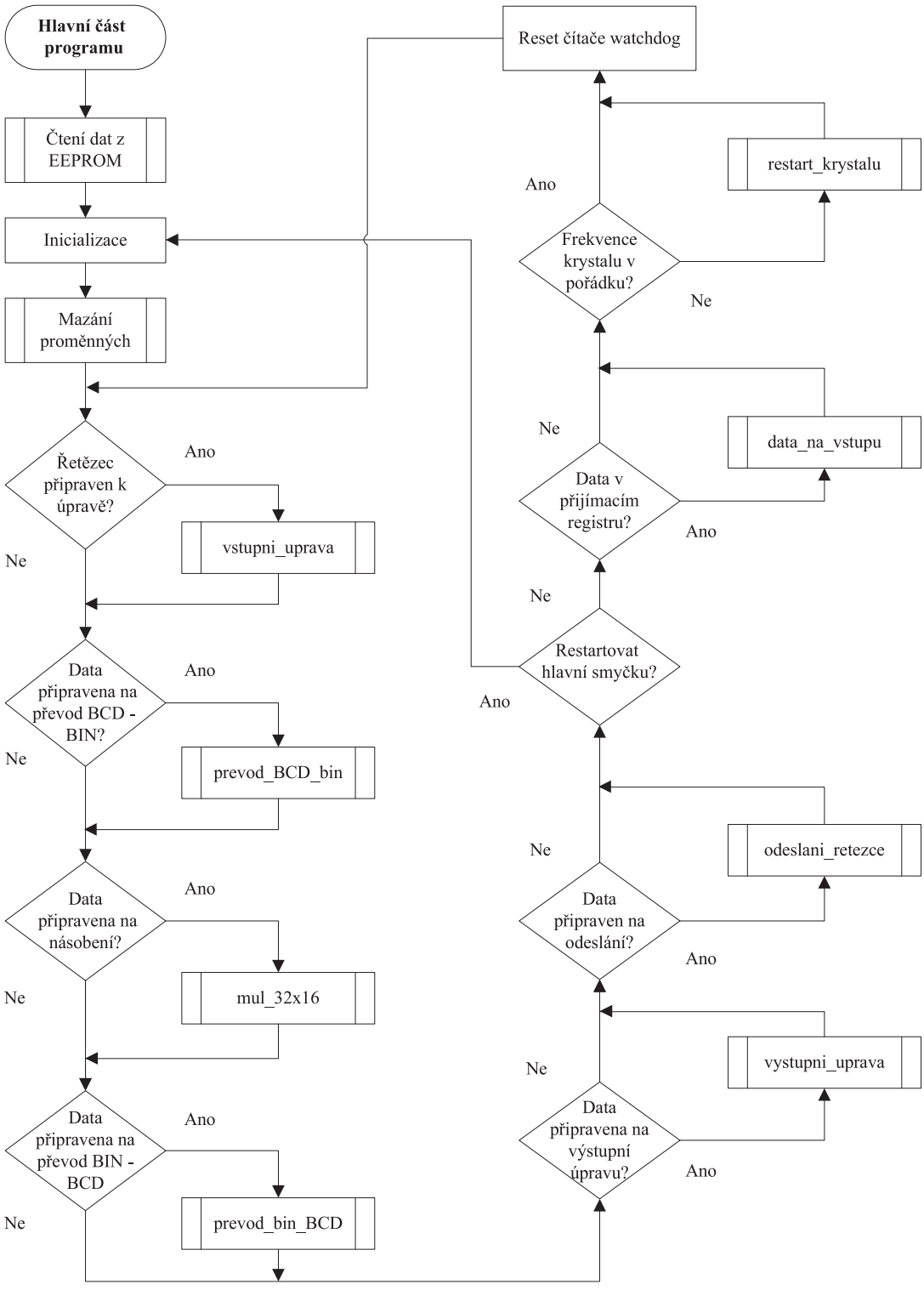
- *SC1* – Přijetí a ověření ASCII „start znaku“ T.
- *SC2* – Přijetí a ověření prvního „stop znaku“ (CR).
- *SC3* – Přijetí druhého „stop znaku“ (LF).
- *SC4* – Ověření druhého „stop znaku“ (LF).
- *SC5* – Povolení spuštění procedury *vstupni_uprava*.
- *SC6* – Povolení spuštění procedury *prevod_BCD_bin*.
- *SC7* – Indikace převodu prvního znaku v proceduře *vystupni_uprava*.
- *SC8* – Povolení spuštění procedury *mul_32x16*.
- *SC9* – Povolení spuštění procedury *prevod_bin_BCD*.
- *SC10* – Povolení spuštění procedury *vystupni_uprava*.
- *SC11* – Povolení spuštění procedury *odeslani_retezce*.
- *SC12* – Povolení skoku na inicializaci, tedy tzv. restartu programu.

5.1 Hlavní procedury obslužného programu

5.1.1 Příjem řetězce

Zdrojový kód procedury je uveden v příloze B.2. Procedura obstarává příjem celého řetězce a průběžně kontroluje, zda má řetězec správný formát a neobsahuje chyby.

Procedura je spuštěna nastavením bitu *RCIF* v registru *PIR1*. Což indikuje naplnění přijímacího registru RCREG přijímače UART daty znaku. Pokud je přijímaný znak první, proběhne nejdříve kontrola, zda se jedná o „start znak“ vyjadřující písmeno T v ASCII. Současně s kontrolou je do pomocné proměnné *CMAXDIG*



Obr. 5.2: Vývojový diagram hlavní části programu.

přesunuta hodnota z registru *MAXDIG* vyjadřující maximální počet přijímaných číselných znaků. Pokud kontrola neproběhne úspěšně, je celý program restartován.

Pokud proběhne úspěšně je nastaven bit *SC1* indikující přijetí „start znaku“ a procedura je ukončena.

Při dalším spuštění procedury (nová data v registru *RCREG*) je kontrola start znaku vynechána a kontroluje se zda přijatý znak vyjadřuje číselnou hodnotu či ne.

Pokud ano, je znak převeden z ASCII na BCD formát odečtením hexadecimálního čísla 30 a uložen do přijímací paměti. Také je dekrementován registr *CMAXDIG* určující maximální počet následně přijímaných znaků (+1). Následně je testována nenulovost tohoto registru. V případě, že je hodnota registru nulová, je nastaven bit *SC12* a po návratu z procedury je celý program restartován (řetězec obsahoval počet číselných znaků vyšší, než bylo povolené maximum).

Pokud se nejedná o znak vyjadřující číselnou hodnotu, je provedena kontrola, zda jde o první „stop znak“, tedy znak CR v ASCII. V případě, že se jedná o znak jiné hodnoty je nastaven bit *SC12* a po návratu z procedury je celý program restartován. Pokud je znak ověřen, je nastaven bit *SC2* indikující správný příjem prvního „stop znaku“ a poté je procedura ukončena.

Při následném spuštění procedury je provedena kontrola druhého „stop znaku“, tedy znaku LF v ASCII. Pokud přijatý znak nemá tuto hodnotu, je nastaven bit *SC12* a po návratu z procedury je program restartován. Pokud je přijatý znak shodný s předpokládanou hodnotou, je nastaven bit *SC4* indikující správné přijetí druhého „stop znaku“. Na konci procedury je povoleno spuštění následující funkce (vstupní úprava) a vypnut kontinuální příjem přijímače UART. Výstupem funkce je řetězec uložený v registrech *P0* až *P11*, kde znaky vyjadřující číselnou informaci jsou převedeny na čísla ve formátu BCD a ukončující znaky jsou ponechány ve formátu ASCII.

5.1.2 Vstupní úprava

Zdrojový kód procedury je uveden v příloze B.3. Procedura slouží k oddělení číselných hodnot (tj.znaků vyjadřujících číslo) v řetězci od hodnot ukončujících znaků a k uložení číselných hodnot do paměti v pořadí vhodnějším pro následné zpracování.

Procedura začíná kontrolou, zda byl v řetězci přijat alespoň jeden znak vyjadřující číselnou informaci hodnoty změřené ultrazvukovým tloušťkoměrem. Pokud řetězec číselnou informaci neobsahoval, je procedura ukončena a celý program restartován. Pokud řetězec číselnou informaci obsahoval, do pomocných proměnných jsou nejdříve uloženy hodnoty „stop znaků“. Následuje nastavení ukazatele na začátek paměti, kam se má číselná informace z řetězce uložit. Následující cyklus od posledního registru projde celou přijímací paměť a znaky vyjadřující číslo uloží do nových registrů *R0* až *R7*. Cyklus končí dosažením začátku přijímací paměti (registru *P0*) a nastavením povolení pro následující proceduru (převod z BCD do binárního tvaru).

5.1.3 Převod řetězce z BCD do binárního tvaru

Vývojový diagram je uveden na Obr.5.3. Zdrojový kód procedury je uveden v příloze B.4. Po spuštění procedury jsou vymazány její pomocné proměnné. Následuje výpočet opakování převodu, který vychází z konstanty maximálního počtu přijatých čísel (proměnná *MAXDIG*) a proměnné vyjadřující počet čísel do překročení maximálního počtu přijatých čísel (proměnná *CMAXDIG*). Počet opakování je vyjádřen proměnnou *RMAXDIG* z které je následně vypočítán ukazatel na první převáděné číslo (první přijaté číslo resp. číslo s nejvyšším řádem).

Převod začíná přičtením tohoto čísla do paměti výsledku převodu. Následuje dekrementace počtu opakování a zjištění zda je nutné převod opakovat. Pokud ne, procedura končí povolením následující procedury násobení. Pokud ano, je výsledek převodu uložen (zálohován) do pomocných proměnných *ADAT1* až *ADAT4*. Následně je paměť výsledku převodu dvakrát rotována vlevo. Tím je dosaženo vynásobení 4x. Přenos mezi registry je při rotaci paměti přenášen v bitu *C* registru *STATUS*. Po rotaci je k paměti výsledku přičtena zálohovaná hodnota z proměnných *ADAT1* až *ADAT4*. Tím je dosaženo celkového násobení 5x (včetně násobení rotací). V dalším kroku je výsledek převodu opět rotován vlevo, čímž je dosaženo celkového požadovaného násobení 10x. Po násobení je zvýšen obsah ukazatele. Následuje přičtení dalšího čísla k výsledku převodu, dekrementace počtu opakovaní a opět kontrola počtu opakování. Celý cyklus se opakuje až do splnění podmínky *RMAXDIG=0*.

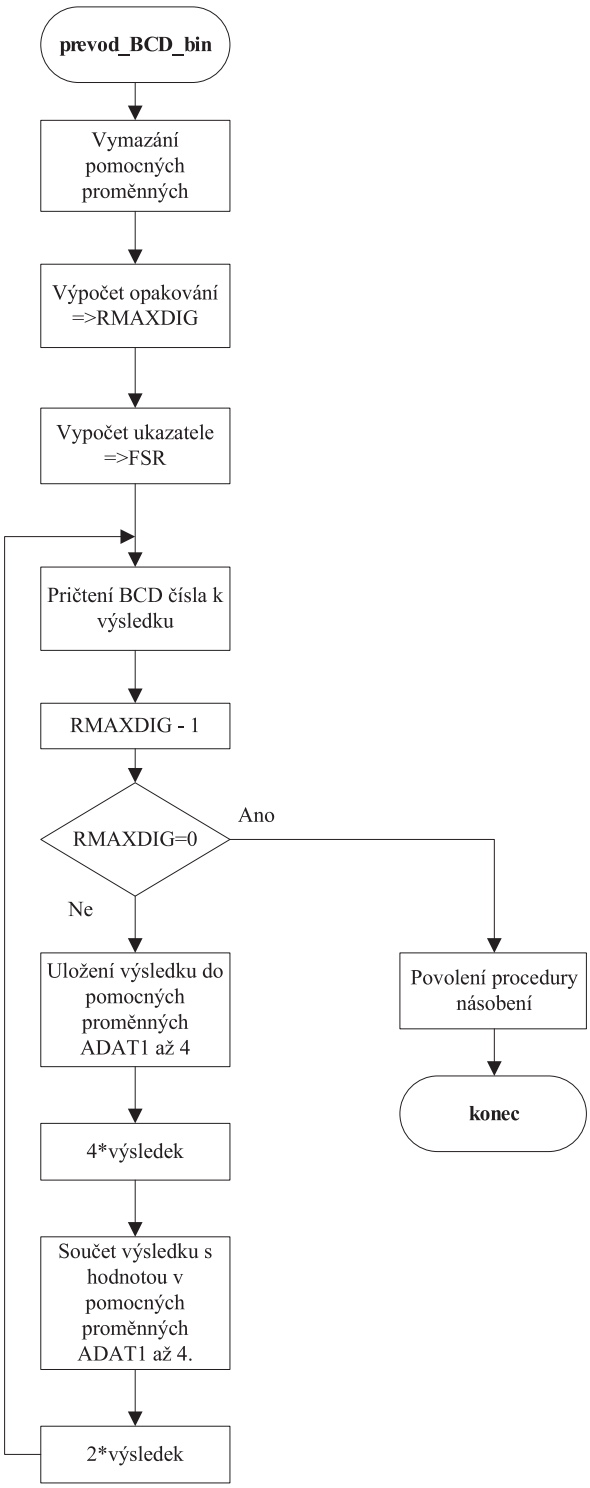
Výsledkem procedury je binární číslo uložené v registrech *V0* až *V3*.

5.1.4 Násobení

Vývojový diagram je uveden na Obr.5.4. Zdrojový kód procedury je uveden v příloze B.5. Procedura po spuštění načte z pomocných proměnných konstantu násobitele (korekční konstantu) o dvou složkách. Nižší část je načtena do proměnné *N0*, vyšší část do proměnné *N1*. K paměťovému prostoru výsledku převodu do binárního tvaru je přidáno dalších 16 bitů (2 registry).

Následně je zjištěno zda LSb násobitele má hodnotu logické 1. Pokud ano, je vstupní binární číslo (mezivýsledek) přičteno do registrů výsledku. Po této operaci je paměť s násobitelem rotována doprava. Bitová hodnota LSb násobitele se rotací nepřevádí do MSb. Vnitřní přenos LSb z *N1* do MSb v *N0* je však zajištěn. Paměť s mezivýsledkem je naopak rotována doleva. Přenos mezi registry mezivýsledku je zajištěn programově.

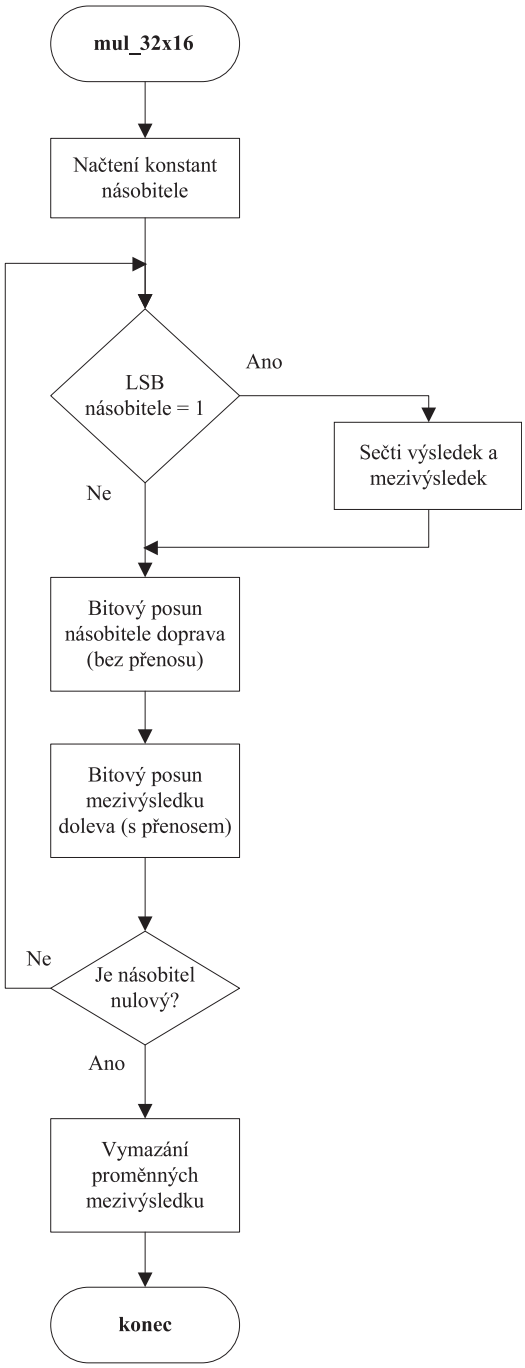
Po rotaci je zjištěno, zda má násobitel hodnotu nula. Pokud je nenulový, běh procedury pokračuje opětovným zjištěním zda LSb výsledku má hodnotu 1. Pokud



Obr. 5.3: Vývojový diagram procedury pro převod z BCD na binární formát.

však podmínka nulovosti násobitele splněna je, jsou vymazány proměnné mezivýsledku a nastaveno povolení pro následující proceduru (převod z binárního tvaru do BCD) a tím procedura končí.

Výsledkem procedury je binární číslo uložené v registrech *C0* až *C5*.



Obr. 5.4: Vývojový diagram procedury násobení.

5.1.5 Převod z binárního tvaru na BCD řetězec

Zdrojový kód procedury je uveden v příloze B.6. Procedura využívá principu popsaného v kapitole možnosti řešení. Rovnice však byly upraveny pro převod na

dekadická čísla o maximálně 13 číslicích a jsou uvedeny v příloze A.

Z hlediska rozsahu vstupní hodnoty je procedura značně nadimenzována. Za běžného provozu na jejím vstupu může být dekadické číslo o maximálně 10 číslicích. Větší rozsah vstupní hodnoty je umožněn záměrně pro případ, že by v budoucnu byla nutná změna korekční konstanty, na konstantu s větším počtem desetinných míst. Úprava procedury pro vyšší rozsah vstupní hodnoty je totiž záležitost problematická a pravděpodobně by si vyžádala přepsání větší části kódu.

Převod začíná od čísla nejnižšího řádu *d_0*. Do paměťového prostoru dvojice registrů *BP0* a *BP1* jsou pomocí podprogramu *add_part* dle příslušné rovnice sečteny násobené hexadecimální číslice. Pro násobení je využit podprogram *mul_by_con*, který přebírá v pracovním registru uloženou hodnotu hexadecimálního číslice. Následně ji vynásobí číslem uloženým v pomocné proměnné *MBCC*. Po sečtení všech číslic dané rovnice je přičten výstupní registr (převodní funkce) odpovídající řádu právě převáděné číslice. V tomto případě se jedná o registr *B0*. Následně je volán podprogram *mod_10*. Ten nejdříve zkontroluje nenulovost pomocných proměnných *BP0* a *BP1* a následně využitím odečítání nejdříve hodnoty 100, pak hodnoty 10 rozdělí hodnotu v pomocných proměnných na počty stovek, desítek a jednotek. Počet stovek je uložen v proměnné *BPP2*, desítek v proměnné *BPP1* a jednotky jsou uloženy v pracovním registru *W*. Tímto podprogram končí.

Po návratu do hlavní části procedury je hodnota v pracovním registru uložena do registru výstupu převodu, který odpovídá řádu právě převáděné číslice. Ve zmíněném případě do registru *B0*. Počet desítek je z proměnné *BPP1* přičten do registru řádu o jedno vyššího (*B1*), počet stovek do řádu o dva vyššího než-li převáděná číslice (*B2*).

Popsaným způsobem se provedou všechny převody od *d_0* do *d_12*. Procedura končí nastavením *SC10=1*, tedy povolením následující procedury výstupní úpravy. Výstupem procedury je 13 BCD čísel uložených v registrech *B0* až *B12*.

5.1.6 Výstupní úprava

Vývojový diagram je uveden na Obr.5.5. Zdrojový kód procedury je uveden v příloze B.7. Procedura začíná vložením hodnoty d'252' do registru *BCA*. Tento registr je v paměti umístěn na místě desetinné čárky a hodnota 252 po průchodu podprogramem pro převod na ASCII (přičtení h'30') dá ve výsledku hodnotu reprezentující znak desetinné čárky v ASCII.

Následuje nastavení ukazatele na konec paměti výsledku převodu z binárního do BCD formátu, tedy registr *B12*. Ukazatel je nastaven z proměnné *BCOUNT*. Hodnota je z registru (*B12*) vyčtena a je ověřena její nulovost.

Pokud je hodnota nulová ověří se, zda již bylo podprogramem pro převod na ASCII převedeno nějaké číslo (*SC7*). Pokud ano (*SC7=1*), je číslo v registru převedeno z BCD na ASCII podprogramem *preved_na_ASCII*. Pokud ne (*SC7=0*), je inkrementována proměnná *BCOUNT*.

Pokud je hodnota vyčtená z registru *B12* nenulová, je spuštěn podprogram pro převod z BCD na ASCII *preved_na_ASCII*. Prvním spuštěním tohoto podprogramu je nastavena proměnná *SC7*.

Následně je zjištěno, zda ukazatel (*FSR*) ukazuje na konec paměti. Pokud ne, je jeho hodnota inkrementována a procedura pokračuje opětovným vyčtení konstanty z dalšího paměťového místa s následnou kontrolou nulovosti.

Pokud je hodnota ukazatele (*FSR*) shodná s adresou konce paměti, je zjištěno zda proměnná *BCOUNT* má hodnotu větší, než-li je hodnota adresy registru před desetinnou čárkou (*B6*).

Pokud ne, je povolena následující procedura (odeslaní řetězce) a odstraněno povolení pro proceduru vstupní úpravy. Tímto procedura končí.

Pokud ano, je do proměnné *BCOUNT* uložena adresa registru před desetinnou čárkou (*B6*) a do tohoto registru je vložena hodnota h'30', tedy číslo nula v ASCII. Následně je povolena následující procedura (odeslání řetězce) a odstraněno povolení pro proceduru vstupní úpravy.

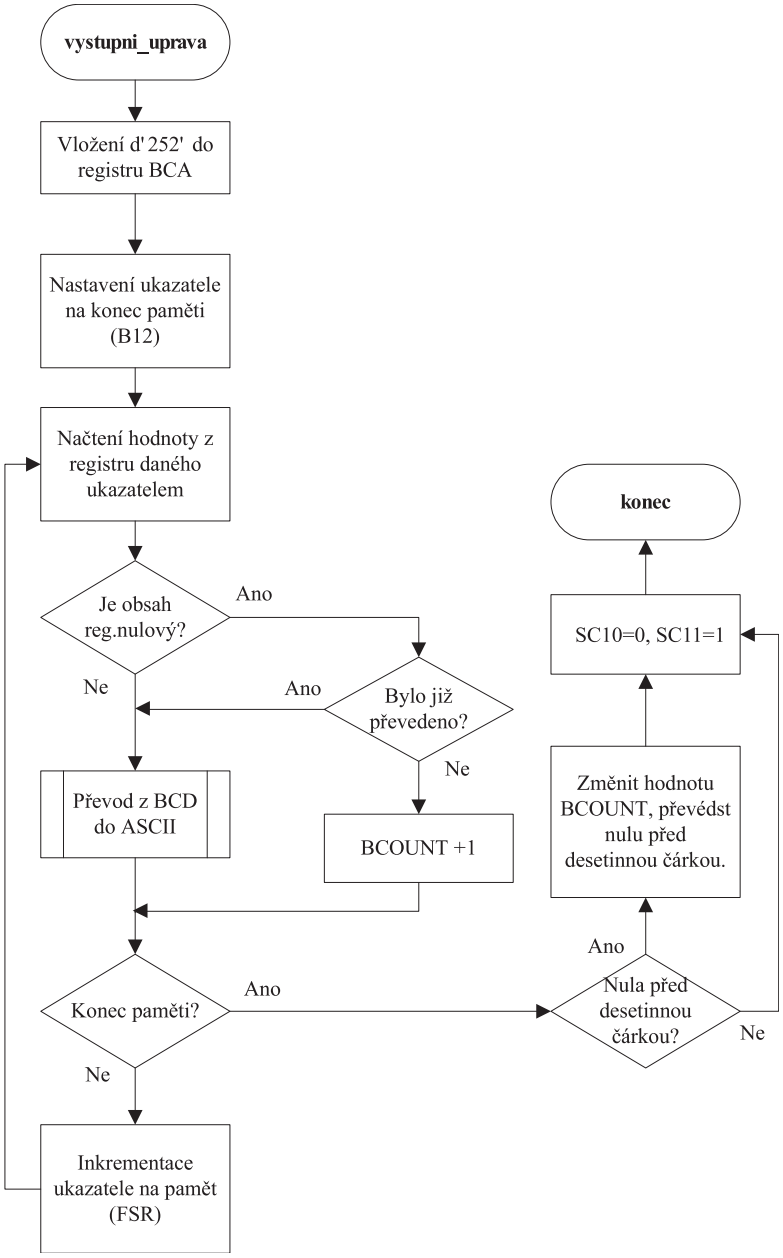
Výstupem procedury je část paměti *B0* až *B12* převedená na znaky v ASCII a proměnná *BCOUNT* určující první číslo, které se bude v následující proceduře (odeslání řetězce) odesílat.

5.1.7 Odeslání řetězce

Zdrojový kód procedury je uveden v příloze B.8. Procedura začíná nastavením vysílače obvodu UART pro vysílání přenosovou rychlosti 9600Bd. Následuje načtení proměnné *BCOUNT* do ukazatele. V proměnné *BCOUNT* je uložena adresa prvního odesílaného znaku.

Za pomocí ukazatele je znak vyčten z paměti a je zavolán podprogram pro odeslání znaku. V tomto podprogramu je hodnota znaku vložena do vysílacího registru *TXREG* a následně je v cyklu testován bit *TXIF* registru *PIR1*. Pokud je bit *TXIF* nastaven končí testování. Nastavení tohoto bitu indikuje přenos znaku do vysílacího posuvného registru *TSR*.

Podprogram se vrací do místa, kde byl původně zavolán a následuje kontrola ukazatele na konec vysílané paměti. Pokud hodnota ukazatele není shodná s adresou konce paměti, je inkrementován a následuje další vyčtení z registru a odeslání až do chvíle, kdy je konec paměti dosažen.



Obr. 5.5: Vývojový diagram procedury pro výstupní úpravu.

Po dosažení konce paměti jsou pomocí funkce pro odeslání znaku odeslány ukončující znaky CR a LF. Následně je zrušeno povolení pro proceduru odeslání řetězce a nastaven bit SC12. Což způsobí restart programu po návratu do hlavní smyčky.

5.2 Ověření funkčnosti

Ověření funkčnosti bylo provedeno za použití osobního počítače vybaveného dvěma sériovými porty. Zapojení je uvedeno na Obr.5.6.

Na počítači byly spuštěny dvě instance programu Terminál v1.9b (viz. Obr.5.7). V instanci nastavené na port připojený na přijímací vodiče adaptéru byla nastavena přenosová rychlost 2400Bd, v instanci nastavené na port připojený na vysílací vodiče adaptéru byla nastavena přenosová rychlost 9600Bd.

Pro simulaci běžných pracovních podmínek bylo vytvořeno makro s typickým vstupním řetězcem a nastavena perioda opakování 300ms. Poté bylo odesílání spuštěno. V druhé instanci byly kontrolovány přijímané řetězce.

Pomocí funkce counter programu Terminal byla při snížení vysílací periody zjištěna maximální rychlost zpracování signálu. Za jednu sekundu je zařízení schopno zpracovat cca 10 řetězců. Což je trojnásobek zadáním požadované rychlosti.

Programem Terminal byly taktéž simulovány chybové stavy vstupního řetězce:

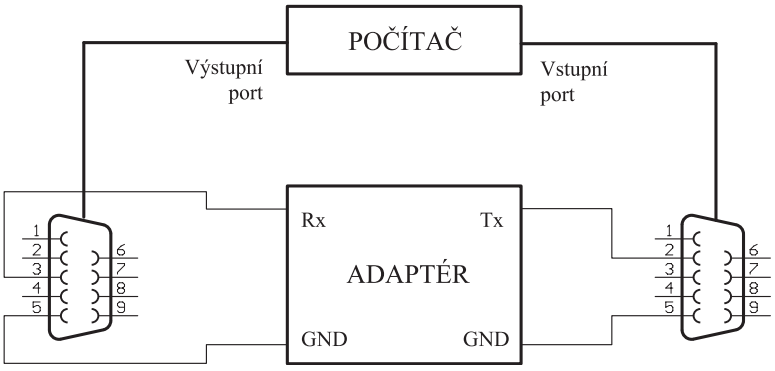
- start znak neodpovídá ASCII znaku T,
- počet číselných znaku v řetězci je vyšší než nastavený,
- první ukončující znak neodpovídá ASCII znaku CR,
- druhý ukončující znak neodpovídá ASCII znaku LF,
- mezi start znakem a ukončujícími znaky není vložen žádný znak vyjadřující číselnou hodnotu,
- mezi číselnými znaky vloženy nečíselné znaky,
- náhodná sekvence znaků.

Všechny tyto chybové stavy zařízení filtruje a na výstup neodesílá žádné informace.

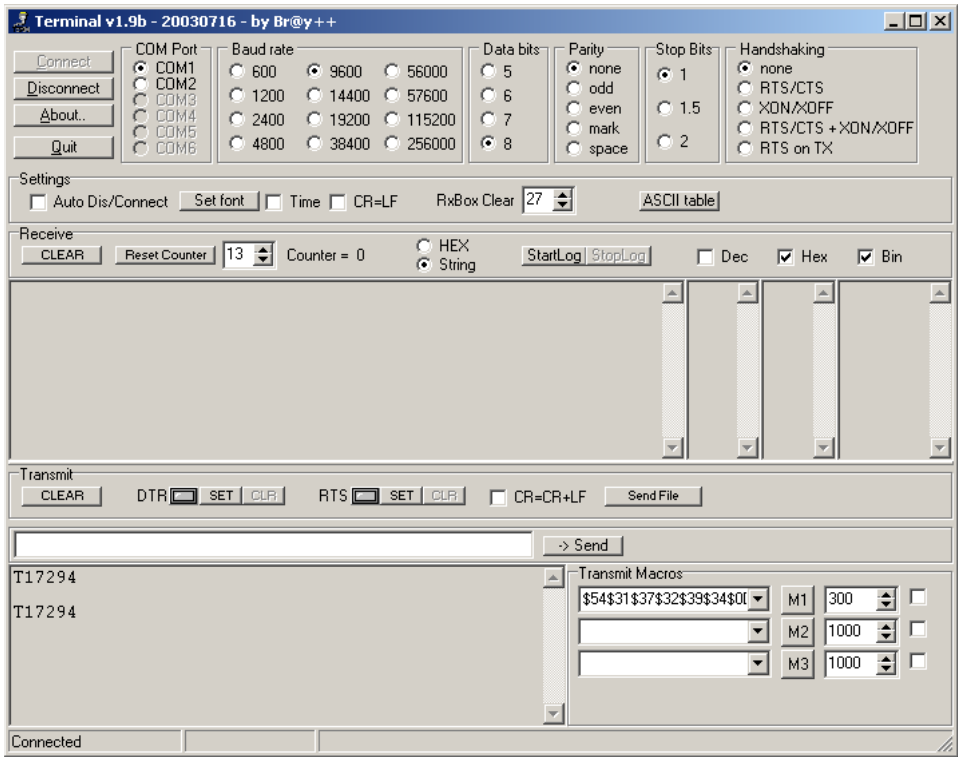
5.3 Přenositelnost obslužného programu

Obslužný program se vzhledem ke zvolenému typu mikrokontroléru vyznačuje dvěma specifiky:

- nastavení „předděliče“ čítače Watchdog je provedeno automaticky,
- v hlavní smyčce programu se nachází procedura pro kontrolu stability externího oscilátoru.



Obr. 5.6: Blokové schéma propojení počítače s adaptérem.



Obr. 5.7: Instance programu Terminal - příjem řetězce na počítači.

V případě, že by bylo nutné změnit cílovou součástku, je nutno zabezpečit odstranění kontroly externího oscilátoru, pokud ji mikrokontrolér nepodporuje. Případně nastavit čítači Watchdog správný „předdělič“.

Obslužný program byl kromě mikrokontroléru PIC16F913 testován také na mikrokontroléru PIC16F873A.

6 ZÁVĚR

V práci jsou stručně popsány varianty řešení dílčích částí zadaného úkolu. Následuje popis hardwarové konstrukce adaptéru a popis funkce obslužného programu pro mikrokontrolér. V praktické části práce byla navržena deska plošných spojů, která byla osazena zvolenými součástkami včetně mikrokontroléru. Mikrokontrolér byl naprogramován obslužným programem. Funkce zařízení byla ověřena na typických vstupních datech a byla také ověřena schopnost zařízení filtrovat chybné vstupní data. Vytvořený adaptér splňuje svými parametry požadavky zadaní.

LITERATURA

[1] GOFTON, P.W. *Sériová komunikace*. Přeložil Bohumil Kvapil. Vydání 1. Praha: Grada Publishing, 2001. 240s. ISBN 80-7169-131-3.

[2] HRBÁČEK, Jiří. *Komunikace mikrokontroléru o okolím 1*. Vydání 1. Praha: BEN - technická literatura, 1999. 152s. ISBN 80-86056-42-2.

[3] HRBÁČEK, Jiří. *Komunikace mikrokontroléru o okolím 2*. Vydání 1. Praha: BEN - technická literatura, 2000. 148s. ISBN 80-86056-73-2.

[4] HRBÁČEK, Jiří. *Moderní učebnice programování PIC 1.díl*. Vydání 1. Praha: BEN - technická literatura, 2004. 77s. ISBN 80-7300-136-5.

[5] HRBÁČEK, Jiří. *Moderní učebnice programování PIC 2.díl*. Vydání 1. Praha: BEN - technická literatura, 2007. 140s. ISBN 978-80-7300-137-7.

[6] JASIO, Lucio Di; WILMSHURST, Tim; IBRAHIM, Dogan et al. *PIC Microcontrollers: Know It All*. Oxford: Elsevier, 2008. 910s. ISBN 978-0-7506-8615-0.

[7] PALACHERLA, Amar. *AN555 - Software Implementation of Asynchronous Serial I/O*. Chandler: Microchip, 2002. 38s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[8] PAYSON, John. *Binary to BCD unpacked 16 bit to 5 digit* [online]. Cambridge, Massachusetts, USA 2001–2009, [cit. 18. 5. 2009]. Dostupné z URL: <http://www.piclist.com/techref/microchip/math/radix/b2bu-16b5d.htm>.

[9] PEROUTKA, Oldřich. *Mikrokontroléry PIC16F87X*. Vydání 1. Praha: BEN - technická literatura, 2005. 256s. ISBN 80-7300-139-X.

[10] SANCHEZ, Julio; CANTON, Maria P. *Microcontroller Programming*. The Microchip PIC. Florida: CRC Press, 2007. 824s. ISBN 08-4937-189-9.

[11] VACEK, Václav. *Učebnice programování PIC*. Vydání 1. Praha: BEN - technická literatura, 2000. 139s. ISBN 80-86057-87-2.

[12] *PIC16F87XA Data Sheet*. Chandler: Microchip, 2002. 234s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[13] *PIC16F913 Data Sheet*. Chandler: Microchip, 2006. 303s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[14] *MAXIM +5V-Powered, Multichannel RS-232 Drivers/Receivers (19-4323; Rev 15; 1/06)*. Sunnyvale: Maxim IC, 2006. 36s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.maxim-ic.com/quick_view2.cfm?qv_pk=1798>.

[15] *Yageo: Data sheet surface-mount ceramic multilayer capacitors, General purpose, Class 1, NP0, 16 V TO 50 V, (V. 0)*. Taiwan: Yageo, 2009. 15s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.yageo.com/pdf/UPY-GP_NP0_16V-to-50V_0.pdf>.

[16] *CA45 Series Solid Electrolyte Tantalum Chip Capacitor*. Japan: Shjinpei, 2009. 3s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.gme.cz/_dokumentace/dokumenty/907/907-029/dsh.907-029.1.pdf>.

[17] *Quartz SMD HC49USSMD*. Hameln: Auris, 2008. 1s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.auris-gmbh.de/root/img/pool/pdf_files/HC49USSMD.pdf>.

[18] *L78L00 Series - Positive voltage regulators (Rev. 10)*. Geneva: STMicroelectronics, 2005. 26s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.gme.cz/_dokumentace/dokumenty/934/934-005/dsh.934-005.1.pdf>.

[19] *LE00AB/C Series - Verz low drop voltage regulators with inhibit*. Geneva: STMicroelectronics, 2002. 26s. [cit. 18. 5. 2009]. Dostupné z URL: <www.datasheetcatalog.com/datasheets_pdf/L/E/5/0/LE50CZ.shtml>.

[20] *TS4148 0.5AMPS High Speed Switching Diode (Version: C08)*. Taiwan: Taiwan Semiconductors, 2008. 2s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.gme.cz/_dokumentace/dokumenty/934/934-005/dsh.934-005.1.pdf>.

[21] *137 Series 2510 Connectors*. Praha: GME, 2008. 1s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.gme.cz/_dokumentace/dokumenty/800/800-087/dsh.800-087.1.pdf>.

[22] *Datasheet MAX232, MAX232I* . Dallas: Texas instruments, 2004. 17s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.ti.com/lit/gpn/max232>.

[23] *Datasheet HIN232, HIN236, HIN237, HIN238,HIN239, HIN240, HIN241 (FN3138.16)*. Milpitas: Intersil, 2008. 21s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.intersil.com/data/fn/fn3138.pdf>.

[24] *Datasheet ICL232 (FN3020.7)*. Milpitas: Intersil, 2008. 7s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.intersil.com/data/fn/fn3020.pdf>.

[25] *Datasheet TC232*. Chandler: Microchip, 1996. 6s. [cit. 18. 5. 2009]. Dostupné z URL: <http://datasheet.digchip.com/295/295-3-136340-C232COE.pdf>.

[26] *Datasheet ST232B, ST232C*. Geneva: STMicroelectronics, 2004. 19s. [cit. 18. 5. 2009]. Dostupné z URL: <http://www.st.com/stonline/products/literature/ds/6420.pdf>.

[27] DUANE, Brett. *Aplication note AN949 - Making Your Oscillator Work*. Chandler: Microchip, 2004. 8s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[28] *PIC16F627A/628A/648A Data Sheet*. Chandler: Microchip, 2007. 178s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[29] *PIC16F631/677/685/687/689/690 Data Sheet*. Chandler: Microchip, 2008. 306s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[30] *PIC16F688 Data Sheet*. Chandler: Microchip, 2007. 202s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[31] *PIC16F631/677/685/687/689/690 Data Sheet*. Chandler: Microchip, 2007. 306s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[32] *PIC16F72X/PIC16LF72X Data Sheet*. Chandler: Microchip, 2009. 266s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[33] *PIC16F7X7 Data Sheet*. Chandler: Microchip, 2004. 276s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[34] *PIC16F7X Data Sheet*. Chandler: Microchip, 2002. 174s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[35] *PIC16F882/883/884/886/887 Data Sheet*. Chandler: Microchip, 2009. 328s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[36] *PIC16F87/88 Data Sheet*. Chandler: Microchip, 2002. 228s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

[37] *PIC16F913/914/916/917/946 Data Sheet*. Chandler: Microchip, 2007. 330s. [cit. 18. 5. 2009]. Dostupné na webu: <http://www.microchip.cz>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ASCII americký normalizovaný kód pro výměnu informací – American Standard Code for Information Interchange

BCD Binárně kódované desítkové číslo – Binary Coded Decimal

Bd Baud

bit/s bity za sekundu

CAN Controller-Area Network

CR návrat vozíku – Carriage Return

DCE ukončující zařízení datového okruhu – Data Communication Equipment

DMIPS/MHz Dhrystone MIPS/MHz

DPS deska plošných spojů

DTE koncové zařízení přenosu dat – Data Terminal Equipment

EEPROM Elektricky mazatelná a programovatelná paměť – Electrically Erasable Programmable Read-Only Memory

EIA sdružení pro elektronický průmysl – Electronic Industries Association

EPROM Elektricky programovatelná paměť – Electrically Programmable Read Only Memory

FIFO First-In First-Out

ICSP Sériové programování v cílovém obvodu – In Circuit Serial Programming

LF posun o řádek – Line Feed

LSb Nejméně významný bit – Least Significant bit

MAPS Microchip Advanced Part Selector

MIPS Milion instrukcí za sekundu – Million Instructions Per Second

MSb Bit s největším významem – Most Significant bit

PWM Pulsní šířková modulace – Pulse Width Modulation

RAM Paměť s náhodným přístupem – Random Access Memory

RISC	Počítač s redukovanou instrukční sadou – Reduced Instruction Set Computer
RXD	přijímaná data – Received Data
SCI	Sériové komunikační rozhraní – Serial Communication Interface
SG	signálová země – Signal Ground
SMD	součástka pro povrchovou montáž – Surface Mount Device
SMT	technologie povrchové montáže – Surface Mount Technology
TXD	vysílaná data – Transmitted Data
USART	Univerzální synchronní asynchronní přijímač vysílač – Universal Synchronous Asynchronous Receiver Transmitter
USB	univerzální sériová sběrnice – Universal Serial Bus

SEZNAM PŘÍLOH

A VZTAHY PRO PŘEVOD Z BINÁRNÍHO TVARU NA BCD ČÍSLA

Hexadecimální číslo:

$$H = h\_10 * 16^{10} + h\_9 * 16^9 + h\_8 * 16^8 + h\_7 * 16^7 + h\_6 * 16^6 + h\_5 * 16^5$$

$$+h\_4 * 16^4 + h\_3 * 16^3 + h\_2 * 16^2 + h\_1 * 16^1 + h\_0 * 16^0.$$

Decimální číslo:

$$D = d\_12 * 10^{12} + d\_11 * 10^{11} + d\_10 * 10^{10} + d\_9 * 10^9 + d\_8 * 10^8 + d\_7 * 10^7$$

$$+d\_6 * 10^6 + d\_5 * 10^5 + d\_4 * 10^4 + d\_3 * 10^3 + d\_2 * 10^2 + d\_1 * 10^1$$

$$+d\_0 * 10^0.$$

Vyjádření složek hexadecimálního čísla:

$$h\_0 * 16^0 = h\_0 * 1 = h\_0 * 10^0,$$

$$h\_1 * 16^1 = h\_1 * 16 = h\_1 * 10^1 + 6 * h\_1 * 10^0,$$

$$h\_2 * 16^2 = h\_2 * 256 = 2 * h\_2 * 10^2 + 5 * h\_2 * 10^1 + 6 * h\_2 * 10^0,$$

$$h\_3 * 16^3 = h\_3 * 4096 = 4 * h\_3 * 10^3 + 9 * h\_3 * 10^1 + 6 * h\_3 * 10^0,$$

$$h\_4 * 16^4 = h\_4 * 65536 = 6 * h\_4 * 10^4 + 5 * h\_4 * 10^3 + 5 * h\_4 * 10^2$$

$$+3 * h\_4 * 10^1 + 6 * h\_4 * 10^0,$$

$$h\_5 * 16^5 = h\_5 * 1048576 = h\_5 * 10^6 + 4 * h\_5 * 10^4 + 8 * h\_5 * 10^3$$

$$+5 * h\_5 * 10^2 + 7 * h\_5 * 10^1 + 6 * h\_5 * 10^0,$$

$$h\_6 * 16^6 = h\_6 * 16777216 = h\_6 * 10^7 + 6 * h\_6 * 10^6 + 7 * h\_6 * 10^5$$

$$+7 * h\_6 * 10^4 + 7 * h\_6 * 10^3 + 2 * h\_6 * 10^2 + h\_6 * 10^1$$

$$+6 * h\_6 * 10^0,$$

$$h\_7 * 16^7 = h\_7 * 268435456 = 2 * h\_7 * 10^8 + 6 * h\_7 * 10^7 + 8 * h\_7 * 10^6$$

$$+4 * h\_7 * 10^5 + 3 * h\_7 * 10^4 + 5 * h\_7 * 10^3 + 4 * h\_7 * 10^2$$

$$+5 * h\_7 * 10^1 + 6 * h\_7 * 10^0,$$

$$h\_8 * 16^8 = h\_8 * 4294967296 = 4 * h\_8 * 10^9 + 2 * h\_8 * 10^8 + 9 * h\_8 * 10^7$$

$$+4 * h\_8 * 10^6 + 9 * h\_8 * 10^5 + 6 * h\_8 * 10^4 + 7 * h\_8 * 10^3$$

$$+2 * h\_8 * 10^2 + 9 * h\_8 * 10^1 + 6 * h\_8 * 10^0,$$

$$h\_9 * 16^9 = h\_9 * 68719476736 = 6 * h\_9 * 10^{10} + 8 * h\_9 * 10^9 + 7 * h\_9 * 10^8$$

$$+h\_9 * 10^7 + 9 * h\_9 * 10^6 + 4 * h\_9 * 10^5 + 7 * h\_9 * 10^4 + 6 * h\_9 * 10^3$$

$$+7 * h\_9 * 10^2 + 3 * h\_9 * 10^1 + 6 * h\_9 * 10^0,$$

$$h\_10 * 16^{10} = h\_10 * 1099511687776 = h\_10 * 10^{12} + 9 * h\_10 * 10^{10} + 9 * h\_10 * 10^9$$

$$+5 * h\_10 * 10^8 + h\_10 * 10^7 + h\_10 * 10^6 + 6 * h\_10 * 10^5 + 2 * h\_10 * 10^4 +$$

$$7 * h\_10 * 10^3 + 7 * h\_10 * 10^2 + 7 * h\_10 * 10^1 + 6 * h\_10 * 10^0.$$

Seskupení jednotlivých dekadických řádů:

$$d\_0' = h\_0 + 6 * (h\_1 + h\_2 + h\_3 + h\_4 + h\_5 + h\_6 + h\_7 + h\_8 + h\_9 + h\_10),$$

$$d\_1' = h\_1 + 5 * h\_2 + 9 * h\_3 + 3 * h\_4 + 7 * h\_5 + h\_6 + 5 * h\_7 + 9 * h\_8 + 3 * h\_9$$

$$+7 * h\_10,$$

$$d\_2' = 2 * h\_2 + 5 * h\_4 + 5 * h\_5 + 2 * h\_6 + 4 * h\_7 + 2 * h\_8 + 7 * h\_9 + 7 * h\_10,$$

$$d\_3' = 4 * h\_3 + 5 * h\_4 + 8 * h\_5 + 7 * h\_6 + 5 * h\_7 + 7 * h\_8 + 6 * h\_9 + 7 * h\_10,$$

$$d\_4' = 6 * h\_4 + 4 * h\_5 + 7 * h\_6 + 3 * h\_7 + 6 * h\_8 + 7 * h\_9 + 2 * h\_10,$$

$$d\_5' = 7 * h\_6 + 4 * h\_7 + 9 * h\_8 + 4 * h\_9 + 6 * h\_10,$$

$$d\_6' = h\_5 + 6 * h\_6 + 8 * h\_7 + 4 * h\_8 + 9 * h\_9 + h\_10,$$

$$d\_7' = h\_6 + 6 * h\_7 + 9 * h\_8 + h\_9 + h\_10,$$

$$d\_8' = 2 * h\_7 + 2 * h\_8 + 7 * h\_9 + 5 * h\_10,$$

$$d\_9' = 4 * h\_8 + 8 * h\_9 + 9 * h\_10,$$

$$d\_10' = 6 * h\_9 + 9 * h\_10,$$

$$d\_11' = 0,$$

$$d\_12' = h\_10,$$

B ZDROJOVÝ KÓD OBSLUŽNÉHO PROGRAMU

B.1 Podprogram pro čtení dat z EEPROM

```
nacteni_konstant
;cteni korekcnich konstant z EEPROM
    movlw   h'01'               ; Adresa nižší části korekční konst.->W
    call    read_EEPROM         ; Procedura čtení z EEPROM
    movwf   MULT0               ; Uložení do RAM
    movlw   .1
    call    cekej               ; Procedura čekání
    movlw   h'00'               ; Adresa vyšší části korekční konst.->W
    call    read_EEPROM         ; Procedura čtení z EEPROM
    movwf   MULT1               ; Uložení do RAM
    movlw   .1
    call    cekej               ; Procedura pro čekání
;cteni delky retezce z EEPROM
    movlw   h'02'               ; Adresa konst. délky řetězce ->W
    call    read_EEPROM         ; Procedura čtení z EEPROM
    movwf   MAXDIG              ; Uložení do RAM
    return

read_EEPROM
    bankpg2                     ; Makro pro přepnutí do banky 2
    movwf   EEADRL              ; Vložení adresy registru EEPROM
    clrf    EEADRH
    bankpg3                     ; Makro pro přepnutí do banky 3
    bcf     EECON1,EEPGD
    bsf     EECON1,RD           ; Spustit čtení
    bankpg2                     ; Makro pro přepnutí do banky 2
    nop
    movf    EEDATL,W            ; W=Obsah registru EEPROM
    bankpg0                     ; Makro pro přepnutí do banky 0
    return
```

B.2 Podprogram pro příjem dat a převod na BCD

```
data_na_vstupu
    movf    PCOUNT,W            ; Ukazatel na přijímací paměť->W
    movwf   FSR
    call    prijem_retezce
    btfss   SC1                 ; Pokud SC1=0,konec podprogramu data_na_vstupu
    return
    btfss   SC3                 ; Pokud SC3=0,konec podprogramu data_na_vstupu
    return
    btfss   SC4                 ; Pokud SC4=0,nastav SC12=1 => zahození řetězce
    bsf     SC12
    return                      ; Konec podprogramu data_na_vstupu

;--

W_ASCII_to_BCD_not
    movf    BP1,W
    btfss   SC2                 ; Pokud nebyl přijat první ukonč. znak(SC2=0)-skok
    goto    kontrorola_prvni_k_znaku
    bsf     SC3                 ; Identifikace příjmu druhého ukonč. znaku
    movwf   INDF                ; Ulož do paměti (na místo, kam ukazuje FSR)
    decf    PCOUNT,F
    return                      ; Konec podprogramu W_ASCII_to_BCD
kontrorola_prvni_k_znaku
    xorlw   h'0D'               ; Exkluz. součet přij.znaku a jeho předpokl.hodnoty
    btfss   STATUS,Z            ; Pokud se čísla nerovnají => zahoď řetězec (SC12=1)
    bsf     SC12
    btfsc   STATUS,Z            ; Pokud se čísla rovnají (výsl. nula) nastav SC2=1
    bsf     SC2                 ; Identifikace, že byl přijat první ukončující znak
    movf    BP1,W               ; Načtení přij. znaku z pomocné proměnné
    movwf   INDF                ; Ulož do paměti (na místo, kam ukazuje FSR)
    decf    PCOUNT,F            ; Ukazatel na přijímací paměť -1
    return
W_ASCII_to_BCD
    movwf   BP1                 ; Záloha přijatého znaku    do pomocné proměnné
    movlw   h'30'
    subwf   BP1,W               ; Odečti od př. znaku h'30', W=výsledek
    btfss   STATUS,C
    goto    W_ASCII_to_BCD_not  ; Pokud je výsledek záporné číslo - skok
    movlw   h'3A'               ; Odečti od př. znaku h'3A', W=výsledek
    subwf   BP1,W
    btfsc   STATUS,C
    goto    W_ASCII_to_BCD_not  ; Pokud je výsledek záporné číslo - skok
    addlw   h'0A'               ; Přičti k výsledku h'0A'=> BCD číslo
    movwf   INDF                ; Ulož do paměti (na místo, kam ukazuje FSR)
    decf    PCOUNT,F            ; Ukazatel na přijímací paměť -1
    decf    CMAXDIG,F           ; Dekrementace max.rozsahu přijímaní
    movf    CMAXDIG,F           ; Ověření zda není nula
    btfsc   STATUS,Z
    bsf     SC12                ; Pokud je nula => zahození řetězce
    return

;--

kontrola_start_znaku
    xorlw   h'54'               ; Exkluz. součet přij.znaku a jeho předpokl.hodnoty
    btfsc   STATUS,Z            ; Pokud se čísla rovnají (výsl. nula) nastav SC1=1
    bsf     SC1                 ; Identifikace příjmu „start znaku"
    goto    kontrola_start_znaku_konec
```

```
prijem_retezce
    movf    RCREG,W
    btfss   SC1                     ; Kontrola zda již byl přijat „start znak"
    goto    kontrola_start_znaku
    btfsc   SC1
    goto    prijem_retezce_pok
kontrola_start_znaku_konec
    movf    MAXDIG,W                ; Načtení maximální velikosti řetězce
    movwf   CMAXDIG                 ; Uložení do pomocné proměnné
    incf    CMAXDIG,F
    return                          ; Konec podprogramu prijem_retezce
prijem_retezce_pok
    call    W_ASCII_to_BCD
    btfss   SC3
    return                          ; Konec podprogramu prijem_retezce
    xorlw   h'0A'                   ; Exkluz. součet přij.znaku a jeho předpokl.hodnoty
    btfss   STATUS,Z                ; Pokud je výsledek nula,
    return                          ; konec podprogramu prijem_retezce
    bsf     SC4                     ; Identifikace správného řetězce
    bsf     SC5                     ; Povolení spuštění podprogramu vstupni_uprava
    bcf     RCSTA,CREN              ; Vypnutí kontinuálního čtení
    return                          ; Konec podprogramu prijem_retezce
```

B.3 Podprogram pro vstupní úpravu řetězce

```
vstupni_uprava
    movf    CMAXDIG,W           ; Kontrola zda bylo v řetězci přijato alespoň
    subwf   MAXDIG,W            ; jedno číslo, pokud ne -> zahodit řetězec
    btfss   STATUS,C
    goto    vstupni_uprava_err
    incf    PCOUNT,F
    movf    PCOUNT,W
    movwf   FSR
    movf    INDF,W
    movwf   RS1                 ; Uložení konečných znaků h'0D' a h'0A' do
    incf    FSR,F               ; pomocných proměnných
    incf    PCOUNT,F
    movf    INDF,W
    movwf   RS0
    movlw   h'61'               ; Ukazatel na začátek sady registrů R7-R0
    movwf   RA
vstupni_uprava_repeat
    incf    PCOUNT,F
    movf    PCOUNT,W
    movwf   FSR
    movf    INDF,W
    movwf   RSP
    movf    RA,W
    movwf   FSR
    movf    RSP,W
    movwf   INDF
    decf    RA,F
    call    kontrola_P0         ; Kontrola konce sady registrů (P0)
    btfss   SC6
    goto    vstupni_uprava_repeat
    return                      ; Konec podprogramu vstupni_uprava
kontrola_P0
    movf    PCOUNT,W
    xorlw   h'38'
    btfsc   STATUS,Z            ; Pokud je konec paměti nastav SC6=1
    bsf     SC6                 ; Povolen převod BCD na bin
    return
vstupni_uprava_err
    bcf     SC5                 ; Odstranění povolení pro proceduru vstupni_uprava
    bsf     SC12                ; Zahození řetězce
    return
```

B.4 Podprogram pro převod z BCD na binární formát

```
prevod_BCD_bin
    clrf     V3                  ; Nulování V0-V3
    clrf     V2
    clrf     V1
    clrf     V0
    movf     MAXDIG,W            ; Výpočet opakování
    movwf    RMAXDIG
    incf     RMAXDIG,F
    movf     CMAXDIG,W
    subwf    RMAXDIG,F
    movf     RMAXDIG,W
    sublw    h'62'               ; Výpočet ukazatele na nejvyšší přij.řád
    movwf    FSR
prevod_BCD_bin_smycka
    movf     INDF,W              ; Vyčtení hodnoty z paměti
    call     add_32v2            ; Přičtení k celku
    decf     RMAXDIG,F           ; Pokud je znak poslední, skok
    btfsc    STATUS,Z
    goto     prevod_BCD_bin_ret
    call     uloz_pred           ; Záloha výsledku před násobením
    call     mul_2xv2            ; Násobení 2x
    call     mul_2xv2            ; Násobení 2x
    call     add_32_p            ; Přičtení zálohovaného výsledku (nasob.celk.5x)
    call     mul_2xv2            ; Násobení 2x ...celkem tedy 10x
    incf     FSR,F               ; Navýšení ukazatele
    goto     prevod_BCD_bin_smycka
prevod_BCD_bin_ret
    bsf      SC8                 ; Povolení spuštění procedury mul_32x16
    return                       ; Konec podprogramu prevod_BCD_bin

mul_2xv2                         ; Podprogram pro 32-bitové násobení 2x
    bcf      STATUS,C
    rlf      V3,F                ; Přenos mezi registry při posunu je
    rlf      V2,F                ; přenášen v C
    rlf      V1,F
    rlf      V0,F
    bcf      STATUS,C
    return

add_32_p                         ; 32-bitové sčítání se zálohovanou předchozí
    movf     ADAT4,W             ; hodnotou
    addwf    V3,F
    movf     ADAT3,w
    btfsc    STATUS,C
    incfsz   ADAT3,w
    addwf    V2,F
    movf     ADAT2,w
    btfsc    STATUS,C
    incfsz   ADAT2,w
    addwf    V1,F
    movf     ADAT1,w
    btfsc    STATUS,C
    incfsz   ADAT1,w
    addwf    V0,F
    return
```

```
add_32v2                                ; 32-bitové přičítání
    addwf    V3,F
    clrw
    btfsc    STATUS,C
    incfsz   V2,w
    addwf    V2,F
    btfsc    STATUS,C
    incfsz   V1,w
    addwf    V1,F
    btfsc    STATUS,C
    incfsz   V0,w
    addwf    V0,F
    return

uloz_pred                               ; Záloha mezivýsledku do pomocných proměnných
    movf     V3,W
    movwf    ADAT4
    movf     V2,W
    movwf    ADAT3
    movf     V1,W
    movwf    ADAT2
    movf     V0,W
    movwf    ADAT1
    return
```

B.5 Podprogram pro binární násobení korekční konstantou

```
mul_32x16
    movf      MULT0,W              ; Načtení konstant pro násobení
    movwf     N0
    movf      MULT1,W
    movwf     N1
mul32x16_repeat
    btfsc     N0,0                 ; Pokud LSB=1 volej proceduru add_52_p
    call      add_52_p
    call      mul32x16_shift
    movf      N0,F                 ; Dokud nejsou konstanty nulové opakuj
    btfss     STATUS,Z
    goto      mul32x16_repeat
    movf      N1,F
    btfss     STATUS,Z
    goto      mul32x16_repeat
    clrf      V3                   ; Vymazání proměnných
    clrf      V2
    clrf      V1
    clrf      V0
    clrf      VP
    clrf      VPP
    bsf       SC9                  ; Povolení spuštění procedury prevod_bin_BCD
    return                         ; Konec podprogramu mul_32x16

mul32x16_shift                     ; Bitový posun výsledku vlevo
    bcf       STATUS,C
    rlf       V3,F
    rlf       V2,F
    rlf       V1,F
    rlf       V0,F
    rlf       VP,F
    rlf       VPP,F
    bcf       STATUS,C
    rrf       N1,F                 ; Bitový posun násobitele vpravo
    rrf       N0,F
    bcf       STATUS,C
    return

add_52_p                           ; 52-bitové sčítání výsledku s mezivýsledkem
    movf      V3,W
    addwf     C5,F
    movf      V2,W
    btfsc     STATUS,C
    incfsz    V2,W
    addwf     C4,F
    movf      V1,W
    btfsc     STATUS,C
    incfsz    V1,W
    addwf     C3,F
    movf      V0,W
    btfsc     STATUS,C
    incfsz    V0,W
    addwf     C2,F
    movf      VP,W
    btfsc     STATUS,C
    incfsz    VP,W
    addwf     C1,F
    movf      VPP,W
    btfsc     STATUS,C
    incfsz    VPP,W
    addwf     C0,F
    return
```

B.6 Podprogram pro převod z binárního formátu na BCD

```
prevod_bin_BCD
    bcf     STATUS,Z
    bcf     STATUS,C
    clrf    BP0
    clrf    BP1
;--d_0
    movf    C5,W                ; h_0*1
    andlw   b'00001111'
    call    add_part
    movlw   d'6'                ; h_1*6
    movwf   MBCC
    swapf   C5,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_2*6
    movwf   MBCC
    movf    C4,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_3*6
    movwf   MBCC
    swapf   C4,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_4*6
    movwf   MBCC
    movf    C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_5*6
    movwf   MBCC
    swapf   C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_6*6
    movwf   MBCC
    movf    C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_7*6
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_8*6
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_9*6
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
```

```
    movlw   d'6'                ; h_10*6
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    call    mod_10
    movwf   B0
    movf    BPP1,W
    addwf   B1,F
    movf    BPP2,W
    addwf   B2,F
    clrf    BP0
    clrf    BP1
;--d_1
    swapf   C5,W                ; h_1*1
    andlw   b'00001111'
    call    add_part
    movlw   d'5'                ; h_2*6
    movwf   MBCC
    movf    C4,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'9'                ; h_3*9
    movwf   MBCC
    swapf   C4,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'3'                ; h_4*3
    movwf   MBCC
    movf    C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'7'                ; h_5*7
    movwf   MBCC
    swapf   C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    C2,W                ; h_6*1
    andlw   b'00001111'
    call    add_part
    movlw   d'5'                ; h_7*5
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'9'                ; h_8*9
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'3'                ; h_9*3
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
```

```
    movlw   d'7'                ; h_10*7
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B1,W
    call    add_part
    call    mod_10
    movwf   B1
    movf    BPP1,W
    addwf   B2,F
    movf    BPP2,W
    addwf   B3,F
    clrf    BP0
    clrf    BP1
;--d_2
    movlw   d'2'                ; h_2*2
    movwf   MBCC
    movf    C4,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'5'                ; h_4*5
    movwf   MBCC
    movf    C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'5'                ; h_5*5
    movwf   MBCC
    swapf   C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'2'                ; h_6*2
    movwf   MBCC
    movf    C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'4'                ; h_7*4
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'2'                ; h_8*2
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'7'                ; h_9*7
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
```

```
    movlw   d'7'                ; h_10*7
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B2,W
    call    add_part
    call    mod_10
    movwf   B2
    movf    BPP1,W
    addwf   B3,F
    movf    BPP2,W
    addwf   B4,F
    clrf    BP0
    clrf    BP1
;--d_3
    movlw   d'4'                ; h_3*4
    movwf   MBCC
    swapf   C4,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'5'                ; h_4*5
    movwf   MBCC
    movf    C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'8'                ; h_5*8
    movwf   MBCC
    swapf   C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'7'                ; h_6*7
    movwf   MBCC
    movf    C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'5'                ; h_7*5
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'7'                ; h_8*7
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_9*6
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
```

```
    movlw   d'7'                ; h_10*7
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B3,W
    call    add_part
    call    mod_10
    movwf   B3
    movf    BPP1,W
    addwf   B4,F
    movf    BPP2,W
    addwf   B5,F
    clrf    BP0
    clrf    BP1
;--d_4
    movlw   d'6'                ; h_4*6
    movwf   MBCC
    movf    C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'4'                ; h_5*4
    movwf   MBCC
    swapf   C3,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'7'                ; h_6*7
    movwf   MBCC
    movf    C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'3'                ; h_7*3
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_8*6
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'7'                ; h_9*7
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'2'                ; h_10*2
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B4,W
    call    add_part
    call    mod_10
    movwf   B4
    movf    BPP1,W
    addwf   B5,F
    movf    BPP2,W
    addwf   B6,F
    clrf    BP0
    clrf    BP1
```

```
;--d_5
    movlw   d'7'                ; h_6*7
    movwf   MBCC
    movf    C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'4'                ; h_7*4
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'9'                ; h_8*9
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'4'                ; h_9*4
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'6'                ; h_10*6
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B5,W
    call    add_part
    call    mod_10
    movwf   B5
    movf    BPP1,W
    addwf   B6,F
    movf    BPP2,W
    addwf   B7,F
    clrf    BP0
    clrf    BP1
;--d_6
    swapf   C3,W                ; h_5*1
    andlw   b'00001111'
    call    add_part
    movlw   d'6'                ; h_6*6
    movwf   MBCC
    movf    C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'8'                ; h_7*8
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'4'                ; h_8*4
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'9'                ; h_9*9
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
```

```
    movf    C0,W                ; h_10*1
    andlw   b'00001111'
    call    add_part
    movf    B6,W
    call    add_part
    call    mod_10
    movwf   B6
    movf    BPP1,W
    addwf   B7,F
    movf    BPP2,W
    addwf   B8,F
    clrf    BP0
    clrf    BP1
;--d_7
    movf    C2,W                ; h_6*1
    andlw   b'00001111'
    call    add_part
    movlw   d'6'                ; h_7*6
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'9'                ; h_8*9
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    swapf   C1,W                ; h_9*1
    andlw   b'00001111'
    call    add_part
    movf    C0,W
    andlw   b'00001111'         ; h_10*1
    call    add_part
    movf    B7,W
    call    add_part
    call    mod_10
    movwf   B7
    movf    BPP1,W
    addwf   B8,F
    movf    BPP2,W
    addwf   B9,F
    clrf    BP0
    clrf    BP1
;--d_8
    movlw   d'2'                ; h_7*2
    movwf   MBCC
    swapf   C2,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'2'                ; h_8*2
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'7'                ; h_9*7
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
```

```
    movlw   d'5'                ; h_10*5
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B8,W
    call    add_part
    call    mod_10
    movwf   B8
    movf    BPP1,W
    addwf   B9,F
    movf    BPP2,W
    addwf   B10,F
    clrf    BP0
    clrf    BP1
;--d_9
    movlw   d'4'                ; h_8*4
    movwf   MBCC
    movf    C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'8'                ; h_9*8
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'9'                ; h_10*9
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B9,W
    call    add_part
    call    mod_10
    movwf   B9
    movf    BPP1,W
    addwf   B10,F
    movf    BPP2,W
    addwf   B11,F
    clrf    BP0
    clrf    BP1
;--d_10
    movlw   d'6'                ; h_9*6
    movwf   MBCC
    swapf   C1,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movlw   d'9'                ; h_10*9
    movwf   MBCC
    movf    C0,W
    andlw   b'00001111'
    call    mul_by_con
    call    add_part
    movf    B10,W
    call    add_part
    call    mod_10
    movwf   B10
    movf    BPP1,W
    addwf   B11,F
    movf    BPP2,W
    addwf   B12,F
    clrf    BP0
    clrf    BP1
```

```
;--d_11
    movf    B11,W
    call    add_part
    call    mod_10
    movwf   B11
    movf    BPP1,W
    addwf   B12,F
    clrf    BP0
    clrf    BP1
;--d_12
    movf    C0,W            ; h_10*1
    andlw   b'00001111'
    call    add_part
    movf    B12,W
    call    add_part
    call    mod_10
    movwf   B12
    clrf    BP0
    clrf    BP1
    bsf     SC10
    return

mul_by_con                  ; Procedura pro násobení h_x číslem uloženým
    movwf   MBCP            ; v registru MBCC
    movf    MBCP,F
    btfsc   STATUS,Z        ; Kontrola nenulovosti, pokud MBCP=0 konec
    goto    mul_by_con_end
    clrw
mul_by_con_repeat
    addwf   MBCP,W          ; Cyklus přičítáni, pokud MBCC=0 konec
    decfsz  MBCC,F
    goto    mul_by_con_repeat
mul_by_con_end
    return

mod_10                      ; Procedura k rozdělení b na složky řádů
    movf    BP0,F
    btfsc   STATUS,Z        ; Pokud BP0=0, zkontroluj BP1, jinak spusť algoritmus
    goto    mod_10_BP1_check
    goto    mod_10_start
mod_10_BP1_check
    movf    BP0,F
    btfss   STATUS,Z        ; Pokud BP1=0, vynuluj BPP1,BPP2,W a konec, jinak
    goto    mod_10_start    ; spusť algoritmus
    clrf    BPP1
    clrf    BPP2
    clrw
    return
mod_10_start
    clrf    BPP1            ; Nulování pomocných registrů a pracovního registru
    clrf    BPP2
    clrw
    movlw   STO
mod_10_start_100repeat
    incf    BPP2,F
    bcf     STATUS,C        ; Odečti d'100' od BP0, pokud C=0(záporný výsledek),
    subwf   BP0,F           ; dekrementuj 1x BP1, pokud je však BP1 záporné pak
    btfsc   STATUS,C        ; přičti k BP0 hodnotu 100 a pokračuj na další část
    goto    mod_10_start_100repeat
    movf    BP1,F
    btfsc   STATUS,Z
    goto    mod_10_start_10start
    decf    BP1,F
    goto    mod_10_start_100repeat
```

```
mod_10_start_10start
    movlw    STO
    decf     BPP2,F           ; Přičti nadbytečně odečtenou 100 a odstraň spatně
    addwf    BP0,F            ; indikovanou 100
    movlw    DESET
    movf     BP0,F            ; Pokud BP0=0, vynuluj pracovní registr a konec
    btfss    STATUS,Z
    goto     mod_10_start_10repeat
    clrw
    return
mod_10_start_10repeat
    incf     BPP1,F
    subwf    BP0,F
    btfsc    STATUS,C
    goto     mod_10_start_10repeat
    addwf    BP0,W
    decf     BPP1,F
    return

add_part
    bcf      STATUS,C         ; 16-bitové přičítání
    addwf    BP0,F
    btfsc    STATUS,C
    incf     BP1,F
    return
```

B.7 Podprogram pro výstupní úpravu řetězce

```
vystupni_uprava              ; Procedura pro zjištění počtu vysílaných znaků
    movlw    d'252'          ; Příprava registru označujícího desetinnou čárku
    movwf    BCA
    movf     BCOUNT,W        ; a jejich převod z BCD na ASCII
    movwf    FSR             ; Nastavení ukazatele na nejvyšší registr BCD
vystupni_uprava_repeat
    movf     INDF,W          ; Načtení čísla d_x z paměti
    btfsc    STATUS,Z        ; Pokud je registr nulový proveď násl. instr.
    btfsc    SC7             ; Byl již převedeno nějaké číslo?, pokud ne (SC7=0)
    call     preved_na_ASCII ; přeskoč násl. instr.
    btfss    SC7             ; Pokud SC7=0 proveď násl.instr.
    incf     BCOUNT,F        ; Inkrementace ukazatele na první nenulový registr
    movf     FSR,W           ; Kontrola konce procházené paměti
    xorlw    h'47'
    btfsc    STATUS,Z        ; Pokud je konec procházené paměti, proveď násl.instr.
    goto     vystupni_uprava_fin
    incf     FSR,F           ; Inkrementace ukazatele na procházenou paměť
    goto     vystupni_uprava_repeat
vystupni_uprava_fin
    bcf      STATUS,C
    movf     BCOUNT,W        ; Je-li BCOUNT>h'43' jedná se o desetinné číslo
    sublw    h'43'           ; začínající nulou proto do BCOUNT vložit h'43'
    btfsc    STATUS,C        ; Pokud je BCOUNT<=h'43' proveď násl. instr.
    goto     vystupni_uprava_fin_return
    movlw    h'43'
    movwf    BCOUNT
    movlw    h'30'           ; Převod nuly před desetinnou čárkou
    movwf    B6
    bcf      SC10            ; Odstranění povolení pro vystupni_uprava
    bsf      SC11             ; Povolení procedury odeslani_retezce
    return                   ; Konec podprogramu vystupni_uprava
vystupni_uprava_fin_return
    bcf      SC10            ; Odstranění povolení pro vystupni_uprava
    bsf      SC11            ; Povolení procedury odeslani_retezce
    return                   ; Konec podprogramu vystupni_uprava

preved_na_ASCII
    btfss    SC7             ; Pokud je první převáděný znak nastav SC7
    bsf      SC7
    addlw    h'30'
    movwf    INDF
    bcf      STATUS,C
    return
```

B.8 Podprogram pro odeslání řetězce

```
odeslani_retezce
bankpg1                         ; Makro pro nastavení banky 1
    movlw    b'00100110'        ; Nastaveni vysílače UART
    movwf    TXSTA
    movlw    d'25'
    movwf    SPBRG
bankpg0                         ; Makro pro nastavení banky 0
    movf     BCOUNT,W           ; Ukazatel na začátek odesílané paměti
    movwf    FSR
odeslani_retezce_repeat
    movf     INDF,W             ; Vyčtení znaku a odeslání
    call     odeslani_retezce_odesli_znak
    movf     FSR,W              ; Kontrola zda je konec odesílané paměti
    xorlw    h'47'
    btfsc    STATUS,Z           ; Pokud je konec od.paměti odešli konečné znaky
    goto     odeslani_retezce_fin
    incf     FSR,F
    goto     odeslani_retezce_repeat
odeslani_retezce_fin
    movf     RS0,W              ; Odeslání znaku h'0A'
    call     odeslani_retezce_odesli_znak
    movf     RS1,W              ; Odeslání znaku h'0D'
    call     odeslani_retezce_odesli_znak
    bcf      SC11               ; Odstranění povolení pro odeslani_retezce
    bsf      SC12               ; Povolení pro odstranění řetězce (restart)
    return                      ; Konec podprogramu odeslani_retezce

odeslani_retezce_odesli_znak ; Procedura pro odeslání znaku
    movwf    TXREG
    nop
    btfss    PIR1,TXIF          ; Testuj dokud nebudou data vložena do TSR
    goto     $-1                ; $
    bcf      STATUS,Z           ; Mazáni příznakového bitu
    return                      ; Konec podprogramu odeslani_retezce_odesli_znak
```

C ASCII TABULKA

DEC	HEX	ZNAK	DEC	HEX	ZNAK	DEC	HEX	ZNAK	DEC	HEX	ZNAK
0	00	NULL	32	20	SPACE	64	40	@	96	60	‘
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	”	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	ˆ	126	7E	˜
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Tab. C.1: Nižší část ASCII tabulky.

D OBSAH DOPROVODNÉHO CD

Součástí této bakalářské práce je i CD, které má následující adresářovou strukturu.

/asm-zdroj – Zdrojový kód obslužného programu.

/konstrukce – Schéma zapojení, podklad pro tvorbu DPS.