VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

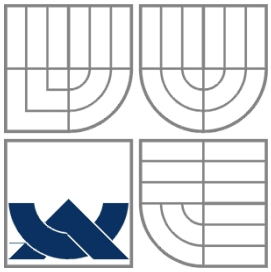
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

ŘÍZENÍ KOMUNIKACE PO SBĚRNICI USB

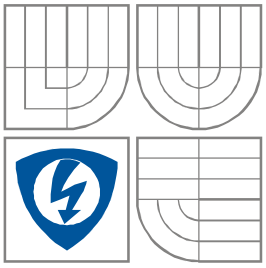
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. ZDENĚK HLAVICA
AUTHOR

BRNO 2008



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

ŘÍZENÍ KOMUNIKACE PO SBĚRNICI USB

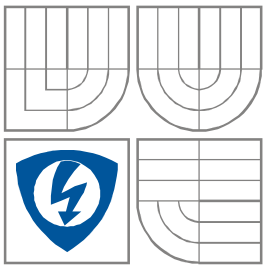
CONTROL OF THE COMMUNICATION ON THE USB

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Zdeněk Hlavica
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Aleš Prokeš, Ph.D.
SUPERVISOR

BRNO, 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor

Elektronika a sdělovací technika

Student:	Hlavica Zdeněk Bc.	***ID:***	89356
Ročník:	2	***Akademický rok:***	2007/2008

NÁZEV TÉMATU:

Řízení komunikace po sběrnici USB

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s metodami komunikace po sběrnici USB a prostudujte používané standardy a jejich vlastnosti. Vytvořte přehled dostupných řadičů typu "host" a uveďte jejich základní dělení a strukturu. Navrhněte způsob připojení řadiče USB k mikroprocesoru pomocí sběrnice SPI nebo osmibitového portu. Seznamte se strukturou paměťových médií typu "flash disc", se způsobem ukládání, mazání a čtení dat, tvorbou adresářů a způsobem přístupu k datům.

Navrhněte způsob propojení zvoleného řadiče USB s vývojovou deskou mikroprocesoru Silicon Laboratories C8051F120 a realizujte jej. Vytvořte SW pro obsluhu řadiče a program pro záznam dat na USB flash disk. Je požadována možnost vytvoření adresáře, souboru vč. uložení data a času vzniku, čtení souboru, případně formátování disku a mazání dat. Předpokládejte ukládání relativně dlouhých sekvencí dat.

DOPORUČENÁ LITERATURA:

[1] AXELSON, J. USB Mass Storage Designing and Programming Devices and Embedded Hosts. Madison WI: Lakeview Research LLC, 2006.

[2] BURKHARD, K. USB - měření, řízení a regulace pomoí sběrnice USB. Praha: BEN - technická literatura, 2002.

[3] 8051 Mixed-Signal Microcontrollers [online]. Dostupné na WWW: http://www.silabs.com/tgwWebApp/public/index.htm

Termín zadání: 5.10.2007 ***Termín odevzdání:*** 30.5.2008

Vedoucí práce: doc. Ing. Aleš Prokeš, Ph.D.

prof. Dr. Ing. Zbyněk Raida
předseda oborové rady

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:	Zdeněk Hlavica
Bytem:	Kelč 345, Kelč, 756 43
Narozen/a (datum a místo):	3. června 1984 ve Valašském Meziříčí

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 53, Brno, 602 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika
(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

   ☐ disertační práce
   ☒ diplomová práce
   ☐ bakalářská práce
   ☐ jiná práce, jejíž druh je specifikován jako .....
   (dále jen VŠKP nebo dílo)

Název VŠKP:	Řízení komunikace po sběrnici USB
Vedoucí/ školitel VŠKP:	doc. Ing. Aleš Prokeš, Ph.D.
Ústav:	Ústav radioelektroniky
Datum obhajoby VŠKP:	_____

   VŠKP odevzdal autor nabyvateli*:

   ☒ v tištěné formě – počet exemplářů: 2
   ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

* hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 30. května 2008

……..	……
Nabyvatel	Autor

Abstrakt

Cílem diplomové práce Komunikace po sběrnici USB je realizace hardwaru a softwaru pro mikrokontrolér C8051F120 umožňující záznam dat na paměťové médium flash disk. Zahrnuje studium sběrnice USB, potřebných přenosových protokolů a souborového systému. Také se zabývá hostitelskými řadiči USB a obvodem MAX3421E firmy Maxim vybraným pro vývoj diplomové práce. Je požadována schopnost vytvářet adresáře a soubory včetně záznamu data vytvoření, čtení a záznam dat do souborů v souborovém systému FAT16.

Klíčová slova

USB, Univerzální sériová sběrnice, Flash disk, Bulk-Only, MAX3421E, FAT16, File Allocation Table, USB Hostitel

Abstract

The aim of Master's thesis Control of the communication on the USB is realization hardware and software designed for microcontroller C8051F120 what is enabling recording of data on memory medium Flash drive. It includes study of USB, transfer protocols, which are needed and file system. The thesis deals about host type USB controllers and about USB controller MAX3421E fy Maxim, which has been choosen for development of software. It is required ability to create directories and files including recording data to file and ability of reading data from files in the file system FAT16.

Key words

USB, Universal Serial Bus, Flash drive, Bulk-Only, MAX3421E, FAT16, File Allocation Table, USB Host

Bibliografická citace

HLAVICA, Z. *Řízení komunikace po sběrnici USB.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 128 s. Vedoucí diplomové práce doc. Ing. Aleš Prokeš, Ph.D.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma Komunikace po sběrnici USB jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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 30. května 2008 .....
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce doc. Ing. Aleši Prokešovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne 30. května 2008 .....
podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Diplomová práce Řízení komunikace po sběrnici USB se zabývá praktickým řešením realizace hardwaru a softwaru, umožňující přenos dat mezi mikrokontrolérem C8051F120 firmy Silicon Laboratories a rozšířeným paměťovým médiem USB flash diskem. Hlavním cílem a výstupem práce je realizace softwaru pro uvedený mikrokontrolér a také realizace vzorku hardwaru, umožňující manipulaci se soubory na běžném USB flash disku v požadovaném souborovém systému.

Práce je rozdělena na části, týkající se vždy jedné dílčí oblasti, jejichž pochopení je nezbytné k dosažení hlavního cíle. Jak naznačuje vlastní název práce, začíná popisem vlastní sběrnice USB, její architektury, vlastností a způsobem přenosu dat v rozsahu, který je nutný k pochopení dalších kapitol či realizovaných algoritmů softwaru.

Další část práce se zabývá vlastním cílovým paměťovým zařízením - USB flash diskem. Protože cílem práce je komunikace, nezabývá se hardwarovým řešením tohoto zařízení, nýbrž jen pohledem na zařízení z hlediska potřebné komunikace, použitých přenosových protokolů a logické organizace paměťového pole.

Základem každé úspěšné komunikace je zvládnutí daného přenosového protokolu. Proto je velká část práce věnována právě popisu transportního protokolu *Bulk-Only*, navrženého pro přenos dat po sběrnici USB, a který používají nejen USB flash disky.

Následuje stručný úvod do problematiky rozhraní *SCSI*, které používá pro komunikaci s počítačovým systémem spousta současných zařízení včetně USB flash disků. Protože tento standard je značně rozsáhlý, popis je omezen prakticky jen na implementované příkazy v této diplomové práci.

Se znalostmi základních příkazů sady *SCSI* je již možné přistupovat k paměťovému poli záznamových zařízení podporující toto rozhranní. Data jsou obvykle v paměťovém poli sdružena do souborů a ty jsou pak organizována do tzv. souborového systému. Dle zadání je požadována schopnost práce v souborovém systému *FAT16*. Struktura a způsob práce s tímto systémem je tedy předmětem další kapitoly práce.

V další části je pojednáno o některých dostupných řadičích USB, které dokáží zprostředkovat funkci hostitele sběrnice USB. Použití obvodu tohoto druhu je nezbytné k uskutečnění komunikace s periferními zařízeními USB, kterým je také USB flash disk. V následující kapitole je přehledově popsán vybraný řadič MAX3421E a v další kapitole návrh jeho připojení k vývojové desce s výše uvedeným mikrokontrolérem.

Poslední a nejrozsáhlejší část práce se týká vlastního komunikačního softwaru, pro jehož realizaci jsou čerpány informace ze všech předchozích kapitol práce. Tato část slouží především jako návod k použití uživatelských funkcí softwaru a zároveň jako doprovodný popis, který umožní lépe porozumět řešení použitých algoritmů.

2 Sběrnice USB

Standard sběrnice USB (*Universal Serial Bus*) definuje především architekturu sběrnice USB, elektrické a mechanické vlastnosti sběrnice, přenosový protokol a s ním spojený datový tok na sběrnici a také zařízení pro větvení sběrnice - USB rozbočovače.

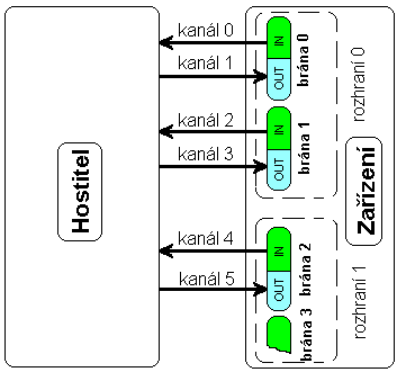
Ve verzi specifikace 1.1 jsou definovány dva základní druhy rozhraní USB. První je zařízení s nízkou rychlostí (*low-speed devices*, - *LS*) - 1,5Mbit/s. Patří sem např. počítačové myši, klávesnice a herní pákové ovladače. Druhé je zařízení s plnou rychlostí (*full-speed devices*, - *FS*), - 12Mbit/s a jsou určena především pro přenos dat jako je komunikace s tiskárnou, zvuk nebo videodata (webkamery). Současná verze specifikace 2.0 rozšířila standard o další druh zařízení - zařízení s vysokou rychlostí přenosu (*high-speed devices*, - *HS*) - 480Mbit/s, které umožňují efektivní připojení zařízení pro rychlý přenos dat. Protože zadání diplomové práce určuje použití USB řadiče pracujícího s přenosovou rychlostí *full-speed*, stručný popis je omezen pouze na oblast specifikace týkající se této přenosové rychlosti. Informace o sběrnici USB jsou čerpány z [1], [2], [10], [11], [12], [13] a [16].

2.1 Fyzická vrstva USB

Fyzická vrstva je nejnižší vrstva přenosového modelu USB. Zde je stručně nastíněna architektura sběrnice a elektrické vlastnosti signálové části sběrnice. Mechanické vlastnosti, napájení a kódování dat lze nalézt v [1] popř. v [2].

2.1.1 Architektura USB

Architekturu sběrnice USB tvoří hostitel, rozbočovače a koncová zařízení. Hostitel byl dříve osazován pouze do osobních počítačů, dnes je již vyrábí jako samostatné integrované obvody. Jednotlivé přenosy jsou uspořádány do logických kanálů, kterými probíhá patřičná komunikace, např. viz obr. 1.

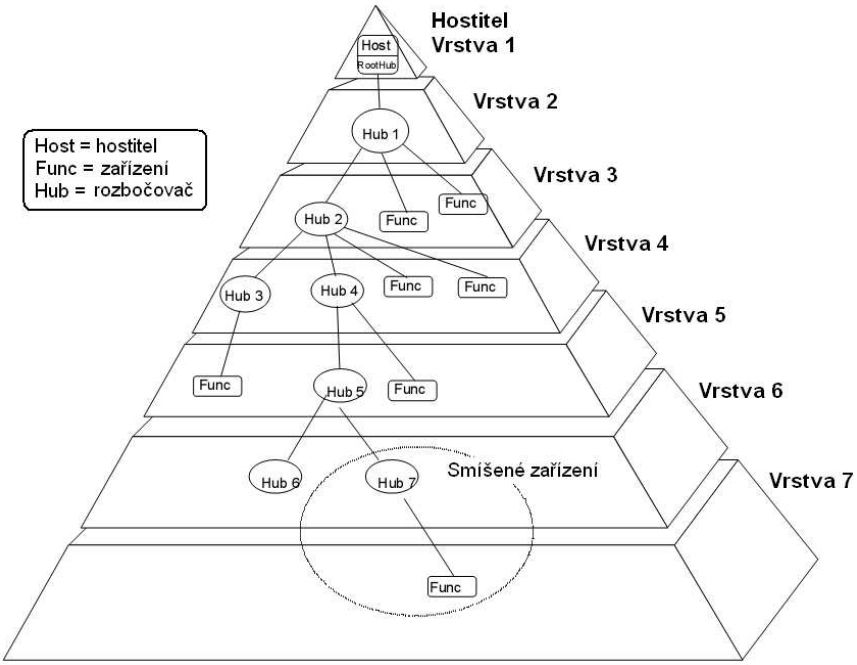


Obr. 1 Znázornění toku dat mezi hostitelem a zařízením

Na obr. 2 (převzatý z [1]) je znázorněna stromová topologie sběrnice USB. Hostitel je vždy na úplném počátku hierarchie sběrnice a na jedné sběrnici USB může být pouze jeden. Hostitel přiděluje přenosové médium metodou výzvy. Jednotlivá zařízení na sběrnici jsou

identifikovaná jedinečnou adresou, kterou vždy po připojení přidělí hostitel. Pokud zařízení přijme pověření se svou adresou, zareaguje na něj patřičným způsobem. Pomocí rozbočovačů lze připojit na sběrnici až 127 zařízení.

Zařízení USB (*DEVICE*) obsahuje minimálně jednu cílovou bránu. Brána (*endpoint*) je jakýsi logický koncový bod komunikace. Každá brána má obvykle svoji vysílací a přijímací paměť pro datové pakety. Hostitel tedy směřuje jednotlivé pakety vždy na konkrétní bránu v konkrétním zařízení. Brána 0 je obousměrná, ostatní brány můžou být nastaveny jako jen vstupní nebo jen výstupní. Zařízení *FS* může mít kromě brány 0 až 15 dalších bran. Maximální velikost paketu pro brány zařízení *FS* je 64 bajtů (jsou myšleny užitečná data v datovém paketu). Maximální velikost paketu pro bránu 0 může být nastavena na 8, 16, 32 nebo 64 bajtů (určuje výrobce daného zařízení). Jednotlivé brány jsou sdruženy do rozhraní (*interfaces*). Každé rozhraní tvoří vždy jednu samostatnou funkci zařízení.



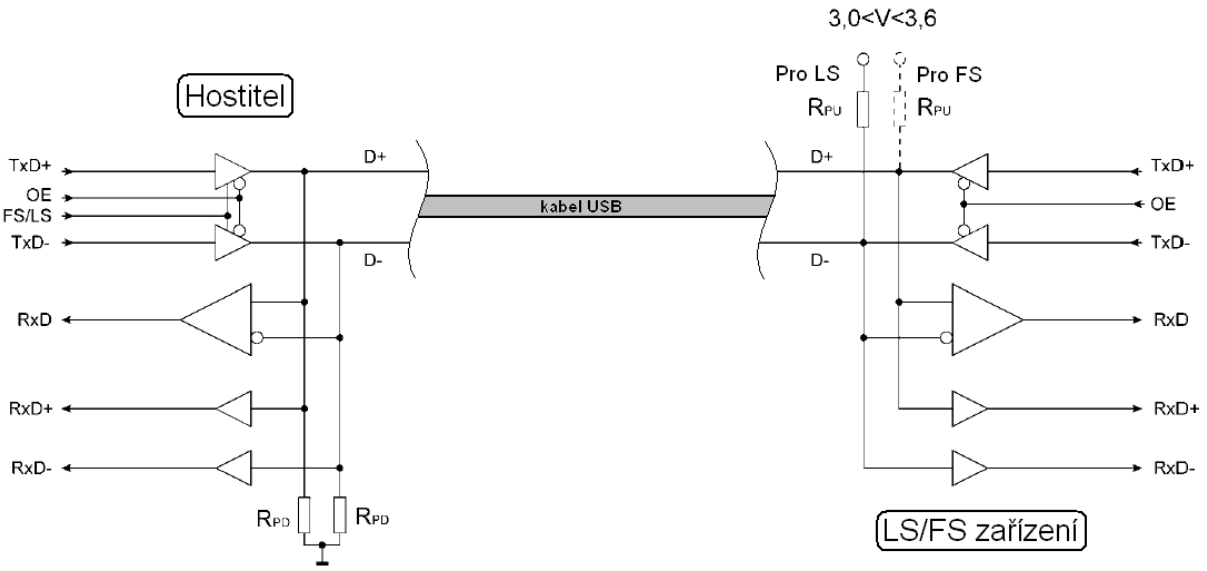
Obr. 2 Stromová struktura sběrnice USB

2.1.2 Elektrické vlastnosti sběrnice

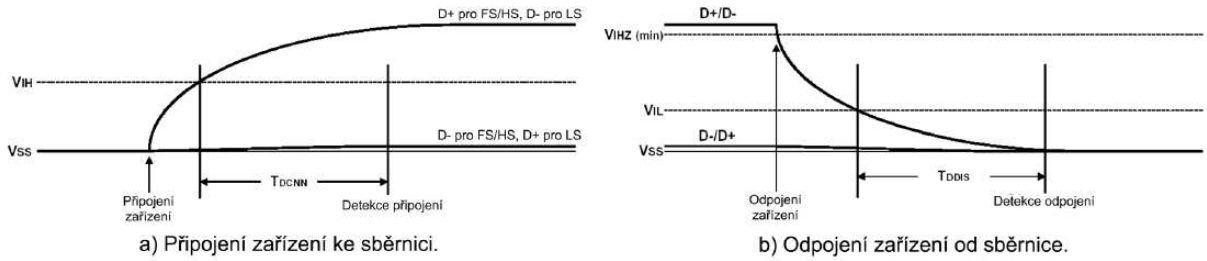
Datové vodiče USB tvoří dvojice *D+* a *D-* s diferenciálním přenosem signálu. Nízká úroveň signálu je vyhodnocena když je $V_{OL} \leq 0{,}3V$ při zátěži $R_{PU}=1{,}5k\Omega$ proti napětí 3,6V. Vysoká úroveň signálu je vyhodnocena když je $V_{OH} \geq 2{,}8V$ při zátěži $R_{PD}=15k\Omega$ proti napětí 0V. Pro umožnění obousměrného poloduplexního přenosu musejí mít budiče sběrnice třístavový výstup.

Zapojení budičů a přijímačů signálu *FS* a *LS* USB transceiveru je uvedeno na obr. 3. Rezistory R_{PD} zaručují, že bez připojeného zařízení bude napětí na datových vodičích *D+* a *D-* nulové. USB periferie má vždy na jednom ze signálových vodičů zapojen rezistor R_{PU} proti napětí 3,0 – 3,6V. Při připojení zařízení k hostiteli dojde vlivem tohoto rezistoru ke změně stavu. Pokud je R_{PU} připojen k lince *D-*, je detekováno zařízení typu *LS*. Pokud je R_{PU} připojen k lince *D+* je detekováno zařízení typu *FS*.

Stavy na sběrnici USB jsou definovány jako "*rozdílová 1*", "*rozdílová 0*" a "*jednoznačná 0*" (*SE0 – single-ended 0*). USB používá paketový přenos. Paket začíná symbolem *SOP* a končí symbolem *EOP*. Na obr. 4 je znázorněn časový průběh signálů *D+* a *D-* při připojení a odpojení zařízení USB. V tab. 1 jsou uvedeny základní stavy vyskytující se na sběrnici USB při její činnosti.



Obr. 3 Schéma připojení budičů ke sběrnici USB



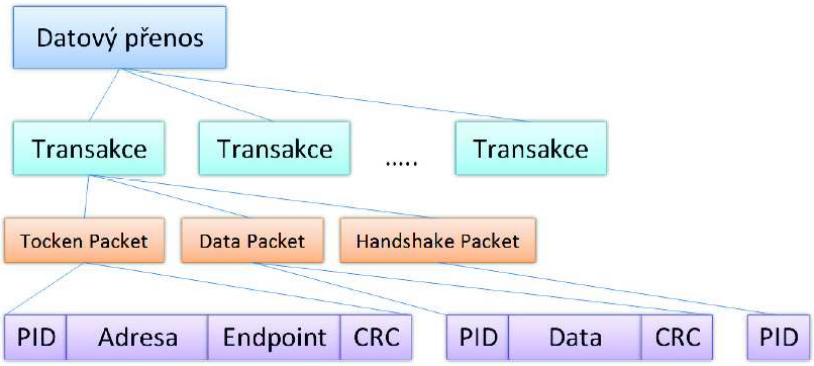
Obr. 4 Časový průběh signálů při připojení a odpojení zařízení

Tab. 1 Základní stavy sběrnice USB

Stav sběrnice	Definice stavu
Rozdílová "1"	$(V_{D+} - V_{D-}) > 200mV$ a $U_{D+} > V_{IH(min)}$
Rozdílová "0"	$(V_{D-} - V_{D+}) > 200mV$ a $U_{D-} > V_{IL(max)}$
Jednoznačná "0" (SE0)	$V_{D+} - V_{D-} < V_{IL(max)}$
Datový stav "J"	Pro FS zařízení: "Rozdílová 1". Pro LS zařízení: "Rozdílová 0"
Datový stav "K"	Pro FS zařízení: "Rozdílová 0". Pro LS zařízení: "Rozdílová 1"
Klidový stav pro FS	$V_{D+} > V_{IHZ(min)}$ a $V_{D-} < V_{IL(max)}$
Klidový stav pro LS	$V_{D-} > V_{IHZ(min)}$ a $V_{D+} < V_{IL(max)}$
Začátek paketu (SOP)	Přechod z klidového stavu do stavu "K"
Konec paketu (EOP)	SE0 na dobu dvou bitů následovaný stavem "J" na dobu jednoho bitu.
Připojení zařízení	Klidový stav po dobu nejméně 2ms
Odpojení zařízení	SE0 po dobu nejméně 2,5µs
Signál reset	V_{D+} a $V_{D-} < V_{IL(max)}$ po dobu nejméně 10ms

2.2 Linková vrstva USB

Na úrovni linkové vrstvy jsou zavedené rámce (*frames*), do kterých jsou libovolně umisťovány přenášené pakety. Rámec začíná přenosem paketu *SOF*, v případě *LS* je použit symbol konce paketu *EOP*. Rámce mají délku pro *FS* a *LS* zařízení 1ms±500ns. Tok dat na USB je znázorněn na obr. 5.



Obr. 5 Tok dat na USB

2.2.1 Pakety

Každý paket USB je dlouhý celočíselný počet bajtů. Detailní strukturu jednotlivých paketů lze nalézt v [1]. Typy paketů jsou uvedeny v tab. 2. Každý paket začíná synchronizačním polem *SYNC* s hodnotou 128, zajišťující synchronizaci přijímače (je použito kanálové kódování *NRZI*).

Tab. 2 Typy paketů

Skupina paketů	Typ paketu	Popis
Pověřovací (Token)	OUT	Přenos dat od hostitele k zařízení
	IN	Přenos dat ze zařízení k hostiteli
	SETUP	Konfigurační přenos hostitele
	SOF	Označuje začátek rámce
Datový (Data)	DATA0	Sudý datový paket
	DATA1	Lichý datový paket
Potvrzovací (Handshake)	ACK	Potvrzení bezchybného příjmu dat
	NAK	Data nebyla přijata nebo vyslána
	STALL	Brána zařízení je pozastavena nebo
		konfigurační požadavek není podporován
Speciální (Special)	PRE	Označuje LS přenos tak, aby rozbočovače
		aktivovaly LS zařízení k nim připojené

Pakety *OUT* (vysílací), *IN* (přijímací) a *SETUP* (konfigurační – speciální případ vysílacího paketu) tvoří pověřovací pakety. Obsahují adresu cílového zařízení a adresu USB brány (*endpoint*) v rámci tohoto zařízení. *LS* zařízení může obsahovat maximálně tři brány, *FS* zařízení může mít až šestnáct bran. Pomocí pověřovacích paketů hostitel zahajuje jednotlivé USB přenosy. Za pakety *OUT* a *SETUP* vždy okamžitě následuje datový paket *DATA0* nebo *DATA1*, nesoucí užitečná data. Při odeslání paketu *IN* vyšle zařízení za nedefinovanou dobu na sběrnici paket *DATA0* nebo *DATA1* hostiteli, nebo bezprostředně odešle potvrzovací paket *NAK* nebo *STALL*.

Ve skupině datových paketů jsou sudý datový paket (*DATA0*) a lichý datový paket (*DATA1*). Při vysílání se tyto pakety střídají, což v případě chyby přenosu nebo ztráty potvrzovacího paketu umožňuje snadno chybu napravit jednoduchým mechanismem opakováním paketů. Před každým datovým paketem musí být na sběrnici vyslán paket *SETUP*, *OUT* nebo *IN*.

Potvrzovací pakety slouží k potvrzení přijetí datových paketů. *ACK* posílá hostitel jako potvrzení správného příjmu paketu ze zařízení (přenos *IN*), nebo *ACK* posílá zařízení jako potvrzení správného příjmu datového paketu od hostitele (přenos *OUT* nebo *SETUP*). *NAK* pošle zařízení hostiteli v případě, že brána není připravena přijmout nebo odeslat data. *STALL* zařízení posílá při nemožnosti přijmout nebo vyslat data (hostitel musí pro obnovení přenosu resetovat brány), nebo může zařízení odeslat *STALL* k indikaci, že zadaný řídící požadavek na bránu 0 není podporován. V tomto případě je brána v tzv. stavu *HALT*.

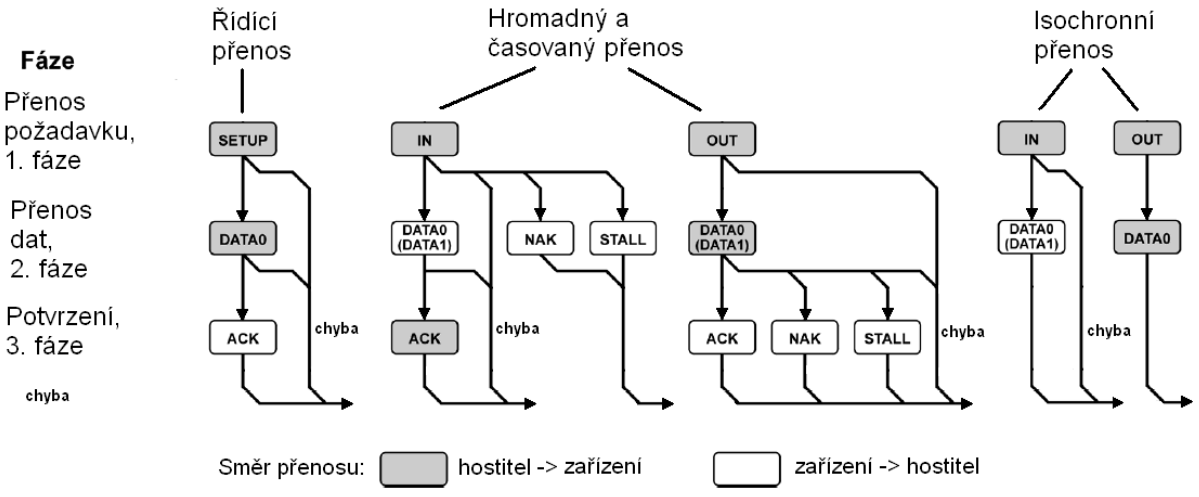
Paket *PRE* slouží pro komunikaci se zařízením *LS* přes rozbočovač. Paket bude hostitelem odeslán před každým přenášeným paketem, směřovaným k *LS* zařízení, rozbočovači.

2.2.2 Typy přenosů po sběrnici USB

Specifikace USB definuje čtyři druhy přenosu po USB sběrnici:

- Řídící přenos (*Control Transfer*)
- Časovaný přenos (*Interrupt Transfer*)
- Isochronní přenos (*Isochronous Transfer*)
- Hromadný přenos (*Bulk Transfer*)

Jednotlivé přenosy mají vždy dvě nebo tři fáze, viz obr. 6. Pozn.: *LS* zařízení mohou využívat pouze řídící a časovaný přenos.



Obr. 6 Typy přenosů USB

Řídící přenos (*Control Transfer*) se skládá ze dvou nebo tří fází. Na obr. 6 je znázorněna pouze první část řídícího přenosu a to přenos požadavku, kdy je vyslán hostitelem paket s konfiguračním pověřením (*SETUP*) následovaný datovým paketem s požadavkem (*DATA0*). Pokud je požadavek v pořádku přijat, zařízení vyšle potvrzení úspěchu. Ostatní fáze řídícího přenosu probíhají stejně jako hromadný přenos, viz kap. 2.2.3.

Další typ přenosu je hromadný. Pro přenos ze zařízení do hostitele vyšle hostitel *IN* paket, na který zařízení odpoví odesláním datového paketu *DATA0/1* nebo paketem *NAK*, jestliže není brána připravena odeslat data. Pokud brána odešle paket *STALL* je pozastavena. Přijme-li hostitel bezchybný datový paket, odešle do zařízení potvrzovací paket *ACK*, nebo byl-li paket poškozen, nezašle žádné potvrzení, což zařízení vyhodnotí jako chybu přenosu. Při vysílání dat do zařízení je situace obdobná. Při hromadném přenosu je zaručeno bezchybné doručení dat kontrolou kontrolních součtů každého paketu.

Časovaný přenos (*Interrupt Transfer*) má stejná pravidla jako přenos hromadný. Rozdíl spočívá ve způsobu přidělování přenosové kapacity média. Hostitel vysílá s definovanou periodou pověření, kdy je zařízení schopno vysílat či přijímat data. Perioda, s jakou se má hostitel dotazovat, je uložena v deskriptoru příslušné brány zařízení, viz kap. 2.3.4. Typické zařízení využívající tento druh přenosu je PC klávesnice nebo myš.

Izochronní přenos (*Isochronous Transfer*) má pouze dvě fáze. Narozdíl od hromadného přenosu se zde nepotvrzuje přijetí dat. Proto není zaručeno jejich doručení ani bezchybnost. Používá se k přenosu souvislého toku dat, např. pro zvukové karty, přičemž je zaručená určitá uživatelská přenosová rychlost či datový tok.

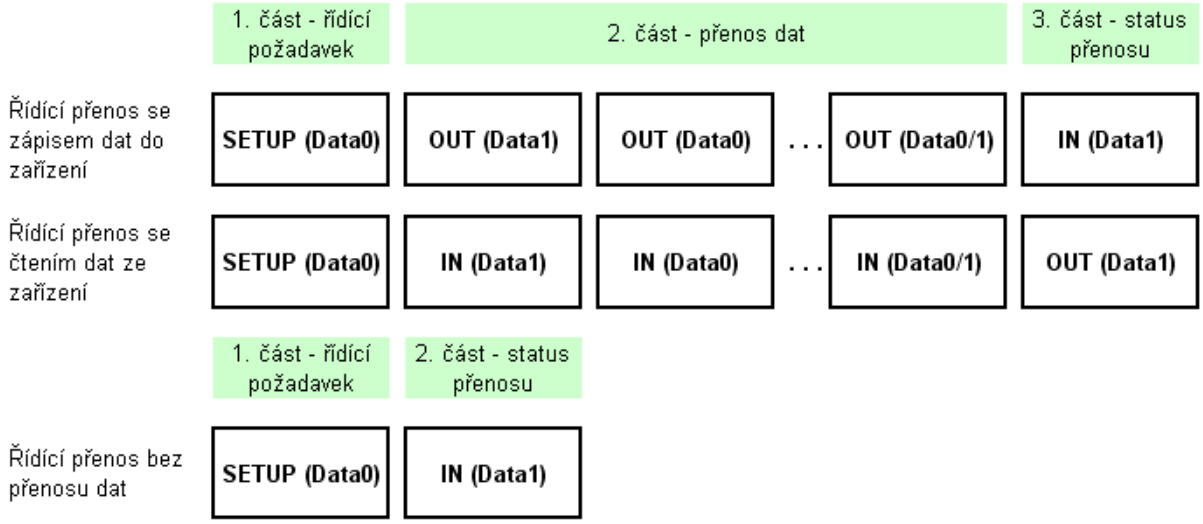
2.2.3 Řídící přenos (Control Transfer)

Po připojení zařízení na sběrnici USB a po provedení jeho resetu se nachází ve stavu obecného zařízení. Bez ohledu na druh zařízení, jeho funkci nebo účel, jediný možný způsob komunikace v tomto stavu je pomocí řídícího přenosu. Řídící přenos podporuje pouze brána 0 (*endpoint 0*) USB zařízení, která je výhradně určena pro tento účel. Řídící přenos tedy slouží k zjištění informací o připojeném zařízení a zvolení jedné z jeho možných konfigurací. Tento proces se nazývá enumerace zařízení. Teprve po úspěšné enumeraci je USB periferní zařízení připraveno k použití pro účel, ke kterému je určeno.

Jsou tři typy řídícího přenosu. Prvním typem je řídící přenos se zápisem dat do zařízení, druhým typem je řídící přenos se čtením dat ze zařízení a posledním je řídící přenos bez přenosu dat. Má vždy dvě nebo tři části. První část je pověření (*SETUP*) s paketem požadavku, druhá část je vlastní přenos dat (pakety *IN*, *OUT* a *DATA0/1*) a třetí část je stavová (pakety IN, OUT a *DATA1*). Stavová část je tvořena paketem s nulovou délkou dat a opačným směrem přenosu než byla vysílána data. Při vysílání dat se musí střídat sudé (*DATA0*) a liché (*DATA1*) datové pakety, přičemž se začíná sudým paketem. Ukončovací paket přenosu je vždy lichý (*DATA1*). Na obr. 7 je znázorněná struktura jednotlivých typů řídích přenosů.

Každý blok (obdélník) na obr. 7 představuje přenos tří paketů na sběrnici USB. Např. v první části přenosu je hostitelem vyslán pověřovací paket *SETUP*, hned za ním následuje datový paket *DATA0*, který nese vlastní data požadavku (viz kap. 2.5) a jako reakci na správné doručení paketu zařízení USB odpovídá potvrzovacím paketem *ACK*. Stejným

způsobem pracuje transakce *OUT* a *IN* v dalších dvou fázích řídícího přenosu. Pakety *SETUP*, *OUT* a *IN* nejsou nositeli dat. Data jsou odesílány jako samostatné datové pakety *DATA0* nebo *DATA1*, před jejichž odesláním musí být vyslán paket *SETUP*, *OUT* nebo *IN*. Pouze při transakci *IN*, kdy je datový paket *DATA0/1* odesílán ze zařízení hostiteli, potvrzovací paket *ACK* posílá hostitel do zařízení.



Obr. 7 Struktura řídících přenosů

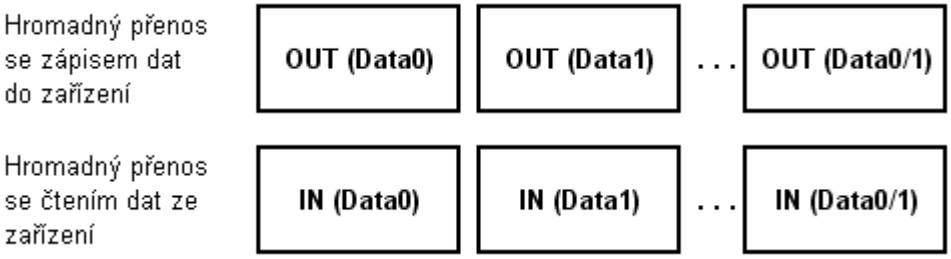
Pakety musí splňovat určitá pravidla. První datový paket je typu *DATA0*, je odeslán za paketem *SETUP* a má velikost 8 bajtů, představujících vlastní USB požadavek. Další datový paket je *DATA1* a při dalším přenosu se pakety *DATA0* a *DATA1* neustále střídají. Množství dat v datových paketech musí být rovno maximálnímu množství, které může být bránou 0 odesláno. Tento parametr je uveden v deskriptoru zařízení (viz kap. 2.3.1) a může být 8, 16, 32 nebo 64 bajtů. Menší paket, než je uveden v deskriptoru zařízení, lze odeslat pouze tehdy, pokud se jedná o poslední datový paket v datové části přenosu, nebo pokud je to první datový paket v datové části přenosu a množství celkových přenášených dat je nižší než maximální velikost paketu, který brána 0 podporuje. Datový paket ve třetí části přenosu je potvrzovací a má velikost 0 bajtů.

V řídícím požadavku je hostitelem uvedeno celkové požadované množství dat k odeslání či příjmu pro celý řídící přenos. Pokud si hostitel vyžádá větší množství dat, než může zařízení poskytnout, může zařízení odeslat pouze data, která má k dispozici. Hostitel pak detekuje kratší paket než očekává a transakci ukončí. V případě, že toto menší množství dat, je celočíselným násobkem velikosti brány, musí zařízení ještě na další žádost hostitele (*IN*) odeslat paket s nulovou délkou dat, aby mohl hostitel detekovat krátký paket. Poté ještě proběhne stavová část přenosu a transakce se ukončí.

Pokud je řídící brána pozastavena (stav *HALT*, brána vrací paket *STALL*), další odeslaný paket *SETUP* bránu inicializuje do výchozího stavu a brána musí správně přijmout požadavek. Řídící brána je pozastavena při nemožnosti provést požadavek nebo je to reakce na nepodporovaný požadavek.

2.2.4 Hromadný přenos (Bulk Transfer)

Struktura hromadného přenosu je na obr. 8. Hromadný přenos dat můžou využívat libovolné brány (pokud ho podporují) USB zařízení. Jsou dva typy hromadného přenosu a to přenos dat z hostitele do zařízení (transakce *OUT*) a přenos dat ze zařízení hostiteli (transakce *IN*).



Obr. 8 Struktura hromadného přenosu

Pravidla přenosu jsou stejná jako ve druhé fázi řídícího přenosu. Skládá se pouze s transakcí *IN* nebo *OUT*, přičemž se musejí střídat datové pakety *DATA0* a *DATA1*. Po resetu USB zařízení hromadný přenos začíná vždy paketem *DATA0*.

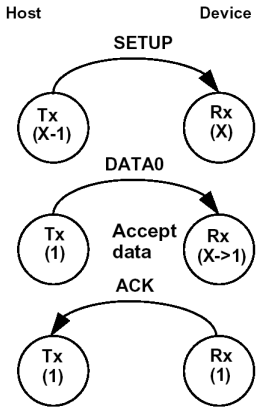
Hromadný přenos tedy nemá speciálně definovaný začátek a konec jako řídící přenos. Na sběrnici jsou vysílány data oběma směry jak je zrovna potřeba. Ztrátu synchronizace mezi hostitelem a zařízením musí řešit hostitel ve vyšší vrstvě.

Každému datovému paketu opět předchází pověřovací paket *IN* nebo *OUT*. Množství dat v datových paketech je opět rovno velikosti hromadné brány. Tento parametr je uveden v deskriptoru příslušné brány. Pouze při odesílání nižšího množství dat než velikost brány nebo pokud se jedná o poslední paket v kontinuálním datovém přenosu může být množství dat v paketu nižší. Pokud zařízení odešle hostiteli nižší množství dat v paketu než očekává, hostitel ukončí celou transakci. Pokud zařízení odešle hostiteli větší množství dat v paketu než očekává, je to chyba a přenos je nutné zopakovat.

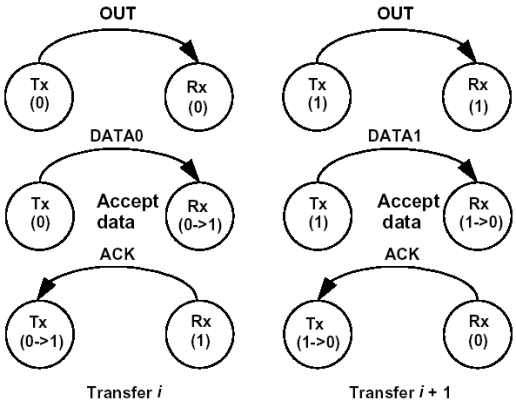
2.2.5 Synchronizace paketů a jejich opakování

Na obr. 9 (převzatý z [1]) je znázorněn tok paketů v první části řídícího přenosu – odeslání požadavku do zařízení. Hostitel vyšle paket *SETUP*, následuje datový paket *DATA0* nesoucí 8 bajtů požadavku. Zařízení tyto dva pakety přijme a pokud jsou smysluplné a bezchybné transakci potvrdí hostiteli potvrzovacím paketem *ACK*. Čísla v závorkách na obr.9 znázorňují jak si hostitel a zařízení nastavují vnitřní sekvenční bity (*Data Toggle Bits*), určující, zda je příště očekáván sudý nebo lichý datový paket.

Na obr. 10 (převzatý z [1]) je znázorněna transakce *OUT*, tedy přenos dat z hostitele do zařízení. Přenos začíná vysláním paketu *OUT*, následuje např. datový paket *DATA0* a po jeho správném doručení zařízení odpovídá potvrzovacím paketem *ACK*. V dalším přenosu je očekáván datový paket *DATA1* atd. Transakce *IN* pracuje obdobně, jen je každý přenos zahájen paketem *IN*, datové pakety *DATA0/1* směřují ze zařízení k hostiteli a potvrzovací pakety *ACK* posílá hostitel do zařízení.

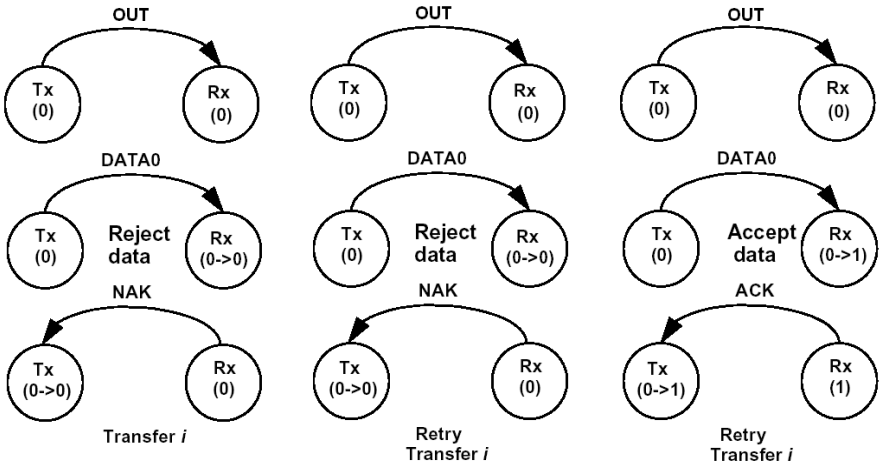


Obr. 9 Přenos požadavku – 1. část řídícího přenosu



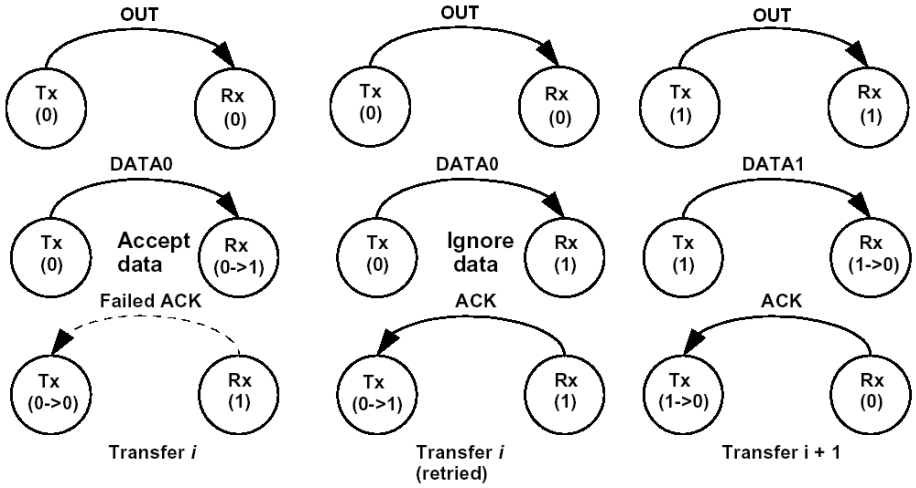
Obr. 10 Transakce OUT

Na obr. 11 (převzatý z [1]) je znázorněna rovněž transakce *OUT*, ale v prvních dvou případech nebyla brána schopna přijmout data a vrátila potvrzovací paket *NAK*. V tomto případě hostitel opakuje neustále stejný paket a sekvenční bit nemění, dokud nebude paket akceptován. Obdobně v případě transakce *IN*, kdy zařízení vrací *NAK*, se hostitel neustále snaží přijmout data ze zařízení opakovaným posíláním paketu *IN* na danou bránu, přičemž sekvenční bit nemění dokud odpovídající datový paket nepřijme.



Obr. 11 Transakce OUT - neakceptovaná data

Na obr. 12 (převzatý z [1]) je uveden příklad ztráty potvrzovacího paketu na sběrnici. Nastala situace kdy zařízení akceptovalo data, ale hostitel podle ztráty *ACK* považuje datový paket za ztracený, proto svůj sekvenční bit nezmění a odešle ten stejný paket na sběrnici znovu. Zařízení tento paket přijme, ale protože nyní očekává paket *DATA1* nikoli *DATA0*, přijatá data bude ignorovat. Avšak správné přijetí opakovaného paketu hostiteli potvrdí odesláním *ACK*. V této chvíli hostitel mění sekvenční bit a odesílá následující paket *DATA1*. Obdobná situace nastává při ztrátě samotného datového paketu, např. když nebude u přijatého paketu odpovídat kontrolní součet. Zařízení v tomto případě žádný potvrzovací paket neodešle záměrně. Hostitel zaznamená time-out a paket bude opakovat. Z uvedeného vyplývá, že díky střídání lichého a sudého datového paketu lze účinně na sběrnici eliminovat vlivy ztráty či poškození paketů při jejich přenosu. Ekvivalentní princip platí pro přenosy ze zařízení do hostitele.



Obr. 12 Ztráta potvrzovacího paketu

2.3 Deskriptory

Deskriptory slouží k popisu zařízení připojeného na sběrnici a tím zajišťují její univerzálnost. Deskriptory jsou pevně definované datové struktury uložené v paměti USB zařízení, které hostitel ze zařízení vyčte po jeho připojení na sběrnici.

Je definováno několik základních typů deskriptorů. Kromě nich existují ještě další typy deskriptorů, např. deskriptory specifické pouze pro určité třídy USB zařízení. Jsou to především deskriptor zařízení, konfigurace, rozhraní, brány a textového řetězce. Tyto deskriptory mají hierarchickou strukturu. Na vrcholu je vždy jeden deskriptor zařízení. Ten se rozvětvuje na jednotlivé konfigurace. Pod každý deskriptor konfigurace patří deskriptory rozhraní a pod ně zase deskriptory bran, kterými jsou daná rozhraní tvořena. Každý deskriptor rozhranní s přidruženými bránami tvoří jednu funkci zařízení, přičemž každé rozhraní může mít několik alternativních nastavení. V jedné konfiguraci může být více rozhraní, a proto fyzicky jedno USB zařízení se může z hlediska hostitele jevit jako několik zařízení s různými funkcemi.

2.3.1 Deskriptor zařízení

Deskriptor zařízení (*device descriptor*) má velikost 18 bajtů a v USB zařízení je pouze jeden. Struktura deskriptoru zařízení je v tab. 3. Obsahuje číslo USB specifikace, se kterou jsou deskriptory kompatibilní, třídu zařízení, maximální velikost paketu při přenosu bránou 0, čísla *ID* pro identifikaci výrobku, indexy deskriptorů textových řetězců sériového čísla, výrobce apod. a počet dostupných konfigurací USB zařízení. Pokud je index textového řetězce 0, znamená to že jej zařízení neobsahuje. Deskriptor zařízení je první deskriptor, který se hostitel snaží ze zařízení přečíst. Je to z důvodu potřeby parametru *bMaxPacketSize0*, protože až na speciální případy musí mít datové pakety maximální velikost brány. Protože až do přečtení tohoto deskriptoru hostitel neví jak velký paket může na bránu 0 odeslat, specifikace USB definuje, že minimální velikost brány 0 je 8 bajtů. Hostitel tedy nejprve pošle požadavek pro přečtení osmi bajtů deskriptoru zařízení, čímž zjistí *bMaxPacketSize0*. Teprve potom přečte všech 18 bajtů deskriptoru zařízení.

Tab. 3 Struktura deskriptoru zařízení

Jméno pole	Ofset (B)	Délka (B)	Popis
bLength	0	1	Velikost deskriptoru.
bDescriptorType	1	1	Kód deskriptoru zařízení (0x01).
bcdUSB	2	2	BCD kódované číslo specifikace se kterou jsou deskriptory kompatibilní.
bDeviceClass	4	1	Identifikátor třídy zařízení. Pokud je roven nule, každé rozhranní v konfiguraci má nastavenou třídu nezávisle na ostatních rozhranních. Pokud je roven 255, zařízení nepatří do žádné definované třídy.
bDeviceSubCalss	5	1	Identifikátor podskupiny třídy zažízení.
bDeviceProtocol	6	1	Identifikátor protokolu závislý na podskupině a třídě zařízení.
bMaxPacketSize0	7	1	Maximální velikost dat, při jednom přenosu bránou 0.
idVendor	8	2	Identifikátor výrobce, který přiděluje USB-IF.
idProduct	10	2	Identifikátor výrobku, který si volí výrobce.
bcdDevice	12	2	BCD kódovaná verze výrobku.
iManufacture	14	1	Index textového řetězce popisující výrobce nebo nula.
iProduct	15	1	Index textového řetězce popisující výrobek nebo nula.
iSerialNumber	16	1	Index textového řetězce sériového čísla nebo nula.
bNumConfiguration	17	1	Počet možných konfigurací.

2.3.2 Deskriptor konfigurace

Deskriptor konfigurace (*configuration descriptor*) má základní velikost 9 bajtů a jejich počet je roven počtu dostupných konfigurací. Struktura deskriptoru konfigurace je v tab. 4. První deskriptor konfigurace má vždy index 0 a postupně se inkrementuje s dalšími konfiguracemi.

Deskriptor konfigurace obsahuje počet rozhraní v této konfiguraci, identifikátor konfigurace pro její aktivaci řídícím požadavkem, vlastnosti zařízení při použití této konfigurace, index deskriptoru textového řetězce pro její popis a maximální odebíraný proud zařízením. Důležitý parametr je *wTotalLength*, který udává celkovou velikost deskriptoru konfigurace. Do deskriptoru konfigurace patří všechny deskriptory rozhraní a deskriptory

bran, které k daným rozhraním konfigurace patří (nebo i další typy deskriptorů pokud je konfigurace obsahuje jako např. *HID* deskriptor).

Tab. 4 Struktura deskriptoru konfigurace

Jméno pole	Ofset (B)	Délka (B)	Popis
bLength	0	1	Velikost deskriptoru.
bDescriptorType	1	1	Kód deskriptoru zařízení (0x02).
wTotalLength	2	2	Celková délka všech deskriptorů posílaných spolu s deskriptorem konfigurace.
bNumInterfaces	4	1	Počet rozhraní v konfiguraci.
bConfigurationValue	5	1	Identifikátor konfigurace. Tato hodnota je použita pro výběr této konfiguracepožadavkem SetConfiguration.
iConfiguration	6	1	Index textového řetězce popisující tuto konfiguraci nebo nula, pokud není konfigurace popsána.
bmAttributes	7	1	Vlastnosti zařízení v konfiguraci (vlastní napájení, možnost vzbuzení hostitele, viz. spec.).
MaxPower	8	1	Maximální proud odebíraný zařízením ze sběrnice při použití této konfigurace. Jednotkou jsou 2mA.

Nejprve si hostitel ze zařízení přečte prvních 9 bajtů deskriptoru konfigurace kdy zařízení vrací strukturu uvedenou v tab. 4. Poté načte celý deskriptor konfigurace, kdy hostitel zadá požadovanou délku dat *wTotalLength*. Tím přečte vše, co k dané konfiguraci patří, kromě deskriptorů textových řetězců nebo jiných speciálních deskriptorů. Načtené deskriptory jsou pak logicky seřazeny. Prvních 9 bajtů je vlastní deskriptor konfigurace, následuje deskriptor prvního rozhraní, za ním jsou deskriptory bran patřících k tomuto rozhraní, za nimi je deskriptor dalšího rozhranní atd.

2.3.3 Deskriptor rozhraní

Deskriptor rozhraní (*interface descriptor*) má velikost 9 bajtů a v jedné konfiguraci jich může být více. Struktura deskriptoru rozhraní je v tab. 5.

Tab. 5 Struktura deskriptoru rozhraní

Jméno pole	Ofset (B)	Délka (B)	Popis
bLength	0	1	Velikost deskriptoru.
bDescriptorType	1	1	Kód deskriptoru zařízení (0x04).
bInterfaceNumber	2	1	Identifikátor rozhraní. První rozhraní v konfiguraci má identifikátor 0.
bAlternateSetting	3	1	Identifikátor alternativního nastavení rozhraní.
bNumEndpoints	4	1	Počet bran používaný rozhraním bez řídící brány.
bInterfaceClass	5	1	Identifikátor třídy zařízení. Hodnota 0xFF znamená, že rozhraní nepatří k žádné definované třídě.
bInterfaceSubClass	6	1	Identifikátor podskupiny třídy zařízení.
bInterfaceProtokol	7	1	Identifikátor protokolu závislý na třídě a podskupině třídy zařízení.
iInterface	8	1	Index textového řetězce popisující toto zozhraní nebo nula pokud není rozhraní popsáno.

Tento deskriptor je důležitý hlavně kvůli identifikaci již existujících tříd zařízení, jako například *Mass Storage, HID* apod. Popis parametrů deskriptoru rozhraní pro USB flash disk bude uveden později.

2.3.4 Deskriptor brány

Deskriptor brány (*endpoint descriptor*) má velikost 7 bajtů a v jednom rozhraní jich může být více. Struktura deskriptoru rozhraní je v tab. 6. Obsahuje adresu brány, která zároveň určuje zda se jedná o bránu typu *IN* nebo *OUT*. Dále obsahuje typ přenosu pro tuto bránu a maximální velikost dat v jednom paketu při přenosu touto bránou. Pro zařízení *full-speed* je maximální možná velikost brány pro hromadný přenos 64 bajtů. Parametr *bInterval* určuje periodu dotazování na časovaný typ brány. Kromě brány 0 je každá jiná brána buď typu *OUT* nebo *IN*, nemůže být pro přenos dat obousměrná.

Tab. 6 Struktura deskriptoru brány

Jméno pole	Ofset (B)	Délka (B)	Popis
bLength	0	1	Velikost deskriptoru.
bDescriptorType	1	1	Kód deskriptoru zařízení (0x05).
bEndpointAddress	2	1	Adresa brány v rámci zařízení. D0-D3 je číslo brány. D7 určuje směr toku dat: 0 = od hostitele do zařízení (OUT), 1 = od zařízení k hostiteli (IN).
bmAttributes	3	1	Vlastnosti brány. D1, D0 určuje typ brány: 0 = řídící (control), 1 = izochronní, 2 = pro objemná data (bulk), 3 = časovaná (interrupt).
wMaxPacketSize	4	2	Maximální velikost dat v jednom přenosu.
bInterval	6	1	Čas v milisekundách, který určuje interval dotazování na časovaný typ brány.

2.3.5 Deskriptor podporovaných jazyků

Deskriptor podporovaných jazyků (*language descriptor*) má velikost minimálně 4 bajty a v zařízení je pouze jeden. Struktura deskriptoru rozhraní je v tab. 7. Deskriptor obsahuje identifikátory jazyků, ve kterých má zařízení dostupné deskriptory textových řetězců. Tyto kódy jsou uvedeny v [25]. Deskriptor podporovaných jazyků má stejný kód jako deskriptory textových řetězců (0x03), index má však 0, zatímco indexy deskriptorů textových řetězců jsou 1 a výše (při žádosti na přečtení daného deskriptoru se zadává jeho kód a index). USB zařízení na evropském trhu používají angličtinu.

Tab. 7 Struktura deskriptoru podporovaných jazyků

Jméno pole	Ofset (B)	Délka (B)	Popis
bLength	0	1	Velikost deskriptoru.
bDescriptorType	1	1	Kód deskriptoru zařízení (0x03).
wLANGID[0]	2	2	Identifikátor 1. podporovaného jayzka.
…	…	…	
wLANGID[x]	N	2	Identifikátor x. podporovaného jayzka.

2.3.6 Deskriptor textového řetězce

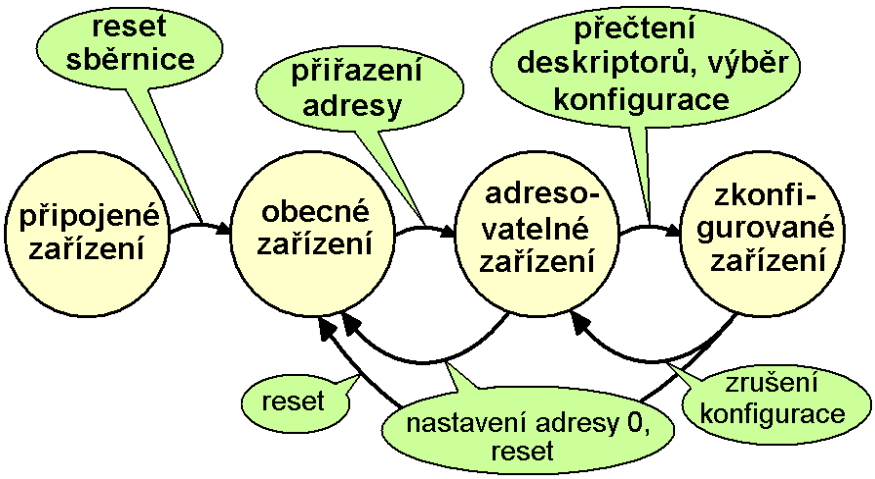
Deskriptor textového řetězce (*string descriptor*) má velikost závislou na počtu znaků v textovém řetězci. V zařízení jich může být více. Struktura deskriptoru rozhraní je v tab. 8. V řetězci se používá kódování znaků *UNICODE* délky 16 bitů.

Tab. 8 Struktura deskriptoru textového řetězce

Jméno pole	Ofset (B)	Délka (B)	Popis
bLength	0	1	Velikost deskriptoru.
bDescriptorType	1	1	Kód deskriptoru zařízení (0x03).
bString	2	2*N	Textový řetězec dlouhý N znaků kódovaný UNICODE.

2.4 Enumerace zařízení

Enumerace je proces rozpoznání zařízení (typ, účel, vlastnosti) a zvolení jedné z jeho možných konfigurací. Během celé doby, co je zařízení na sběrnici připojeno, se nachází buď ve stavu připojeného zařízení, obecného zařízení, adresovatelného zařízení nebo zkonfigurovaného zařízení, viz obr. 13.



Obr. 13 Stavy zařízení připojené na sběrnici USB

Po připojení zařízení na sběrnici hostitel vyšle signál *USB reset* – jednoznačná nula na dobu minimálně 10ms a zařízení přejde do stavu obecného zařízení. Ve stavu obecného zařízení má výchozí adresu 0 a také je již možná komunikace s hostitelem. K tomuto účelu slouží dříve popsaný řídící přenos a základní brána 0. Další řešení enumerace závisí na konkrétním algoritmu hostitele. Aby zařízení mohlo přejít do zkonfigurovaného stavu, musí mu hostitel nejdříve přiřadit novou adresu v rámci sběrnice. Také vyčte dostupné deskriptory a na základně získaných informací zvolí požadovanou konfiguraci. Nastavením jedné z možných konfigurací zařízení přechází do zkonfigurovaného stavu. V tomto stavu jsou aktivní všechna rozhraní zvolené konfigurace. Do adresovatelného stavu může hostitel zařízení vrátit nastavením konfigurace s indexem 0. Do stavu obecného zařízení lze přejít aktivací signálu *USB reset*.

2.5 USB řídící požadavky

Aby bylo možné připojené zařízení na USB sběrnici identifikovat a rozpoznat, musí všechna USB zařízení podporovat stejný základní komunikační protokol. Jak již bylo řečeno dříve, k tomuto účelu slouží brána 0 a řídící typ přenosu. V řídícím přenosu se k tomuto účelu v první fázi předává zařízení tzv. řídící požadavek. Délka požadavku je 8 bajtů, aby mohl být vyslán hostitelem v rámci jednoho paketu, protože minimální velikost brány 0 podle specifikace je 8 bajtů. Za pověřovacím paketem *SETUP* následuje datový paket *DATA0* nesoucí standardní řídící požadavek. Na obr. 7 je tento datový paket součástí pole *SETUP*, kde je naznačen sudým datovým paketem *DATA0*. Kromě standardních požadavků může vlastní řídící požadavky definovat i třída zařízení nebo výrobek. V tom případě bude v požadavku uvedeno, že je požadavek specifický pro třídu nebo pro výrobek.

2.5.1 Struktura řídících požadavků

Požadavek určuje komu je adresován a to buď pro zařízení, rozhraní nebo bránu. Dále určuje druh požadavku, který může být standardní, specifický pro třídu nebo specifický pro výrobek. Důležitou součástí je kód, který určuje co má požadavek provést za operaci a také množství bajtů a směr, ve kterém budou mezi hostitelem a zařízením přenášena data v datové fázi řídícího přenosu. Zbylé dva parametry jsou doplňkové a jejich význam může být u každého požadavku jiný.

Tab. 9 Struktura požadavku

Jméno pole	Ofset (B)	Délka (B)	Popis
bmRequestType	0	1	Vlastnosti požadavku: D7: Směr pěnosu dat v rámci požadavku 0 - od hostitele k zařízení 1 - ze zařízení k hostiteli D6, D5: Typ požadavku 0 - standardní 1 - specifický pro třídu 2 - specifický pro výrobek D4 - D0: Příjemce požadavku 0 - zařízení 1 - rozhraní 2 - brána 3 - ostatní
bRequest	1	1	Kód požadavku
wValue	2	2	1. parametr závislý na požadavku
wIndex	4	2	2. parametr požadavku, většinou index nebo ofset
wLength	6	2	Délka dat přenášených s požadavkem do/ze zařízení

V tab. 9 je znázorněn význam jednotlivých bajtů požadavku. Požadavek se přenáší od bajtu s ofsetem 0. Poslední tři parametry jsou dvoubajtové, kde se jako první přenáší méně významný bajt. V tab. 10 je uveden seznam standardních požadavků s jejich kódy v dekadické podobě a stručný popis funkce. Některé požadavky plní rozdílnou funkci podle toho, zda jsou adresovány pro zařízení, rozhraní nebo bránu.

Tab. 10 Přehled standardních požadavků

Požadavek	Kód	Popis
GetStatus	0	Zjištění stavu zařízení
ClearFeature	1	Vypnutí definované funkce zařízení
SetFeature	3	Zapnutí definované funkce zařízení
SetAddress	5	Nastavení adresy zařízení v rámci sběrnice
GetDescriptor	6	Přečtení deskriptoru zařízení,konfigurace nebo textu
SetDescriptor	7	Změna nebo přidání deskriptoru
GetConfiguration	8	Zjištění vybrané aktuální konfigurace
SetConfiguration	9	Nastavení aktuální konfigurace
GetIntrface	10	Zjištění výběru alternativního nastavení rozhraní
SetIntrface	11	Výběr alternativního nastavení rozhraní
SynchFrame	12	Určení synchronizačního rámce

V tab.11 je uvedena konfigurace standardních řídících požadavků. Každé zařízení ne vždy podporuje všechny USB řídící požadavky. Některé specifické požadavky však musí podporovat všechny USB zařízení (*SetAddress*, *SetConfiguration*, *GetDescriptor*, *SetInterface*). Dále budou popsány jen nejdůležitější požadavky. Celkový detailní popis lze nalézt v [1].

Tab. 11 Konfigurace standardních požadavků

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0000 0000b	CLEAR_FEATURE (0x01)	Feature Selector	0	0	Žádná
0000 0001b			Rozhraní		
0000 0010b			Brána		
1000 0000b	GET_CONFIGURATION (0x08)	0	0	1	Číslo konfigurace
1000 0000b	GET_DESCRIPTOR (0x06)	Kód deskriptoru & Index	0 nebo language ID	Délka deskriptoru	Deskriptor
1000 0001b	GET_INTERFACE (0x0A)	0	Rozhraní	1	Číslo alt. Rozhraní
1000 0000b	GET_STATUS (0x00)	0	0	2	Status zařízení
1000 0001b			Rozhraní		Status rozhraní
1000 0010b			Brána		Status brány
0000 0000b	SET_ADDRESS (0x05)	Adresa zařízení	0	0	Žádná
0000 0000b	SET_CONFIGURATION (0x09)	Číslo konfigurace	0	0	Žádná
0000 0000b	SET_DESCRIPTOR (0x07)	Kód deskriptoru & Index	0 nebo language ID	Délka deskriptoru	Deskriptor
0000 0000b	SET_FEATURE (0x03)	Feature Selector	0	0	Žádná
0000 0001b			Rozhraní		
0000 0010b			Brána		
0000 0001b	SET_INTERFACE (0x0B)	Alternativní rozhraní	Rozhraní	0	Žádná
1000 0010b	SYNCH_FRAME (0x0C)	0	Brána	2	Rámec

2.5.2 Požadavek SET_ADDRESS

Po resetu zařízení na USB sběrnici má zařízení nastavenou adresu 0. Požadavek *SET_ADDRESS* slouží k přidělení jiné adresy v rozsahu 1-127, aby se zařízení dostalo do adresovatelného stavu. Tento požadavek je vhodné provést jako první. I když je na sběrnici pouze jedno zařízení, musí být požadavek na změnu adresy použit, protože pokud není zařízení v adresovatelném stavu, některé další důležité požadavky nefungují. Požadavek má kód 5 a v parametru *wValue* je předána nová adresa zařízení. Zařízení změní svou adresu až po úspěšném proběhnutí stavové fáze řídícího přenosu. Pokud je zadaná adresa větší než 0 zařízení přejde do adresovatelného stavu, jinak zůstane ve stavu obecného zařízení s adresou 0. Pokud je zařízení v adresovatelném stavu a v požadavku je zadána adresa 0, zařízení přejde do stavu obecného zařízení. Pokud je zařízení ve zkonfigurovaném stavu (byla zvolena konfigurace požadavkem *SET_CONFIGURATION*), chování zařízení na požadavek *SET_ADDRESS* není specifikováno.

2.5.3 Požadavek GET_DESCRIPTOR

Požadavek *GET_DESCRIPTOR* slouží k přečtení libovolného deskriptoru z USB zařízení. Požadavek má kód 6. V parametru *wValue* je předán kód požadovaného deskriptoru a index deskriptoru. V *LSB* je uveden index a v *MSB* je uveden kód. Např. pro deskriptor zařízení *wValue* = 0x0100, deskriptor textového řetězce s indexem 1 *wValue* = 0x0301, deskriptor první konfigurace *wValue* = 0x0200. V parametru *wIndex* lze uvést kód jazyka, ve kterém chceme přečíst deskriptor textového řetězce. Podporované jazyky jsou uvedené v deskriptoru podporovaných jazyků. Obvykle postačí zadat do parametru *wIndex* hodnotu 0. V parametru *wLength* je třeba uvést množství dat, které chceme přijmout. Nejlépe je zadat přímo velikost deskriptoru, např. pro deskriptor zařízení *wLength* = 18. Pokud je zadáno menší množství dat než velikost deskriptoru, zařízení vrátí odpovídající počet od začátku deskriptoru, pokud je zadáno vetší množství než je velikost deskriptoru, zařízení vrátí pouze počet bajtů odpovídající velikosti deskriptoru. V tomto případě hostitel rozpozná konec dat v přenosu podle kratšího paketu než je velikost řídící brány. Požadavek *GET_DESCRIPTOR* je platný požadavek pro jakýkoli aktuální stav zařízení.

2.5.4 Požadavek SET_CONFIGURATION

Požadavek *SET_CONFIGURATION* slouží k aktivaci (výběru) zvolené konfigurace zařízení. I když většina USB zařízení má pouze jedinou možnou konfiguraci, přesto je nutné ji požadavkem *SET_CONFIGURATION* zvolit, aby byla aktivována. Pokud má USB zařízení více možných konfigurací, může být v jeden okamžik aktivována pouze jedna. V parametru *wValue* je předán identifikátor konfigurace získaný z deskriptoru dané konfigurace (parametr *bConfigurationValue*). Pokud je zařízení ve stavu obecného zařízení, reakce na tento požadavek není specifikována. Pokud je zařízení v adresovatelném stavu a identifikátor konfigurace je platný, zařízení přejde do zkonfigurovaného stavu. Pokud je zařízení ve zkonfigurovaném stavu a identifikátor konfigurace je platný, daná konfigurace je aktivována a zařízení zůstane ve zkonfigurovaném stavu. Pokud je zadán identifikátor konfigurace 0, zařízení se vrací do adresovatelného stavu.

2.5.5 Požadavek SET_INTERFACE

Jednotlivá rozhraní v konfiguraci jsou číslována a každé představuje jiné zařízení. Každé rozhraní má pevně přiřazené jednotlivé brány, přičemž všechna rozhraní v jedné konfiguraci jsou aktivována. Může však nastat situace, kdy je pro jedno zařízení (rozhraní) vhodné měnit parametry rozhraní, např. parametry brány. Toho se často využívá např. pro změnu velikosti paketu pro izochronní přenos v rámci jediného rozhraní (např. USB webkamery). Standard tento problém řeší zavedením alternativních rozhraní k danému rozhraní. Zatímco jednotlivá hlavní rozhraní, které mají parametr *bAlternateSetting* roven 0, musejí mít přiděleny různé brány, alternativní rozhraní může mít přidělené stejné brány jako dané hlavní rozhraní, protože je v jeden okamžik aktivní pouze jedno z nich. Např. rozhraní 0 může mít další 3 alternativní rozhraní které používají stejné brány, přičemž všechna jsou rozhraní 0, hlavní má *bAlternateSetting* = 0, první alternativní má 1, druhé má 2 a třetí 3. Všechna alternativní rozhraní mají parametr *bInterfaceNumber* roven svému hlavnímu rozhraní, ale parametr *bAlternateSetting* má různý od nuly. Pokud existují alternativní rozhraní, požadavek *SET_INTERFACE* slouží k jejich výběru. Požadavek *SET_INTERFACE* má kód 11. V parametru *wValue* je předáno číslo nastavovaného alternativního rozhraní (lze zadat i 0), v parametru *wIndex* je předáno číslo rozhraní, ke kterému volíme alternativu. Tento požadavek je platný pouze pro zkonfigurovaný stav zařízení.

2.5.6 Požadavek GET_STATUS

Tento požadavek je závislý na tom, zda je adresováno zařízení, rozhraní nebo brána (parametr *bmRequestType* požadavku). Zde jako nejdůležitější uvedu pouze případ požadavku na bránu.

Zařízení může na základě nejrůznějších situací pozastavit libovolnou bránu do stavu *HALT*, tzn. že nepřijímá ani neodesílá data, jen na jakýkoli příchozí paket vrací paket *STALL*. Výjimkou je brána 0, která i ve stavu *HALT* dokáže přijmout paket *SETUP* s požadavkem, kdy stav *HALT* automaticky ruší. Pokud je potřeba zjistit, zda je brána pozastavena, lze použít požadavek *GET_STATUS*. Parametr *wValue* je pak 0, v parametru *wIndex* je uvedeno číslo dotazované brány, přičemž je nutné respektovat, že brány typu *OUT* mají adresy 0x00 - 0x0F a brány *IN* 0x80 – 0x8F. Parametr *wLength* je nastaven na hodnotu 2. Ve stavu obecného zařízení není reakce na tento požadavek specifikována. Pokud je zařízení v adresovatelném stavu a číslo zadané brány je jiné než 0, zařízení vrací chybu pozastavením brány 0. Ve zkonfigurovaném stavu lze adresovat brány které jsou v dané konfiguraci dostupné, při zadání jiné brány vrací opět chybu. Zařízení na požadavek vrátí 2 bajty ve formátu uvedeného v tab.12.

Tab. 12 Data požadavku *GET_STATUS* na bránu

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Rezervováno (nastaveno na 0)															HALT

Pokud je brána pozastavena, parametr *HALT* je nastaven na 1. Všechny ostatní bity jsou nastaveny na 0.

2.5.7 Požadavek CLEAR_FEATURE

Tento požadavek je opět závislý na tom, zda je adresováno zařízení, rozhraní nebo brána (parametr *bmRequestType* požadavku). Zde jako nejdůležitější uvedu pouze případ požadavku na bránu.

V případě, že zařízení pozastaví bránu do stavu *HALT*, hostitel musí mít možnost tento stav brány zrušit. K tomu slouží požadavek *CLEAR_FEATURE*.

Stav *HALT* slouží k tomu, aby zařízení hostitele upozornilo na vnitřní chybu nebo chybu transportního protokolu vyšší vrstvy. Zařízení nemůže samo od sebe hostiteli odeslat data. Chybu nebo nemožnost odpovídající reakce na hostitelovy příkazy může ohlásit pozastavením brány. Hostitel pak podnikne patřičné kroky k nápravě.

Parametry požadavku musí být nastaveny následovně. Parametrem *wIndex* se jako v předchozím případě zadává číslo brány. Parametr *wLength* je nastaven na 0. V parametru *wValue* je zadán tzv. *feature selector*. Pro případ adresování brány požadavkem *CLEAR_FEATURE* to je selektor *ENDPOINT_HALT*, který má hodnotu 0. Ve stavu obecného zařízení není reakce na tento požadavek specifikována. Pokud je zařízení v adresovatelném stavu a číslo zadané brány je jiné než 0, zařízení vrací chybu pozastavením brány 0. Ve zkonfigurovaném stavu lze adresovat brány které jsou v dané konfiguraci dostupné, při zadání jiné brány vrací opět chybu.

3 USB Flash disk

Záznamové zařízení typu USB flash disk (dále jen flash disk), v anglickém jazyce označované jako zařízení *Flash Drive*, patří do skupiny zařízení označované jako *USB Mass Storage Devices*, kterou specifikuje třída zařízení *USB MSC* (*Universal Serial Bus Mass Storage Class*), přehled specifikace viz [14]. Skupina *USB MSD* sdružuje zařízení pro ukládání dat na různá paměťová média komunikující s hostitelským zařízením přes sběrnici USB. *USB Mass Storage* zařízení mohou používat různé druhy paměťových médií jako pevné disky, pružné disky (floppy mechaniky), kompaktní disky (CD nebo DVD mechaniky) a různé druhy flash paměťových karet nebo obvodů (*MultiMediaCard - MMC*, *SD Memory Card - SD*, *CompactFlash - CF*). Flash disk používá jako paměťové médium paměť typu flash, charakterizované vždy jako nevyměnitelné médium.

Třída *USB MSC* definuje transportní protokoly pro přenos dat mezi hostitelem a záznamovým zařízením, specifikuje bootovatelné zařízení a také specifikuje možné příkazové sady pro jednotlivé typy zařízení.

3.1 Flash disk vs. třída USB MSC

Prakticky všechny flash disky používají dnes stejné komunikační protokoly a příkazové sady. Ze třídy *USB MSC* to je transportní protokol *Bulk-Only Transport* viz kap. 4 a dále používají podle této třídy transparentní příkazovou sadu *SCSI* viz kap. 5. Dále budou

uvedeny některé parametry USB deskriptorů specifické pro flash disk. Ostatní obecné parametry jsou dostatečně popsány v kap. 2.3 nebo v [1].

3.1.1 Deskriptor zařízení

V deskriptoru zařízení jsou tři parametry *bDeviceClass*, *bDeviceSubClass* a *bDeviceProtocol* umožňující identifikovat třídu zařízení. I když má dané USB zařízení pouze jedno rozhraní (funguje jen jako USB flash disk), tyto parametry bývají nastavené na hodnoty 0 a jsou uvedeny až v deskriptoru daného rozhraní, kde musejí být uvedeny vždy.

V deskriptoru zařízení je také index textového řetězce sériového čísla výrobku *iSerialNumber*. Pokud je tento deskriptor textového řetězec v zařízení přítomný (*iSerialNumber* je větší než 0), splňuje dle třídy také určitá pravidla. Textový řetězec sériového čísla má minimálně 12 znaků a můžou být použity pouze znaky 0x30 – 0x39 („0“ – „9“) a 0x41 – 0x046 („A“ – „F“). Znaky jsou kódovány v 16-ti bitovém *UNICODE*.

3.1.2 Deskriptor konfigurace

V parametru *bNumInterfaces* je uveden počet rozhraní v konfiguraci. V dané konfiguraci (zařízení jich může mít více) musí být alespoň jedno rozhraní flash disku, tedy hodnota *bNumInterfaces* musí být minimálně 1.

3.1.3 Deskriptor rozhraní

Deskriptor rozhraní flash disku musí být specificky vyplněn. Transportní protokol *Bulk-Only* používá pro komunikaci (kromě brány 0) dvě brány, tedy rozhraní flash disku musí obsahovat minimálně tyto dvě brány. Parametr *bNumEndpoints* musí mít tedy hodnotu minimálně **2**.

Parametr *bInterfaceClass* je nastaven na hodnotu **0x08**, což je *Mass Storage Class*. Parametr *bInterfaceSubClass* je nastaven na hodnotu **0x06**, což indikuje příkazovou sadu *SCSI*. A jako poslední důležitý parametr *bInterfaceProtocol* je nastaven na hodnotu **0x50**, což je *Bulk-Only* transportní protokol.

3.1.4 Deskriptory bran

Bulk-Only transportní protokol požaduje pro komunikaci řídící bránu 0 a další dvě brány, jednu typu *OUT* a jednu typu *IN*, obě s hromadným typem přenosu. V rozhraní jsou tedy dva deskriptory bran.

Parametr *bmAttributes* je nastaven na hodnotu **0x02**, tedy hromadný typ přenosu. Maximální velikost paketu těmito bránami, parametr *wMaxPacketSize*, může být nastaven na hodnoty **8**, **16**, **32** nebo **64** bajtů (při komunikaci *full-speed*). Poslední důležitý parametr je *bEndpointAddress*. Adresy bran můžou být libovolné, záleží na výrobci a na počtu bran, které obsahuje USB transceiver ve flash disku. Adresy brány *OUT* a *IN* jsou vždy rozdílné. Bit 7 parametru *bEndpointAddress* určuje zda se jedná o bránu *IN* nebo *OUT* viz kap. 2.3.4.

3.2 Architektura a paměťové pole

Flash disk podporuje přenosovou rychlost odpovídající zařízením typu *Full-speed* nebo *High-speed*, tedy 12Mb/s nebo 480Mb/s nebo oboje. Většina *High-speed* zařízení podporuje i přenosovou rychlost odpovídající zařízením *Full-speed*.

Flash disk se skládá z USB transceiveru (USB zařízení typu periferie - *Device*), mikrokontroléru, který obsahuje řídící firmware a paměti typu flash. Pro komunikaci používá řídící bránu 0 s řídícím typem přenosu a dvě další brány (*OUT* a *IN*) s hromadným typem přenosu. Používá transportní protokol *Bulk-Only* a příkazovou sadu *SCSI*.

Na jedné paměti flash disku může být umístěno více logických jednotek (jednotné paměťové pole se souborovým systémem). Toto rozdělení musí provést firmware daného disku, tedy rozhoduje o něm výrobce flash disku. Jeden flash disk může mít výrobcem vytvořených až 16 logických jednotek, tedy může mít naformátovaných až 16 disků. Většina flash disků má však pouze jedinou logickou jednotku, pak např. při použití operačního systému Windows XP může mít naformátován pouze jediný disk. V transportním protokolu *Bulk-Only* hostitel přímo adresuje vybranou logickou jednotku flash disku.

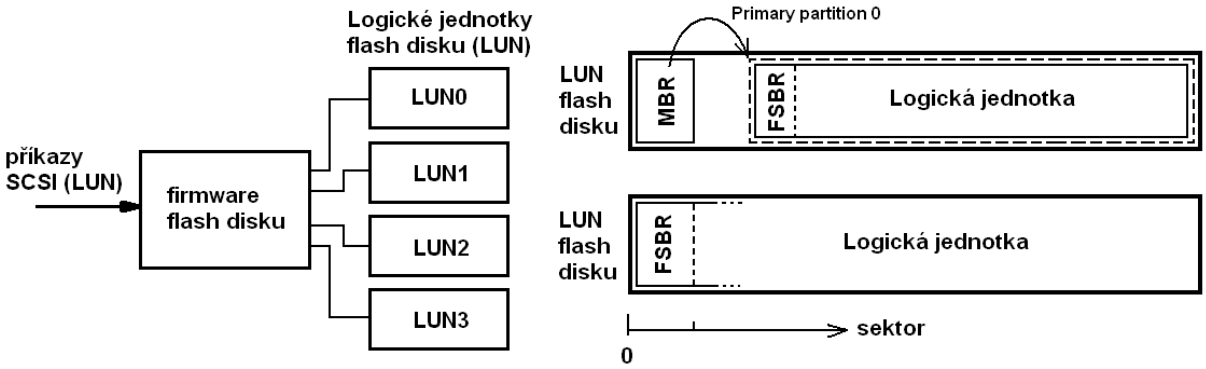
Flash disk používá adresování *LBA* (*Logical Block Addressing*). Logická jednotka je rozdělena na bloky dat stejné velikosti – sektory. Sektor má většinou velikost 512 bajtů, ale může mít i větší (1024, 2048, 4096 atd.). Pomocí příkazů *SCSI* aplikovaných v transportním protokolu *Bulk-Only* lze sektory z flash disku číst a stejně tak je zapisovat. S menší jednotkou než sektor nelze na logické jednotce flash disku manipulovat.

Data jsou na logické jednotce ukládány v tzv. souborovém systému. Dnes nejznámější a nejrozšířenější souborové systémy používající operační systém Windows jsou *FAT16*, *FAT32* a *NTFS*. Použití standardních souborových systémů na flash disku není podmínkou. O významu dat v jednotlivých sektorech a o jejich organizaci se stará pouze hostitel. Pro firmware flash disku je logická jednotka jen soubor určitého počtu sektorů, bez znalosti jejich obsahu nebo významu. Flash disk lze tedy používat i jako obyčejné paměťové pole.

Většina flash disků je už od výrobce naformátována na souborový systém *FAT16* (uživatel však může snadno souborový systém změnit). Jen disky o kapacitách větších než 4GB nemohou používat souborový systém *FAT16* pokud nemají sektor větší než 512 bajtů. Pak je nutné použít např. souborový systém *FAT32* nebo *NTFS*.

Organizace dat souborového systému v polovodičových pamětích se odvíjí od pevných disků, pro které byly původně souborové systémy navrhovány. Ve většině paměťových zařízeních je první sektor na médiu (sektor 0) *Master Boot Record* sektor (*MBR*). *MBR* sektor na paměťových polích slouží mimo jiné k případnému dělení disků na více oddílů (*DOS*). *MBR* obsahuje datovou strukturu obsahující tabulku, která může definovat následující čtyři sekce (*partition*). Každá taková sekce se jeví jako samostatný svazek, ve kterém může být definována logická jednotka. Na začátku *partition* je pak *File System Boot Record* (*FSBR*, např. *FAT16 Boot Record*) sektor, který je rovněž začátkem logické jednotky a definuje základní parametry logické jednotky. Na obr. 14 je znázorněno formátování logické jednotky flash disku s *MBR* sektorem a bez něj. Levá část obr. 14 znázorňuje uspořádání logických jednotek firmwarem flash disku (*LUN*). Každý *SCSI* příkaz od hostitele je směřován jedné logické jednotce (při transportování *SCSI* příkazu je nutné zadat číslo logické jednotky flash

disku, pro kterou je příkaz adresován). Logické jednotky flash disku jsou indexovány od nuly a může jich být až 16.



Obr. 14 Způsoby formátování logických jednotek flash disku

Protože třída *USB MSC* definuje, že rozdělení na logické jednotky provádí firmware zařízení, *MBR* sektor není na flash disku potřeba, zbytečně pak snižuje užitečnou kapacitu disku jak jde vidět z obr.14. Přesto hodně výrobců takto flash disky formátuje. Operační systém Windows XP podporuje flash disky s oběma variantami formátování. Pokud tedy logická jednotka flash disku má první sektor *MBR*, v *MBR* sektoru je vyplněn pouze odkaz na *Primary Partition 0* viz kap. 6.1, která definuje začátek a velikost logické jednotky, která je pak v paměťovém poli o něco dále. Ostatní tři odkazy na další *partiion* v tabulce *MBR* jsou nevyplněny.

Pokud logická jednotka flash disku není naformátována s *MBR* sektorem, první sektor je přímo *File System Boot Record* sektor a tedy také začátek dané logické jednotky disku.

4 Bulk-Only transportní protokol

Bulk-Only transportní protokol používá pro komunikaci mezi hostitelem a zařízením jednu bránu typu *IN* a jednu typu *OUT*. Pro řešení určitých situací dále definuje dva řídící požadavky (pro řídící bránu 0) specifické pro třídu a to *Bulk-Only Mass Storage Reset* a *Get Max LUN*.

Úspěšná komunikace transportním protokolem *Bulk-Only* má dvě nebo tři fáze. Přenos příkazu, datová fáze (u některých příkazů je vynechána) a stavová fáze (přenos statusu přenosu hostiteli). V první fázi hostitel pošle do zařízení příkazový blok (např. s příkazem *SCSI*) ve struktuře nazývané *Command Block Wrapper* (*CBW*). V datové fázi posílá hostitel nebo zařízení data. Ve stavové fázi posílá zařízení hostiteli informace ve struktuře nazývané *Command Status Wrapper* (*CSW*). *Mass Storage* ani *SCSI* specifikace nedefinují jak dlouho by měl hostitel čekat na vrácení dat ze zařízení.

4.1 Požadavek Bulk-Only Mass Storage Reset

Požadavek specifický pro třídu *Mass Storage*, užívaný k resetování *Mass Storage* zařízení a jeho přidružených rozhraní. Tento „*class-specific*“ požadavek připraví zařízení k příjmu dalšího *CBW* od hostitele. Hostitel posílá požadavek na řídící bránu 0 zařízení. Zařízení po provedení požadavku uchová hodnotu *data-toggle* bitů (bity indikující, zda příští datový paket na příslušných branách bude lichý nebo sudý) a případný stav *HALT* na branách hromadného přenosu. Dokud nebude *Bulk-Only Mass Storage Reset* dokončen bude zařízení vracet *NAK* ve stavové fázi požadavku. V tab.13 je uvedena konfigurace jednotlivých parametrů požadavku.

- *bmRequestType*: Class, Interface, host to device
- *bRequest* nastavit na 255 (FFh)
- *wValue* nastavit na 0
- *wIndex* nastavit na číslo daného rozhraní
- *wLength* nastavit na 0

Tab. 13 Konfigurace požadavku *Bulk-Only Mass Storage Reset*

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001b	11111111b	0000h	Interface	0000h	žádná

4.2 Požadavek Get Max LUN

Stejně jako požadavek *Bulk-Only Mass Storage reset* je *Get Max LUN* požadavek specifický pro třídu *Mass Storage*.

Zařízení *Mass Storage* může realizovat několik logických jednotek, které sdílí společné prvky a vlastnosti zařízení. Hostitel používá pole *bCBWLUN* ve struktuře *CBW* (viz kap. 4.3) k určení, pro kterou logickou jednotku zařízení je *CBW* určen. Požadavek *Get Max LUN* slouží k určení kolik logických jednotek zařízení (flash disk) podporuje. Logické jednotky v zařízení musí být číslovány postupně od *LUN 0* až do *LUN 15* (00h - 0Fh). V tab. 14 je uvedena konfigurace jednotlivých parametrů požadavku.

- *bmRequestType*: Class, Interface, device to host
- *bRequest* nastavit na 254 (FEh)
- *wValue* nastavit na 0
- *wIndex* nastavit na číslo daného rozhraní
- *wLength* nastavit na 1

Tab. 14 Konfigurace požadavku *Get Max LUN*

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001b	11111110b	0000h	Interface	0001h	1 bajt

Zařízení odpoví na požadavek vrácením jednoho bajtu dat, který bude obsahovat číslo nejvyšší *LUN* jakou zařízení podporuje. Např. pokud zařízení podporuje čtyři *LUN* číslované

0 až 3, pak vrácená hodnota by měla být 3. Jestliže není žádná *LUN* přidružená k zařízení, vrácená hodnota bude 0. Hostitel nesmí posílat *CBW* pro neexistující logickou jednotku. Zařízení, které nepodporuje vícenásobné *LUN* může na požadavek *Get Max LUN* vrátit paket *STALL*.

4.3 Command Block Wrapper (CBW)

Hostitel posílá *CBW* do zařízení na bránu typu *OUT* určenou pro hromadný přenos dat v první fázi přenosu. *CBW* obsahuje příkazový blok (*CDB – Command Descriptor Block*) a ostatní informace potřebné pro přenos. Velikost *CBW* je 31 bajtů. Tab. 15 ukazuje strukturu *CBW*.

- ***dCBWSignature*** (32 bitů) – hodnota **43425355h** – identifikuje, že tato struktura je *CBW*. Jako první se posílá *LSB* (55h).

- ***dCBWTag*** (32 bitů) – tato hodnota (libovolná, volí hostitel) bude očekávána v *CSW*, kterým zařízení odpoví na požadavek jako stavová část přenosu.

- ***dCBWDataTransferLength*** (32 bitů) - Jestli-že bit 7 v *bmCBWFlags* = 0, je to počet bajtů, které bude hostitel posílat zařízení v datové fázi přenosu. Jestli-že bit 7 v *bmCBWFlags* = 1, je to počet bajtů které bude hostitel akceptovat při příjmu dat ze zařízení v datové fázi přenosu.

Tab. 15 Struktura parametrů v *CBW*

	Bit							
Bajt, Ofset	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0-3 (00h-03h)	dCBWSignature							
4-7 (04h-07h)	dCBWTag							
8-11 (08h-0Bh)	dCBWDataTransferLength							
12 (0Ch)	bmCBWFlags							
13 (0Dh)	Reserved (0)				bCBWLUN			
14 (0Eh)	Reserved (0)			bCBWCBLength				
15-30 (0Fh-1Eh)	CBWCB							

- ***bmCBWFlags*** (8 bitů) - Určuje směr přenosu dat v datové fázi přenosu. Bit 7 je bit *Direction*. Bit 7 = 0 pro *OUT* (z hostitele do zařízení), bit 7 = 1 pro *IN* (ze zařízení do hostitele). V případě že není datová fáze přenosu, je bit 7 ignorován. Ostatní bity jsou nastaveny na 0.

- ***Reserved bity*** (4 a 3 bity)– všechny rezervované bity v *CBW* nastavit na hodnotu 0.

- ***bCBWLUN*** (4 bity) - Pro zařízení s vícenásobnými *LUN* specifikuje, ke které *LUN* bude směrován požadavek. Jinak nastavit na 0.

- ***bCBWCBLength*** (5 bitů) - Délka *CDB* (*Command Descriptor Block*) v poli *CBWCB* v bajtech. Rozsah hodnoty je 1-16.

- ***CBWCB*** (128 bitů) - Příkazový blok, který má zařízení vykonat.

Pole *CBWCB* obsahuje tzv. *Command Descriptor Block (CDB)*. *CDB* je datová struktura, která obsahuje vlastní příkaz (např. *SCSI*) a doplňkové informace potřebné k provedení příkazu. Pole *CBWCB* má vždy délku 16 bajtů, ale mnoho *CDB* je kratších. V tomto případě jsou zbývající bajty nastaveny na 0.

Ve většině případů *bCBWCBLength* určuje délku *CDB* v poli *CBWCB* bez zbývajících bajtů (nastavených na 0).

Pole *bmCBWFlags* určuje směr přenosu dat v datové fázi přenosu. *dCBWDataTransferLength* určuje kolik bajtů bude hostitel posílat nebo kolik bajtů bude hostitel očekávat při příjmu.

4.3.1 Platnost a smysluplnost CBW

Hostitel interpretuje své záměry pro zařízení ve struktuře *CBW*. Pro každý přijatý *CBW* zařízení vykonává dvě verifikace. Při první kontroluje zda je přijatý *CBW* platný a ve druhé ověřuje zda informace obsažené v *CBW* mají smysluplný obsah.

Zařízení považuje *CBW* za platný pokud:

- *CBW* byl přijat poté co zařízení odeslalo *CSW* nebo po resetu,
- *CBW* má délku 31 bajtů,
- a *dCBWSignature* je rovno 43425355h.

Zařízení považuje obsah *CBW* za smysluplný pokud:

- nejsou nastaveny žádné rezervované bity (*Reserved bits*),
- *bCBWLUN* obsahuje číslo platné *LUN*, kterou zařízení podporuje,
- a *bCBWCBLength* a obsah *CBWCB* souhlasí s parametrem deskriptoru rozhraní *bInterfaceSubClass*, který určuje příkazovou sadu daného *Mass Storage* zařízení.

4.4 Command Status Wrapper (CSW)

CSW posílá zařízení hostiteli ve stavové fázi přenosu transportního protokolu *Bulk-Only*. *CSW* má velikost 13 bajtů. Tab.16 ukazuje strukturu *CSW*.

Tab. 16 Struktura parametrů v *CSW*

	Bit							
Bajt, Ofset	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0-3 (00h-03h)	dCSWSignature							
4-7 (04h-07h)	dCSWTag							
8-11 (08h-0Bh)	dCSWDataResidue							
12 (0Ch)	bCSWStatus							

- ***dCSWSignature*** (32 bitů) – hodnota **53425355h** - identifikuje, že tato struktura je *CSW*. Jako první se posílá *LSB* (55h).

- ***dCSWTag*** (32 bitů) - hodnota *dCBWTag*, která byla přijata v *CBW* od hostitele.

- ***dCSWDataResidue*** (32 bitů) - pro datové přenosy *OUT* je hodnota *dCSWDataResidue* rozdíl mezi *dCBWDataTransferLength* a počtem bajtů, který zařízení zpracovalo. Pro přenosy *IN* představuje tato hodnota rozdíl mezi *dCSWDataTransferLength* a počtem platných bajtů, které zařízení poslalo hostiteli zpět, tedy bez bajtů, které byly poslány zpět, aby vyplnily počet do hodnoty *dCWBDataTransferLength*.

- ***bSCWStatus*** - 00h = příkaz proveden (*Command Passed*), 01h = příkaz nebyl proveden (*Command Failed*), 02h = chyba (*Phase Error*)- hostitel by měl provést *reset recovery* (zotavovací reset, viz kap. 4.5).

V poli *dCSWDataResidue* zařízení určuje, zda byla přijata a zpracována všechna data, které hostitel požadoval v *CBW* k odeslání, nebo zda zařízení odeslalo všechna data, které si hostitel žádal.

V přenosech, které mají datovou fázi přenosu, *dCSWDataResidue* obsahuje rozdíl mezi *dCBWDataTranferLength* v *CBW* a množstvím dat, které zařízení zpracovalo. Jestliže zařízení zpracovalo všechna data, která hostitel poslal (tedy *dCBWDataTranferLength* bajtů), *dCSWDataResidue* je nula. V přenosech, ve kterých posílá zařízení data hostiteli, *dCSWDataResidue* obsahuje rozdíl mezi *dCBWDataTranferLength* v *CBW* a množstvím platných dat, které zařízení poslalo bez výplňových bajtů. Jestliže zařízení odešle *dCBWDataTranferLength* platných bajtů, *dCSWDataResidue* je nula.

Pole *bCSWStatus* určuje zda byl příkaz vykonán bez chyby. Hodnota 00h znamená úspěšné vykonání příkazu. Hodnota 01h znamená, že příkaz selhal a hostitel by měl okamžitě použít příkaz *SCSI REQUEST SENSE* (pro flash disk) k obdržení statusu zařízení. Hodnota 02h znamená, že by hostitel měl vykonat zotavovací reset zařízení (*reset recovery*).

4.4.1 Platnost a smysluplnost CSW

Hostitel provádí dvě verifikace každého *CSW*, který příjme. Při první kontroluje zda je *CSW* platný a ve druhé ověřuje zda informace obsažené v *CSW* mají smysluplný obsah.

Hostitel považuje *CSW* za platný pokud:

- *CSW* má délku 13 bajtů,
- *dCSWSignature* je rovno 53425355h,
- *dCSWTag* odpovídá *dCBWTag* z odpovídajícího *CBW*.

Hostitel považuje obsah *CSW* za smysluplný pokud platí jedno z následujících:

- *bCSWStatus* je 00h nebo 01h a *dCSWDataResidue* je menší nebo rovno *dCBWDataTransferLength*,
- *bCSWStatus* je 02h.

4.5 Zotavovací reset (Reset Recovery)

Zotavovací reset slouží k uvedení zařízení do výchozího stavu, aby bylo schopné při ztrátě synchronizace nebo chybě přijmout nový *CBW* od hostitele. Pro zotavovací reset musí hostitel provést řídící požadavky v následujícím pořadí:

1. ***Bulk-only Mass Storage Reset***. Po dokončení požadavku je zařízení připraveno přijmout nový *CBW*. Reset by neměl změnit stav *data-toggle* bitů a okolnosti stavu *HALT* bran. *Bulk-olny mass stotage reset* je specifický řídící požadavek pro třídu *Mass Storage*.

2. ***Clear Feature (ENDPOINT_HALT)*** požadavek pro bránu *IN*. Zařízení resetuje stav *data-toggle* brány na *DATA0* (bude vysílán lichý datový paket). *Clear Feature* je standardní USB řídící požadavek.

3. ***Clear Feature (ENDPOINT_HALT)*** požadavek pro bránu *OUT*. Zařízení resetuje stav *data-toggle* brány na *DATA0* (bude očekáván lichý datový paket).

4.6 Třídy chyb

Při komunikaci hostitele se zařízením pomocí transportního protokolu *Bulk-Only* mohou nastat různé druhy chybových situací. Jsou čtyři možné druhy chyb, které mohou nastat.

4.6.1 CBW není platný

Jestliže přijatý *CBW* není platný, zařízení pozastaví bránu *IN* hromadného přenosu. Zařízení také buď pozastaví bránu *OUT* hromadného přenosu, nebo bude akceptovat příchozí data od hostitele, které pak zahodí. Zařízení bude setrvávat v tomto stavu dokud hostitel neprovede zotavovací reset zařízení.

4.6.2 Vnitřní chyba zařízení

Při detekci vnitřní chyby zařízení, při které zařízení potřebuje provést reset, odpověď zařízení hostiteli bude jedna z následujících možností:

- pozastaví příslušnou bránu hromadného přenosu a vrátí *Phase Error* status (*bCSWStatus = 02h*),
- pozastaví obě brány hromadného přenosu dokud se neprovede zotavovací reset zařízení (*reset recovery*).

4.6.3 Rozpory hostitele a zařízení

Po rozpoznání, že *CBW* je platný a smysluplný a nenastala vnitřní chyba, se může zařízení ocitnout v situaci, kdy nemůže vyhovět očekávání hostitele podle bitu *Direction* v *bmCBWFlags* a *dCBWDataTransferLength*. V některých případech může pak zařízení požadovat zotavovací reset. V těchto případech zařízení vrátí *Phase Error* status (*bCSWStatus = 02h*).

4.6.4 Chyba příkazu

Po rozpoznání, že *CBW* je platný a smysluplný, zařízení může selhat při pokusu o uspokojení požadavku. Zařízení pak vrátí *Command Failed* status (*bCSWStatus = 01h*).

4.7 Třináct možných případů při komunikaci

Při komunikaci *Bulk-Only* transportním protokolem může nastat základních třináct případů, ve kterých se komunikace může ocitnout. V následujících kapitolách budou uvedeny požadavky na hostitele a zařízení při přenosech, kde budou jednotlivé případy označeny.

4.7.1 Hostitel neočekává datovou fázi přenosu

Tento případ nastane když *dCBWDataTransferLength* je nula. To signalizuje, že hostitel neočekává odesílání ani příjem dat z nebo do zařízení.

Hlavní požadavky pro tento případ jsou:

- Hodnota bitu *Direction* nemá vliv na výsledek těchto případů.

Požadavky na hostitele jsou:

1. Hostitel musí poslat platný a smysluplný *CBW*.
   - Hostitel může nastavit nebo vymazat *Direction* bit.
2. Hostitel se musí pokusit o příjem *CSW*.
3. Pokud je brána při příjmu *CSW* pozastavena, pak:
   - hostitel musí resetovat (clear) bránu *IN* hromadného přenosu,

- hostitel se musí pokusit přijmout *CSW* znovu.

4. Když je *CSW* platný a smysluplný, pak:
   - (případ 1) *bCSWStatus = 00h* nebo *01h* a *dCSWDataResidue* je nula,
   - (případ 2 nebo 3) jestliže je *bCSWStatus = 02h*, pak:
     hostitel musí ignorovat hodnotu *dCSWDataResidue*,
     hostitel musí provést zotavovací reset zařízení.

Požadavky na zařízení jsou:

1. Zařízení musí přijmout *CBW*.
2. Když je *CBW* platný a smysluplný, tak:
   - zařízení se pokusí vykonat příkaz nebo požadavek,
   - (případ 1) jestliže zařízení nemá žádná data k odeslání či přijetí, pak:
     zařízení musí nastavit *bCSWStatus = 00h* nebo *01h*,
     zařízení musí nastavit *dCSWDataResidue* na nulu,
   - (případ 2 nebo 3) Jestliže zařízení má data k odeslání či přijetí, pak:
     zařízení musí nastavit *bCSWStatus = 02h*.
3. Zařízení musí vrátit platný a smysluplný *CSW*.
   - Zařízení může pozastavit bránu *IN* hromadného přenosu, když se *bCSWStatus* nerovná *00h* nebo *01h*.

Když je *dCBWDataTransferLength* nula, hostitel očekává odeslání požadavku bez datové fáze přenosu. Ve většině běžných situacích (případ 1), zařízení souhlasí, že požadavek neobsahuje datovou fázi přenosu, nastaví *bCSWStatus* na 00h nebo 01h a *dCSWDataResidue* na nulu.

Jestliže zařízení očekává odeslání (případ 2) nebo příjem (případ 3) dat v datové fázi přenosu když hostitel neočekává žádná data, zařízení nastaví *bCSWStatus* na 02h a může pozastavit bránu *IN* hromadného přenosu. Při příjmu *bCSWStatus* = 02h, hostitel ignoruje *dCSWDataResidue* a vykoná zotavovací reset nebo resetuje porty zařízení.

4.7.2 Hostitel očekává příjem dat ze zařízení

Tento případ nastane když *dCBWDataTransferLength* není nula a bit *Direction* je nastaven na 1 (*IN*).

Požadavky na hostitele jsou:

1. Hostitel musí odeslat platný a smysluplný *CBW*.
2. Hostitel se musí pokusit o příjem dat ze zařízení.
3. Pokud se brána při příjmu dat uvede do stavu *HALT* (pozastavena), pak:
   - hostitel musí akceptovat přijatá data,
   - hostitel musí resetovat (*clear feature*) bránu *IN* hromadného přenosu.
4. Hostitel se musí pokusit o příjem *CSW*.
5. Pokud je brána při příjmu *CSW* ve stavu *HALT*, pak:
   - hostitel musí resetovat (clear) bránu *IN* hromadného přenosu,
   - hostitel se musí pokusit přijmout *CSW* znovu.
6. Když je *CSW* platný a smysluplný, pak:

- (případ 4, 5 nebo 6) jestliže je *bCSWStatus = 00h* nebo *01h* tak hostitel musí určit množství platných přijatých dat jako rozdíl mezi *dCBWDataTranferLength* a hodnotou *dCSWDataResidue*,
- (případ 7 nebo 8) Jestliže je *bCSWStatus = 02h*, pak:
  hostitel musí ignorovat hodnotu *dCSWDataResidue*,
  hostitel musí provést zotavovací reset zařízení.

Požadavky na zařízení jsou:

1. Zařízení musí přijmout *CBW*.
2. Když je *CBW* platný a smysluplný, tak:
   - zařízení se pokusí vykonat příkaz nebo požadavek,
   - (případ 6) jestliže zařízení chce odeslat *dCBWDataTranferLength* dat, pak:
     zařízení musí odeslat *dCBWDataTranferLength* bajtů dat,
     zařízení musí nastavit *bCSWStatus = 00h* nebo *01h*,
     zařízení musí nastavit *dCSWDataResidue* na nulu,
   - (případ 4 nebo 5) jestliže zařízení chce odeslat méně dat než hostitel požaduje, pak:
     zařízení musí odeslat svá zamýšlená data,
     (zařízení může odeslat plný počet dat tak, že zbývající bajty do hodnoty *dCBWDataTranferLength* nastaví na nulu.
     Jestliže zařízení odešle méně dat než hostitel požaduje, tak:
     zařízení může ukončit přenos krátkým paketem (kratší paket než velikost brány),
     zařízení musí uvést bránu *IN* hromadného přenosu do stavu *HALT*.)
     zařízení musí nastavit *bCSWStatus = 00h* nebo *01h*,
     zařízení musí nastavit *dCSWDataResidue* na hodnotu rozdílu mezi *dCBWDataTranferLength* a množstvím platných odeslaných dat,
   - (případ 7 nebo 8) jestliže zařízení chce buď odeslat více dat než hostitel požaduje, nebo chce přijmout data od hostitele, tak:
     zařízení může odeslat data do hodnoty *dCBWDataTranferLength*,
     jestliže zařízení pošle méně dat než požaduje hostitel, tak:
     zařízení může ukončit přenos krátkým paketem,
     zařízení musí uvést bránu *IN* hromadného přenosu do stavu *HALT*,
     jestliže zařízení pošle *dCBWDataTranferLength* bajtů dat, tak:
     zařízení může uvést bránu *IN* hromadného přenosu do stavu *HALT*,
     zařízení musí nastavit *bCSWStatus* na *02h*.
3. Zařízení musí vrátit platný a smysluplný *CSW*.

Když je *dCBWDataTransferLength* větší než nula a bit *Direction* v *bmCBWFlags* = 1, hostitel očekává příjem dat v datové části přenosu. Ve většině běžných situacích (případ 6), chce zařízení odeslat *dCBWDataTranferLength* bajtů. Zařízení nastaví *bCSWStatus* na 00h nebo 01h a *dCSWDataResidue* na nulu.

Jestliže zařízení neočekává vysílání dat (případ 4) nebo chce vyslat menší množství dat než *dCBWDataTransferLength* bajtů (případ 5), zařízení může vyplnit data nulovými bajty do požadované délky nebo nemusí poslat žádná data nebo pošle méně dat. Pro první případ (výplň nulovými bajty) zařízení nastaví *bCSWStatus* na 00h nebo 01h a nastaví

dCSWDataResidue na hodnotu rozdílu mezi *dCBWDataTransferLength* a množstvím odeslaných dat bez výplňových nulových bajtů (*pad bytes*).

Jestliže očekává odeslání většího množství dat než *dCBWDataTransferLength* bajtů (případ 7) nebo očekává příjem dat od hostitele (případ 8), zařízení může poslat *dCBWDataTransferLength* bajtů. Při odeslání menšího množství dat než *dCBWDataTransferLength* bajtů, zařízení musí pozastavit bránu *IN* hromadného přenosu a nastavit *bCSWStatus* na 02h. Při poslání *dCBWDataTransferLength* bajtů může zařízení pozastavit bránu *IN* hromadného přenosu a musí nastavit *bCSWStatus* na 02h. Při příjmu *bCSWStatus* = 02h, hostitel ignoruje *dCSWDataResidue* a vykoná zotavovací reset nebo resetuje porty zařízení.

4.7.3 Hostitel očekává odesílání dat do zařízení

Tento případ nastane když *dCBWDataTransferLength* není nula a bit *Direction* je nastaven na 0 (*OUT*).

Hlavní požadavky pro tento případ jsou:

- Hostitel nesmí odeslat paket s nulovou délkou.

Požadavky na hostitele jsou:

1. Hostitel musí odeslat platný a smysluplný *CBW*.
2. Hostitel musí odeslat data do zařízení.
   - Hostitel může odeslat krátký paket pouze k ukončení datového přenosu.
3. Pokud nastane stav *HALT* při odesílání dat, pak:
   - hostitel musí resetovat (*clear feature*) bránu *OUT* hromadného přenosu.
4. Hostitel se musí pokusit přijmout *CSW*.
5. Pokud je brána při příjmu *CSW* ve stavu *HALT*, pak:
   - hostitel musí resetovat (*clear feature*) bránu *IN* hromadného přenosu,
   - hostitel se musí pokusit přijmout *CSW* znovu.
6. Když je *CSW* platný a smysluplný, pak:
   - (případ 9, 11 nebo 12) jestliže je *bCSWStatus = 00h* nebo *01h* tak hostitel musí určit množství zpracovaných dat zařízením jako rozdíl mezi *dCBWDataTranferLength* a hodnotou *dCSWDataResidue*,
   - (případ 10 nebo 13) jestliže je *bCSWStatus = 02h*, pak:
     hostitel musí ignorovat hodnotu *dCSWDataResidue*,
     hostitel musí provést zotavovací reset zařízení.

Požadavky na zařízení jsou:

1. Zařízení musí přijmout *CBW*.
2. Když je *CBW* platný a smysluplný, tak:
   - zařízení se pokusí vykonat příkaz nebo požadavek,
   - (případ 9, 11 nebo 12) jestliže zařízení chce zpracovat méně nebo rovno množství dat, které hostitel chce odestlat, tak:
     zařízení musí přijmout data,

zařízení může buď akceptovat všech *dCBWDataTranferLength* bajtů dat, nebo ukončit předčasně přenos uvedením brány *OUT* hromadného přenosu do stavu *HALT*,

zařízení musí nastavit *bCSWStatus = 00h* nebo *01h*,

zařízení musí nastavit *dCSWDataResidue* na hodnotu rozdílu mezi *dCBWDataTranferLength* a množstvím dat zpracovaných zařízením,

- (případ 10 nebo 13) jestliže zařízení chce zpracovat více dat než hostitel požaduje nebo chce odesílat data do hostitele, tak:

  zařízení může přijmout data maximálně do hodnoty *dCBWDataTranferLength* bajtů,

  zařízení musí buď akceptovat všech *dCBWDataTranferLength* bajtů dat, nebo předčasně ukončit přenos uvedením brány *OUT* hromadného přenosu do stavu *HALT*,

  zařízení musí nastavit *bCSWStatus = 02h*.

3. Zařízení musí vrátit platný a smysluplný *CSW*.
   - Jestliže *bCSWStatus* není *00h* nebo *01h*, zařízení může uvést bránu *IN* hromadného přenosu do stavu *HALT*.

Když je *dCBWDataTransferLength* větší než nula a *Direction* bit v *bmCBWFlags* = 0, host očekává odesílání dat v datové fázi přenosu. Ve většině běžných situacích (případ 12), zařízení očekává příjem *dCBWDataTransferLength* bajtů dat. Zařízení nastaví *bCSWStatus* na 00h nebo 01h a nastaví *dCSWDataResidue* na nulu.

Jestliže zařízení očekává příjem menšího množství dat než *dCBWDataTransferLength* bajtů (případ 11) nebo neočekává žádná data (případ 9), zařízení může akceptovat *dCBWDataTransferLength* bajtů (doporučeno) nebo může předčasně ukončit přenos uvedením brány *OUT* hromadného přenosu do stavu *HALT*. V prvním případě (při akceptování dat) zařízení nastaví *bCSWStatus* na 00h nebo 01h a nastaví *dCSWDataResidue* na hodnotu rozdílu mezi *dCBWDataTransferLength* a množstvím zpracovaných dat. Množství zpracovaných dat zařízením může být menší nebo rovno množství dat přijatých a akceptovaných zařízením. Uvedení brány *OUT* hromadného přenosu do stavu *HALT* v tomto případě může zapříčinit problém v OS Windows, proto většina zařízení akceptuje *dCBWDataTransferLength* bajtů dat a nastaví *dCSWDataResidue* na vhodnou hodnotu.

Jestliže zařízení očekává příjem většího množství dat než *dCBWDataTransferLength* bajtů (případ 13) nebo očekává odesílání dat hostiteli (případ 10), zařízení může akceptovat data do hodnoty *dCBWDataTransferLength* bajtů nebo může předčasně ukončit přenos uvedením brány *OUT* hromadného přenosu do stavu *HALT*. Pro první případ (při akceptování dat), zařízení nastaví *bCSWStatus* na 02h. Jestliže je *bCSWStatus* = 02h, zařízení může uvést bránu *IN* hromadného přenosu do stavu *HALT*. Při příjmu *bCSWStatus* = 02h, hostitel ignoruje *dCSWDataResidue* a vykoná zotavovací reset zařízení nebo resetuje porty zařízení.

4.7.4 Pozastavení brány (HALT)

Třída *Mass Storage* je unikátní ve způsobu použití *STALL* na hromadných branách. V jiných USB třídách může odesílatel naznačit konec přenosu vysláním krátkého paketu, kterým je datový paket nulové délky nebo nějaký paket kratší než *wMaxPacketSize* (v některých případech to je povoleno i v *Bulk-Only* protokolu, ale pro maximální kompatibilitu

se doporučuje pozastavování bran). Zařízení třídy *Mass Storage* použije pro tento účel vysílání paketu *STALL* stejně jako pro indikaci ostatních chybových situací.

Poté co brána hromadného přenosu vrátí paket *STALL*, je ve stavu *HALT*. K obnovení komunikace s bránou musí hostitel vyslat řídící požadavek *ClearFeature(ENDPOINT_HALT)* s adresou příslušné brány v poli *wIndex* požadavku.

Brána 0 může také použít *STALL*. Při příjmu *Get Max LUN* požadavku, zařízení s pouze jedinou *LUN* může vrátit *STALL* jako odpověď, že zařízení tento požadavek nepodporuje. Brána se pak vrací do normálního režimu při příjmu dalšího paketu *SETUP*.

Mass Storage zařízení ***musí*** pozastavit jednu nebo obě brány hromadného přenosu v následujících situacích:

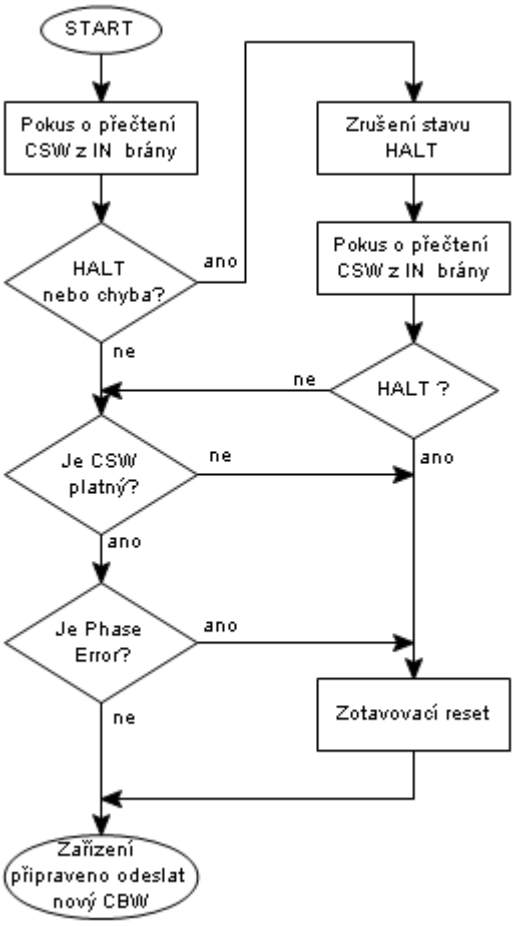
- jestliže zařízení pošle méně dat než bylo požadováno v datové fázi přenosu, zařízení musí uvést bránu *IN* pro hromadný přenos do stavu *HALT*,
- jestliže příjme *CBW*, který není platný, musí uvést bránu *IN* hromadného přenosu do stavu *HALT* a dále musí buď nastavit bránu *OUT* hromadného přenosu do stavu *HALT*, nebo akceptovat a vyřadit všechny data přijímaná bránou,
- při vnitřní chybě vyžadující reset, zařízení musí buď uvést bránu do stavu *HALT* v každém datovém přenosu a nastavit *bCSWStatus* = 02h, nebo uvést brány hromadného přenosu *IN* a *OUT* do stavu *HALT* dokud nebude vykonán zotavovací reset.

Zařízení *Mass Storage* ***může*** pozastavit brány hromadného přenosu v následujících situacích:

- jestliže zařízení vyžaduje odeslat více dat než hostitel specifikuje v *CBW*, poté co odešle požadované množství dat, zařízení může uvést bránu *IN* hromadného přenosu do stavu *HALT*,
- jestliže zařízení očekává k přijetí rozdílné množství dat než hostitel specifikuje v *CBW*, zařízení může uvést bránu *OUT* hromadného přenosu do stavu *HALT*,
- jestliže zařízení určí během datové fáze přenosu, že nemůže dokončit požadavek, zařízení může uvést bránu *IN* a *OUT* hromadného přenosu do stavu *HALT*.

4.8 Příjem CSW

Specifikace *Bulk-Only* transportního protokolu definuje algoritmus vyčítání *CSW* ze zařízení hostitelem. Zařízení odešle každý *CSW* bránou *IN* hromadného přenosu. Na obr. 15 je znázorněn algoritmus, který by měl hostitel použít pro každý přenos *CSW* při vstupu do stavové fáze přenosu.



Obr. 15 Algoritmus stavové fáze přenosu

5 SCSI

SCSI (***Small Computer System Interface***) je standardní rozhraní a sada příkazů pro výměnu dat mezi externími nebo interními zařízeními a počítačovou sběrnicí. Veškeré informace o tomto standardu jsou dostupné v [26].

Standard *SCSI* je velice rozsáhlý, proto jsou v této kapitole stručně popsány pouze příkazy *SCSI* a z nich jen příkazy použité v této diplomové práci (potřebné pro obsluhu flash disku).

Příkazy *SCSI* jsou rozděleny do dvou specifikací. *SPC-2* nebo novější *SPC-3* (*SCSI Primary Commands*) definuje primární *SCSI* příkazy. *SBC-2* nebo novější *SBC-3* (*SCSI Block Commands*) definují *SCSI* blokové příkazy, viz [6], [7], [8], [9].

5.1 Implementované příkazy

Každé zařízení specifikuje svou příkazovou sadu v odpovědi na příkaz *SCSI INQUIRY*. Příkazové sady specifikuji seznamy povinných příkazů. Např. zařízení, které vrátí v odpovědi na příkaz *INQUIRY PERIPHERAL DEVICE TYPE = 00h* (*direct accses block device*) a *VERSION = 05h* (*SPC-3*), by mělo implementovat všechny příkazy definované jako povinné ve specifikaci *SPC-3*. V praxi však mnoho zařízení neimplementují všechny povinné příkazy.

Prakticky pro umožnění komunikace by mělo zařízení implementovat minimálně tyto příkazy:

- *INQUIRY*
- *READ CAPACITY(10)*
- *READ(10)*
- *REQUEST SENSE*
- *TEST UNIT READY*
- *WRITE(10)* (pro zapisovatelná zařízení)

Příkaz *INQUIRY* slouží k zjištění obecných informací o zařízení (jakou příkazovou sadu podporuje, identifikační čísla výrobku a výrobce atd.). Příkazem *READ CAPACITY(10)* lze zjistit počet sektorů v dané logické jednotce a také velikost jednoho sektoru v bajtech. Příkazem *READ(10)* lze číst sektory ze zařízení, naopak příkaz *WRITE(10)* umožňuje zápis sektorů do zařízení. *TEST UNIT READY* oznamuje, že jednotka je připravená. *REQUEST SENSE* slouží pro vyčtení hlášení o chybě při selhání příkazu, kdy zařízení vrací hostiteli tzv. *Sense data*.

Hostitel nebo zařízení mnohdy požadují ještě další příkazy navíc. Když zařízení tvrdí, že podporuje danou příkazovou sadu, pro hostitele to znamená slušný předpoklad toho, že zařízení implementuje všechny povinné příkazy z dané příkazové sady.

Při příjmu nepodporovaného příkazu nesmí zařízení zkolabovat. Správná odpověď na nepodporovaný příkaz je:

- vrácení *bCSWStatus = 01h* (*Command Failed*) v *CSW*,
- v *Sense* datech nastaví *SENSE KEY = 05h* (*ILLEGAL REQUEST*) a nastaví *ADDITIONAL SENSE CODE = 20h* (*INVALID COMMAND OPERATION CODE*).

Z hlediska hostitele, který předpokládá, že připojené *Mass Storage* zařízení je flash disk, jako minimum pro úspěšnou komunikaci je použití příkazů *READ CAPACITY(10)*, *READ(10)* a *WRITE(10)*. Tyto tři příkazy jsou zcela základní a podporuje je každý flash disk.

5.1.1 Příkaz READ CAPACITY(10)

Příkaz *READ CAPACITY(10)* slouží k zjištění počtu sektorů v *LUN* a velikosti jednoho sektoru. První informace je důležitá pro formátování disku, druhá je nezbytná pro použití příkazů *READ* a *WRITE* a také pro potřebné výpočty při správě souborového systému na disku. Tab. 17 ukazuje *CDB* příkazu *READ CAPACITY(10)* dle *SBC-2*.

Tab. 17 Příkaz READ CAPACITY(10)

Bajt	Bit							
	7	6	5	4	3	2	1	0
0	Operační kód (25h)							
1	Rezervováno							zrušeno
2	(MSB)	LBA adresa						
5								(LSB)
6		Rezervováno						
7								
8	Rezervováno							PMI
9	Control							

Délka příkazu je 10 bajtů, tedy *bCBWCBLength* v *CBW* bude nastaven tuto hodnotu. Pro základní funkci příkazu je uveden v prvním bajtu operační kód 25h a všechny ostatní bajty jsou nastaveny na hodnotu 0. Tab. 18 ukazuje strukturu, kterou hostitel obdrží v datové fázi přenosu. Délka této struktury je 8 bajtů, tedy *dCBWDataTransferLength* v *CBW* bude nastaven na hodnotu 8. Bit *Direction* = 1 (*IN*).

Tab. 18 Data příkazu READ CAPACITY(10)

Bajt	Bit							
	7	6	5	4	3	2	1	0
0	(MSB)	Nejvyšší dostupná LBA adresa						
3								(LSB)
4	(MSB)	Velikost sektoru v bajtech						
7								(LSB)

Pokud počet sektorů převyšuje rozsah 32-bitového čísla, zařízení může vrátit jako nejvyšší adresu *LBA* hodnotu FFFFFFFFh. Druhá vrácená hodnota je velikost sektoru v bajtech (většina flash disků má velikost sektoru 512 bajtů).

5.1.2 Příkaz READ(10)

Příkaz *READ(10)* slouží ke čtení sektorů z dané *LUN* disku. Menší ucelenou jednotku než jeden sektor nelze přečíst. Tab. 19 ukazuje *CDB* příkazu *READ(10)*.

Tab. 19 Příkaz READ(10)

Bajt	Bit							
	7	6	5	4	3	2	1	0
0	Operační kód (28h)							
1	RDPROTECT			DPO	FUA	Reserved	FUA_NV	Zrušeno
2	(MSB)	LBA adresa sektoru						
5								(LSB)
6	Reserved			Group number				
7	(MSB)	Počet sektorů						
8								(LSB)
9	Control							

Příkaz má délku 10 bajtů. Nutné je vyplnit operační kód 28h, *LBA* adresu prvního čteného sektoru a počet čtených sektorů od zadané adresy *LBA*. Pro základní funkci lze všechny ostatní parametry nastavit na hodnotu 0. *LBA* adresa je přímo číslo daného sektoru od začátku *LUN*. *LBA* adresa prvního sektoru je 0, druhého 1 atd.

V *CBW* je nutné do parametru *dCBWDataTransferLength* vložit celkový čtený počet bajtů (velikost sektoru x počet čtených sektorů). Bit *Direction* = 1 (*IN*). V datové fázi přenosu zařízení odešle odpovídající počet bajtů.

5.1.3 Příkaz WRITE(10)

Příkaz *WRITE(10)* slouží k zápisu sektorů na danou *LUN* disku. Menší ucelenou jednotku než jeden sektor nelze zapsat. Tab. 20 ukazuje *CDB* příkazu *WRITE(10)* dle specifikace *SBC-2*.

Tab. 20 Příkaz WRITE(10)

Bajt	Bit							
	7	6	5	4	3	2	1	0
0	Operační kód (2Ah)							
1	WRPROTECT			DPO	FUA	Reserved	FUA_NV	Zrušeno
2	(MSB)							
5	LBA adresa sektoru							(LSB)
6	Reserved			Group number				
7	(MSB)							
8	Počet sektorů							(LSB)
9	Control							

Příkaz má rovněž délku 10 bajtů, tedy *bCBWCBLength* v *CBW* bude nastaven hodnotu 10. V příkazu je nutné vyplnit operační kód 2Ah, *LBA* adresu prvního zapisovaného sektoru a počet zapisovaných sektorů od zadané adresy *LBA*. Pro základní funkci lze všechny ostatní parametry nastavit opět na hodnotu 0.

V *CBW* je nutné do parametru *dCBWDataTransferLength* vložit celkový počet zapisovaných bajtů (velikost sektoru x počet zapisovaných sektorů). Bit *Direction* = 0 (*OUT*). V datové fázi přenosu hostitel odešle do zařízení odpovídající počet bajtů.

6 FAT16

Souborový systém *FAT16* rozděluje datovou oblast logické jednotky na bloky nazývané clustery, jejichž velikost je násobkem mocniny dvou a velikosti sektoru. Celkový počet clusterů v jedné logické jednotce může být až 2^{16}, neboť číslo clusteru je 16-bitová hodnota. Uložený soubor je tedy rozdělen na jednotlivé clustery a informaci o tom, které clustery patří k danému souboru jsou uloženy v tabulce *FAT* (*File Allocation Table*). Protože je fyzická velikost souboru na disku pouze celočíselnými násobky clusteru, poslední culster

souboru není většinou zaplněn celý. Proto často vzniká u tohoto systému jistá ztráta užitečného místa logické jednotky. Informace o souborovém systému *FAT16* jsou čerpány z [17] a [18].

Jak bylo zmíněno v kap. 3.2 první sektor v příslušném paměťovém poli bývá často *MBR*. I když logická jednotka flash disku může začínat v tomto případě přímo *FAT16 Boot Record* sektorem, spousta výrobců formátuje flash disky včetně *MBR* sektoru, proto zde bude popsán tento systém také.

6.1 Master Boot Record sektor (MBR)

Pokud je *MBR* na disku přítomen, je to vždy první sektor na disku s *LBA* adresou 0. Může obsahovat vykonávací kód (pokud vykonávací kód není obsažen, pole je vyplněno nulami) a na konci *MBR* jsou čtyři tabulky popisující jednotlivé *partition* na disku (*partition* – jednotná část paměťového pole disku). Bez použití *Extended partition* lze rozdělit disk pomocí *MBR* na čtyři svazky (*Extended partition* – začíná *Extended Boot Record* sektorem, který umožňuje danou *partition* dále rozdělit – používá Windows 98). Struktura *MBR* je uvedena v tab.21.

Tab. 21 Struktura MBR

Ofset	Délka	Popis
0h	446B	Vykonávací kód ("Executable code")
1BEh	16B	1. Partition entry - informace o 1.partition
1CEh	16B	2. Partition entry - informace o 2.partition
1DEh	16B	3. Partition entry - informace o 3.partition
1EEh	16B	4. Partition entry - informace o 4.partition
1FEh	2B	"Executable mark" = 55AAh

MBR je optimalizován pro sektor délky 512 bajtů, menší sektor nemůže ve standardních zařízeních být. Pokud má disk větší sektory než 512 bajtů, prvních 512 bajtů je *MBR* podle tab. 21, zbylá část sektoru je vyplněna nulami. V tab. 22 je uvedena struktura tabulek popisujících jednotlivé *partition* (*partition entry*). Všechny čtyři tabulky mají strukturu stejnou.

Tab. 22 Struktura tabulky partition

Ofset	Délka	Popis
0h	1B	Stav partition, 0h - neaktivní, 80h - aktivní
1h	1B	Začátek partition - hlava
2h	2B	Začátek partition - cylindr/sektor
4h	1B	Typ partition, 04h - FAT16 menší než 32MB, 06h - FAT16 vetší než 32MB, 0Eh - jako 06h používající LBA 13h Extensions
5h	1B	Konec partition - hlava
6h	2B	Konec partition - cylindr/sektor
8h	4B	Počet sektorů mezi MBR a začátkem partition
Ch	4B	Délka partition v sektorech

Z údajů z tabulek pro každou *partition* jsou pro práci se souborovým systémem podstatné jen typ *partition*, podle kterého lze identifikovat *FAT16* a počet sektorů mezi *MBR* sektorem a *Boot Record* sektorem vlastní *partition*.

6.2 FAT16 Boot Record sektor

FAT16 Boot Record sektor je první sektor v *partition*. *LBA* adresa se vypočítá takto:

FAT16 Boot Rec. = (Počet sektorů mezi *MBR* a začátkem *partition*).

V tab. 23 je uvedena struktura *FAT16 Boot Record* sektoru. Tento sektoru obsahuje všechny potřebné údaje o *partition*. Vícebajtové hodnoty jsou uloženy od *LSB*. *ASCII* řetězce jsou uloženy zleva doprava a nevyužité pole jsou vyplněny *ASCII* znaky mezery (0x20). Na konci sektoru je tzv. *Executable Mark*. Nerovná-li se *Executable mark* hodnotě 0x55AA, sektor není platný.

Tab. 23 Struktura *FAT16 Boot Record* sektoru

Ofset	Délka	Popis
0h	3B	Instrukce skoku na začátek vykonavacího kódu
3h	8B	ASCII řetězec názvu operačního systému
Bh	2B	Velikost sektoru v bajtech
Dh	1B	Počet sektorů na jeden cluster
Eh	2B	Počet sektorů mezi FAT16 Boot Rec. sektorem a začátkem 1. FAT
10h	1B	Počet alokačních tabulek FAT (obvykle 2)
11h	2B	Počet vstupů kořenového adresáře (každý má 32B)
13h	2B	Počet sektorů v partition, pouze pro FAT16 menší než 32MB
15h	1B	Popisovač média (u HDD a flash pamětí bývá F8h)
16h	2B	Počet sektorů v jedné FAT
18h	2B	Počet sektorů na stopu
1Ah	2B	Počet hlav
1Ch	4B	Počet skrytých sektorů ve FAT
20h	4B	Počet sektorů v partition, pouze pro FAT16 větší než 32MB
24h	2B	Číslo logické jednotky v partition
26h	1B	Signatura rozšíření (29h)
27h	4B	Sériové číslo partition
2Bh	11B	ASCII řetězec názvu disku (disk volume)
36h	8B	ASCII řetězec názvu file systému "FAT16"
3Eh	448B	Vykonávací kód ("Executable code")
1FEh	2B	"Executable mark" = 55AAh

6.3 Alokační tabulka FAT

LBA adresa začátku první *FAT*:

1. FAT = (*FAT16 Boot Rec*.) + (Počet sektorů mezi *FAT16 Boot Rec*. a začátkem 1. *FAT*).

Všechny alokační tabulky souborového systému (většinou bývají dvě) jsou umístěny v *partition* hned za sebou. *FAT* je složená až z 2^{16} buněk délky 2 bajty. Počet sektorů *FAT* je uveden ve *FAT16 Boot Record* sektoru. Počet buněk *FAT* je nutné vypočítat z parametrů *partition*, resp. z délky datové oblasti. První buňka obsahuje u flash pamětí a HDD hodnotu

F8FFh a ve druhé bývá hodnota FFFFh. První dvě buňky, představující clustery 0 a 1, jsou tedy vyhrazené. Clustery 0 a 1 nejsou na disku fyzicky přítomny. Datová oblast jednotky začíná clusterem číslo 2. Každá buňka *FAT* náleží jednomu clusteru jednotky. Zde je vhodné poznamenat, že clustery se týkají pouze datové oblasti jednotky. Tab. 24 ukazuje možné hodnoty, kterých můžou buňky *FAT* nabývat. O způsobu záznamu souboru nebo adresáře v alokační tabulce bude pojednáno dále.

Tab. 24 Možné hodnoty buňek FAT

Hodnota	Popis
0000h	Volný cluster
0002h - FFEFh	Cluster je část souboru nebo adresáře
FFF0h - FFF6h	Vyhrazený cluster
FFF7h	Vadný cluster
FFF8h - FFFFh	Poslední cluster souboru nebo adresáře

6.4 Kořenový adresář FAT16 (Root Directory)

V souborovém systému *FAT16* mohou být ukládány na logickou jednotku soubory a adresáře. Adresáře jsou speciálním typem souboru a v paměťovém poli jsou ukládány stejně jako soubory. Výjimkou je kořenový adresář (*Root Directory*), který je v souborovém systému *FAT16* umístěn mimo datovou oblast logické jednotky. Kořenový adresář má na rozdíl od ostatních adresářů pevně danou velikost, kterou nejde dále zvětšovat (je uvedena ve *FAT16BR*). Tato velikost je daná formátováním a obvykle je nastavena na 512 vstupů (vstup = *entry*). Kořenový adresář leží hned za poslední alokační tabulkou. Adresa *LBA* začátku kořenového adresáře je

Začátek kořenového adresáře = (začátek 1.FAT) + (délka jedné FAT)*(počet FAT).

V kořenovém adresáři můžou být uloženy podadresáře, soubory nebo *LFN* (*Long File Name* – dlouhý název souboru nebo adresáře). První vstup je obvykle *disk volume*. Pokud se používají dlouhá jména souborů a adresářů, tak se kořenový adresář velmi rychle zaplní.

6.5 Datová oblast FAT16

Datová oblast logické jednotky je cílové místo pro ukládání vlastních souborů a adresářů. Je umístěna hned za kořenovým adresářem a její konec je zároveň koncem logické jednotky. Může se stát, že za datovou oblastí je ještě několik nevyužitelných sektorů do konce *partition*. *LBA* adresu začátku datové oblasti lze vypočítat podle rovnice:

Začátek dat = (Začátek kořenového adresáře) + (počet vstupů do kořenového adresáře) * 32 / (velikost sektoru).

Celá datová oblast je rozdělena na clustery. Jak bylo uvedeno dříve, cluster je nejmenší datová jednotka *FAT16*, kterou lze zaznamenat do alokační tabulky. Počet sektorů v clusteru může být 1, 2, 4, 8, 16, 32, 64 a 128. Maximální možná kapacita logické jednotky s *FAT16* je 4294,2464MB pro velikost sektoru 512 bajtů. Pro větší kapacity je třeba použít zařízení s větší velikostí sektoru nebo jiný souborový systém. Záznam adresářů (mimo

kořenový adresář) a vlastní data uložených souborů jsou pak ukládána do jednotlivých clusterů datové oblasti disku. *LBA* adresu prvního sektoru v clusteru je

1.sektor clusteru = (Začátek dat) + (cluster - 2) * (počet sektorů na cluster).

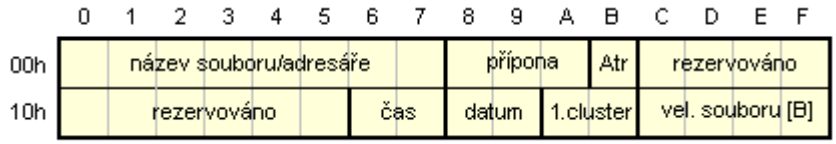
Předchozí kapitoly popisují rozdělení paměťového pole disku na jednotlivé oblasti. Na obr. 16 je toto rozdělení znázorněno graficky pro jedinou *partition* na disku. Na obr. 16 je *LBA* adresa 0 na spodní části obrázku, poslední *LBA* adresa partition pak na horní.

Obr. 16 Organizace paměťového pole disku

Mezi některými oblastmi bývá nevyužité místo, které je obvykle velké maximálně několik desítek sektorů. Teoreticky by tyto oblasti nemusely na disku být. Např. pokud je použito formátování jako na obr. 16, *FAT16 Boot Record* sektor bývá často v sektoru 63, což je zřejmě z důvodu, aby *partition* začínala na novém cylindru disku (první pevné disky měly 63 sektorů na stopu, což je bezpředmětné u pamětí flash). Přesto např. operační systém Windows XP formátuje flash disk s těmito nevyužitými oblastmi. Velikost nevyužité oblasti na konci paměťového pole (podle adresování *LBA*) záleží na tom kolik sektorů zůstane za posledním clusterem, jejichž počet už nestačí pro vytvoření dalšího clusteru při rozvržení formátování. Může se stát, že poslední sektor posledního clusteru je zároveň poslední sektor *partition*. V tomto případě nevyužité místo na konci není.

6.6 Vstupy (Entries)

Vstupy (*Entries*) jsou záznamy délky 32 bajtů, které jsou ukládány v adresářích. Můžou to být hlavičky souborů a adresářů, dlouhé názvy souborů (*LFN*) a tzv. *Disk Label*“, který bývá často umístěn na začátku kořenového adresáře a v oblasti pro jméno souboru obsahuje 11 znaků dlouhý řetězec, popisující danou logickou jednotku. Obr. 17 ukazuje grafické znázornění struktury jednoho vstupu.



Obr. 17 Struktura vstupu FAT16

Tab. 24 popisuje význam jednotlivých parametrů vstupu. V polích pro název souboru a přípony je uložen název odpovídající formátu DOS8.3. Uprostřed názvu nesmí být mezera a je složen z písmen velké abecedy a čísel, popř. dalších speciálních znaků. Název i přípona jsou zarovnány v příslušných polích vlevo. Nevyužité pole řetězců musí být vyplněno znaky mezera (20h).

Tab. 25 Význam parametrů vstupu (*Entry*)

Ofset	Délka	Popis
0h	8B	ASCII řetězec názvu souboru
8h	3B	ASCII řetězec přípony souboru
Bh	1B	Atribut souboru bit 0 - pouze ke čtení (Read Only) bit 1 - skrytý (Hidden) bit 2 - systémový soubor (System) bit 3 - "disk label" bit 4 - adresář (Directory) bit 5 - archivní (Archive)
Ch	10B	Rezervováno
16h	2B	Čas vytvoření nebo poslední změny bit 0-4 - sekundy vyděléné dvěma bit 5-10 - minuty bit 11-15 - hodiny
18h	2B	Datum vytvoření nebo poslední změny bit 0-4 - den bit 5-8 - měsíc bit 9-15 - rok - 1980, např. 28 pro rok 2008
1Ah	2B	Počáteční cluster souboru / adresáře
1Ch	4B	Délka souboru v bajtech

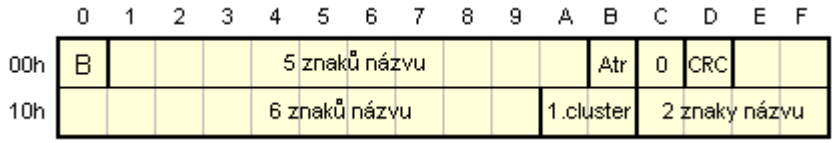
V rezervovaném poli uchovává např. operační systém Windows XP další datumy vztahující se k manipulaci se souborem jako jeho vytvoření a poslední otevření. V poli datum a čas je zaznamenána poslední změna v souboru.

Pokud je vstup smazán, nemaže se celý, pouze na místo prvního bajtu s ofsetem 0x00 je uložena hodnota 0xE5. V poli počáteční cluster je uloženo číslo prvního clusteru daného

souboru nebo adresáře. V posledním poli je uchovávána skutečná velikost souboru v bajtech. Adresáře mají toto pole nastaveno na hodnotu 0.

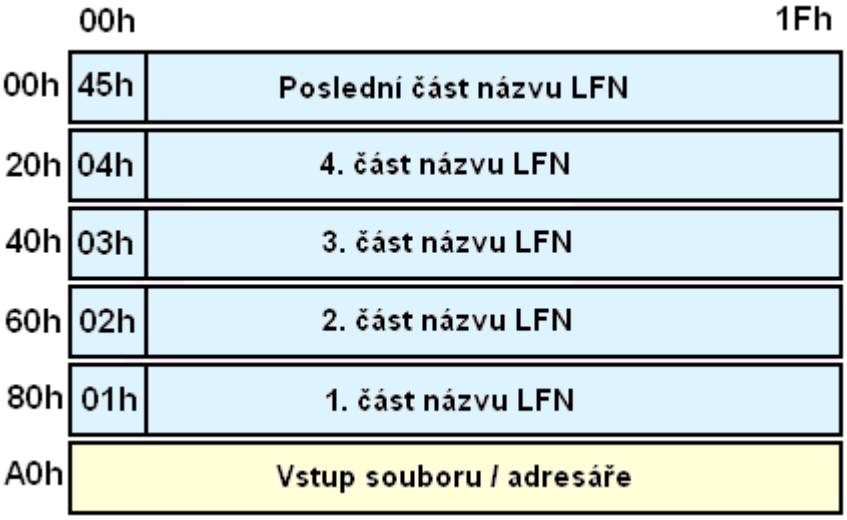
Vstup souboru nemůže obsahovat atribut adresáře, *disk label* a celková hodnota nesmí být 0x0F. Vstup adresáře musí mít naopak nastaven atribut adresáře.

Vstupy *LFN* slouží pro ukládání dlouhých názvů souborů nebo adresářů. Protože operační systém DOS *LFN* nepodporuje, vstup *LFN* je označen tak, že celková hodnota atributového bajtu je 0x0F. DOS všechny vstupy s nepovolenými kombinacemi atributů ignoruje. Obr. 18 ukazuje strukturu vstupu *LFN*.



Obr. 18 Struktura vstupu LFN FAT16

Jméno je v *LFN* uloženo v šestnácti bitovém kódu, proto může obsahovat širokou škálu různých znaků. V atributu je uložena hodnota 0x0F. Řetězec znaků je v *LFN* vstupu rozdělen na tři části. Každý znak má velikost dva bajty. První je uložen vždy *LSB*. Celý název souboru nebo adresáře je rozdělen na části po třinácti znacích a pro každou část je použit jeden vstup *LFN*. Příklad uspořádání vstupů *LFN* jednoho názvu je na obr. 19.



Obr. 19 Způsob řazení LFN v adresářích

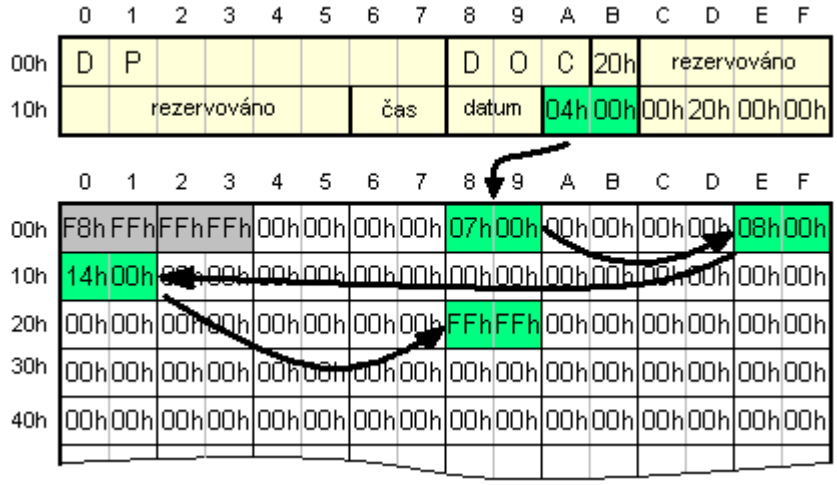
Jednotlivé části názvu jsou řazeny pozpátku a jsou umístěny před vlastním krátkým vstupem adresáře nebo souboru. *LFN* položky jednoho názvu a příslušný vstup souboru/adresáře musí být vždy u sebe jak je naznačeno na obr. 19. Bajt *B* obsahuje pořadí částí názvu souboru. *LFN* s pořadovým číslem 0x01 je první část názvu. K pořadovému číslu poslední části názvu je vždy přičtena hodnota 0x40. Podle toho, že je hodnota bajtu *B* větší než 0x40 je rozpoznáno, že je to poslední část názvu *LFN*. Z těchto podmínek lze snadno vypočítat maximální možnou délku názvu *LFN*, kterou podporuje souborový systém *FAT16* → 0x3F * 13 + 13 = 832 znaků. Název *LFN* je vždy jeden řetězec obsahující jméno, příponu a

také znak tečka. Pokud je na konci řetězce v poslední části názvu *LFN* ještě místo, je na konec řetězce vložena hodnota 0x0000 a dále už jen hodnoty 0xFFFF.

Pokud název při vytváření souborů nebo adresářů splňuje podmínky názvu DOS 8.3, tak je vytvořen jen vstup souboru/adresáře. Způsob výpočtu kontrolního součtu v poli CRC je uveden např. v [18].

6.7 Záznam ve FAT

Na obr. 20 je ukázán jednoduchý příklad čtení souboru z tabulky *FAT* (stejným způsobem funguje práce s adresáři). Na horní straně obr. 20 je příklad vstupu se jménem "DP.DOC" a na spodní straně obrázku je příklad začátku tabulky *FAT*. Z atributového bajtu s hodnotou 20h je patrné, že je nastaven pouze archiv bit a jde tedy o vstup souboru. Z parametru 1.cluster je přečteno číslo prvního clusteru souboru, které je v tomto případě 0004h. Poté je načtena z *FAT* buňka číslo 0004h. V této buňce je uloženo číslo clusteru (a také další buňky *FAT*), kde soubor pokračuje, v tomto případě 0007h. Stejným postupem se pokračuje dokud se nenarazí na buňku s hodnotou FFF8h-FFFFh, která udává, že jde o poslední cluster souboru.



Obr. 20 Způsob práce s tabulkou FAT

6.8 Adresáře

Soubory jsou v systému *FAT16* uloženy ve stromové struktuře. Na vrcholu je kořenový adresář (umístěn ve *FAT16* mimo datovou oblast a není dělen na clustery), ze kterého se musí vždy začít, aby se dostalo k cílovému souboru. V kořenovém adresáři můžou být hlavičky souborů a dalších adresářů, ve kterých můžou být opět další adresáře a tento strom se může dále větvit dokud je v jednotce volné místo. Adresáře mají stejnou strukturu jako soubory, jen ve vstupu mají nastaven atribut *Directory*. V pracovních clustrech adresářů jsou pak uloženy vstupy souborů a dalších podadresářů. Pokud je cluster adresáře zcela zaplněn je rozšířen o další cluster stejně jako soubor. Pouze kořenový adresář nejde dále rozšiřovat.

V prvním clusteru každého adresáře je na začátku uložen vstup adresáře s názvem „.“ (tečka) a hned za ním je vstup adresáře s názvem „..“ (dvě tečky). Příponu tyto dva adresáře nemají. Oba tyto adresáře nemají nastaveny žádné další atributy. Adresář „tečka“ má uloženo na místě prvního clusteru ve vstupu číslo clusteru kde sám leží (tedy obsahuje číslo prvního clusteru adresáře, ve kterém je umístěn). Vstup adresáře „dvě tečky“ zase obsahuje na místě prvního clusteru číslo prvního clusteru adresáře o úroveň výše. Pokud se pohybujeme v adresářové struktuře systému *FAT16*, tento adresář umožňuje návrat o úroveň výše bez potřeby zálohovat čísla clusterů adresářů, do kterých se bylo vnořováno.

Při vytvoření adresáře je obsah celého prvního clusteru adresáře vymazán na hodnoty 0 a na začátku jsou vytvořeny adresáře „.“ a „..“. Pokud je při prohledávání adresáře nalezena na místě prvního znaku vstupu hodnota 0, znamená to, že je to konec záznamu v adresáři a dále jej není třeba prohledávat (vstupy se mažou hodnotou E5h na první pozici vstupu). Operační systém Windows XP při vytváření nového vstupu nepřepisuje smazané vstupy. Nový vstup ukládá na úplný konec záznamu adresáře kde jsou nuly. V kořenovém adresáři adresáře „.“ a „..“ samozřejmě nejsou.

7 USB řadiče typu „HOST“

USB řadiče typu *HOST* (*HOST* = hostitel) zprostředkovávají úlohu hostitele, tedy řídícího prvku na počátku hierarchie sběrnice USB. Dříve byl hostitelský obvod pouze součástí osobního počítače, dnes se již vyskytuje poměrně široký výběr obvodů vykonávající funkci hostitele. Velké množství USB obvodů typu *HOST* bývají součástí složitějších obvodů jako jsou mikrokontroléry nebo jiné mikroprocesorové systémy, kde bývají na čipu zabudovány jako periferie daného obvodu. Tato diplomová práce se však zajímá o USB řadiče jako samostatné integrované obvody připojitelné do již stávajících mikroprocesorových systémů.

7.1 Srovnání dostupných obvodů typu „HOST“

Snadno dostupné USB řadiče jsou integrované obvody firem Cypress, Atmel, Maxim, Philips a FTDI. V tab. 26 jsou uvedeny některé řadiče typu *HOST* i s jejich základními parametry, detailní popis řadičů je v [4], [19], [20], [21], [22], [23] a [24].

Všechny uvedené obvody mohou pracovat i ve funkci *slave* , tedy jako USB periferie (*DEVICE*). K tomu musí obsahovat komunikační brány (*endpoints*), ty však pro funkci hostitele nejsou potřeba. Dále všechny obvody obsahují na čipu obvod *SIE*, který se stará o paralelně-sériový převod a kódování dat a také obsahují *USB tranceiver*.

CY7C67300 firmy Cypress a AT43USB370 firmy Atmel jsou USB řadiče pro aplikace s vysokými nároky. Mají spoustu možností, podporují všechny typy přenosů po USB a jsou přizpůsobené pro komunikaci s řídícím 32-bitovým mikroprocesorovým systémem. CY7C67300 může mít datovou sběrnici konfigurovanou na 16 bitů (podporuje 3 různé způsoby připojení na řídící mikropočítač).

CY7C67300 a VNC1L firmy FTDI obsahují na čipu mikrokontrolér, který může zprostředkovat některé komunikační funkce a tím zjednodušit programové vybavení na straně řídícího mikropočítače.

Tab. 26 Srovnání dostupných obvodů typu *HOST*

Typ	Výrobce	USB verze	OTG	Datová sběrnice	SPI
SL811HS	Cypress	1.1	NE	8 bitů	NE
CY7C67300	Cypress	2.0	ANO	16 bitů (HPI)	**ANO**
AT43USB370	Atmel	2.0	NE	32 bitů	NE
MAX3421E	Maxim	2.0	NE	---	**ANO**
ISP1160	Philips	2.0	NE	16 bitů	NE
ISP1362	Philips	2.0	ANO	16 bitů	NE
VNC1L	FTDI	2.0	NE	8 bitů (4 porty)	**ANO**
Typ	**SIE na čipu**	**Funkce Slave**	**Počet USB**	**Nap. napětí**	**Pouzdro**
SL811HS	ANO	ANO	1	3,3V	TQFP-48
CY7C67300	ANO	ANO	2	3,3V	TQFP-100
AT43USB370	ANO	NE	1	1,8 a 3,3V	LQFP-100
MAX3421E	ANO	ANO	1	3,3V	TQFP-32
ISP1160	ANO	NE	2	3,3 nebo 5V	LQFP-64
ISP1362	ANO	ANO	2	3,3V	LQFP-64
VNC1L	ANO	ANO	2	3,3V	LQFP-48

Vysvětlivky:
OTG - *On The Go* (viz dále)
SPI – *Serial Peripheral Interface*, sériový komunikační kanál
SIE – *Serial Interface Engine* (viz dále)

Řadič MAX3421E je zvláštní tím, že jako jediný podporuje komunikaci s řídícím mikropočítačem jen přes rozhraní *SPI*. Některé další obvody *SPI* podporují také, ale umožňují zároveň přenos dat po paralelní sběrnici.

Obvod SL811HS je první řadič typu *HOST* na trhu, který má standardní osmi bitové rozhraní přizpůsobené pro komunikaci s běžnými osmi bitovými mikrokontroléry. Jako jediný s uvedených řadičů nepodporuje nejnovější specifikaci sběrnice verzi 2.0.

Obvody CY7C67300 a ISP1362 podporují nový standard *On The Go*, který umožňuje propojení dvou různých periferních zařízení vzájemně bez nutnosti použití osobního počítače (např. připojení digitálního fotoaparátu přímo na tiskárnu apod.).

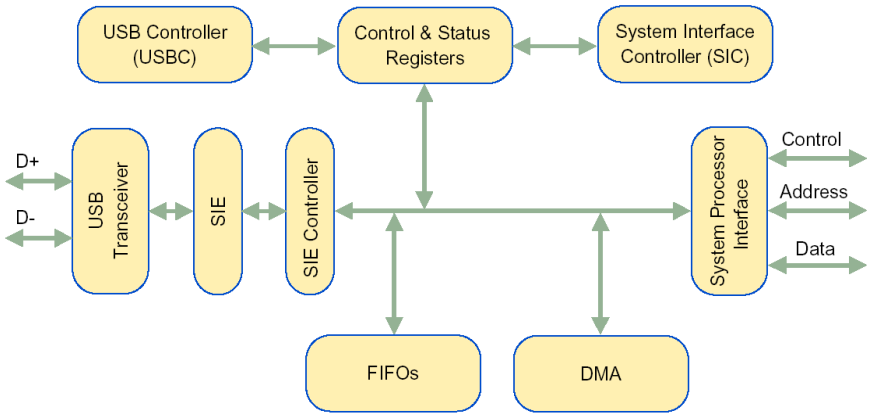
Pro nenáročné aplikace jsou vhodné řadiče SL811HST a MAX3421E. Tyto řadiče splňují veškeré požadavky na komunikaci po sběrnici USB s přenosovou rychlostí *FS*, snadnou programovou obsluhou a úzkou komunikační sběrnici, což jsou hlavní předpoklady pro použití v malých mikroprocesorových systémech.

7.2 Obecná struktura USB řadiče typu „HOST“

Na obr. 21 (převzatý z [19]) je blokové schéma USB řadiče typu *HOST* s jeho základními funkčními bloky. Jednotlivé konkrétní obvody řadičů se liší pouze nepatrně. Prakticky na většinu USB řadiče uvedené v kap. 7.1 lze toto blokové schéma aplikovat. USB

hostitelský řadič má několik bloků provádějících specifickou funkci. U některých jednodušších řadičů můžou být některé bloky sdruženy do jednoho.

Některé řadiče mohou mít samozřejmě i další speciální bloky navíc, umožňující další funkce, např. hlídání a řízení napětí sběrnice. V následujících kapitolách bude stručně pojednáno o jednotlivých blocích na obr. 21.



Obr. 21 Obecná struktura hostitelského USB řadiče

7.2.1 USB Transceiver

(*Transceiver* – přijímač + vysílač) *USB Transceiver* provádí signalizaci na sběrnici USB na úrovni fyzické vrstvy a umožňuje přijímat a vysílat data na sběrnici dle specifikace USB co do úrovní napětí, rozhodovacích úrovní atd. (viz kap 2.1.2), většinou podporuje přenosovou rychlost 1,5Mb/s (*LS*) nebo 12Mb/s (*FS*). Výstup *USB Tranceiveru* vede obvykle přes externí rezistory přímo na USB konektor.

7.2.2 Serial Interface Engine (*SIE*)

Blok *SIE* vykonává následující funkce:

- obnova hodin a dat z příchozího datového toku na sběrnici USB,
- sériově-paralelní převod,
- *NRZI* kódování/dekódování,
- výpočet a kontrola *CRC*,
- generování signalizace pro fyzickou vrstvu pro *Full-speed* i *Low-speed* zařízení,
- detekce připojení a odpojení zařízení na sběrnici,
- vytváří pověřovací pakety (*IN*, *OUT*, *SOF*, *SETUP*),
- provádí vkládání (odstraňování) bitů do datového toku pro zajištění synchronizace.

7.2.3 SIE Controller

Tento blok slouží jako rozhraní mezi *SIE* a blokem *USBC* (*USB Controller*). Dekóduje příkazy přijaté z *USBC* a aktualizuje stav po skončení transakce dat. Tento blok také řídí

paměť *FIFO* (slouží jako buffer pro data) a poskytuje přístup bloku *USBC* do *FIFO* pro interní operace s daty.

7.2.4 USB Controller (*USBC*)

Tento vnitřní mikrokontrolér je určený pro správu USB protokolu v módu *Host* i *Slave*. Kontrolní a stavové registry bývají mapovány do jeho datové paměti pro snadnější a rychlejší přístup. Firmware, běžící v tomto mikrokontroléru, určuje operační mód obvodu (*Host/Slave*).

7.2.5 Systém Interface Controller (*SIC*)

Tento mikrokontrolér slouží jako rozhraní mezi mikrokontrolérem *USBC* a externím procesorem. Jeho firmware spravuje datový tok do a ze systémového procesoru. Také může poskytovat základní USB ovladač určený pro systémovou aplikaci. *SIC* obsahují jen některé řadiče.

7.2.6 FIFO

Blok FIFO paměti slouží jako datový buffer pro vysílaná a přijímaná data na USB sběrnici. V módu *Slave* je většinou velikost bufferu menší než v módu *Host*, protože část paměti zaberou brány periferie – koncové body (*endpoints*).

7.2.7 Control And Status Registers

Tento blok slouží k nastavení obvodu a k řízení jeho činnosti. Mikrokontroléry *USBC* a *SIC* sdílí tyto registry s blokem *System Processor Interface*.

7.2.8 System Processor Interface

Poskytuje většinou paralelní (někdy i sériové - *SPI*) rozhraní mezi *FIFO*, kontrolními a stavovými registry a externím procesorem. Mimo datové sběrnice používá pro řízení přenosu dat další řídící vodiče.

7.2.9 DMA

Blok *DMA* (*Direct Memory Access*) poskytuje podporu *DMA* pro systémový procesor k přenosu dat mezi jeho datovou pamětí a pamětí *FIFO* řadiče USB. Řadič *DMA* systémového procesoru řídí *DMA* operace pomocí standardních *DMA* žádostí. Ne všechny řadiče DMA podporují.

7.3 Obsluha USB řadičů

USB řadiče jsou navrhovány tak, aby jejich programová obsluha byla co nejjednodušší. Proto jsou zcela optimalizovány dle požadavků USB specifikace tak, aby co nejvíce operací prováděl řadič automaticky, což vede k výraznému zjednodušení ovládacího programu.

Všechny řadiče uvedené v kap. 7.1 kompletně zprostředkovávají vše potřebné pro realizaci fyzické vrstvy USB hostitele a zprostředkovávají také velkou část linkové vrstvy. Provádějí všechny ověření chybových součtů paketů i jejich výpočty při odesílání nových paketů, starají se o *time-out* při odesílání a příjmu paketů, reagují na speciální stavy na sběrnici USB, dokáží generovat USB reset sběrnice.

Řadiče jsou přizpůsobeny i pro vysokorychlostní transakce, kdy při odesílání jednoho paketu, může řídící mikrokontrolér připravovat USB řadič na další transakci.

Všechny pakety vytváří samotný řadič, řízením se pouze nastavuje, který paket má být sestaven a odeslán. Řadiče automaticky na dané události odesílají potvrzovací pakety (*ACK*, v případě periferie i *NAK* a *STALL*) a také je automaticky očekávají a přijímají. Při odesílání paketů *SETUP* nebo *OUT*, za kterými musí následovat odeslání datového paketu *DATA0/1*, bude datový paket odeslán opět automaticky (musí být předem připravený v paměti řadiče). Stejně tak při odeslání paketu *IN*, bude automaticky přijat datový paket *DATA0/1* ze zařízení. Poté řadič (např. přerušením) informuje řídící mikrokontrolér o nové události na sběrnici, kdy lze ze stavových registrů řadiče vyčíst, zda byl úspěšně přijat datový paket, zda-li zařízení odeslalo *NAK* (nebo *STALL*), či zařízení nereagovalo a nastal *time-out*.

Řídící mikrokontrolér tedy pouze do paměti řadiče ukládá data, která mají být přenesena v datovém paketu. Při odesílání datových paketů ještě určí zda má být odeslán lichý nebo sudý datový paket a jeho velikost. Některé řadiče si řeší střídání lichých a sudých datových paketů automaticky. Poté nastaví USB řadič na odeslání paketu *SETUP* nebo *OUT*. Řadič tento pověřovací paket odešle a automaticky vytvoří příslušný datový paket, který odešle bezprostředně poté a očekává příjem potvrzovacího paketu ze zařízení. Pro příjem datového paketu odešle řadič pověřovací paket *IN* a data z příchozího datového paketu řadič uloží na předem nastavené místo do jeho paměti. V příslušném registru poskytne informaci o tom, zda se jedná o lichý nebo sudý datový paket.

Celá USB komunikace pomocí USB hostitelského řadiče se tedy z hlediska řídícího programu skládá z odesílání paketů *SETUP*, *OUT* a *IN* (v případě *LS* nebo *FS* komunikace). Všechny ostatní pakety (potvrzovací a datové) jsou odesílány nebo přijímány automaticky. Řadič pouze podává informaci o tom, jaký potvrzovací, či datový paket byl v dané transakci přijat. Řídící program hostitele musí pomocí těchto jednoduchých transakcí paketů realizovat odpovídající typ přenosu (řídící, hromadný, viz kap. 2.2.2) včetně případného opakování paketů při chybách dle kap. 2.2.5. Samotný typ přenosu (např. hromadný) dokáže ošetřit chyby vzniklé na fyzickém přenosovém médiu např. vlivem rušení apod., kdy může být ztracen potvrzovací paket nebo v přenesených datech může být chyba (ošetřeno kontrolními součty). Chyby vzniklé ve vyšších vrstvách musí odstraňovat např. použitý transportní protokol.

8 USB řadič MAX3421E

MAX3421E je USB periferní a hostitelský řadič obsahující digitální logiku a analogové obvody nezbytné k implementaci *FS* USB periferie a *LS*/*FS* USB hostitele odpovídající specifikaci USB verze 2.0. Vestavěný tranceiver má ESD ochranu do ±15kV. Řadič je řízen pomocí interních registrů, k nimiž se přistupuje přes sběrnici *SPI* s maximálním hodinovým kmitočtem 26MHz. Řídící mikrokontrolér může použít tří nebo čtyř vodičové připojení ke sběrnici *SPI*. Řadič má napájení 3,0 – 3,6V, přičemž interface *SPI* může být napájen napětím v rozmezí 1,4 – 3,6V. MAX3421E obsahuje také 8 univerzálních vstupů a 8 výstupů ovládané přes *SPI* interface a umožňují tak rozšířit možnosti USB zařízení bez použití dalších signálů z řídícího mikrokontroléru. Obvod může pracovat v rozmezí teplot –40 až +85°C a je dostupný v pouzdrech TQFP 32 a TQFN 32. V této kapitole bude stručně popsán tento řadič a to pouze z hlediska funkce hostitele USB. Detailní popis je dostupný v [4] a [5].

8.1 Základní rysy

Obecné vlastnosti:

- Obvod je řešen bez použití vnitřního mikroprocesoru.
- Programově kompatibilní s obvodem MAX3420E – USB periferní řadič s SPI rozhraním.
- Může pracovat jako *Full-speed* USB periferie nebo *Full/Low-speed* USB hostitel.
- Integrovaný USB transceiver.
- Softwarově ovladatelné *Pull-up* a *Pull-down* rezistory na signálech sběrnice *D+* a *D-*.
- Programovatelné 3 nebo 4 vodičové *SPI* rozhraní, 26MHz.
- Umožňuje nezávisle napájení rozhraní *SPI* na napájením *SIE*.
- Interní komparátor pro detekování napětí *VBUS* pro *Self-powered* periferie.
- Ochrana ESD na signálech *D+* a *D-* a *VBCOM*.
- Výstupní pin *INT* signalizující vnitřní události, umožňující vyvolat přerušení řídícího systému. Je programovatelný na úroveň nebo na indikaci hranami obou polarit.
- Osm univerzálních vstupů a výstupů.
- Výstupní pin *GPX* umožňující indikovat změnu na univerzálních vstupech obvodu.
- Inteligentní USB *Serial Interface Engine* (*SIE*)
- Automatické řízení toku dat a režim provozu se zdvojenými přenosovými zásobníky.

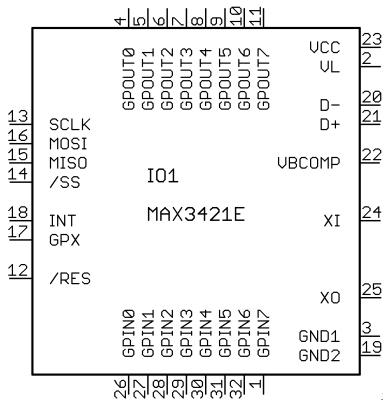
Vlastnosti pro funkci hostitele:

- V režimu hostitele je dostupných 23 řídících registrů.
- Přenosové zásobníky jsou typu *FIFO*
  SNDFIFO: vysílací paměť – zdvojená o kapacitě 64B
  RCVFIFO: přijímací paměť – zdvojená o kapacitě 64B
- Řízení střídání paketů *DATA0*/*DATA1* a kontrola sekvence.
- Chybová kontrola všech přenosů.
- Automatická generace paketů *SOF* v intervalu 1ms.
- Automaticky synchronizuje přenos paketů se začátkem rámce (*SOF*/*EOP*).

- Podává hlášení o výsledku přenosů paketů.
- Podporuje připojení USB hubů.
- Podporuje isochronní přenos.

8.1.1 Schématická značka a vývody

Na obr. 22 je uvedená schématická značka obvodu MAX3421E. Je využito všech 32 vývodů pouzdra TQFP popř. TQFN.



Obr. 22 Schématická značka obvodu MAX3421E

VCC - Napájení USB transceiveru a logického jádra obvodu napětím 3,0 – 3,6V. Je doporučeno zapojit mezi *VCC* a zem keramický kondenzátor 1μF.

VL - Napájení převodníku napětí rozhraní obvodu napětím 1,4 – 3,6V. Toto napětí bude odpovídat na rozhraní *SPI*, *GPOUT* a *GPIN* pinech logické jedničce. Je rovněž doporučeno zapojit na zem keramický kondenzátor 1μF.

VBCOMP - Vstup napěťového komparátoru, pomocí kterého lze určit přítomnost napětí *VBUS* na sběrnici USB pro aplikace *self-powered* periferie.

D+, *D-* - Rozdílové datové signály sběrnice USB.

XI, *XO* - Oscilátorový obvod interního oscilátoru. Nutno zapojit krystal o jmenovitém kmitočtu 12MHz a kondenzátory na zem o kapacitě maximálně 18pF. Lze rovněž do vstupu *XI* zavést hodinový signál se střídou 0,5 z externího zdroje.

GND1, *GND2* - Zemní vývody obvodu.

SCLK, *MOSI*, *MISO*, */SS* - Standardní komunikační rozhraní *SPI*.

INT -Výstupní signál, informující o vnitřních procesech obvodu, určený k vyvolání přerušení v řídícím systému. Je nastavitelný na úroveň (pak je signalizace prováděna úrovní log. 0) nebo na hrany obou polarit.

GPX - Pin indikující některé vnitřní stavy obvodu nebo může být párovým pinem přerušení k pinu *INT*.

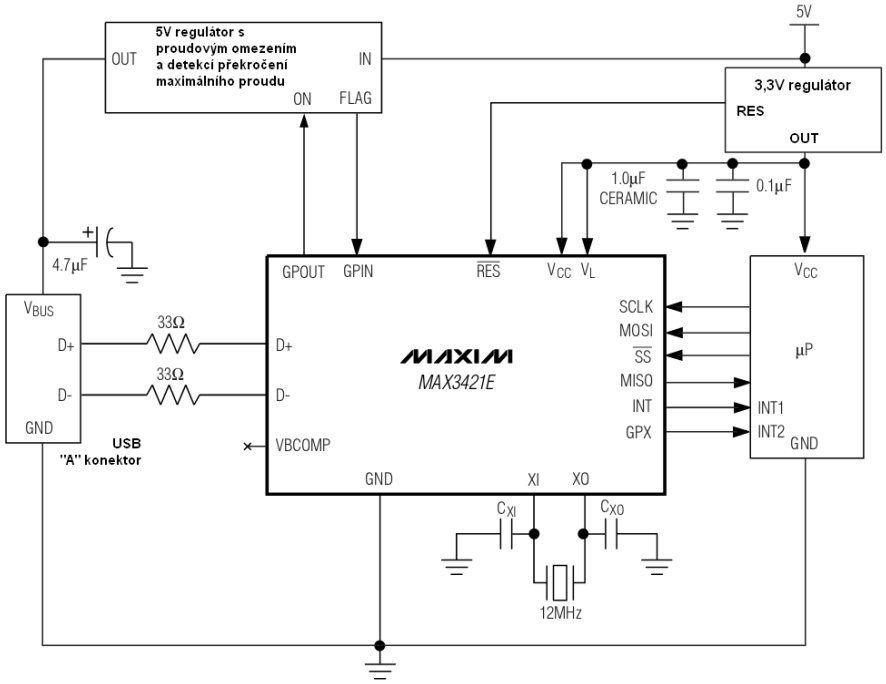
/RES - Resetovací vstup obvodu. Obvod je možné resetovat rovněž přes rozhraní *SPI* se stejným účinkem.

GPIN0-7 - Univerzální vstupy. Mohou být konfigurované tak, že jejich změna vyvolá signál přerušení na pinu *GPX*.

GPOUT0-7 - Univerzální výstupy ovladatelné přes rozhraní *SPI* obvodu.

8.1.2 Doporučené zapojení obvodu v módu hostitele

Na obr. 23 (převzatý z [4]) je výrobcem doporučené zapojení obvodu pro režim hostitele. Signály sběrnice D+ a D- jsou vedené na USB konektor přes rezistory 33Ω ±1%. Rezistory 15kΩ, které musejí být u hostitele zapojeny k zemi jsou integrované uvnitř obvodu a elektronicky připojitelné. Je zde použit lineární regulátor 5V pro napájení sběrnice, který dává informaci o překročení proudu a přes vstupní *GPIN* může vyvolat přerušovací signál na výstupu *GPX*. Výstupem *GPOUT* je možné přes rozhranní *SPI* napájení sběrnice zapínat a vypínat. Regulátor 3,3V pro napájení obvodu má výstupní signál pro resetování řadiče USB a napájí jak jádro řadiče (*VCC*) tak rozhraní (*VL*). S řídícím mikrokontrolérem řadič komunikuje přes rozhraní *SPI* a můžou být využity oba přerušovací signály *INT* a *GPX*.



Obr. 23 Doporučené zapojení obvodu

8.1.3 Přístup k registrům a přenosovým zásobníkům

Řídící mikrokontrolér přistupuje k obvodu pouze přes rozhraní *SPI*. V režimu hostitele je dostupných 23 řídících registrů, pomocí kterých je obvod ovládán.

Každá relace začíná nastavením signálu /SS na log. 0 a končí nastavením na log. 1. První bajt poslaný do řadiče je příkazový a jeho struktura je znázorněna v tab. 27. První bit *ACKSTAT* je relevantní pouze pro režim periferie, v módu hostitele je nastaven na hodnotu 0. Bit *DIR* určuje, zda se bude do řadiče zapisovat nebo číst (*DIR* = 1 pro zápis, *DIR* = 0 pro čtení). Pole *Reg* slouží pro zadání adresy požadovaného registru. Následující bajt jsou pak vlastní přenášená data.

Tab. 27 Struktura příkazového bajtu SPI

b7	b6	b5	b4	b3	b2	b1	b0
Reg4	Reg3	Reg2	Reg1	Reg0	0	DIR	ACKSTAT

Lze také číst nebo zapisovat do několika registrů řadiče během jediné relace. V obvodu funguje automatická inkrementace adresy registru. V příkazovém bajtu se uvede první registr, ke kterému je potřeba přistoupit a po jeho zapsání je automaticky inkrementována vnitřní adresa. Další bajt bude zapsán do následujícího registru atd. Vnitřní inkrementace adresy funguje od registru R13 po R20, Pak se zastaví na registru R20. Po adresování registru R21 (a vyšším) opět obvod automaticky inkrementuje adresu až po registr R31. Registry R1, R2 a R4 jsou přístupy do pamětí FIFO obvodu. Opakovaným zápisem nebo čtením z těchto registrů (i během jediné relace) se plní a vyčítá příslušná paměť.

MAX3421E podporuje dva režimy sběrnice SPI – *half-duplex* a *full-duplex*. V obou režimech podporuje konfiguraci *SPI(0, 0)* nebo *SPI(1, 1)*, tedy buď je *SCLK* v klidu na log.0 a aktivní hrana je první (0, 0), nebo je *SCLK* v klidu na log.1 a aktivní hrana je druhá (1, 1).

V režimu *half-duplex* je využit pouze datový signál *MOSI*, *MISO* je nevyužit. Pokud je v příkazovém bajtu požadováno čtení, po příjmu příkazu je *MOSI* přepnut na výstupní signál a čtená data jsou přes něj vysouvána do řídícího obvodu. V režimu *full-duplex* jsou standardně využity oba datové vodiče *MISO* a *MOSI*. Výchozí stav nastavení sběrnice po resetu MAX3421E je *half-duplex*, a proto pokud řídící obvod tento režim nepodporuje, je třeba před prvním čtením dat z řadiče jej přepnout na standardní *full-duplex*.

8.2 Registrová sada MAX3421E pro režim hostitele

V režimu hostitele je dostupných 23 řídících registrů jejichž seznam a názvy jednotlivých bitů jsou znázorněny v tab. 28. Po resetu je MAX3421E přepnut do režimu periferie z důvod programové kompatibility s obvodem MAX3420E. Po resetu řadiče je nutné jej nejdříve přepnout do režimu hostitele nastavením bitu *HOST* v registru *MODE*. Teprve poté bude význam registrů zcela odpovídat tab. 28. Dále budou popsány ty registry a jejich bity, které jsou potřeba pro základní ovládání řadiče, podrobné vysvětlení funkce je v [5].

Registr *RCVFIFO* je přístup do přijímací paměti *FIFO* řadiče. Po příjmu paketu pak pomocí opakovaného čtení regisru *RCVFIFO* lze vyčíst přijatá data. Podobně registr *SNDFIFO* představuje přístup do vysílací paměti *FIFO* řadiče. Opakovaným zápisem do registru *SNDFIFO* lze naplnit vysílací paměť požadovanými daty. Registr *SUDFIFO* slouží jako přístup do paměti řídícího požadavku. Zápisem osmi bajtů požadavku do registru

SUDFIFO jsou předána řadiči data řídícího požadavku pro transakci *SETUP*. V registru *RCVBC* je po příjmu datového paketu v transakci *IN* uložen počet přijatých bajtů. Tento počet bajtů je pak možné vyčíst z přijímací paměti *FIFO*. Naopak registr *SNDBC* určuje, kolik bajtů bylo zapsáno do vysílací paměti *FIFO* pro transakci *OUT*. Tento počet nastavuje po naplnění vysílací *FIFO* řídící mikrokontrolér a jeho zápisem je vynulován příznakový bit přerušení *SNDBAVIRQ*, čímž se indikuje připravenost vysílací *FIFO* pro přenos.

Tab. 28 Seznam registrů řadiče MAX3421E v režimu hostitele

Reg.	Jméno	b7	b6	b5	b4	b3	b2	b1	b0	Přístup
R0	-	0	0	0	0	0	0	0	0	-
R1	**RCVFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R2	**SNDFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R3	-	0	0	0	0	0	0	0	0	-
R4	**SUDFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R5	-	0	0	0	0	0	0	0	0	-
R6	**RCVBC**	0	BC6	BC5	BC4	BC3	BC2	BC1	BC0	RSC
R7	**SNDBC**	0	BC6	BC5	BC4	BC3	BC2	BC1	BC0	RSC
R8	-	0	0	0	0	0	0	0	0	-
R9	-	0	0	0	0	0	0	0	0	-
R10	-	0	0	0	0	0	0	0	0	-
R11	-	0	0	0	0	0	0	0	0	-
R12	-	0	0	0	0	0	0	0	0	-
R13	**USBIRQ**	0	VBUSIRQ	NOVBUSIRQ	0	0	0	0	OSCOKIRQ	RC
R14	**USBIEN**	0	VBUSIE	NOVBUSIE	0	0	0	0	OSCOKIE	RSC
R15	**USBCTL**	0	0	CHIPRES	PWRDOWN	0	0	0	0	RSC
R16	**CPUCTL**	PULSEWID1	PULSEWID0	0	0	0	0	0	IE	RSC
R17	**PINCTL**	0	0	0	FDUPSPI	INTLEVEL	POSINT	GPXB	GPXA	RSC
R18	**REVISION**	b7	b6	b5	b4	b3	b2	b1	b0	R
R19	-	0	0	0	0	0	0	0	0	-
R20	**IOPINS1**	GPIN3	GPIN2	GPIN1	GPIN0	GPOUT3	GPOUT2	GPOUT1	GPOUT0	RSC
R21	**IOPINS2**	GPIN7	GPIN6	GPIN5	GPIN4	GPOUT7	GPOUT6	GPOUT5	GPOUT4	RSC
R22	**GPINIRQ**	GPINIRQ7	GPINIRQ6	GPINIRQ5	GPINIRQ4	GPINIRQ3	GPINIRQ2	GPINIRQ1	GPINIRQ0	RC
R23	**GPINIEN**	GPINIEN7	GPINIEN6	GPINIEN5	GPINIEN4	GPINIEN3	GPINIEN2	GPINIEN1	GPINIEN0	RSC
R24	**GPINPOL**	GPINPOL7	GPINPOL6	GPINPOL5	GPINPOL4	GPINPOL3	GPINPOL2	GPINPOL1	GPINPOL0	RSC
R25	**HIRQ**	HXFRDNIRQ	FRAMEIRQ	CONDETIRQ	SUSDNIRQ	SNDBAVIRQ	RCVDAVIRQ	RWUIRQ	BUSEVENTIRQ	RC
R26	**HIEN**	HXFRDNIE	FRAMEIE	CONDETIE	SUSDNIE	SNDBAVIE	RCVDAVIE	RWUIE	BUSEVENTIE	RSC
R27	**MODE**	DPPULLDN	DMPULLDN	DELAYISO	SEPIRQ	SOFKAENAB	HUBPRE	LOWSPEED	HOST	RSC
R28	**PERADDR**	0	b6	b5	b4	b3	b2	b1	b0	RSC
R29	**HCTL**	SNDTOG1	SNDTOG0	RCVTOG1	RCVTOG0	SIGRSM	SAMPLEBUS	FRMRST	BUSRST	LS
R30	**HXFR**	HS	ISO	OUTNIN	SETUP	EP3	EP2	EP1	EP0	LS
R31	**HRSL**	JSTATUS	KSTATUS	SNDTOGRD	RCVTOGRD	HRSLT3	HRSLT2	HRSLT1	HRSLT0	R

R – lze číst, S – lze nastavit, C – lze smazat, LS – registr citlivý na zápis (load sensitive) - zápisem do registru se spouští nějaká vnitřní operace.

Při resetu řadiče je vypnut interní oscilátor 12MHz. Příznakový bit *OSCOKIRQ* v registru *USBIRQ* je nastaven po resetu řadiče na 1 v okamžiku připravenosti interního oscilátoru. Příznakový bit *OSCOKIRQ* se maže zapsáním hodnoty 1.

Nastavením bitu *CHIPRES* na 1 v registru *USBCTL* je vyvolán reset řadiče, jenž má stejný účinek jako nastavení pinu obvodu */RES* na log. 0. Stav reset je zrušen nastavením bitu

CHIPRES na hodnotu 0. V registru *CPUCTL* je důležitý bit *IE*. Nastavením *IE* na hodnotu 1 je povolen výstupní signál *INT*.

Nastavením bitu *FDUPSPI* v registru *PINCTL* na hodnotu 1 je zapnut režim sběrnice *SPI full-duplex*. Nastavením zpět na hodnotu 0 je aktivován režim *half-duplex*. Pokud je bit *INTLEVEL* nastaven na hodnotu 0, výstupní signál *INT* je nastaven na indikaci hranami. V tomto režimu je každá povolená událost řadiče zvlášť indikována jednou hranou signálu *INT*. Polaritu této hrany je možné nastavit bitem *POSINT*. Je-li *POSINT* nastaven na hodnotu 0, *INT* je v klidovém stavu nastaven na log. 1 a události jsou indikovány sestupnými hranami. Je-li *POSINT* nastaven na hodnotu 1, chování je opačné. Je-li však *INTLEVEL* nastaven na hodnotu 1, indikace signálem *INT* je prováděna úrovní log. 0. Signál *INT* pak zůstává v log. 0 dokud nejsou vynulovány všechny příznaky povolených přerušení. V tomto režimu je pin *INT* konfigurován jako otevřený kolektor a je nutné připojit externí *pull-up* rezistor. Nastavení bitu *POSINT* nemá v tomto režimu vliv.

V registru *HIRQ* jsou příznaky vnitřních pochodů řadiče a rovněž fungují jako spouštěcí příznaky povolených přerušení registrem *HIEN*. Příznak *HXFRDNIRQ* indikuje dokončení přenosu. Příznak *FRAMEIRQ* je nastaven pokaždé když je vyslán paket *SOF* na začátku USB rámce. Příznak *CONDETIRQ* je nastaven po připojení nebo odpojení zařízení na sběrnici USB. Když je tento příznak nastaven, jsou současně aktualizovány bity *JSTATUS* a *KSTATUS* v registru *HRSL*, podle kterých lze pak určit, zda bylo připojeno zařízení *LS* nebo *FS*. Příznak *RCVDAVIRQ* je nastaven, pokud přijímací paměť *FIFO* obsahuje přijatá data. Tento příznak musí po přenosu *IN* vynulovat řídící mikrokontrolér (na rozdíl od příznaku *SNDBAVIRQ*, který je vynulován zápisem do registru *SNDBC*). Příznaky registru *HIRQ* se nulují zápisem hodnoty 1 do příslušných bitů.

Registrem *HIEN* se nastavuje, zda jednotlivé příznaky v registrech *HIRQ* budou vyvolávat přerušovací signál na pinu *INT* obvodu. Nastavením na hodnotu 1 je přerušení povoleno.

V registru *MODE* se nastavením bitu *HOST* na hodnotu 1 přepne řadič USB do režimu hostitele. Při vývoji řídícího softwaru bylo zjištěno, že toto přepnutí funguje přibližně až po 3ms po zrušení stavu reset řadiče. Bitem *LOWSPEED* je řadič přepnut do režimu komunikace se zařízením *LS*. Bit *HUBPRE* nastaví řídící mikrokontrolér na hodnotu 1 když bude komunikovat se zařízením *LS* přes USB hub, kdy řadič vyšle před transakcí paket *PRE*. Bit *SOFKAENAB* zapne automatické generování paketu *SOF* v intervalu 1ms jako začátek USB rámce. Nastavením bitu *DELAYISO* na hodnotu 1 budou vysílány isochronní pakety hned po příštím začátku USB rámce. Nastavením bitů *DPPULLDN* a *DMPULLDN* na hodnotu 1 jsou elektronicky připojeny vnitřní rezistory 15kΩ mezi linky *D+*, *D-* a *GND*.

Do registru *PERADDR* uloží řídící mikrokontrolér adresu cílového zařízení v rámci sběrnice USB.

Registr *HCTL* je *load-sensitive*. Zápisem hodnoty 1 do bitu *BUSRST* je vyvolán reset sběrnice USB (jednoznačná 0 po dobu 50ms). Po ukončení resetu je automaticky vynulován bit *BUSRST* (lze ho testovat během resetu) a je nastaven příznakový bit *BUSEVENTIRQ*. Nastavením bitu *FRMRST* na hodnotu 1 dojde k vynulování čítače USB rámců. Nastavením bitu *SAMPLEBUS* na hodnotu 1 je sejmut vzorek aktuálního stavu sběrnice a podle něj jsou aktualizovány bity *JSTATUS* a *KSTATUS*. Párové bity *RCVTOG0/1* a *SNDTOG0/1* slouží k nastavení řadiče podle toho, jaký datový paket bude příště vyslán (*DATA0/1*) nebo jaký je

očekáván (*DATA0/1*). Z těchto párů musí být vždy nastaven pouze jeden. Pokud budou nastaveny oba nebo pokud budou oba nulové, jejich nastavení nemá vliv na činnost řadiče. Řadič si tyto hodnoty uloží do vnitřních *data-toggle* bitů a ty si během přenosu automaticky dále střídá. Pokud pak bude potřeba komunikovat s jinou bránou nebo jiným zařízením, aktuální stav vysílacích a přijímacích *data-toggle* bitů je dostupný v registru *HRSL* v bitech *SNDTOGRD* a *RCVTOGRD*. Tyto hodnoty si řídící mikrokontrolér zálohuje a při příští komunikaci se stejnou bránou podle nich nastaví párové bity v registru *HCTL*.

Registr *HXFR* je rovněž *load-sensitive*. Zápisem do tohoto registru je zahájena transakce. Do pole *EP0-3* je zapsáno číslo cílové brány a podle bitů *SETUP*, *OUTNIN*, *ISO* a *HS* je proveden patřičný přenos podle tab. 29.

Tab. 29 Nastavení registru HXFR pro různé typy přenosů

Typ přenosu	HS	ISO	OUTNIN	SETUP	Hodnota hex
SETUP	0	0	0	1	10
BULK-IN	0	0	0	0	0-ep
BULK-OUT	0	0	1	0	2-ep
HS-IN	1	0	0	0	8-ep
HS-OUT	1	0	1	0	A-ep
ISO-IN	0	1	0	0	4-ep
ISO-OUT	0	1	1	0	6-ep

Transakce *SETUP* vyšle paket *SETUP* s požadavkem (paket *DATA0*) na bránu 0, transakce *BULK-IN* představuje přenos *IN* (vyšle paket *IN* a čeká na příjem paketu *DATA0/1*), transakce *BULK-OUT* představuje přenos *OUT* (vyšle paket *OUT* a následně paket *DATA0/1*). Obě transakce *BULK* musí adresovat bránu s hromadným nebo časovaným typem přenosu (včetně brány 0). Transakce *HS-IN* a *HS-OUT* (*handshake*) slouží pro usnadnění programování řídícího přenosu. Tvoří stavovou fázi řídícího přenosu – řadič vyšle paket *OUT* nebo *IN* na bránu 0 a odešle popř. přijme datový paket *DATA1* s nulovou délkou dat. Transakce *ISO-IN* a *ISO-OUT* slouží pro transakce isochronním přenosem.

Registr *HRSL* je pouze pro čtení a poskytuje důležité informace. Obsahuje bity *JSTATUS*, *KSTATUS*, *RCVTOGRD* a *SNDTOGRD*, které byly zmíněny dříve. Mimo ně obsahuje tento registr pole *HRSLT0-3*, které podává kód o výsledku poslední transakce. Možné kódy jsou uvedeny v tab. 30.

Tab. 30 Kódy HRSLT výsledku transakce

HRSLT	Název	Význam
0x00	hrSUCCESS	Transfer úspěšný.
0x01	hrBUSY	SIE je zaneprázdněn, transfer byl zrušen.
0x02	hrBADREQ	Neplatná hodnota v HXFR registru.
0x03	hrUNDEF	rezervováno
0x04	hrNAK	Periferie vrátila paket NAK.
0x05	hrSTALL	Periferie vrátila paket STALL.
0x06	hrTOGERR	Chyba v sekvenci DATA0/1
0x07	hrWRONGPID	Přijatý špatný PID.
0x08	hrBADBC	Špatný počet bajtů.
0x09	hrPIDERR	Chybně přijat PID.
0x0A	hrPKTERR	Chyba v přijatém paketu (EOP).
0x0B	hrCRCERR	Chyba CRC.
0x0C	hrKERR	Místo odpovědi stav K.
0x0D	hrJERR	Místo odpovědi stav J.
0x0E	hrTIMEOUT	Zařízení neodpovědělo v požadovaném čase.
0x0F	hrBABBLE	Zařízení vysílalo příliš dlouho.

8.3 Programování přenosů hostitele

Pro každý vysílaný paket musí být správně nastaveno:

- adresa zařízení v registru *PERADDR*,
- číslo brány v registru *HXFR*,
- požadovaná transakce v registru *HXFR*,
- data: *OUT* data v *SNDFIFO* (v registru *SNDBC* počet bajtů), *IN* přijatá data v *RCVFIFO* (*RCVBC* bajtů).

Adresa zařízení a data ve *FIFO* zůstávají nezměněny dokud je nezmění řídící mikrokontrolér. To znamená, že např. při transakci *OUT* a chybě přenosu není třeba znovu ukládat data do *FIFO*, stačí znovu inicializovat přenos v *HXFR* registru.

Výsledek každého přenosu je indikován v poli *HRSLT*, který indikuje 16 různých možných výsledků. Hodnota v *HRSLT* je aktualizována až po nastavení příznakového bitu *HXFRDNIRQ*. Pro aplikace, které nepoužijí přerušení *HXFRDNIRQ*, může být pro použití detekce ukončení transferu použito neustále testování kódu *hrBUSY* pole *HRSLT*.

8.3.1 Programování BULK-IN přenosu

Řídící mikrokontrolér zapíše do registru *HXFR* hodnotu 0000eeee(b) k zahájení přenosu *IN*. *SIE* řadiče odešle paket *IN* do zařízení s adresou *PERADDR* na bránu *EP[3:0]*. Na odpověď čeká 6,5 bitových period na sběrnici USB. Jestli-že zařízení odpoví odesláním paketu *DATA0/1*, *SIE* uloží data z paketu do přijímací paměti *RCVFIFO* a určí jejich počet. *SIE* zkontroluje správnost paketu a počet přijatých bajtů uloží do registru *RCVBC*, nastaví kód výsledku přenosu *HRSLT* a nastaví příznakový bit konce transakce *HXFRDNIRQ*. V závislosti na výsledku přenosu *SIE* nastaví nebo nenastaví bit *RCVDAVIRQ*, který indikuje platná data v *RCVFIFO*.

Pokud jsou přijatá data bezchybná (*HRSLT* = 0x00), *SIE* odešle do zařízení potvrzovací paket *ACK*, neguje svůj vnitřní *data-toggle* bit a nastaví příznak *RCVDAVIRQ*, pro indikaci, že jsou dostupná platná data v *RCVFIFO*.

Pokud jsou přijatá data bezchybná, ale byl přijat paket nesprávné sekvence (sudý místo lichého nebo naopak), *SIE* rovněž odešle potvrzovací paket *ACK*, ale neneguje svůj vnitřní *data-toggle* bit a také nenastaví příznak platnosti dat *RCVDAVIRQ*. *SIE* pak nastaví jako výsledek přenosu *HRSLT = hrTOGERR*.

Pokud je v přijatých datech chyba, *SIE* neodešle potvrzovací paket *ACK*, neneguje vnitřní *data-toggle* bit a rovněž nenastaví *RCVDAVIRQ*.

Podle výsledku přenosu v *HRSLT* provede řídící mikrokontrolér patřičné kroky. Je-li přenos úspěšný, přečte z registru *RCVBC* počet přijatých bajtů a vyčte je z paměti *RCVFIFO*. Poté vynuluje bit *RCVDAVIRQ* (zapsáním hodnoty 1).

8.3.2 Programování BULK-OUT přenosu

Řídící mikrokontrolér nejdříve zkontroluje zda je *SNDBAVIRQ* nastaven na hodnotu 1, což indikuje že je vysílací paměť připravena pro zápis dat. Je-li *FIFO* dostupná, zapíše do ní až 64 bajtů dat opakovaným zápisem do registru *SNDFIFO*. Počet vysílaných dat poté uloží do registru *SNDBC*, čímž se automaticky vynuluje bit *SNDBAVIRQ*, který indikuje, že jsou data paketu připravena ve *FIFO*.

Pokud řídící mikrokontrolér odesílá paket na stejnou bránu jako v minulé transakci *OUT*, nemusí nastavovat *SNDTOG0* nebo *SNDTOG1* v registru *HCTL* pro určení, zda se má vysílat paket *DATA0* nebo *DATA1*. Pokud však odesílá paket pro jinou bránu, musí aktualizovat nastavení *SNDTOG0/1* ze zálohy z konce posledního transferu na tuto bránu.

Pokud je nastaven počet odesílaných bajtů *SNDBC* = 0, bude odeslán příslušný paket *DATA0/1* bez dat.

K zahájení přenosu zapíše mikrokontrolér do registru *HXFR* hodnotu 0010eeee(b), kde eeee je číslo cílové brány. *SIE* odešle paket *OUT* na zařízení s adresou *PERADDR* a ten je okamžitě následován datovým paketem *DATA0/1* s délkou dat odpovídající hodnotě v registru *SNDBC*. *SIE* poté čeká na odpověď 6,5 bitových period na sběrnici USB.

Pokud *SIE* přijme ze zařízení potvrzovací paket nebo nastane *time-out*, nastaví příznakový bit konce transakce *HXFRDNIRQ* na hodnotu 1 a aktualizuje výsledek přenosu *HRSLT*. Pokud přijme potvrzovací paket *ACK*, neguje vnitřní *data-toggle* bit přenosu *OUT*, indikuje v *HRSLT* úspěšný přenos kódem *hrSUCCESS* a nastaví bit *SNDBAVIRQ* na hodnotu 1 pro indikaci volné vysílací paměti.

Pokud *SIE* nepřijme ze zařízení potvrzovací paket *ACK*, přenos bude muset být zřejmě zopakován. Pokud přijme paket *NAK*, stačí znovu inicializovat přenos zapsáním do registru *HXFR* = 0010eeee(b). *SIE* použije stejnou adresu v *PERADDR*, stejnou hodnotu počtu vysílaných bajtů v registru *SNDBC* a pošle stejná data z vysílací *FIFO*.

8.3.3 Programování SETUP přenosu

1. Přenos požadavku

Řídící mikrokontrolér zapíše 8 bajtů řídícího požadavku do paměti *SUDFIFO*. K této paměti není přidružen žádný registr počtu bajtů, protože řídící požadavek musí mít vždy přesně 8 bajtů. Poté zahájí přenos zapsáním hodnoty 00010000(b) do registru *HXFR*. *SIE* odešle do zařízení s adresou *PERADDR* na bránu 0 paket *SETUP* následován paketem *DATA0* s osmi bajty řídícího požadavku z paměti *SUDFIFO*.

SIE poté čeká 18 bitových period na sběrnici USB pro odpověď ze zařízení. Pak nastaví příznakový bit konce přenosu *HXFRDNIRQ* na hodnotu 1. Pokud zařízení neodpoví, přenos končí s výsledkem *hrTIMEOUT*. Pokud zařízení vrátí potvrzovací paket *ACK*, přenos končí s výsledkem *hrSUCCESS*.

2. Datová fáze přenosu

Pokud je požadována datová fáze přenosu, je programována pomocí *BULK-IN* a *BULK-OUT* přenosů. Některé požadavky jako např. *SET_ADDRESS* datovou fázi nemají, protože veškeré potřebné informace pro zařízení jsou obsaženy v osmi bajtech řídícího požadavku.

3. Stavová fáze přenosu

Stavová fáze přenosu obsahuje paket *DATA1* s nulovou délkou dat a opačného směru (*OUT*/*IN*) než byl přenos v předchozí datové fázi. Pro stavovou fázi lze tedy opět použít přenosů *BULK-IN* a *BULK-OUT*. Pro usnadnění však MAX3421E poskytuje speciální *HS-OUT* a *HS-IN* transakce, které plní úlohu stavové fáze řídícího přenosu.

HS-OUT:

Hostitel posílá ve stavové fázi přenosu paket *OUT* v řídícím přenosu *IN* (čtení ze zařízení). K odeslání paketu *OUT* s paketem *DATA1* bez dat na bránu 0 stačí zapsat do registru *HXFR* hodnotu 0xA0. Pakety budou odeslány na zařízení s adresou *PERADDR*.

SIE poté čeká na odpověď 6,5 bitových period na sběrnici USB, neodpoví-li do této doby, nastane výsledek přenosu *hrTIMEOUT*. *SIE* nastaví příznakový bit konce přenosu *HXFRDNIRQ* na hodnotu 1 a aktualizuje kód výsledku přenosu *HRSLT*.

HS-IN:

Hostitel posílá ve stavové fázi přenosu paket *IN* v řídícím přenosu *OUT* (čtení ze zařízení nebo přenos bez datové fáze). K odeslání paketu *IN* na bránu 0 stačí zapsat do registru *HXFR* hodnotu 0x80. Pakety budou odeslány na zařízení s adresou *PERADDR*.

SIE poté čeká na odpověď 6,5 bitových period na sběrnici USB. Pak nastaví příznakový bit konce přenosu *HXFRDNIRQ* na hodnotu 1 a aktualizuje kód výsledku přenosu *HRSLT*. Jestli-že zařízení odpoví odesláním paketu *DATA1* s nulovou délkou dat, *SIE* automaticky odešle do zařízení potvrzovací paket *ACK*. Výsledek přenosu pak bude *hrSUCCESS*.

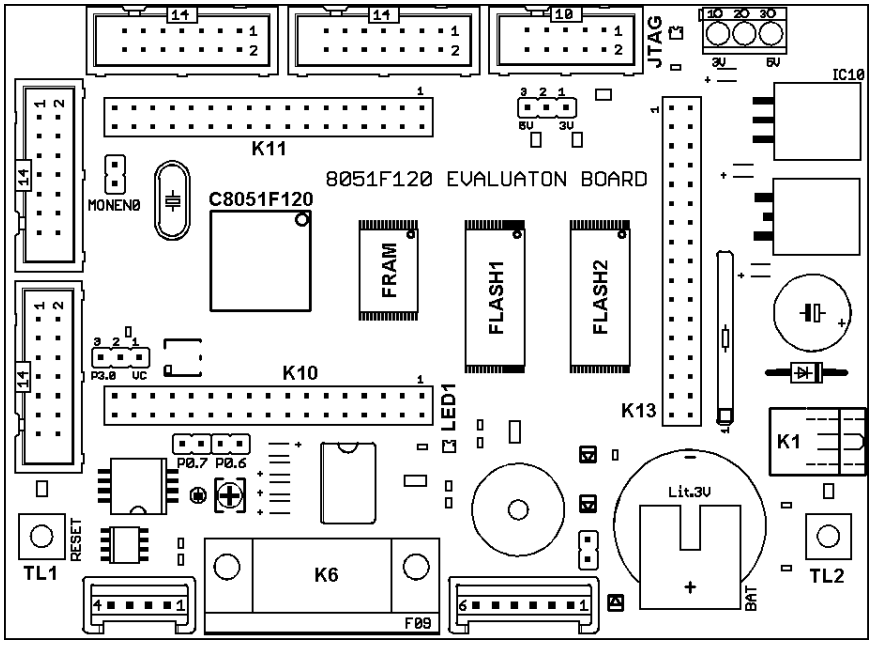
9 Propojení USB řadiče MAX3421E s vývojovou deskou SiLabs C8051F120

Jako USB hostitelský řadič pro realizaci diplomové práce byl zvolen obvod MAX3421E firmy Maxim. Byl vybrán pro svoji programovou jednoduchost, nenáročnost na napájení, jednoduché komunikační sběrnici a také pro solidní dokumentaci od výrobce včetně programovacího průvodce.

Dle požadavků zadání bylo nutné navrhnout vhodné propojení řadiče z vývojovou deskou Silicon Laboratories C8051F120, která byla poskytnuta pro vývoj diplomové práce.

9.1 Vývojová deska Silicon Laboratories C8051F120

Na obr. 24 je rozložení prvků na vývojové desce C8051F120. Dále bude pojednáno o prvcích a konektorech vývojové desky, které jsou potřeba pro vývoj komunikačního programu s flash diskem.



Obr. 24 Rozložení prvků na desce SiLabs C8051F120

Základem vývojové desky C8051F120 je stejnojmenný mikrokontrolér firmy Silicon laboratories, viz [3]. Tento mikrokontrolér disponuje instrukčním souborem poměrně rozšířeného mikrokontroléru 8051. Je napájen napětím 3,3V.

Konektor K1 slouží k napájení vývojové desky napětím +5 až 20V, jsou osazeny integrované stabilizátory napětí +5V a +3,3V. Konektor K13 slouží pro připojení další externí paměti. Pokud není aktivována žádná paměť a není aktivován interface pro připojení externí

paměti, lze využít porty P4-P7 pomocí tohoto konektoru. Porty P4-P7 nejsou vyvedeny na žádný jiný konektor. Konektor K10 poskytuje připojení na porty P0 a P2 mikrokontroléru. Je zde také připojeno ovládání LED1, a vyvedeno tlačítko TL2. Na konektoru je přítomno napájecí napětí 5V a GND.

Konektor K6 je typu CANON9, tvořící rozhranní RS232. Napevno je k tomuto konektoru připojen UART0 mikrokontroléru. Tlačítko TL1 je reset mikrokontroléru.

Na konektoru K11 je přístup na port P1 mikrokontroléru, výstupy DAC a vstupy ADC převodníků, vstupy analogových komparátorů. Na konektoru je vyvedeno napájecí napětí 3,3V. Zbývající port P3 mikrokontroléru je zamýšlen pro připojení alfanumerického displeje a není vyveden na žádném z konektorů K10, K11 a K13.

9.1.1 Zapojení konektorů vývojové desky

Dále budou popsány konektory vývojové desky Silicon Laboratories C8051F120, které jsou potřeba a také dle požadavků zadání volné pro připojení řadiče USB.

V tab. 31 je uvedeno zapojení konektoru K10. Na tomto konektoru je vyvedena sběrnice *SPI* potřebná pro připojení zvoleného USB řadiče. Dle zadání je pro potřeby práce volný port P2.

Tab. 31 Zapojení konektoru K10

Konektor K10					
Pin	**Funkce**	**Popis**	**Pin**	**Funkce**	**Popis**
1	LED1	L - svítí, H - nesvítí	**18**	P2.1	port P2
2	SP	siréna, H - píská	**19**	P2.2	port P2
3	TLEXT	TL2, sepnuto - L	**20**	P2.3	port P2
4	VDD3B	3,3V zálohované BAT	**21**	P2.4	port P2
5	R1OUT	RS232 - CTS	**22**	P2.5	port P2
6	T1IN	RS232 - RTS	**23**	P2.6	port P2
7	P0.0	port P0 (TX0)	**24**	P2.7	port P2
8	P0.1	port P0 (RX0)	**25**	SDA	sběrnice I2C
9	P0.2	port P0 (SCK)	**26**	SCL	sběrnice I2C
10	P0.3	port P0 (MISO)	**27**	RST	reset C8051F120
11	P0.4	port P0 (MOSI)	**28**	INT8583	---
12	P0.5	port P0 (NSS)	**29**	K48	---
13	P0.6	port P0 (SDA)	**30**	K410	---
14	P0.7	port P0 (SCL)	**31**	K47	---
15	VCC5	stab. napětí 5V	**32**	K49	---
16	DGND	digitální zem	**33**	VCC5	stab. napětí 5V
17	P2.0	port P2	**34**	DGND	digitální zem

V tab. 32 je uvedeno zapojení konektoru K11. Na tomto konektoru je k dispozici port P1, na kterém jsou vstupy externího přerušení mikrokontroléru (při aktivaci periferií dle zadání) a také napájecí napětí 3,3V pro řadič USB.

Tab. 32 Zapojení konektoru K11

Konektor K11					
Pin	**Funkce**	**Popis**	**Pin**	**Funkce**	**Popis**
1	CP1-	vstup - komparátoru 1	**18**	DAC1	výstup DAC1
2	CP1+	vstup + komparátoru 1	**19**	VREF01	spojení VREF0 a VREF2
3	CP0-	vstup - komparátoru 0	**20**	VREF	spojení VREFD a VREF
4	CP0+	vstup + komparátoru 0	**21**	P1.0	port P1.0 / INT0
5	VDD3A	filtrované napětí 3,3V	**22**	P1.1	port P1.1 / INT1
6	AGND	analogová zem	**23**	P1.2	port P1.2
7	AIN0	vstup ADC0	**24**	P1.3	port P1.3
8	AIN1	vstup ADC1	**25**	P1.4	port P1.4
9	AIN2	vstup ADC2	**26**	P1.5	port P1.5
10	AIN3	vstup ADC3	**27**	P1.6	port P1.6
11	AIN4	vstup ADC4	**28**	P1.7	port P1.7
12	AIN5	vstup ADC5	**29**	A	---
13	AIN6	vstup ADC6	**30**	B	---
14	AIN7	vstup ADC7	**31**	C	---
15	VCC3	stab. napětí 3,3V	**32**	D	---
16	DGND	digitální zem	**33**	VCC3	stab. napětí 3,3V
17	DAC0	výstup DAC0	**34**	DGND	digitální zem

9.2 Schéma zapojení modulu USB řadiče MAX3421E

Na obr. 25 je uvedeno schéma zapojení modulu řadiče MAX3421E a jeho připojení k vývojové desce Silicon Laboratories C8051F120. Modul je připojen na konektory K10 a K11 vývojové desky.

Číslování součástek je společné s modulem ethernetového řadiče, který je dle požadavků zadání na stejné desce s plošnými spoji, není však předmětem této práce.

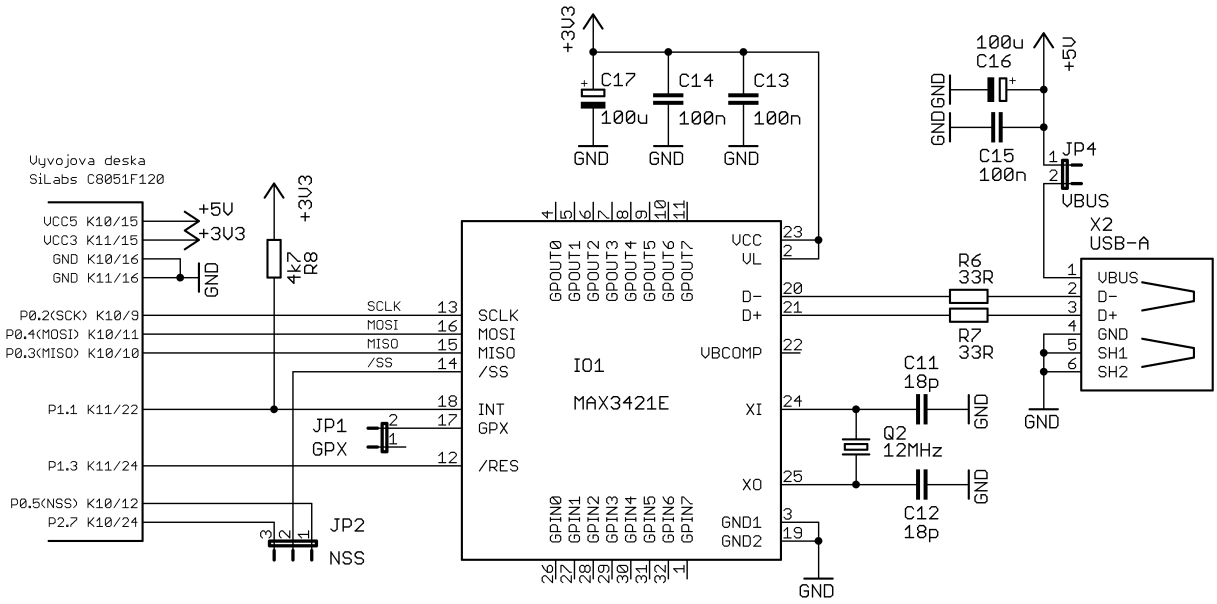
Do konektoru USB X2 (zásuvka typ A = hostitel) je přes propojku JP4 přivedeno napájení sběrnice USB +5V, které pro jednoduchost nemá zajištěno proudové omezení. JP4 umožňuje odpojit napájení sběrnice z konektoru X2 pro případ, že by byl modul využit jako USB periferie (s patřičnou konektorovou redukcí). Datové vodiče jsou do konektoru X2 připojeny podle technické dokumentace k obvodu přes rezistory R6 a R7.

Hodinový obvod tvoří vnitřní oscilátor s vnějšími kondenzátory C11, C12 a krystalem Q2 s jmenovitým kmitočtem 12MHz.

MAX3421E je napájen napájecím napětím 3,3V přivedeným z konektoru K11 do pinu *VCC* (napájení *USB tranceiveru*) a do pinu *VL* (napájení *SIE* a obvodů rozhraní). Oba zemní piny *GND1* a *GND2* jsou připojeny na zem digitálních částí vývojové desky. Napájení je filtrováno kondenzátory C13, C14 a C17.

Mikrokontrolér vývojové desky i MAX3421E pracují se stejným napájecím napětím, tedy mohou být propojovány přímo. Komunikační rozhraní řadiče MAX3421E tvoří standardní sériová sběrnice *SPI*. Sběrnice *SPI* (*SCLK*, *MOSI*, *MISO*) vývojové desky je z konektoru K10 přímo napojena do řadiče MAX3421E. Dle požadavků zadání je možné aktivační signál */SS* přepínat pomocí JP2 mezi signálem *NSS* sběrnice *SPI* vývojové desky a

portem P2.7, který je dostupný rovněž na konektoru K10. Je to z důvodu, že signál *NSS* mikrokontroléru vývojové desky je spolu s ostatními vodiči sběrnice *SPI* vyveden ještě na další konektor, umožňující připojení dalšího zařízení přes rozhraní *SPI*. Připojení aktivačního signálu řadiče USB na jiný port mikrokontroléru umožní současnou funkci s případným dalším zařízením.

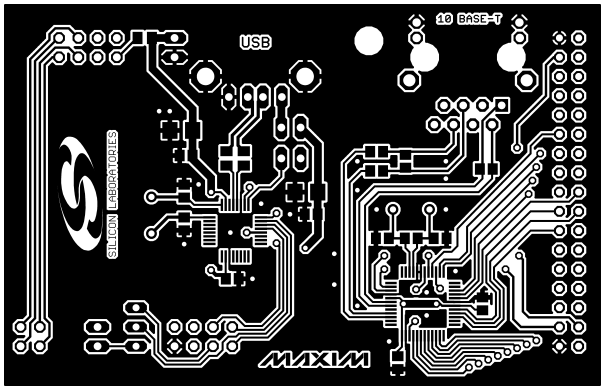


Obr. 25 Schéma zapojení modulu řadiče MAX3421E

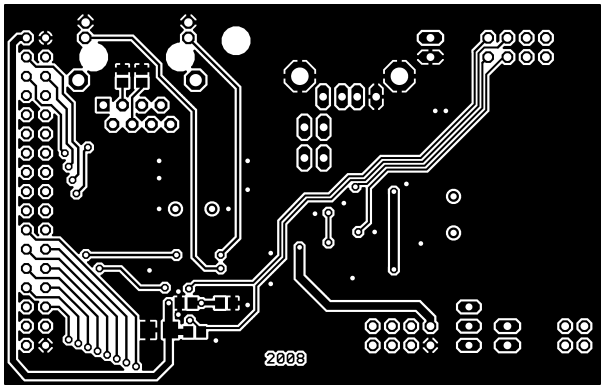
Na port P1.1, kde je dostupné externí přerušení *INT1*, je připojen signál přerušení z řadiče MAX3421E. Pomocí tohoto signálu poskytuje řadič informaci o dokončení veškerých komunikačních procesů. Pin *INT* řadiče USB je programově konfigurovatelný na obě polarity hrany nebo zápornou úroveň s nutností připojení *pull-up* rezistoru. Pomocný signál *GPX* z řadiče USB není třeba pro potřeby práce využít, proto je pro budoucí použití vyveden na konektor JP1. Univerzální piny *GPIN* a *GPOUT* nejsou pro potřeby práce využity.

9.3 Deska s plošnými spoji modulu řadiče MAX3421E

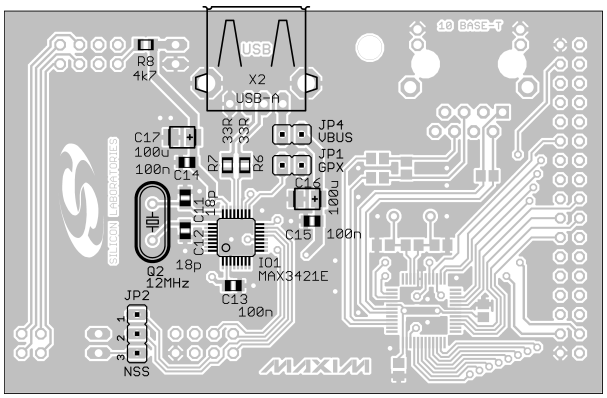
Na obr. 26 a obr. 27 je navržená deska s plošnými spoji pro řadič USB MAX3421E. Deska je podle požadavku zadání společná s modulem ethernetového řadiče (tato část desky byla předem k dispozici). Na osazovacích nákresech na obr. 28 a obr. 29 jsou uvedeny pouze součástky modulu USB. Deska má rozměry 79x50,4mm. Je navržená tak, aby ji bylo možno přímo zasunout do konektorů K10, K11 a K13 vývojové desky Silicon Laboratories C8051F120.



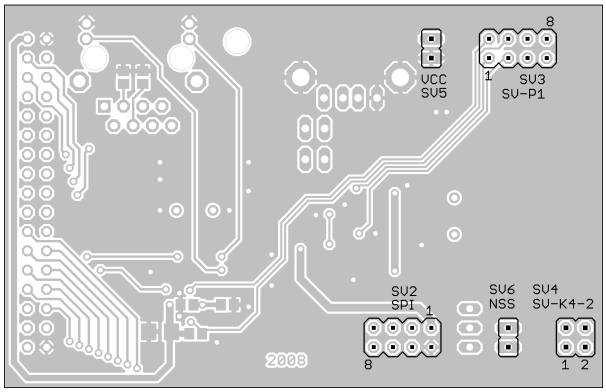
Obr. 26 Horní strana desky s plošnými spoji modulu USB (1:1)



Obr. 27 Spodní strana desky s plošnými spoji modulu USB (1:1)



Obr. 28 Osazení součástek na horní straně desky s plošnými spoji modulu USB



Obr. 29 Osazení součástek na spodní straně desky s plošnými spoji modulu USB

10 Software pro komunikaci s flash diskem

Na počátku vývoje softwaru pro komunikaci mikrokontroléru 8051 s USB flash diskem je nutné stanovit požadavky a základní architekturu softwaru. Je požadováno, aby základní USB přenosy – řídící a hromadný – zaručovaly kompletní ošetření proti chybám přenosu s opakováním ztracených nebo chybných paketů. Není požadována možnost připojení USB hubu, cílem je komunikace pouze s jedním připojeným zařízením na sběrnici. Funkce provádějící enumeraci připojeného zařízení bude akceptovat pouze zařízení *full-speed* a připojený flash disk funkce identifikuje podle parametrů deskriptorů uvedených v kap. 3.1. Pokud bude připojené zařízení obsahovat více rozhraní odpovídající flash disku, bude pracovat pouze s rozhraním s nejnižším indexem v konfiguraci rovněž s nejnižším indexem, kde bylo toto rozhraní nalezeno.

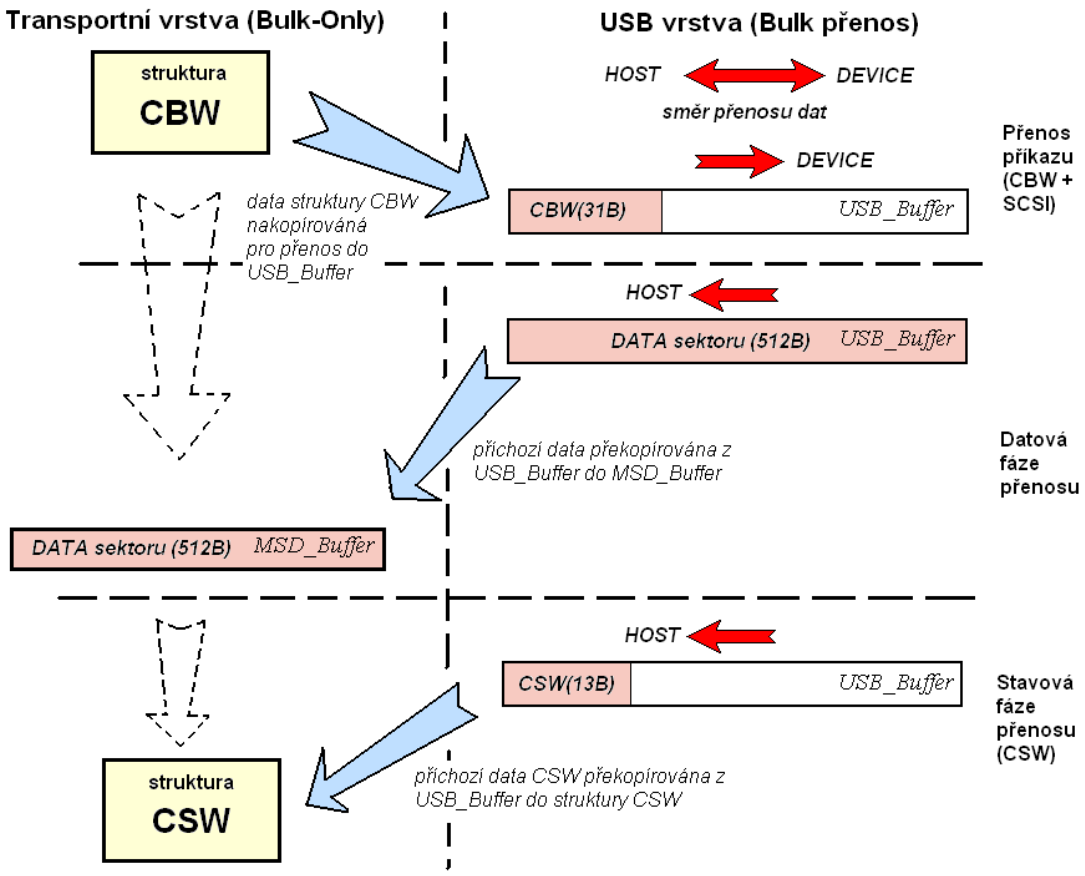
Přenos pomocí transportního protokolu *Bulk-Only* bude ošetřen proti chybám přenosu. Úspěšný přenos tímto protokolem však ještě neznamená úspěch provedení zadaného příkazu. Pro zápis a čtení sektorů z logické jednotky flash disku budou použity základní *SCSI* příkazy *WRITE(10)* a *READ(10)*. Úspěšné provedení těchto příkazů bude vyhodnoceno tehdy, pokud po úspěšném transportu budou přenesena všechna požadovaná data a status výsledku provedení příkazu bude hlásit úspěch, v jiném případě zahlásí software nedefinovanou chybu v komunikaci. Je také požadováno, aby software umožnil pracovat s více logickými jednotkami najednou a pouze se souborovým systémem *FAT16*.

Dalším požadavkem je, aby software rezervoval pro své potřeby co nejméně systémových prostředků mikrokontroléru C8051F120. Časová zpoždění budou realizována jednoduchým cyklem. Aby software alokoval co nejméně datové paměti, bude navržen tak, že si dokáže v datové paměti uchovat maximálně pouze data jednoho sektoru logické jednotky, což pro práci se souborovým systémem *FAT16* stačí. K realizaci přenosových funkcí je třeba alokovat zásobník, přes který budou protékat oběma směry data na USB. Tento zásobník bude nazván *USB_Buffer*. Pro trvalé uchování dat načteného sektoru nebo jako zdrojová data pro zápis do sektoru je třeba alokovat druhý zásobník, který bude nazván *MSD_Buffer*. Tímto zásobníkem budou protékat pouze data **datové fáze** přenosu protokolem *Bulk-Only*. Oba tyto zásobníky musejí mít velikost alespoň jednoho sektoru logické jednotky, se kterou pracují. Data první fáze *Bulk-Only* přenosu (*CBW*) a poslední fáze (*CSW*) budou směřována do (a ze)

zásobníku *USB_Buffer* přímo z deklarovaných datových struktur stejných názvů. Obr. 30 znázorňuje tok dat mezi zásobníky při čtení sektoru o velikosti 512 bajtů z flash disku.

Pro práci se soubory je požadováno, aby software umožnil práci s více soubory najednou – více otevřených souborů současně. Kromě možnosti ukládání a čtení dat ze souboru je požadována možnost vytváření adresářů a také možnost volitelně nastavovat atributy, datum a čas změn souborů do jejich hlaviček. Naopak není požadováno, aby software dokázal pracovat s cestami k souborům a adresářům z důvodu omezených systémových prostředků mikrokontroléru 8051 (cesta může mít v operačním systému Windows až 260 znaků). Program se bude moci vnořovat postupně pouze o jednu úroveň zadáním konkrétního adresáře v adresáři současné pozice. Dále je požadována možnost zjištění velikosti volného místa v logické jednotce a také možnost identifikovat konkrétní vložený flash disk. Identifikace bude umožněna načtením prvních šestnácti znaků deskriptoru textového řetězce sériového čísla a také uchováním parametrů *ID Vendor* a *ID Product* z deskriptoru zařízení flash disku.

U koncových uživatelských funkcí pro práci s daty na flash disku je také požadováno, aby v případě selhání podávaly informaci o příčině neúspěchu (např. zadaný adresář nebyl nalezen). Posledním hlavním požadavkem je, že software bude pracovat pouze s krátkými názvy souborů a adresářů, nebude tedy podporovat vstupy *LFN*.



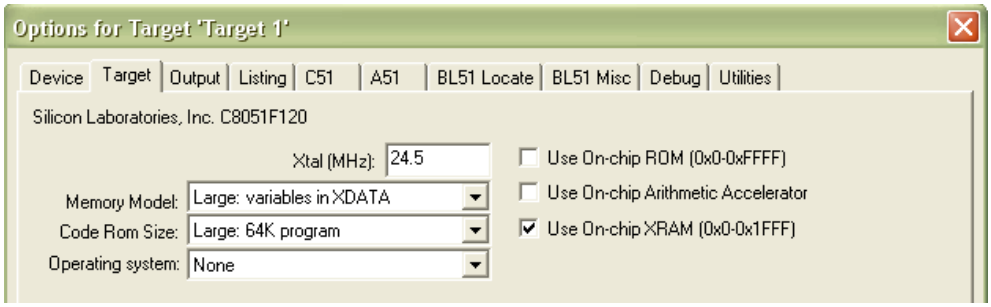
Obr. 30 Tok dat mezi zásobníky při čtení sektoru logické jednotky

Software umožňují komunikaci vývojové desky SiLabs s USB flash diskem je napsán v jazyce C a kompilován ve vývojovém prostředí KEIL µVision3. Moduly softwaru i celý vzorový projekt je na přiloženém CD. Zde uvedené informace slouží jako doprovodný text ke zdrojovým souborům softwaru, který umožní lepší orientaci ve zdrojovém textu.

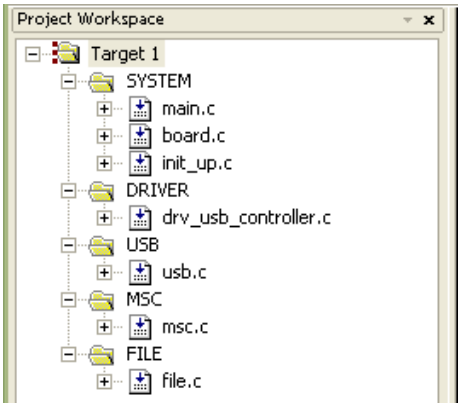
Vzhledem k nárokům programu, je zvolena konfigurace uložení proměnných programu do paměti XRAM mikrokontroléru, viz obr. 31 (*Project / Options for Target…*, záložka *Target*).

Kvůli relativní rozsáhlosti programu, je pro přehlednost a účelnost rozdělen do sedmi modulů, které většinou představují určitou část programové vrstvy komunikace. Protože jednotlivé moduly spolu kooperují, jsou více či méně navzájem provázané. Zcela samostatný modul je ovladač řadiče USB. S novým ovladačem pro jiný řadič USB je nutná případná úprava nastavení hardwaru mikrokontroléru podle použitého komunikačního rozhranní s daným USB řadičem.

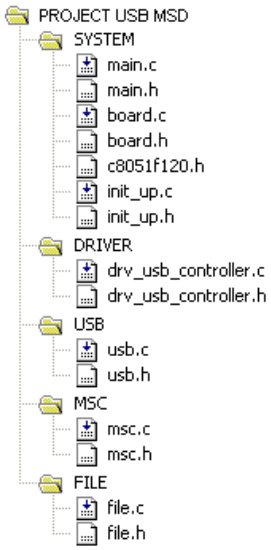
V jednotlivých modulech jsou specificky nastavené cesty k jednotlivým souborům jiných modulů (*#include* "…"). Moduly v projektu musí být na disku uloženy v adresářové struktuře podle obr. 33, v jiném případě je nutné v jednotlivých zdrojových souborech přepsat cesty k daným souborům, které jsou do modulů vkládány. Na obr. 32 je uveden příklad uspořádání jednotlivých modulů v projektu prostředí µVision3 (editor *Project Workspace*), které také zvýší přehlednost při tvorbě programu.



Obr. 31 Konfigurace kompilátoru µVision3



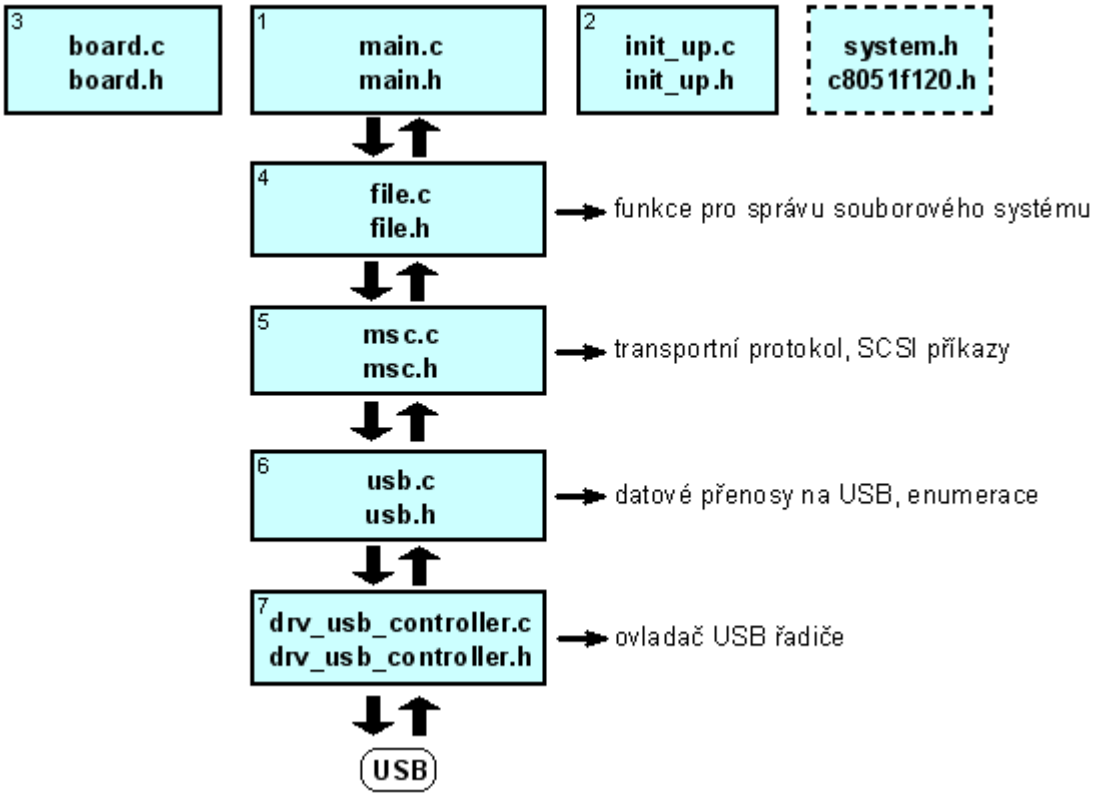
Obr. 32 Příklad přehledného uspořádání modulů v projektu



Obr. 33 Adresářová struktura souborů modulů projektu Project USB MSD

10.1 Programové moduly

Projekt je rozdělen do sedmi programových modulů znázorněných na obr. 34. Na obrázku je naznačen postupný datový tok z jednoho modulu do druhého jak prochází jednotlivými vrstvami komunikace, od USB řadiče až po hlavní modul (*main*).



Obr. 34 Moduly a ostatní zdrojové soubory projektu

1) ***main.c, main,h*** – Hlavní modul programu, obsahující funkci *main*. Z hlavního modulu jsou volány uživatelské funkce pro práci s flash diskem. Modul *main.c* není přímou součástí modulů komunikace, může být uzpůsoben podle uživatele. Ukázka základního provedení modulu *main.c* je v kap.10.9.

2) ***init_up.c, init_up.h*** – Počáteční nastavení hardwaru mikrokontroléru C8051F120. Soubor init_up.c je generován programem *Config2 Silicon Laboratories* a je vložen do modulu *main.c*.

3) ***board.c, board.h*** – Modul obsahuje přiřazení názvů portů mikrokontroléru pro ovládání některých signálů řadiče USB a případně indikačních LED, popř. obsahuje další jednoduché funkce. Modul *board* slouží pro nastavení konkrétní konfigurace zařízení vývojové desky.

4) ***file.c, file.h*** – Obsahuje funkce pro práci se souborovým systémem *FAT16* logické jednotky disku.

5) ***msc.c, msc.h*** – Transportní protokol *Bulk-Only* a funkce pro aplikaci použitých *SCSI* příkazů.

6) ***usb.c, usb.h*** – Realizuje řídící přenos, hromadný přenos *IN* a *OUT*, proceduru enumerace flash disku.

7) ***drv_usb_controller.c, drv_usb_controller.h*** – Obsahuje funkce pro ovládání řadiče USB, ze kterých jsou sestaveny v modulu *usb* jednotlivé typy přenosů.

system.h – Obsahuje obecná indikační makra, které využívají všechny moduly

c8051f120.h – Definice registrů mikrokontroléru C8051F120.

Možnosti nastavení programových modulů pro zkompilování programu dle požadavků uživatele je uveden v kap.10.10. Jednotlivé moduly obsahují větší či menší množství funkcí, které pomáhají realizovat hlavní funkce modulů. Tyto jednoduché funkce není třeba detailně popisovat, proto v následujících kapitolách budou popsány jen hlavní funkce jednotlivých modulů a jejich důležité parametry.

10.2 Programový modul „drv_usb_controller“

Modul *drv_usb_controller* slouží jako ovladač řadiče MAX3421E. Hlavní funkce modulu jsou voleny tak, aby je bylo možné realizovat s širokou škálou jiných řadičů USB, a aby z nich šel vytvořit řídící a hromadný typ přenosu. Při použití jiného řadiče bude potřeba vyměnit pouze tento modul. Jednotlivé funkce jsou zde poměrně detailně popsány pro vysvětlení ovládání řadiče, a také pro usnadnění tvorby ovladače pro jiný řadič. Proměnné a funkce, které používají vyšší moduly pro sestavení komunikace jsou označeny v souboru *drv_usb_controller.h*. Ostatní funkce a proměnné jsou specifické pro řadič MAX3421E umožňující realizaci hlavních funkcí. Modul také obsahuje funkci pro vytváření časových zpoždění, která zaměstná procesor na definovanou dobu.

```
void Delay(unsigned long int time);
```

Funkce *Delay* je realizována cyklem *for*. Parametr *time* funkce zadává počet provedení cyklů a tím také dobu zpoždění. V modulu jsou definována makra jako parametry pro tuto funkci od časů 100µs až po 5s v několika krocích na dekádu (např. *TIME_100ms*). Protože zpoždění této funkce závisí na taktovací frekvenci mikrokontroléru, jsou definována makra pro 5 různým kmitočtů mikrokontroléru.

```
//#define SYSTEM_CLOCK_1                // pro sysclk = 6.125MHz
//#define SYSTEM_CLOCK_2                // pro sysclk = 12.25MHz
//#define SYSTEM_CLOCK_3                // pro sysclk = 24.5MHz
//#define SYSTEM_CLOCK_4                // pro sysclk = 49MHz
#define SYSTEM_CLOCK_5                  // pro sysclk = 98MHz
```

Musí být vždy definováno jedno z těchto maker. Podmíněnou kompilací programu jsou pak platné patřičné definice časů *TIME_*.. pro funkci *Delay*. Pokud mikrokontrolér pracuje s jinou taktovací frekvencí, zvolí se makro pro frekvenci nejbližší vyšší nebo lze hodnoty přepočítat trojčlenkou pro novou frekvenci. Funkci *Delay* používají, všechny ostatní moduly softwaru.

10.2.1 Funkce SPI

Funkce *SPI* provede výměnu dat na sběrnici *SPI*. Je to funkce specifická pro použitý řadič USB. Protože pro ovládání datového registru *SPI0DAT* musí mít mikrokontrolér nastavenou patřičnou stránku *SFR*, funkce provede zálohu aktuální stránky a na jejím konci je stránka v registru *SFRPAGE* opět obnovena.

```
unsigned char SPI(unsigned char SPI_data);
```

Vstupním parametrem jsou data vysílána na sběrnici *SPI* do periferie a vrácený parametr jsou data, která periferie odeslala zpět. Funkce skončí až po provedení přenosu. Pomocí této funkce je vytvořena veškerá komunikace s řadičem USB.

10.2.2 Funkce Register

Funkce *Register* slouží ke čtení nebo zápisu do jednoho registru řadiče MAX3421E a opět se jedná o specifickou funkci pro použitý řadič. Má dva vstupní parametry. *Reg* je příkazový bajt, jehož formát byl uveden v kap. 8.1.3. Parametr *Value* je zapisovaná hodnota do registru. V případě čtení z registru je *Value* libovolná vložená hodnota.

```
unsigned char Register(unsigned char Reg, unsigned char Value);
```

V případě čtení z registru funkce vrací přečtená data. Na počátku funkce nastaví signál *USB_SS* (*slave select* řadiče USB) na log.0 čímž započne relaci. Poté pomocí funkce *SPI* odešle hodnotu *Reg* a poté hodnotu *Value*. Relaci ukončí nastavením signálu *USB_SS* na hodnotu log.1. Funkce vrátí příchozí data z přenosu hodnoty *Value*.

10.2.3 Funkce USB_Controller_Reset

Funkce slouží k resetování řadiče USB. Reset řadiče je proveden nastavením signálu *RST* (deklarovaný v modulu *board* na port P1.3) na hodnotu log.0 po dobu 1ms. Poté čeká na nastavení bitu *OSCOKIRQ* v registru *USBIRQ*, který indikuje stabilní signál z interního oscilátoru řadiče. Poté je bit vymazán zapsáním hodnoty 1.

```
void USB_Controller_Reset();
```

Ukázalo se, že v době několik milisekund po resetu řadiče nelze přepnou řadič do režimu hostitele. Proto funkce nakonec čeká dobu 10ms k ustálení vnitřních procesů řadiče.

10.2.4 Funkce USB_Controller_Init

Funkce *USB_Controller_Init* slouží k počátečnímu nastavení řadiče USB. Nejdříve provede reset řadiče pomocí funkce *USB_Controller_Reset*.

```
void USB_Controller_Init();
```

Protože mikrokontrolér C8051F120 nepodporuje režim provozu sběrnice *SPI half-duplex*, je nutné nejdříve v registru *PINCTL* nastavit bit *FDUPSPI* na hodnotu 1, čímž se řadič přepne do režimu *full-duplex*. Současně se v tomtéž registru nastaví na hodnotu 1 bit *INTLEVEL*, čímž se přepne výstupní pin *INT* řadiče na signalizaci úrovní log.0. Poté se v registru *MODE* nastaví na hodnotu 1 bity *HOST*, *DPPULLDN* a *DMPULLDN*, čímž přejde řadič do režimu hostitele a připojí se vnitřní rezistory 15kΩ na signálové vodiče sběrnice USB. Dále jsou vynulovány bity v registrech *HIEN* a *USBIEN*, čímž jsou zakázány veškeré signalizace přerušení na pin *INT* řadiče. Dále jsou zapsáním hodnot 1 do všech bitů registru *HIRQ* vynulovány všechny příznaky vnitřních procesů obvodu. V registru *CPUCTL* je nastaven na hodnotu 1 bit *IE*, čímž se povolí výstupní pin *INT* řadiče. Nakonec jsou v registru *HIEN* povoleny signalizace přerušení *HXFRDNIE* (ukončení přenosu) a *CONDETIE* (připojení/odpojení zařízení na sběrnici).

10.2.5 Funkce INT_USB_Controller

Funkci *INT_USB_Controller* je potřeba volat při signalizaci požadavku přerušení signálem *INT* řadiče USB. Funkce obsluhy přerušení C8051F120 od externího zdroje není součástí tohoto modulu. Funkce provádí obsluhu dvou různých signalizací řadiče USB – dokončení přenosu a připojení/odpojení zařízení na sběrnici. Přerušení mikrokontroléru je použito z důvodu, aby při odpojení nebo připojení zařízení na sběrnici byly aktualizovány patřičné příznaky programu indikující stav připojení.

```
void INT_USB_Controller();
```

Funkce přečte obsah registru *HIRQ*, který indikuje příčinu přerušení. Jeli nastaven bit *HXFRDNIRQ*, nastaví příznak *USB_A* na hodnotu 1. To signalizuje hlavnímu programu dokončení přenosu. Pokud je nastaven na hodnotu 1 bit *CONDETIRQ*, je dále testováno, zda je toto přerušení povoleno. Není-li povoleno, bit se pouze vynuluje. Je-li povoleno, provede se test, zda bylo připojeno zařízení *FS*. To je zjištěno ze stavu bitů *JSTATUS* a *KSTATUS*

v registru *HRSL*. Jeli nastaven na hodnotu 1 pouze bit *JSTATUS*, bylo připojeno zařízení *FS*. Jakýkoliv jiný stav je vyhodnocen jako zařízení nepřipojeno (i v případě připojení zařízení *LS*). Je-li vyhodnoceno připojení, je nastaven na hodnotu 1 globální příznakový bit *Inserted*, který v každém okamžiku běhu programu informuje o připojeném zařízení. Také je vynulován příznakový bit řadiče (zapsáním hodnoty 1) *RCVDAVIRQ*, čímž je přijímací paměť *FIFO* řadiče připravena přijmout paket. Bylo-li zařízení odpojeno, je okamžitě vypnuto automatické generování paketů *SOF* vynulováním bitu *SOFKAENAB* v registru *MODE*. Neučiní-li se tak, nastává z neznámého důvodu opakovaně přerušení *CONDETIRQ* i bez změn na sběrnici. Nakonec se vynuluje příznakový bit *Inserted* a také *Enumerated* a *Number_Of_LUNs*, o kterých bude pojednáno později.

Protože přerušení *CONDETIRQ* nastávalo při odpojení zařízení ze sběrnice dvakrát za sebou, je aplikováno zpoždění 100ms a poté jsou vymazány všechny příznaky přerušení v registru *HIRQ*.

V modulu je možné definovat makro *CONNECT_LED*. Pokud je toto makro definováno, je proveden překlad funkce *INT_USB_Controller* s funkcí *Blik* z modulu *board*, která při připojení a odpojení zařízení ze sběrnice USB blikne s LED, jejíž připojení na port mikrokontroléru je rovněž definováno v modulu *board*.

```
#define CONNECT_LED
```

10.2.6 Funkce USB_Device_Detect

Funkce slouží k zjištění, zda je na sběrnici USB připojeno zařízení *FS* a zároveň aktualizuje globální příznak připojení *Inserted*. Funkce je potřeba v případě, že je zařízení na sběrnici připojeno v okamžiku, kdy je zapnut nebo resetován hostitel. V tomto případě nenastane přerušení od připojení/odpojení zařízení a globální parametr *Inserted* není aktualizován.

```
bit USB_Device_Detect();
```

K zjištění stavu signálů sběrnice USB je třeba sejmout její vzorek. To je provedeno nastavením na hodnotu 1 bitu *SAMPLEBUS* v registru *HCTL* řadiče. Tímto byly podle aktuálního stavu sběrnice aktualizovány bity *JSTATUS* a *KSTATUS* v registru *HRSL*. Zařízení *FS* je připojeno, je-li na sběrnici stav J. Funkce kromě aktualizace parametru *Inserted* vrací hodnotu 1 pokud je zařízení *FS* připojeno, nebo hodnotu 0 v opačném případě.

10.2.7 Funkce USB_Engine_Reset

Funkce *USB_Engine_Reset* slouží k vyvolání stavu USB reset (jednoznačná 0) sběrnice USB po dobu 50ms, čímž je resetováno připojené zařízení na sběrnici.

```
void USB_Engine_Reset();
```

Funkce nejdříve zakáže všechny signalizace přerušení v registru *HIEN* nastavením všech bitů na hodnotu 0. USB reset je spuštěn nastavením na hodnotu 1 bitu *BUSRST* v registru *HCTL*. Dobu 50ms časuje řadič automaticky. Konec resetu je indikován

automatickým nastavením bitu *BUSRST* na hodnotu 0. Protože se předpokládá použití této funkce pouze po připojení zařízení na sběrnici, je poté spuštěno automatické generování paketu *SOF* nastavením na hodnotu 1 bitu *SOFKAENAB* v registru *MODE*. Poté funkce čeká na nastavení bitu *FRAMEIRQ* v registru *HIRQ*, který indikuje proběhnutí začátku dalšího USB rámce. V [5] je doporučeno, aby před zahájením přenosu po USB resetu proběhl alespoň jeden prázdný rámec. Poté funkce vymaže všechny příznaky v registru *HIRQ* a opět povolí přerušení *HXFRDNIE* a *CONDETIE* v registru *HIEN*.

10.2.8 Funkce USB_SETUP_Token

Funkce *USB_SETUP_Token* provede přenos řídícího požadavku jako první fáze řídícího přenosu. Před voláním funkce musí být naplněna paměť *SUDFIFO* daty řídícího požadavku.

```
void USB_SETUP_Token();
```

V modulu *drv_usb_controller* je definována globální proměnná *USB_Device_Address*, která musí obsahovat před voláním přenosových funkcí adresu cílového zařízení v rámci sběrnice (0-127). Tato adresa je funkcí zkopírována do registru *PERADDR* řadiče.

V modulu lze také definovat makro *PACKET_LED*. Pokud je makro definováno provede se podmíněný překlad s funkcí *LEDX1_ON* z modulu *board*, která rozsvítí LED, indikující přenos paketu na sběrnici USB. Tato LED poté výstižně indikuje činnost hostitele na sběrnici.

```
#define PACKET_LED
```

Poté je zapsána do registru *HXFR* řadiče hodnota 0x10, čímž dojde k zahájení přenosu. Nakonec je volána funkce *USB_Packet_Status*, která vyhodnotí výsledek přenosu a nastaví patřičné příznaky programu (viz kap. 10.2.11).

10.2.9 Funkce USB_IN_Token

Funkce *USB_IN_Token* slouží k přenosu datového paketu ze zařízení do hostitele. Před voláním funkce musí být vynulován bit *RCVDAVIRQ* v registru *HIRQ*, pro indikaci, že je přijímací paměť *FIFO* řadiče připravena přijmout nový datový paket.

```
void USB_IN_Token(unsigned char Endpoint, unsigned char Length, unsigned
char Sequence);
```

Řadič MAX3421E dokáže automaticky střídat sudé a liché datové pakety a pouze v případě chybné sekvence ohlásí chybu. Protože tuto vlastnost nemají všechny běžné USB řadiče, funkce modulu *drv_usb_controller* tuto vlastnost nevyužívají. Aby bylo možné vyhodnotit, za byl přijat lichý nebo sudý datový paket, je třeba před každým odesláním paketu *IN* nastavit počáteční typ očekávaného datového paketu v párových bitech *RCVTOG0* a *RCVTOG1* řadiče MAX3421E. Očekávaný typ paketu je zadán ve vstupním parametru *Sequence* (0 = *DATA0*, 1 = *DATA1*). Podle této hodnoty je funkcí buď nastaven bit *RCVTOG0* nebo bit *RCVTOG1* v registru *HCTL*.

Funkce poté uloží do registru *PERADDR* adresu zařízení z globální proměnné *USB_Device_Address* a podle toho, zda je definováno makro *PACKET_LED* rožne příslušnou LED jako indikaci zahájení přenosu paketu na sběrnici.

Přenos je zahájen zápisem hodnoty 0000eeee(b) do registru *HXFR*, kde eeee je číslo cílové brány zadané vstupním parametrem *Endpoint*. Pro vyhodnocení výsledku přenosu je volána finkce *USB_Packet_Status*, (viz kap. 10.2.11), která také vyhodnotí, zda byl přijat lichý nebo sudý datový paket. Nakonec je třeba vyhodnotit některé další situace. Funkce po dokončení přenosu testuje, zda je nastaven příznakový bit *RCVDAVIRQ* v registru *HIRQ*, který indikuje, že *FIFO* obsahuje přijatá data. Pokud je nastaven, do globální proměnné *RCV_Count* je uložen počet přijatých bajtů z registru *RCVBC*, není-li nastaven, do *RCV_Count* je uložena hodnota 0. Od této chvíle se již registr *RCBC* nebude do nového přenosu znovu číst a počet přijatých bajtů je uložen v proměnné *RCV_Count*. Dále je v modulu definována globální bitová proměnná *PSR_Overflow*, která má hodnotou 1 indikovat, že bylo přijato více bajtů, než bylo očekáváno hostitelem. Tuto funkci mají přímo některé USB řadiče (např. SL811HST firmy Cypress), MAX3421E ji však nemá. Vyhodnocení je provedeno na základě porovnání vstupního parametru *Length* (zadaný očekávaný maximální počet bajtů) a počtu přijatých bajtů *RCV_Count*. Je-li *RCV_Count* větší než *Length*, je indikována chyba nastavením *PSR_Overflow* na hodnotu 1.

10.2.10 Funkce USB_OUT_Token

Funkce *USB_OUT_Token* slouží k odeslání datového paketu z hostitele do zařízení. Před voláním funkce musí být ve vysílací *FIFO* řadiče uložena odesílaná data a také musí být jejich počet uložen v registru *SNDBC*.

```
void USB_OUT_Token(unsigned char Endpoint, unsigned char Length, unsigned
char Sequence);
```

Na počátku podle vstupního parametru *Sequence* (0 = *DATA0*, 1 = *DATA1*) nastaví funkce párové bity *SNDTOG0* a *SNDTOG1*, v registru *HCTL*, čímž se nastaví, zda bude odeslán lichý nebo sudý datový paket. Funkce poté uloží do registru *PERADDR* adresu zařízení z globální proměnné *USB_Device_Address* a podle toho, zda je definováno makro *PACKET_LED* rožne příslušnou LED jako indikaci zahájení přenosu paketu na sběrnici.

Přenos je zahájen zápisem hodnoty 0010eeee(b) do registru *HXFR*, kde eeee je číslo cílové brány zadané vstupním parametrem *Endpoint*. Pro vyhodnocení výsledku přenosu je volána funkce *USB_Packet_Status*, (viz kap. 10.2.11).

10.2.11 Funkce USB_Packet_Status

Funkce *USB_Packet_Status* slouží k vyhodnocení proběhlého přenosu a je volána funkcemi *USB_SETUP_Token*, *USB_IN_Token* a *USB_OUT_Token*, které zahajují přenos na sběrnici USB.

```
void USB_Packet_Status();
```

Na počátku funkce čeká na dokončení probíhajícího přenosu, který je indikován nastavením globální proměnné *USB_A* na hodnotu 1 v obsluze přerušení od řadiče USB. Nenastane-li konec přenosu do určité doby, čekání se přeruší a přenos je vyhodnocen jako *time-out*. K indikaci výsledku přenosu jsou v modulu deklarovány globální příznakové bity:

```
bit PSR_ACK;
bit PSR_Error;
bit PSR_Timeout;
bit PSR_Sequence;
bit PSR_Overflow;
bit PSR_NAK;
bit PSR_STALL;
bit PSR_TOGERR;
```

Funkce vyčte z registru řadiče *HRSL* parametr *HRSLT* a podle kódu výsledku přenosu jsou nastaveny příslušné příznakové bity. *PSR_ACK* je nastaven na hodnotu 1, pokud byl přenos úspěšný, *PSR_Timeout* pokud zařízení neodpovědělo, příznak *PSR_Overflow* je vyhodnocen ve funkci *USB_IN_Token* a indikuje, že bylo přijato více bajtů než bylo hostitelem očekáváno, *PSR_NAK* je nastaven pokud zařízení odeslalo potvrzovací paket *NAK*, *PSR_STALL* je nastaven pokud zařízení odeslalo paket *STALL*. Příznakový bit *PSR_TOGERR* je nastaven, pokud byl přijat jiný typ datového paketu než byl řadičem očekáván. Při všech jiných chybách indikovaných polem *HRSLT* je nastaven příznak *PSR_Error*.

Poté funkce vyhodnotí, zda byl přijat lichý nebo sudý datový paket. Funkce nejdříve vyčte z řadiče hodnotu bitu *RCVTOG1*, jehož hodnota odpovídá požadovanému typu paketu (*DATA0/1*). Pokud přenos nebyl úspěšný nebo je nastaven příznakový bit programu *PSR_TOGERR* je typ přijatého paketu roven negované hodnotě bitu *RCVTOG1*. V opačném případě je typ přijatého paketu roven bitu *RCVTOG1*. Podle této podmínky je nastaven globální příznakový bit *PSR_Seguence* (0 = *DATA0*, 1 = *DATA1*), který udává typ přijatého paketu.

Pokud je definováno makro *PACKET_LED*, provede se podmíněný překlad programu s funkcí *LEDX1_OFF* z modulu *board*, která zhasne LED, indikující přenos paketu po sběrnici. Tato LED byla rozsvícena příslušnými funkcemi zahajující přenos paketů.

10.2.12 Funkce USB_Data_Received

Funkce *USB_Data_Received* slouží k zjištění počtu přijatých bajtů při posledním přenosu ve směru *IN*. Funkce vrací přímo hodnotu počtu bajtů z globální proměnné *RCV_Count*, která byla naplněna při dokončení přenosu *IN*.

```
unsigned char USB_Data_Received();
```

10.2.13 Funkce USB_Buffer_IN

Funkce *USB_Buffer_IN* slouží k vyčtení přijímací paměti *FIFO* řadiče. Vyčtení paměti *RCVFIFO* je provedeno během jediné relace na sběrnici *SPI*.

```
void USB_Buffer_IN(unsigned char *Buffer, unsigned int Offset, unsigned
char Length);
```

Funkce má tři vstupní parametry. Parametr *Buffer* je ukazatel na cílové datové pole typu *unsigned char*, kam budou uložena vyčtená data. Parametr *Offset* určuje ofset v cílovém poli od jeho začátku, odkud budou data ukládána a poslední parametr *Length* udává počet bajtů, které funkce z paměti řadiče vyčte. Po vyčtení požadovaného počtu bajtů je vynulován bit řadiče *RCVDAVIRQ* v registru *HIRQ*, čímž je přijímací *FIFO* připravena přijmout nová data.

10.2.14 Funkce USB_Buffer_OUT

Funkce *USB_Buffer_OUT* slouží k naplnění vysílací paměti *FIFO* řadiče. Naplnění paměti *SNDFIFO* je provedeno během jediné relace na sběrnici *SPI*.

```
void USB_Buffer_OUT(unsigned char *Buffer, unsigned int Offset, unsigned char Length);
```

Funkce má tři vstupní parametry. Parametr *Buffer* je ukazatel na zdrojové datové pole typu *unsigned char*, ze kterého budou uložena data do vysílací paměti řadiče. Parametr *Offset* určuje ofset ve zdrojovém poli od jeho začátku, kde začínají zdrojová data a poslední parametr *Length* udává počet bajtů, které funkce uloží do paměti řadiče. Po naplnění požadovaného počtu bajtů je uložen do registru řadiče SNDBC počet bajtů, který byl do vysílací paměti uložen, čímž je vysílací paměť připravena k odeslání dat.

10.2.15 Funkce USB_Buffer_SETUP

Funkce *USB_Buffer_SETUP* slouží k naplnění paměti řídícího požadavku řadiče *SUDFIFO*. Naplnění paměti *SUDFIFO* je provedeno během jediné relace na sběrnici *SPI*.

```
void USB_Buffer_SETUP(unsigned char *Buffer, unsigned int Offset);
```

Funkce má dva vstupní parametry. Parametr *Buffer* je ukazatel na zdrojové datové pole typu *unsigned char*, ze kterého budou uložena data požadavku do paměti *SUDFIFO* řadiče. Vždy bude ze zdrojového pole přeneseno do řadiče 8 bajtů. Parametr *Offset* určuje ofset ve zdrojovém poli od jeho začátku, kde začínají zdrojová data požadavku.

10.3 Programový modul „usb“

Modul *usb* obsahuje funkce zprostředkovávající datové přenosy po USB a enumeraci zařízení typu flash disk. Modul nepodporuje připojení USB hubu na sběrnici, vždy musí být na sběrnici pouze cílové zařízení. Modul deklaruje globální datové struktury deskriptoru zařízení, konfigurace, rozhraní, brány *OUT* a brány *IN*. Tyto struktury obsahují jen ty parametry jednotlivých USB deskriptorů, které jsou potřeba pro další komunikaci nebo by mohly být dále využity. Ve strukturách deskriptorů jsou informace k připojenému flash disku platné teprve až po úspěšné enumeraci (volání enumerační funkce).

Modul *usb* také deklaruje datové pole pro veškerou datovou komunikaci po sběrnici USB. Jakékoli odchozí nebo příchozí data prochází přes toto pole. Velikost pole musí mít minimálně velikost největší podporované velikosti jednoho sektoru logické jednotky flash

disku. Větší velikost nemá pro funkce modulu *usb* smysl, nižší nemůže pro správnou funkci být (maximální podporovaná velikost sektoru pro kompilaci programu je volená uživatelem, viz kap. 10.8).

```
xdata unsigned char USB_Buffer[MAX_SECTOR_SIZE];
```

Pro sestavení jednotlivých datových přenosů používá modul funkce pro odesílání paketů jen z modulu *drv_usb_controller*.

10.3.1 Funkce USB_Control_Request

Funkce *USB_Control_Request* vykonává USB řídící požadavky, tedy realizuje řídící přenos dle kap. 2.2.3.

```
unsigned char USB_Control_Request(unsigned char bmRequestType, unsigned char bRequest, unsigned int wValue, unsigned int wIndex, unsigned int wLength);
```

Funkce má pět vstupních parametrů, představující řídící požadavek (8 bajtů) dle kap. 2.5. Jak již bylo zmíněno dříve, jsou tři typy řídícího přenosu. Protože žádný řídící požadavek s datovou fází s přenosem dat do zařízení není pro komunikaci s většinou USB zařízení potřeba, funkce tento typ požadavků nepodporuje! Tyto požadavky nesmějí být do funkce zadávány. Funkce podporuje všechny požadavky s datovou fází s přenosem dat ze zařízení do hostitele a všechny požadavky bez datové fáze přenosu. Tyto dva typy představují naprostou většinu všech požadavků.

Funkce *USB_Control_Request* vrací parametr představující výsledek přenosu:

0 OK, požadavek úspěšně proveden,
1 zařízení vrátilo paket *STALL*, buď je požadavek nepodporován, nebo selhalo jeho vykonávání,
2 zařízení neustále vracelo paket *NAK* nebo nastala opakovaná chyba v přenosu.

Funkce *USB_Control_Request* vykonává kompletní řídící přenos, včetně řešení chybových situací. Všechny pakety jsou adresovány na řídící bránu 0 zařízení. Velikost paketu pro řídící bránu je dostupná ve struktuře deskriptoru zařízení *Device_Descriptor*, tedy před prvním použitím funkce musí být tato struktura vhodně nastavena (parametr *bMaxPacketSize0*). Nejdříve funkce odešle *SETUP* paket s datovým paketem požadavku *DATA0*. Pokud zařízení vrátí paket *STALL*, celá funkce končí s tímto výsledkem, tedy zadaný požadavek byl nepodporován. Nastane-li jakákoli jiná chyba, funkce počká dobu definovanou makrem *ERROR_DELAY* a pakety odešle znovu. Maximální počet opakování paketu při chybách je definován makrem *MAX_PACKET_ERROR*.

```
#define MAX_PACKET_ERROR      5000              // (< 65535)
#define ERROR_DELAY           TIME_100us
```

Parametry jsou nastaveny tak, aby se funkce pokoušela paket odeslat po dobu 0,5 sekundy (bez uvažování doby odesílání paktů). Tento čas je volen experimentálně, a lze jej upravovat.

Po úspěšném odeslání požadavku, funkce přechází do datové fáze přenosu nebo do stavové fáze přenosu. V obou případech se přijímá datový paket (funkce bez datové fáze přenosu má ve stavové fázi vždy příjem paketu ze zařízení). Funkce odešle paket *IN* a ze zařízení následně bude přijat datový paket. Pokud se jednalo o přenos bez datové fáze přenosu, funkce končí úspěchem. V datové fázi přenosu se přijímají pakety o maximální velikosti brány *IN*. Kratší paket může odeslat zařízení jen jako poslední paket v datové fázi nebo když chce ukončit datovou fázi. V tomto případě funkce přejde do stavové fáze. V datové fázi funkce neustále opakuje příjem datových paketů, dokud není přijat požadovaný počet bajtů nebo dokud zařízení neukončí přenos krátkým paketem. Přijatá data postupně ukládá do pole *USB_Buffer*.

V případě, že zařízení odešle během datové fáze paket *STALL*, funkce končí s tímto výsledkem jako nepodporovaný požadavek. V případě, že zařízení odešle datový paket delší než velikost brány nebo paket delší než očekává funkce *USB_Control_Request*, funkce počká dobu definovanou makrem *REQUEST_DELAY* a celý požadavek se začne provádět znovu. Maximální počet pokusů o provedení požadavku je definován makrem *MAX_REQUEST_ERROR*. Tyto dva parametry lze upravovat, jsou určeny experimentálně.

```
#define MAX_REQUEST_ERROR     3                 // (< 255)
#define REQUEST_DELAY         TIME_100ms
```

V případě, že v přijatém datovém paketu je chyba nebo zařízení poslalo datový paket, který nebyl očekáván (lichý, sudý) podle bitu *Data_Toggle* (lokální proměnná), funkce počká dobu *ERROR_DELAY* a paket se pokusí přijmout znovu. V případě, že brána *IN* je zaneprázdněná a odeslala paket *NAK*, funkce počká dobu definovanou makrem *NAK_DELAY* a paket se pokusí opět přijmout znovu. Maximální počet pokusů o příjem každého paketu v datové fázi přenosu je *MAX_PACKET_ERROR*. Pokud nebude nějaký paket v datové fázi přenosu přijat i po *MAX_PACKET_ERROR* pokusech, celý požadavek se funkce pokusí provést znovu. Celkový počet opakování celého požadavku však nesmí překročit hodnotu *MAX_REQUEST_ERROR*.

```
#define NAK_DELAY             TIME_100us
```

Pokud funkce přijme všechna data nebo pokud byla datová fáze předčasně ukončena příjmem krátkého paketu, funkce přechází do stavové fáze přenosu. Počet skutečně přijatých bajtů ze zařízení je zaznamenán v globální proměnné *Control_Transfered*.

Při řídícím přenosu s příjmem dat ze zařízení v datové fázi, stavovou fázi tvoří paket *OUT* s datovým paketem *DATA1* s nulovou délkou dat odeslaný do zařízení. Pokud bude paket zařízením potvrzen (*ACK*), funkce úspěšně končí (vrací hodnotu 0). Pokud zařízení vrátí paket *STALL*, funkce končí s tímto výsledkem (vrací hodnotu 1). Pokud nastala jiná chyba, funkce čeká dobu *ERROR_DELAY*, nebo pokud brána vrátila *NAK*, funkce čeká dobu *NAK_DELAY* a paket se pokusí odeslat znovu s maximálním počtem opakování *MAX_PACKET_ERROR*. Nepodaří-li se paket odeslat, funkce se pokusí celý požadavek zopakovat.

Jsou-li vyčerpány všechny pokusy o provedení požadavku, funkce končí a vrací hodnotu 2, oznamující obecnou chybu. Po úspěšném provedení požadavku jsou případná přijatá data uložena v poli *USB_Buffer* a jejich počet je oznamován v proměnné *Contol_Transfered*. Přijatá data jsou v poli *USB_Buffer* pouze do dalšího použití jakékoli přenosové funkce z modulu *usb*.

10.3.2 Funkce USB_Bulk_DATA_OUT

Funkce *USB_Bulk_DATA_OUT* realizuje hromadný přenos dat typu *OUT* podle kap. 2.2.4. Realizuje tedy přenos dat z hostitele do zařízení hromadným typem přenosu. Tato funkce posílá pakety na bránu, jejíž adresa je uložena ve struktuře deskriptoru brány *Endpoint_Descriptor_OUT*. Před prvním použitím funkce je tedy nutné tuto strukturu nejdříve správně vyplnit.

```
unsigned char USB_Bulk_DATA_OUT(unsigned int Length);
```

Funkce *USB_Bulk_DATA_OUT* vrací rovněž parametr představující výsledek přenosu:

0 OK, přenesena všechna data,
1 zařízení vrátilo paket *STALL*, brána je pozastavena,
2 zařízení neustále vracelo paket *NAK* nebo nastala opakovaná chyba v přenosu.

Funkce má jediný vstupní parametr *Length*. Určuje, kolik bajtů z pole *USB_Buffer* má být přeneseno do zařízení. Před použitím funkce je tedy nutné připravit odesílaná data do pole *USB_Buffer* (od začátku). O rozdělení dat na jednotlivé pakety se funkce postará sama. Maximální velikost odesílaných dat nesmí překročit velikost pole *USB_Buffer*. Velikost paketu pro bránu *OUT* musí být předem uložena ve struktuře deskriptoru brány *Endpoint_Descriptor_OUT*.

Protože se jednotlivé datové pakety (lichý, sudý) musejí neustále střídat, je definován globální parametr představující *Data Toggle* bit brány *OUT* (bit *Data_Toggle_OUT*). Na počátku je nastaven na hodnotu *DATA0*, neboť hromadný přenos začíná po resetu vždy sudým datovým paketem. V parametru je během celé doby funkce hostitele uchovávána hodnota, jaký datový paket bude v příštím přenosu odeslán (sudý, lichý).

```
bit Data_Toggle_OUT = DATA0;
```

Funkce neustále odesílá pakety *OUT* následované datovými pakety *DATA0/1* (funkcí *USB_OUT_Token*), které se střídají, dokud nejsou přenesena všechna požadovaná data z pole *USB_Buffer*. V případě, že zařízení vrátí během přenosu paket *STALL*, je pozastavena brána a s tímto výsledkem končí také přenosová funkce (vrací hodnotu 1). Když na odeslaný paket vrátí zařízení paket *NAK*, funkce čeká dobu *NAK_DELAY*, nebo pokud nastane *time-out* nebo chyba, zařízení čeká dobu *ERROR_DELAY* a v obou případech se pokusí odeslat paket znovu. Maximální počet pokusů o přenos jednoho paketu do zařízení je *MAX_PACKET_ERROR*.

V případě vyčerpání maximálního počtu pokusů o přenos paketu funkce končí a vrací hodnotu 2 jako selhání. Při úspěšném odeslání všech bajtů vrací hodnotu 0 jako úspěch. Po ukončení funkce (i při pozastavení brány nebo jiném případě nedokončení přenosu) je počet úspěšně přenesených bajtů do zařízení uložen v globální proměnné *Bulk_Transfered*.

10.3.3 Funkce USB_Bulk_DATA_IN

Funkce *USB_Bulk_DATA_IN* realizuje hromadný přenos dat typu *IN* podle kap. 2.2.4. Realizuje přenos dat ze zařízení do hostitele hromadným typem přenosu. Funkce posílá a přijímá pakety z brány, jejíž adresa je uložena ve struktuře deskriptoru brány

Endpoint_Descriptor_IN. Před prvním použitím funkce je tedy nutné tuto strukturu nejdříve správně vyplnit.

```
unsigned char USB_Bulk_DATA_IN(unsigned int Length);
```

Funkce vrací parametr představující výsledek přenosu:

0 OK, přenesena všechna data (nebo data do ukončení krátkým paketem),
1 zařízení vrátilo paket *STALL*, brána je pozastavena,
2 zařízení neustále vracelo paket *NAK* nebo nastala opakovaná chyba v přenosu.

Funkce má jediný vstupní parametr *Length*, který určuje, kolik bajtů má být přeneseno ze zařízení do pole *USB_Buffer*. O rozdělení celkového množství přijímaných dat na jednotlivé pakety se funkce postará sama. Maximální velikost přijímaných dat nesmí překročit velikost pole *USB_Buffer*. Velikost paketu pro bránu *IN* musí být předem uložena ve struktuře deskriptoru brány *Endpoint_Descriptor_IN*.

Protože se jednotlivé datové pakety (lichý, sudý) musejí neustále střídat, je stejně jako pro přenos *OUT* definován globální parametr představující *Data Toggle* bit brány *IN*. Na počátku je nastaven na hodnotu *DATA0*, neboť hromadný přenos začíná po resetu vždy sudým datovým paketem. V parametru je během celé doby funkce hostitele uchovávána hodnota, jaký datový paket je v příštím přenosu přes bránu *IN* očekáván (sudý, lichý).

```
bit Data_Toggle_IN = DATA0;
```

Funkce neustále odesílá pakety *IN* na bránu hromadného přenosu *IN* následované datovými pakety *DATA0/1*, které posílá brána *IN* zpět hostiteli. Datové pakety se střídají, dokud nejsou přenesena všechna požadovaná data ze zařízení do pole *USB_Buffer*. V případě, že zařízení vrátí během přenosu paket *STALL*, je pozastavena brána a s tímto výsledkem končí také přenosová funkce (vrací hodnotu 1). Když na odeslaný paket *IN* vrátí zařízení paket *NAK*, funkce čeká dobu *NAK_DELAY*, nebo pokud nastane *time-out*, chyba přenosu nebo zařízení odeslalo datový paket, který nebyl podle bitu *Data_Toggle_IN* očekáván, funkce čeká dobu *ERROR_DELAY* a v obou případech se pokusí přijmout paket znovu. Maximální počet pokusů o přenos jednoho paketu ze zařízení je *MAX_PACKET_ERROR*.

V případě vyčerpání maximálního počtu pokusů o přenos paketu funkce končí a vrací hodnotu 2 jako selhání. Při úspěšném příjmu všech bajtů funkce vrací hodnotu 0 jako úspěch. Funkce vrací úspěch i v případě, že zařízení ukončilo přenos odesláním krátkého paketu. Skutečný počet přenesených bajtů ze zařízení do pole *USB_Buffer* v případě úspěchu přenosu je uložen v globální proměnné *Bulk_Transfered*.

10.3.4 Funkce USB_Device_Enumeration

Funkce *USB_Device_Enumeration* slouží k provedení enumerace (rozpoznání a nastavení) zařízení typu flash disk připojeného na USB sběrnici. K tomuto účelu využívá dříve popsanou funkci *USB_Control_Request* k provádění jednotlivých USB řídících požadavků.

```
bit USB_Device_Enumeration();
```

Funkce vrací pouze dvě možné hodnoty:

0 selhání vykonání některého požadavku nebo připojené zařízení není flash disk,
1 připojené zařízení je flash disk, podařilo se nastavit odpovídající konfiguraci.

V modulu *usb* je deklarována globální proměnná *Enumerated*, která po skončení funkce *USB_Device_Enumeration* obsahuje vrácenou hodnotu funkcí.

U požadavku *GET_DESCRIPTOR* je jako délka dat v datové fázi přenosu vždy zadána přesná délka daného deskriptoru, a proto počet skutečně přijatých bajtů není kontrolován. USB zařízení vždy posílá deskriptor celý, pokud by odeslalo pouze část deskriptoru (i když si hostitel zadal jeho celou délku), je to zcela netypický případ. Windows XP zase při enumeraci zadává při načtení deskriptoru zařízení a deskriptoru konfigurace délku dat 255 bajtů. Zařízení ovšem pošle pouze data platná pro daný deskriptor a přenos ukončí krátkým paketem.

Funkce *USB_Device_Enumeration* provede následující kroky:

1) Reset sběrnice USB. Funkce *USB_Engine_Reset* je dostupná v modulu *drv_usb_controller*. Reset sběrnice uvede připojené zařízení do výchozího stavu. USB adresa připojeného zařízení je nyní 0.

2) Ve struktuře *Device_Descriptor* nastaví parametr *wMaxPacketSize0* na hodnotu 8. To je minimální velikost brány 0 podle specifikace USB (v tomto okamžiku ještě není známa maximální velikost paketu pro řídící bránu). Odešle požadavek *SET_ADDRESS* na adresu 0 s požadovanou novou adresou 1. Od tohoto okamžiku budou všechny další pakety adresovány na adresu zařízení 1.

3) Požadavkem *GET_DESCRIPTOR* načte prvních 8 bajtů deskriptoru zařízení. Protože před tímto krokem není známa skutečná velikost brány, nelze načíst větší množství dat než 8 bajtů. Po načtení prvních osmi bajtů deskriptoru zařízení je již známa velikost řídící brány a tato je opravena ve struktuře deskriptoru zařízení. Nyní lze přenášet řídící bránou jakékoli podporované množství dat v jedné řídící transakci.

4) Požadavkem *GET_DESCRIPTOR* je přečten celý deskriptor zařízení a je vyplněna struktura deskriptoru zařízení *Device_Descriptor*. Nyní je znám počet možných konfigurací zařízení a zda je přítomen deskriptor textového řetězce sériového čísla.

5) Pokud připojené zařízení obsahuje deskriptor textového řetězce sériového čísla, prvních 16 znaků je uloženo do globálního textového řetězce *Serial_Number*. Pokud je sériové číslo kratší než 16 platných znaků, ostatní znaky jsou vyplněny znakem „0“. Pro přečtení se použije požadavek *GET_DESCRIPTOR* s indexem daného deskriptoru textového řetězce a načte se pouze první bajt deskriptoru, který udává jeho velikost. Znovu se použije požadavek *GET_DESCRIPTOR* a tentokrát je jako počet dat zadána získaná velikost. Přijatá data se z pole *USB_Buffer* vhodně překopírují do textového řetězce *Serial_Number*, přičemž se předpokládá, že jde o angličtinu a horní bajt z kódu *UNICODE* se neuvažuje.

6) Požadavkem *GET_DESCRIPTOR* načte 9 bajtů deskriptoru první konfigurace. Nyní je z načteného deskriptoru známa celková délka všech deskriptorů patřících k této konfiguraci. Znovu je použit požadavek *GET_DESCRIPTOR* pro načtení deskriptoru

konfigurace, ovšem funkce požaduje celkový počet dat, jehož hodnota byla získána z předešlé transakce. V poli *USB_Buffer* jsou nyní uloženy všechny deskriptory patřící k této konfiguraci počínaje deskriptorem konfigurace.

7) Funkce prozkoumá deskriptory konfigurace uložené v poli *USB_Buffer*. Hledá deskriptor rozhraní a jeho příslušných dvou bran, jehož parametry odpovídají parametrům uvedeným v kap. 3.1. Nalezne-li rozhraní odpovídající flash disku a jeho dvě brány, parametry přečtených deskriptorů jsou uloženy do struktury deskriptoru rozhraní, deskriptoru brány *IN* a deskriptoru brány *OUT*. V tomto okamžiku je rozhodnuto že je připojen flash disk, jsou známy adresy brány *IN* a *OUT* a také maximální velikosti paketů přes tyto brány.

8) Pokud v bodě 7) nebylo nalezeno rozhraní flash disku, funkce opakuje body 6), 7) pro další konfiguraci. Jsou-li prohledány neúspěšně všechny dostupné konfigurace, funkce končí a vrací hodnotu 0 (připojené zařízení neobsahuje rozhraní flash disku).

9) Pokud bylo v bodě 7) rozhodnuto, že připojené zařízení je flash disk a jsou úspěšně získány potřebné parametry pro komunikaci, požadavkem *SET_CONFIGURATION* nastaví konfiguraci, ve které bylo rozhraní flash disku nalezeno. Index dané konfigurace je získán z deskriptoru konfigurace. Nyní je zařízení ve zkonfigurovaném stavu a z největší pravděpodobností je schopné pracovat. Některé flash disky však vyžadují ještě nastavení rozhraní. Požadavkem *SET_INTERFACE* je tedy nastaveno alternativní rozhraní. Hodnota alternativního rozhraní pro požadavek je získána z nalezeného deskriptoru rozhraní flash disku (u většiny flash disků bude tato hodnota 0).

10) Globální proměnná *Enumerated* je nastavena na hodnotu 1. Funkce končí a vrací hodnotu 1 jako úspěšně rozpoznaný a nakonfigurovaný flash disk.

10.4 Programový modul „msc“

Modul *msc* obsahuje funkci realizující přenos pomocí transportního protokolu *Bulk-Only* a také funkce realizující použité *SCSI* příkazy podle kap. 4 a kap. 5.

Jsou deklarovány globální struktury *CBW* a *CSW*, které používá transportní funkce pro přenos. V modulu je rovněž deklarováno pole *MSD_Buffer*. Všechna data která jsou přenášena v **datové fázi** transportního protokolu *Bulk-Only* prochází přes tento zásobník. Pro jeho velikost platí stejná pravidla jako pro *USB_Buffer* modulu *usb*, musí se do něj vejít jeden celý sektor logické jednotky. S daty uložené v poli *MSD_Buffer* již pracuje modul *file* pro správu souborového systému.

```
xdata unsigned char MSD_Buffer[MAX_SECTOR_SIZE];
```

Pro realizaci přenosu transportním protokolem *Bulk-Only* používá modul funkce pro přenos dat hromadným přenosem *USB_Bulk_DATA_OUT*, *USB_Bulk_DATA_IN* a funkci *USB_Control_Request* z modulu *usb*.

10.4.1 Funkce MSD_Reset_Recovery

Zotavovací reset zařízení má být podle kap. 4.5 realizován řídícím požadavkem *Bulk-Only Mass Storage Reset* a požadavkem *CLEAR_FEATURE* pro resetování bran.

```
bit MSD_Reset_Recovery();
```

Funkce vrací dvě možné hodnoty:

0 reset úspěšně proveden,
1 reset se nepodařilo provést.

Experimenty s různými flash disky bylo zjištěno (a je na to upozorňováno i v [16]), že některé flash disky požadavek *Bulk-Only Mass Storage Reset* nepodporují. Při jeho aplikaci docházelo často k selhání funkce zařízení dokud nebylo odpojeno a znovu připojeno napájecí napětí. Proto je tato varianta ve zdrojovém kódu zakomentována, ale při použití u flash disků, které jej podporují, je plně funkční. Místo toho je použit přímo reset sběrnice USB, který rovněž uvede zařízení do výchozího stavu. Zotavovací reset zařízení je možné provést až po enumeraci zařízení. To umožňuje resetovací proceduru zjednodušit, protože všechny parametry pro komunikaci jsou již z enumerační procedury uloženy ve strukturách příslušných deskriptorů. Funkce *MSD_Reset_Recovery* použije po resetu sběrnice USB pouze požadavky *SET_ADDRESS*, *SET_CONFIGURATON* a *SET_INTERFACE*. Požadavky *CLEAR_FEATURE* pro resetování bran se v tomto případě nemusejí provádět. Protože je tímto zařízení uvedeno do výchozího stavu, funkce nastaví indikátory *Data_Toggle_OUT* a *Data_Toggle_IN* na hodnotu *DATA0*, aby byl hostitel synchronní s resetovaným zařízením.

10.4.2 Funkce MSD_Get_Num_Of_LUN

Funkce *MSD_Get_Num_Of_LUN* zjistí počet dostupných logických jednotek na flash disku. K tomuto účelu slouží řídící požadavek *GET_MAX_LUN*, který definuje třída *USB MSC*, viz kap. 4.2.

```
unsigned char MSD_Get_Num_Of_LUN();
```

Funkce může vracet tyto hodnoty:

0 nastala chyba při vykonání požadavku,
1-16 počet dostupných *LUN*.

10.4.3 Funkce MSD_Bulk_Only_Transport

Funkce *MSD_Bulk_Only_Transport* realizuje přenos transportním protokolem *Bulk-Only*. Před použitím funkce je třeba vyplnit strukturu *CBW* pro danou transakci. Pro analýzu přenosu po jeho skončení jsou uložena data ze stavové fáze přenosu do struktury *CSW*.

```
bit MSD_Bulk_Only_Transport();
```

Funkce vrací dvě možné hodnoty:

0 přenos úspěšný,
1 přenos selhal.

V případě této funkce neznamená úspěch transportu úspěch příkazu. Úspěch transportu znamená, že všechny jeho fáze proběhly podle definice transportního protokolu. Samotný příkaz mohl selhat, což je pak vráceno ve struktuře *CSW*. Analýzu úspěchu příkazu je nutné provést po úspěšném provedení transportu podle daných požadavků a hodnot ve struktuře *CSW*.

Pokud nastane jakákoli chyba při přenosu (chyba požadavku, chyba bulk přenosu, neplatné hodnoty v *CSW* apod.) funkce počká dobu *BULK_ONLY_DELAY* a provede celý transport znovu. Počet pokusů o provedení transportu je definován makrem *MAX_BULK_ONLY_ERROR*.

```
#define MAX_BULK_ONLY_ERROR         3                   (< 255)
#define BULK_ONLY_DELAY             TIME_100ms
```

Mezi jednotlivými pokusy o provedení celého transportu funkce aplikuje zotavovací reset zařízení (*MSD_Reset_Recovery*). Pokud by však tento reset selhal, transport končí a vrací hodnotu 1 jako selhání.

Na začátku přenosu funkce zkopíruje parametry struktury *CBW* do pole *USB_Buffer* a odešle je hromadným přenosem *OUT* (funkce *USB_Bulk_DATA_OUT*). Podle parametru *CBW.dCBWDataTransferLength* je patrné, zda příkaz obsahuje datovou fázi přenosu či nikoli. Neobsahuje-li datovou fázi přenosu, funkce pokračuje stavovou fází. Obsahuje-li datovou fázi přenosu, podle parametru *CBW.bCBWFlags* je indikováno, zda se jedná o přenos směrem *OUT* nebo *IN*. Oba typy přenosů vykonávají podobnou činnost.

Při přenosu *OUT* je požadovaný počet dat z pole *MSD_Buffer* zkopírován do pole *USB_Buffer* a tyto jsou odeslány na bránu *OUT* flash disku funkcí *USB_Bulk_DATA_OUT(CBW.dCBWDataTransferLength)*.

Při přenosu *IN* je požadovaný počet dat vyčten z brány *IN* flash disku funkcí *USB_Bulk_DATA_IN(CBW.dCBWDataTransferLength)*. Po úspěšném přenosu jsou přijatá data z pole *USB_Buffer* zkopírována do pole *MSD_Buffer*.

Pokud během těchto transakcí vrátí přenosová funkce pro přenos hodnotu *ST_STALL* (hodnota 1), zařízení pozastavilo příslušnou bránu hromadného přenosu. Funkce použije pro příslušnou bránu požadavek *CLEAR_FEATURE(ENDPOINT_HALT)* pro její resetování a nastaví *Data Toggle* bit příslušné brány v modulu *usb* na hodnotu *DATA0*. Při transportu *IN* se přijatá data zkopírují do pole *MSD_Buffer* a v obou případech se pak pokračuje stavovou fází přenosu. V případě, že během těchto transakcí vrátí přenosová funkce hromadného přenosu hodnotu *ST_NAK* (hodnota 2) jako chyba přenosu, celý transport bude zopakován znovu.

Přejde-li transport do stavové fáze, funkce se pokusí vyčíst ze zařízení status *CSW*. Algoritmus příjmu *CSW* je realizován dle specifikace *Bulk-Only* transportního protokolu viz kap. 4.8. Nepodaří-li se *CSW* přečíst napoprvé, funkce použije požadavek *GET_STATUS* pro bránu *IN*, kterým ověří zda je brána ve stavu *HALT*. Není-li ve stavu *HALT*, funkce projde kontrolou *CSW* a z největší pravděpodobností se celý transport bude opakovat. Je-li brána ve

stavu *HALT*, funkce použije požadavek *CLEAR_FEATURE(ENDPOINT_HALT)* pro bránu *IN* a pokusí se *CSW* načíst znovu. Nepodaří-li se načíst *CSW* ani napodruhé, celý transport se zopakuje.

Pokud se podařilo napoprvé nebo napodruhé *CSW* načíst, uloží se data z pole *USB_Buffer* do struktury *CSW* a provede se kontrola jeho smysluplnosti a platnosti podle kap. 4.4.1.

Pokud je *CSW* smysluplný a platný, otestují se některé parametry. Pokud je hodnota výsledku provádění *SCSI* příkazu *CSW.bCSWStatus* menší než 2 (příkaz vykonán nebo příkaz selhal) a současně je parametr *CSW.dCSWDataResidue <= CBW.dCBWDataTransferLength*, funkce končí a vrací hodnotu 0 jako úspěch přenosu. Pokud však nastala chyba fáze (výsledek vykonávání příkazu je 2) nebo neplatí výše uvedená podmínka, což znamená chybu, celý transport se zopakuje. Po vyčerpání *MAX_BULK_ONLY_ERROR* pokusů o provedení transportu funkce končí a vrací 1 jako selhání transportu.

10.4.4 Funkce SCSI_READ10

Funkce *SCSI_READ10* slouží ke čtení sektorů z logické jednotky flash disku. Funkce přečte jeden zadaný sektor a uloží jej do pole *MSD_Buffer*.

```
bit SCSI_READ10(unsigned char LUN, unsigned long int LBA, unsigned int
Sector_Size)
```

Funkce vrací dvě možné hodnoty:

0 sektor úspěšně přečten,
1 načtení sektoru selhalo.

Funkce má tři vstupní parametry. *LUN* je index logické jednotky flash disku, pro kterou je příkaz adresován, *LBA* je logická adresa čteného sektoru a *Sector_Size* je velikost jednoho sektoru adresované *LUN* v bajtech. Modul *msc* deklaruje globální proměnnou *Loaded_Sector*:

```
xdata unsigned long int Loaded_Sector;
```

V této proměnné se uchovává adresa *LBA* naposledy přečteného sektoru. U některých funkcí se může stát, že by byl opakovaně čtený jeden sektor. Pokud budou data z posledního čtení sektoru v poli *MSD_Buffer* nezměněna, není třeba sektor načítat zbytečně znovu. Všechny funkce, které zasahují (přepisují) do pole *MSD_Buffer*, nastavují proměnnou *Loaded_Sector* na výchozí hodnotu 0. Sektor s *LBA* adresou 0 bude funkcí *SCSI_READ10* přečten z disku vždy znovu. Při přepnutí programu na jinou logickou jednotku (viz kap. 10.5.4) je *Loaded_Sector* rovněž nastaven na výchozí hodnotu 0.

Na začátku vyplní funkce strukturu *CBW* dle kap. 5.1.2. Hodnota *dCBWTag* je neustále zvětšována o konstantu, aby byla v každém přenosu jiná. Počet přenášených dat je nastaven na vstupní parametr *Sector_Size*. Bit *Direction* je nastaven na přenos *IN*. Jako číslo adresované logické jednotky je zadán vstupní parametr *LUN*. Do pole *CBWCB* je uložen *SCSI* příkaz *READ(10)* podle kap. 5.1.2. Parametr *bCBWCBLength* je tedy nastaven na délku *CDB*

příkazu *READ(10)* (10 bajtů). V příkazu je zadána adresa *LBA* čteného sektoru a počet čtených sektorů je nastaven na hodnotu 1.

Je zavolána transportní funkce *MSD_Bulk_Only_Transport*. Bude-li provedena s negativním výsledkem, funkce *SCSI_READ10* končí a vrací hodnotu 1 jako selhání.

V případě úspěchu transportu je provedena analýza výsledku vykonání příkazu v *CSW*. Příkaz je považován za úspěšně provedený pokud *bCSWStatus* hlásí úspěch a *dCSWDataResidue* je nula. Protože flash disk nemá pravděpodobný důvod, aby příkaz neprovedl korektně, např. poslal menší množství dat než je velikost sektoru, každá jiná varianta výsledku se považuje za chybnou.

10.4.5 Funkce SCSI_WRITE10

Funkce *SCSI_WRITE10* slouží k zápisu sektorů na logickou jednotku flash disku. Funkce zapíše do jednoho zadaného sektoru na disku data z pole *MSD_Buffer*.

```
bit SCSI_WRITE10(unsigned char LUN, unsigned long int LBA, unsigned int
Sector_Size);
```

Funkce vrací dvě možné hodnoty:

0 sektor úspěšně zapsán,
1 zápis sektoru selhal.

Funkce má tři vstupní parametry. *LUN* je index logické jednotky flash disku, pro kterou je příkaz adresován, *LBA* je logická adresa zapisovaného sektoru a *Sector_Size* je velikost jednoho sektoru adresované *LUN* v bajtech.

Funkce pracuje obdobně jako funkce *SCSI_READ10*, rozdíly v nastavení struktury *CBW* jsou minimální. Bit *Direction* je nastaven na typ přenosu *OUT* a v *CBWCB* je uložen příkaz *SCSI WRITE(10)* popsán v kap. 5.1.3. Opět je příkaz nastaven na zápis jediného sektoru.

Úspěch výsledku funkce je vyhodnocen stejně jako v případě funkce *SCSI_READ10*.

10.4.6 Funkce SCSI_READ_CAPACITY10

Funkce *SCSI_READ_CAPACITY10* slouží k vykonání příkazu *READ CAPACITY(10)* popsaného v kap. 5.1.1.

```
bit SCSI_READ_CAPACITY10(unsigned char LUN);
```

Funkce vrací dvě možné hodnoty:

0 příkaz úspěšně proveden,
1 příkaz selhal.

Funkce má jediný vstupní parametr *LUN*, což je číslo adresované logické jednotky flash disku. Nastavení struktury *CBW* je obdobné jako v případě funkce *SCSI_READ10*, jen počet přenášených bajtů je nastaven na hodnotu 8, a v poli *CBWCB* je uložen *CDB* příkazu *READ CAPACITY(10)*, viz kap. 5.1.1.

Vyhodnocení úspěchu provedení příkazu je shodné jako v případě předchozích dvou funkcí. Po úspěšném provedení funkce jsou vyčtená data z flash disku uložena v poli *MSD_Buffer*.

10.5 Programový modul „file“

Pomocí programových modulů *msc*, *usb* a *drv_usb_controller* lze číst a zapisovat jednotlivé sektory na logických jednotkách flash disku. Programový modul *file* obsahuje uživatelské funkce podporující práci s logickou jednotkou flash disku naformátovanou na souborový systém *FAT16* popsaný v kap. 6.

Modul obsahuje 38 funkcí, ze kterých zde budou popsány funkce pro aplikační použití modulu.

10.5.1 Přehled uživatelských funkcí

V tab. 33 je uveden přehled všech uživatelských funkcí modulu *file*. Pouze uživatelská funkce *USB_Device_Detect* je součástí modulu ovladače USB řadiče *drv_usb_controller*.

Veškerou uživatelskou práci s flash diskem zprostředkovávají funkce uvedené v tab.33. Ostatní užitečné parametry jsou uvedeny v příslušných datových strukturách modulu *file*. Funkce jsou rozděleny na tři skupiny, podle druhu činnosti.

Protože mikroprocesorový systém s architekturou 8051 je relativně malý, programový modul *file* pro zjednodušení nepracuje s celými cestami umístění souboru. Obsahuje pro pohyb v adresářové struktuře funkce pro vnoření a vynoření o jednu úroveň. Program si poté pamatuje pro každou logickou jednotku zvlášť, ve kterém adresáři aktuálně je a všechny ostatní operace vztahuje k tomuto adresáři.

Flash disk může obsahovat více logických jednotek. Pokud je program zkompilován pro podporu více *LUN*, při práci je vždy nastaven na jednu určitou logickou jednotku. K přepínání na různé logické jednotky slouží funkce *Curr_LUN*. Funkce pracující s otevřeným souborem pracují správně i když je program aktuálně přepnut na jinou *LUN*, než na které leží otevřený soubor.

Tab. 33 Přehled uživatelských funkcí pro práci s flash diskem

Funkce	Popis činnosti
Systémové funkce	
USB_Device_Detect	Funkce slouží pro zjištění, zda je na sběrnici USB připojeno zařízení *FS* .
Start_Flash_Disk	Počáteční celková inicializace připojeného flash disku a rozpoznání souborových systémů na jednotlivých dostupných logických jednotkách.
Curr_LUN (n)	Nastavení aktuální logické jednotky, se kterou program pracuje na jednotku číslo n.
Get_Free_Space	Zjištění volného místa v bajtech na aktuální logické jednotce flash disku.
Operace s adresáři	
CD_ROOT	Nastaví aktuální adresář, se kterým program pracuje, na kořenový adresář aktuální logické jednotky.
CD ("NAME")	Vnoří se z aktuálního adresáře do adresáře NAME, který je umístěný v aktuálním adresáři.
CD_UP	Vynoření v adresářové struktuře o úroveň výše.
DLD ("NAME")	Smaže z aktualního adresáře adresář NAME včetně jeho obsahu.
MKD ("NAME")	Vytvoří v aktuálním adresáři adresář NAME. Pokud již existuje adresář takového názvu, nebude vytvořen.
LIST_RESET	Nastaví strukturu LIST programu na začátek aktuálního adresáře. Je to párová funkce k funkci LIST_NEXT.
LIST_NEXT	Vyhledá v aktuálním adresáři nejbližší adresář nebo soubor od pozice definované ve struktuře LIST a uloží parametry hlavičky do struktury LIST. Opakovaným voláním lze vytvářet seznamy souborů a adresářů v aktuálním adresáři.
Operace se soubory	
REWRITE_FILE (F, "NAME")	Vytvoří a otevře v aktuálním adresáři soubor NAME s parcovní proměnnou F. Existuje-li již soubor stejného názvu bude smazán. Soubor je po vytvoření připraven pro zápis i čtení.
OPEN_FILE (F, "NAME")	Otevře v aktuálním adresáři soubor NAME s pracovní proměnnou F pro čtení i zápis.
CLOSE_FILE (F)	Uzavře otevřený soubor používající pracovní proměnnou F.
READ_FILE (F, B, L)	Načte z otevřeného souboru používající pracovní proměnnou F počet bajtů L do pole B.
WRITE_FILE (F, B, L)	Zapíše na konec otevřeného souboru používající pracovní proměnnou F počet bajtů L z pole B.
FILE_POINTER (F, O)	Nastaví ukazatel pro čtení z otevřeného souboru používající proměnnou F na ofset O od začátku souboru.
RENAME (X , NAME, NEW)	Přejmenuje soubor nebo adresář NAME v aktuálním adresáři na jméno NEW. Parametr X slouží pro specifikaci požadavku zda přejmenovat soubor nebo adresář.
DLF ("NAME")	Smaže z aktualního adresáře soubor NAME.
Set_File_Info (…)	Slouží pro uživatelsky rychlé nastavení struktury FILE_INFO, podle které se ukládají specíální parametry do hlaviček souborů a adresářů (atributy, datum, čas).

10.5.2 Názvy souborů a chybové kódy

Dle požadavků zadání, modul nepracuje se vstupy *LFN*. Všechny názvy souborů nebo adresářů zadávané do jednotlivých uživatelských funkcí musí splňovat formát *DOS8.3*. Jméno musí mít **1 – 8** znaků, přípona **0 – 3**. Jméno je od přípony v zadávaném názvu odděleno tečkou. Pokud jméno nemá příponu, tečku neobsahuje. Je nutné používat pouze znaky velké abecedy, číslovky popř. některé další speciální znaky. Nejdelší název má tedy 12 znaků, 8 jméno + 1 tečka + 3 přípona, nejkratší název má pouze 1 znak jména. Jména jsou vždy textové řetězce, jsou ukončeny číslem 0, tedy řetězec pro název může mít maximálně 13 bajtů. Do žádné z funkcí modulu *file* nesmí být zadán název v jiném formátu.

Názvy bez přípony:	Názvy s příponou:
A	**A.B**
A254	**A254.JP**
AHOJ	**AHOJ.AVI**
TEPLOTA1	**TEPLOTA1.DAT**
DATAPR~1	**DATAPR~1.XLS**

Některé uživatelské funkce z tab. 33 podávají za určitých okolností chybová hlášení ve formě chybového kódu, který upřesňuje okolnosti vzniku chyby. Je deklarována globální proměnná pro uchování chybového kódu.

```
xdata unsigned char Error_Code;
```

V proměnné *Error_Code* je uchován chybový kód naposledy použité uživatelské funkce, která jej ovlivňuje. V tab. 34 jsou uvedeny možné chybové kódy. Chybový kód je potřeba vyhodnotit v případě, že daná funkce hlásí selhání. V jiném případě není hodnota v proměnné *Error_Code* platná.

Tab. 34 Přehled chybových kódů

Kód	Název (makro)	Chyba
0	ERROR_0	Nenastala žádná chyba
1	ERROR_1	Zadaný adresář nebo soubor neexistuje
2	ERROR_2	Zadaný adresář již existuje
3	ERROR_3	Na disku není potřebné volné místo
4	ERROR_4	Zadaná pracovní proměnná souboru je již používána
5	ERROR_5	Zadaná pracovní proměnná souboru není používána
6	ERROR_6	Chyba ve FAT
7	ERROR_7	Čtecí ukazatel je na konci souboru
8	ERROR_8	Zadaný ofset je větší než velikost souboru
9	ERROR_9	Dosaženo konce adresáře
10	ERROR_10	Nelze přejít o úroveň výše, aktualni adresář zůstává
11	ERROR_11	Chyba ve FAT aktuálního adresáře

ERROR_0 – Nenastala žádná chyba. Pokud uživatelská funkce nastaví tento kód, skončila úspěšně nebo nastala chyba v komunikaci, viz kap. 10.5.6.

ERROR_1 – Nastane pokud zadaný název adresáře nebo souboru není nalezen v aktuálním adresáři.

ERROR_2 – Nastane pokud se uživatelská funkce *MKD* pokusí vytvořit adresář, který v aktuálním adresáři již existuje.

ERROR_3 – Nastane pokud při vytvoření adresáře či souboru nebo při zápisu do souboru není na disku potřebný počet volných clusterů. Je to obecná chyba indikující nedostatečné volné místo na disku pro dokončení dané operace.

ERROR_4 – Nastane pokud při otevírání souboru nebo vytváření nového souboru je zadaná pracovní proměnná aktuálně používána pro manipulaci s jiným souborem. Tato chyba může nastat při neopatrně psaném algoritmu pro práci se soubory. Je to ochrana dat na disku při vývoji uživatelského programu.

ERROR_5 – Nastane pokud při zápisu do souboru, čtení ze souboru nebo nastavení čtecího ukazatele souboru není zadaná pracovní proměnná souboru aktuálně používána – není asociována s žádným souborem.

ERROR_6 - Chyba ve *FAT* se vyskytne tehdy, pokud záznam ve *FAT* příslušného souboru, je kratší, než by odpovídalo jeho velikosti v hlavičce. S takovým souborem nelze pracovat, je možné ho však smazat.

ERROR_7 – Nastane pokud při čtení ze souboru je již čtecí ukazatel z daného souboru na jeho konci (počet přečtených bajtů je 0). Nastane např. pokud při čtení ze souboru bude dosaženo jeho konce a poté bude čteno ze souboru znovu.

ERROR_8 – Tato chyba je spjatá s funkcí *FILE_POINTER*. Nastane pokud je zadaný ofset pro nastavení čtecího ukazatele větší než velikost samotného souboru.

ERROR_9 – Tato chyba je spjatá s funkcí *LIST_NEXT*. Nastane, pokud funkce *LIST_NEXT* při vyhledávání vstupů adresářů a souborů v aktuálním adresáři dosáhla jeho konce. K nastavení funkce na začátek adresáře je nutné použít funkci *LIST_RESET*.

ERROR_10 – Chyba nelze přejít o úroveň výše vznikne tehdy, pokud se uživatelskou funkcí *CD_UP* (popř. *CD(„..“)*) chceme vynořit o úroveň výše, ale neexistuje hlavička adresáře, do kterého se funkce vynořuje. Tato chyba je dost specifická a pravděpodobnost jejího vzniku je minimální. Funkce s tímto případem však pro každý případ počítá.

ERROR_11 - Chyba ve *FAT* aktuálního adresáře vznikne tehdy, pokud prohledávaný adresář (např. při otevírání souboru, mazání, prohledávání atd.) obsahuje ve svém záznamu ve *FAT* číslo rezervovaného nebo špatného clusteru. Tyto hodnoty viz tab. 24 nesmějí v záznamu adresáře být a tedy záznam adresáře je chybný. S takovým adresářem by se nemělo dále pracovat, minimálně by se v něm neměly vytvářet nové soubory nebo adresáře.

10.5.3 Funkce Start_Flash_Disk

Funkce *Start_Flash_Disk* slouží k počáteční celkové inicializaci flash disku. Funkce nemá žádný vstupní parametr a nevrací žádnou hodnotu. Používá funkce a proměnné z modulů *msc*, *usb* a *drv_usb_controller*.

Hlavním výstupem funkce je počet dostupných logických jednotek připojeného flash disku v proměnné *Number_Of_LUNs*. Dostupná jednotka je jednotka používající souborový systém *FAT16* přičemž její *FAT16BR* sektor obsahuje všechny potřebné a platné parametry.

```
xdata unsigned char Number_of_LUNs;
```

Modul *file* také deklaruje pole *LUN* struktury *LOGICAL_UNIT*. Do tohoto pole uloží funkce všechny potřebné parametry dostupných logických jednotek a také obsahuje další parametry pro další práci s těmito jednotkami. Velikost pole *LUN* určuje, kolik logických jednotek najednou program podpoří. V kap. 3.2 bylo zmíněno, že flash dik může mít až 16 logických jednotek. Protože většina flash disků má pouze jednu nebo dvě logické jednotky, není třeba v modulu *file* zbytečně kompilovat pole *LUN* s rozměrem 16. Zbytečně by se tak zabírala datová paměť mikrokontroléru, která by nebyla využita. Rozměr tohoto pole (a tedy i počet podporovaných logických jednotek) může být zvolen uživatelem pomocí makra *MAX_LUNS*.

```
#define MAX_LUNS        2

xdata LOGICAL_UNIT      LUN[MAX_LUNS];
```

Pokud bude mít flash disk více jednotek se souborovým systémem *FAT16* než rozměr pole *LUN*, bude akceptován pouze počet jednotek do rozměru pole *LUN* s nejnižšími indexy na disku. Jeden prvek pole *LUN* má následující strukturu (typ *LOGICAL_UNIT*). Obsahuje zajímavé parametry, se kterými je možné případně dále pracovat mimo modul *file*.

```
typedef struct LOGICAL_UNIT
{
 unsigned char Index_LUN;
 unsigned char File_System;
 unsigned long int Start_Partition;
 unsigned long int Sectors;
 unsigned char Sectors_Per_Cluster;
 unsigned long int Start_FAT;
 unsigned char FATs;
 unsigned int Max_Root_Directory_Entries;
 unsigned int Number_Of_Root_Sectors;
 unsigned int Sectors_Per_FAT;
 unsigned char Disk_Volume[11];
 unsigned char FAT_Name[8];
 unsigned long int Start_Root;
 unsigned long int Start_Data_Area;
 unsigned int Clusters;
 unsigned int Current_Directory_Cluster;
 unsigned char Current_Directory[MAX_LENGTH_OF_ENTRY_NAME];
 unsigned int FAT_Pointer;
 unsigned int Sector_Size;
 unsigned char Number_Of_Entry_Per_Sector;
 unsigned int Number_Of_Cluster_Per_Sector;
} LOGICAL_UNIT;
```

Index_LUN – Index logické jednotky na flash disku. Dostupné logické jednotky v poli *LUN* jsou indexovány od 0, ale indexy těchto jednotek na flash disku se mohou lišit. Proto se musí uchovávat fyzický index logické jednotky pro použití v přenosu protokolem *Bulk-Only*.

***LUN* flash disku**	**Souborový systém**	**Index dané jednotky v poli *LUN***
0	**FAT16**	0
1	**FAT16**	1
2	FAT32	-
3	**FAT16**	2
4	FAT32	-
5	NTFS	-
6	FAT32	-
7	**FAT16**	3
…	…	…

File_System – Typ souborového systému. 0x04 je *FAT16* menší než 32MB, 0x06 je *FAT16* větší než 32MB.
Start_Partition – Číslo sektoru začátku *partition*. Pokud je *LUN* flash disku naformátována bez *MBR* sektoru, *Start_Partition* bude 0.
Sectors – Celkový počet sektorů v *partition*.
Sectors_Per_Cluster – Počet sektorů v jednom clusteru.
Start_FAT – Číslo sektoru začátku první FAT.
FATs – Počet FAT.
Max_Root_Directory_Entries – Maximální počet záznamů v kořenovém adresáři (typicky 512 vstupů).
Number_Of_Root_Sectors – Počet sektorů, který zabírá kořenový adresář.
Sectors_Per_FAT – Počet sektorů, který zabírá jedna FAT.
Disk_Volume[11] - *ASCII* řetězec názvu *partition*. Nevyužité znaky jsou vyplněny znakem mezera. Pole *Disk_Volume* není ukončen hodnotou 0, nelze s ním tedy v C zacházet jako s textovým řetězcem.
FAT_Name[8] - *ASCII* řetězec názvu souborového systému. Pro *FAT16* je uložen řetězec „FAT16 ". Pro práci s tímto polem platí totéž co v předchozím případě.
Start_Root – Číslo sektoru začátku kořenového adresáře.
Start_Data_Area – Číslo sektoru začátku datové oblasti disku.
Clusters – Celkový počet clusterů v *partition*.
Current_Directory_Cluster – Číslo prvního clusteru aktuálního adresáře. Tato proměnná uchovává pozici při pohybu v adresářové struktuře logické jednotky funkcemi *CD*, *CD_UP* a *CD_ROOT*. Pro uchování pozice stačí číslo prvního clusteru daného adresáře.
Current_Directory[MAX_LENGTH_OF_ENTRY_NAME] - Textový řetězec obsahující jméno aktuálního adresáře dané logické jednotky. Řetězec je ukončen hodnotou 0, v jazyce C na něj lze aplikovat funkce pro práci s textovými řetězci.
FAT_Pointer – Ukazatel na buňky *FAT*. Uchovává poslední pozici ve *FAT* kde byl naposledy nalezen volný cluster. Při dalším hledání se bude pokračovat od této buňky.
Sector_Size – Velikost jednoho sektoru dané logické jednotky v bajtech.
Number_Of_Entry_Per_Sector – Počet vstupů souborů (*Entry*), který se vleze do jednoho sektoru jednotky.
Number_Of_Cluster_Per_Sector – Počet buněk *FAT*, který se vleze do jednoho sektoru jednotky.

Pokud bude uživatelský program pracovat pouze s jedinou logickou jednotkou flash disku (bude pracovat např. pouze s *Current_LUN* = 0, *LUN[0]*), je vhodné definovat makro *WORK_WITH_ONE_LUN*.

```
#define WORK_WITH_ONE_LUN
```

Je-li toto makro definováno, mírně bude upravena kompilace uživatelských funkcí *WRITE_FILE*, *READ_FILE* a *CLOSE_FILE*. Následkem bude jisté urychlení těchto funkcích v určitých situacích. Pokud je však makro *WORK_WITH_ONE_LUN* definováno, je nepřípustné používat tyto funkce ještě v jiných logických jednotkách, které by mohl program podporovat (pokud je makro *MAX_LUNS* > 1). V určitých situacích by mohla nastat chybná funkce zmiňovaných funkcí a tím poškození dat v logické jednotce.

Když bude uživatelský program napsán pouze pro práci s jedinou logickou jednotkou (což bude pravděpodobně většina případů) makro *WORK_WITH_ONE_LUN* nemusí být definováno, ale pro urychlení funkcí pro práci se souborem je to vhodné.

Funkce *Start_Flash_Disk* provede několik operací. Nastaví aktuální *LUN* programu (*Current_LUN*, viz kap. 10.5.4), *Data Toggle* obou bran hromadného přenosu a počet dostupných logických jednotek (*Number_Of_LUNs*) na hodnotu 0 pro výchozí stav.

Funkcí *USB_Device_Detect* ověří zda je připojeno zařízení *FS* na USB sběrnici. Není-li, funkce končí a počet dostupných jednotek je 0. Je-li připojeno *FS* zařízení, volá funkci *USB_Device_Enumeration* z modulu *usb*. Při neúspěšné enumeraci funkce opět skončí a počet dostupných logických jednotek bude 0. Je-li enumerace provedena úspěšně (je připojen flash disk), volá funkci *MSD_Get_Num_Of_LUN* z modulu *msc* pro zjištění celkového počtu logických jednotek flash disku.

Poté funkce začne procházet všechny dostupné logické jednotky flash disku dokud nenaplní pole *LUN* informacemi o dostupných jednotkách s *FAT16*. Nejdříve použije funkci *SCSI_READ_CAPACITY10* pro zjištění velikosti jednoho sektoru první logické jednotky. Tuto hodnotu si uloží do parametru *Sector_Size* pole *LUN*, protože je potřeba pro použití funkce pro přečtení sektoru. Pokud je sektor jednotky větší než podporuje program (makro *MAX_SECTOR_SIZE*), jednotka se přeskočí a přejde se na další.

Nyní se načte první sektor logické jednotky flash disku (sektor s *LBA* adresou 0). Tento sektor může být buď *MBR* nebo *FSBR*. V obou případech musí být ukončen znakem *Executable Mark*, který indikuje platnost sektoru. *MBR* i *FSBR* sektor mají velikost 512 bajtů. Pokud má logická jednotka sektor větší, jen prvních 512 bajtů představuje *MBR* nebo *FSBR*, ostatní bajty jsou v sektoru nastaveny na hodnotu 0. Rozlišení, o který ze sektorů se jedná, se provede podle prvních tří bajtů, které v obou případech vždy obsahují specifické instrukce procesoru PC. Pokud je první bajt **0xEB** což je instrukce skoku a třetí bajt je **0x90** což je NOP, jedná se o *FSBR* sektor, v jiném případě je to *MBR*, který na této pozici obsahuje instrukci pro manipulaci se zásobníkem nebo nuly v případě, že neobsahuje vykonávací kód. Instrukce skoku ve *FSBR* má skákat na vykonávací kód, který je v zadní části sektoru. I v případě, že *FAT16BR* sektor tento vykonávací kód neobsahuje, instrukce skoku je na začátku vždy přítomná.

Pokud se jedná o *MBR*, podle parametrů tabulky první primární *partition* je ověřeno, zda se jedná o souborový systém *FAT16*. Pokud ano, podle parametrů tabulky v *MBR* je načten první sektor *partition* představující *FSBR* sektor.

Pokud *MBR* není přítomen, jediný způsob ověření zda se jedná o souborový systém *FAT16* je z textového řetězce souborového systému z tabulky *FSBR* sektoru.

Poté jsou z tabulky *FSBR* uloženy a případně dopočítány všechny potřebné parametry do pole *LUN*. Aktuální adresář je nastaven na hodnotu 0, což je kořenový adresář. Do názvu aktuálního adresáře je zapsán řetězec „ROOT“.

Funkce *Start_Flash_Disk* skončí po projití všech logických jednotek flash disku nebo naplnění pole *LUN*. V proměnné *Number_Of_LUNs* bude uložen celkový počet dostupných logických jednotek, přičemž program je nastaven pro práci s logickou jednotkou, která má v poli *LUN* index 0.

10.5.4 Funkce Curr_LUN

Modul *file* umožňuje pracovat s více logickými jednotkami flash disku. Aby nebylo nutné do každé uživatelské funkce zadávat index logické jednotky, pro kterou má být funkce použita, index aktuální logické jednotky je nastaven globálně v proměnné *Current_LUN*.

```
xdata unsigned char Current_LUN;
```

Téměř všechny uživatelské funkce pracují s tou logickou jednotkou pole *LUN* jejíž index je uložen v proměnné *Current_LUN*. Tato proměnná slouží jen pro čtení. K přepnutí programu na jinou logickou jednotku slouží funkce *Curr_LUN*.

```
void Curr_LUN(unsigned char LUN);
```

Funkce *Curr_LUN* má jediný vstupní parametr a to index nové logické jednotky v poli *LUN*. Jeho možný rozsah je od 0 do (*Number_Of_LUNs* – 1). Pojem „aktuální logická jednotka“ bude dále v textu znamenat logická jednotka, na kterou je program aktuálně nastaven v proměnné *Current_LUN*.

10.5.5 Funkce Get_Free_Space

Funkce *Get_Free_Space* slouží ke zjištění volného místa v aktuální logické jednotce. Pro uchování počtu volných bajtů je deklarována globální proměnná *Free_Space*.

```
xdata unsigned long int Free_Space;
```

Voláním funkce *Get_Free_Space* se hodnota v proměnné *Free_Space* aktualizuje podle aktuálního stavu volného místa aktuální logické jednotky. Funkce zjistí počet volných clusterů v první *FAT* a vynásobí jej počtem bajtů v jednom clusteru. Proměnná *Free_Space* je společná pro všechny jednotky, se kterými program pracuje.

Hodnota volného místa nabývá diskrétních hodnot po velikosti clusteru. Pokud je však *Free_Space* = 1, funkce tímto indikuje chybu v komunikaci s flash diskem.

10.5.6 Společné vlastnosti funkcí pro práci se soubory a adresáři

Většina funkcí pro práci s adresáři a soubory z tab. 33 vrací jako parametr výsledek operace dané funkce ve formě čísla datového typu *unsigned char*. Platí to pro funkce:

CD
CD_UP
DLF
DLD
MKD
REWRITE_FILE
OPEN_FILE
CLOSE_FILE
READ_FILE
WRITE_FILE
FILE_POINTER
RENAME
LIST_NEXT

Všechny tyto uživatelské funkce jsou typu *unsigned char* a pro vyhodnocení různých situací vrací tři možné hodnoty:

0 funkce proběhla úspěšně,
1 funkce selhala, kód chyby je uložen v proměnné *Error_Code*,
2 nastala chyba v komunikaci s flash diskem.

Pokud funkce vrátí hodnotu 0, daná operace proběhla úspěšně. Vrátí-li nějaká funkce hodnotu 1, správnost vykonání operace již není zaručena nebo to přímo znamená selhání dané operace. Konkrétní příčina selhání funkce je pak k dispozici v proměnné *Error_Code* popsané v kap. 10.5.2. Pro každou funkci jsou relevantní jen některé chybové kódy. Uživatelský program může podle chybového hlášení provést patřičné kroky k nápravě.

Pokud však funkce vrátí hodnotu 2, nastala chyba v komunikaci, se kterou si opravné algoritmy transportního a USB přenosového protokolu nedokázaly poradit. Chyba v komunikaci může také nastat z důvodu selhání samotného *SCSI* příkazu ve flash diku. Protože funkce modulu *msc* neakceptují jiný výsledek než úspěšné vykonání příkazu, v případě chybového hlášení flash disku nedokáží funkce modulu situaci řešit. V této situaci je zřejmě nejlepším řešením znovu inicializovat flash disk funkcí *Start_Flash_Disk* a požadavek opakovat. Pokud to hardware umožňuje, bylo by dobré také odpojit a znovu připojit napájecí napětí sběrnice USB (některé flash disky se mohou „zaseknout“ až do odpojení napájecího napětí).

10.5.7 Funkce CD_ROOT

Funkce *CD_ROOT* slouží k nastavení aktuálního adresáře logické jednotky na kořenový adresář (*Root Directory*).

```
void CD_ROOT();
```

Funkce nemá žádný vstupní parametr ani žádný nevrací. Do názvu aktuálního adresáře bude uloženo jméno „ROOT“. Funkce tedy slouží k přemístění z libovolného místa adresářové struktury přímo do kořenového adresáře.

10.5.8 Funkce CD

Funkce *CD* slouží ke vnoření se do adresáře ležícího v aktuálním adresáři logické jednotky. Při úspěšném vnoření se tento adresář stává aktuálním.

```
unsigned char CD(unsigned char *Name);
```

Funkce vrací parametr popsaný v kap. 10.5.6. Funkce *CD* má jeden vstupní parametr a to jméno cílového adresáře, který musí ležet v aktuálním adresáři. Speciální případ je, když je do funkce zadán adresář „..“ (dvě tečky). V tomto případě se funkce vynoří o úroveň výše.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_1 - zadaný adresář v aktuálním adresáři neexistuje,
ERROR_10 - nelze přejít o úroveň výše,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

10.5.9 Funkce CD_UP

Funkce *CD_UP* slouží k vynoření se v adresářové struktuře do adresáře o úroveň výše a tento se pak stává adresářem aktuálním.

```
unsigned char CD_UP();
```

Funkce vrací parametr popsaný v kap. 10.5.6 a nemá žádný vstupní parametr. Pokud je aktuálním adresářem kořenový adresář, volání funkce nemá žádný efekt.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_10 - nelze přejít o úroveň výše,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

Zde je nutno podotknout že *ERROR_11* se zde nevztahuje k aktuálnímu adresáři, ale k adresáři, ve kterém se nachází hlavička adresáře, do kterého se funkce vynořuje.

10.5.10 Funkce DLD

Funkce *DLD* slouží ke smazání adresáře z aktuálního adresáře logické jednotky. Funkce smaže i veškerý obsah daného adresáře, proto doba jejího vykonávání závisí na velikosti adresářové struktury a množství a velikosti souborů v daném adresáři.

```
unsigned char DLD(unsigned char *Name);
```

Funkce vrací parametr popsaný v kap. 10.5.6. Funkce *DLD* má jeden vstupní parametr a to jméno mazaného adresáře, který musí ležet v aktuálním adresáři. Funkce pracuje tak, že se vnoří do mazaného adresáře uživatelskou funkcí *CD* a postupně v něm maže jednotlivé soubory. Narazí-li na adresář, vnoří se do něj a od začátku opět maže jednotlivé soubory. Smaže-li veškerý obsah podadresáře, vynoří se o úroveň výše uživatelskou funkcí *CD_UP* a smaže hlavičku podadresáře. Takto funkce pokračuje dokud není smazaný veškerý obsah mazaného adresáře. Nakonec je smazána i jeho hlavička.

Pro vyhledávání souborů a adresářů používá funkce *DLD* uživatelské funkce *LIST_RESET* a *LIST_NEXT*. Proto nelze funkce *LIST_RESET* a *LIST_NEXT* kombinovat s funkcí *DLD*. Funkce *DLD* přepíše parametry ve struktuře *List* (viz kap. 10.5.12) a proto se musí funkce pro listování adresářem znovu inicializovat.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_1 - zadaný adresář v aktuálním adresáři neexistuje,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

10.5.11 Funkce MKD

Funkce *MKD* slouží k vytvoření nového adresáře v aktuálním adresáři logické jednotky. Pokud v aktuálním adresáři již adresář zadaného názvu existuje, funkce skončí neúspěchem a podá patřičný chybový kód.

```
unsigned char MKD(unsigned char *Name);
```

Funkce vrací parametr popsaný v kap. 10.5.6. Má jeden vstupní parametr a to jméno nového adresáře. Při vytvoření adresáře funkce převezme parametry z globální struktury *File_Info* (viz kap. 10.5.22), která obsahuje datum, čas a atributy.

Pokud v aktuálním adresáři není volné místo pro vytvoření hlavičky nového adresáře, funkce rozšíří aktuální adresář o nový cluster. Funkce vytvoří v aktuálním adresáři hlavičku nového adresáře se jménem *Name* a parametry ze struktury *File_Info*. Pro datovou oblast adresáře rezervuje funkce jeden cluster. Tento cluster je celý vymazán (na hodnoty 0) a na první dvě pozice vstupů jsou vytvořeny adresáře „.“ a „..“ vyplněné podle kap. 6.8. Tyto dva adresáře mají nastaven datum a čas shodný s jeho hlavičkou, ale nemají nastaveny žádné speciální atributy. Nakonec je vytvořen ve *FAT* tabulkách patřičný záznam. Všechny uživatelské funkce čtou pouze první *FAT*, ale při zápisu zapisují tytéž informace do všech kopií.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_2 - adresář nebyl vytvořen, protože zadaný adresář již existuje,
ERROR_3 - není dostatek volného místa,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

Nastane-li chyba *ERROR_11* nelze nový adresář v aktuálním adresáři vytvořit, protože v něm není dostatek místa přičemž záznam ve *FAT* adresáře je chybný, tudíž nemůže být rozšířen o nový cluster. Řešením je pouze smazat nějaký soubor nebo adresář s aktuálního adresáře a na jeho pozici bude zapsán nový adresář. Nicméně pracovat s vadným adresářem se nedoporučuje.

10.5.12 Funkce LIST_RESET

Funkce *LIST_RESET* je párovou funkcí k uživatelské funkci *LIST_NEXT*. Tyto funkce slouží k prohledávání adresářů. Je deklarována globální struktura *List*, ve které funkce *LIST_NEXT* eviduje současnou pozici v prohledávaném adresáři a současně do ní ukládá nalezenou hlavičku. Funkce *LIST_RESET* nastaví strukturu *List* na začátek aktuálního adresáře. Funkce *LIST_NEXT* poté nalezne první soubor nebo adresář v tomto adresáři a parametry jeho hlavičky uloží do struktury *List*. Opakovaným voláním funkce *LIST_NEXT* jsou ukládány do struktury *List* hlavičky dalších nalezených adresářů nebo souborů dokud nedojde na konec adresáře. Ve struktuře *List* je uložena hlavička naposledy nalezeného souboru nebo adresáře. Po aplikování funkce *LIST_RESET* lze opustit aktuální adresář bez vlivu na funkci *LIST_NEXT*.

```
void LIST_RESET();
```

Funkce nemá žádný vstupní parametr a ani žádný nevrací. Struktura *List* je datový typ *LIST_DIRECTORY* jejíž struktura je následující.

```
xdata LIST_DIRECTORY   List;

typedef struct LIST_DIRECTORY
{
  unsigned int Cluster;
  unsigned int Entry_Offset;
  unsigned char Sector_Offset;
  unsigned char Entry_Name[MAX_LENGTH_OF_ENTRY_NAME];
  unsigned char Time[2];
  unsigned char Date[2];
  unsigned long int File_Size;
  ATTRIBUTES Attributes;
} LIST_DIRECTORY;
```

Cluster – Číslo clusteru naposledy nalezené hlavičky (vstupu, *entry*).
Entry_Offset – Bajtový offset v sektoru naposledy nalezené hlavičky (vstupu, *entry*).
Sector_Offset – Sektorový ofset v clusteru naposledy nalezené hlavičky (vstupu, *entry*).
Entry_Name – Textový řetězec názvu naposledy nalezené hlavičky.
Time – Čas vytvoření nebo poslední změny souboru.
Date - Datum vytvoření nebo poslední změny souboru.
File_Size – Velikost naposledy nalezeného souboru nebo adresáře v bajtech.

Attributes – Atributy naposledy nalezeného souboru nebo adresáře.

Datum a čas jsou zde uloženy ve stejném formátu jako v hlavičce souboru. Atributy jsou uloženy ve struktuře *Attributes*, která je datovým typem *ATTRIBUTES*. Její struktura je následující.

```
typedef struct ATTRIBUTES
{
  unsigned char Read_Only;
  unsigned char Hidden;
  unsigned char System;
  unsigned char Archive;
  unsigned char Directory;
} ATTRIBUTES;
```

Read_Only – Atribut „jen pro čtení“.
Hidden – Atribut „skrytý“.
System – Atribut „systémový“.
Archive – Atribut „archivní“
Directory – Atribut „adresář“.

Proměnné atributů jsou sice datovými typy *unsigned char*, nabývají však jen dvou možných hodnot. **0** – atribut nenastaven, **1** - atribut je nastaven.

10.5.13 Funkce LIST_NEXT

Funkce *LIST_NEXT* je párovou funkcí k uživatelské funkci *LIST_RESET*. Slouží k vyhledávání hlaviček souborů a adresářů ve zvoleném adresáři. Pomocí těchto funkcí lze například vytvářet seznam adresářů a souborů v daném adresáři.

```
unsigned char LIST_NEXT();
```

Funkce vrací parametr popsaný v kap. 10.5.6 a nemá žádný vstupní parametr. Funkce je řízena strukturou *List*. Funkce nalezne nejbližší hlavičku souboru nebo adresáře od pozice dané ve struktuře *List*. Parametry nalezené hlavičky uloží opět do struktury *List* a aktualizuje v ní poslední pozici.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_9 - je dosaženo konce adresáře.
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

Možný způsob vytvoření seznamu adresářů a souborů pomocí těchto dvou uživatelských funkcí je následující. Pomocí funkcí *CD*, popř. *CD_UP* a *CD_ROOT* se vnoří do požadovaného adresáře. Voláním funkce *LIST_RESET* je struktura *List* nastavena na jeho počátek. Nyní se ve smyčce neustále volá funkce *LIST_NEXT* a mezi jednotlivými voláními je třeba jména nalezených adresářů a souborů kopírovat ze struktury *List* do připraveného pole. Dojde-li funkce *LIST_NEXT* na konec adresáře, bude to oznámeno v chybovém kódu při selhání funkce. Pokud je třeba vytvořit pouze seznam adresářů, stačí ve smyčce ověřovat atribut adresáře nalezené hlavičky a podle toho jej akceptovat či nikoliv.

Pokud je třeba používat funkci *LIST_NEXT* pro současné vyhledávání ve více adresářích je nutné vždy při změně adresáře zálohovat parametry *List.Cluster*, *List.Entry_Offset*, *List.Sector_Offset* a po návratu k původnímu adresáři je opět obnovit. Tyto tři parametry určují počátek hledání funkcí *LIST_NEXT*.

10.5.14 Funkce REWRITE_FILE

Funkce *REWRITE_FILE* slouží k vytvoření nového souboru v aktuálním adresáři. Nově vytvořený soubor je připraven pro zápis a čtení dokud nebude uzavřen funkcí *CLOSE_FILE*.

```
unsigned char REWRITE_FILE(FILE *F, unsigned char *Name);
```

Funkce vrací parametr popsaný v kap. 10.5.6 a má dva vstupní parametry. Prvním parametrem je ukazatel *F* na jeho pracovní proměnnou. Jedná se o strukturu, ve které budou uloženy informace potřebné pro práci s konkrétním souborem. Pracovní proměnná je typu *FILE* a musí být předem deklarována uživatelem. Po celou dobu práce se souborem bude pracovní proměnná přidružena k tomuto souboru. Druhým parametrem je jméno nového souboru. Při vytvoření nového souboru funkce převezme další parametry z globální struktury *File_Info* (viz kap. 10.5.22), ve které jsou uloženy požadované atributy, datum a čas stejně jako v případě vytvoření nového adresáře.

Pracovní proměnná obsahuje po otevření nebo vytvoření souboru některé parametry, se kterými lze pracovat mimo modul *file*. Jsou to např. parametry hlavičky otevřeného souboru (datum, čas, atributy, jméno, velikost), skutečný počet přečtených bajtů funkcí *READ_FILE* nebo informace o dosažení konce souboru při čtení atd. Struktura datového typu *FILE* je následující.

```
typedef struct FILE
{
 unsigned char Sector_Offset;
 unsigned char File_Name[MAX_LENGTH_OF_ENTRY_NAME];
 unsigned char Status;
 unsigned char Writed_To_File;
 unsigned char Read_Sector_Offset;
 unsigned char EOF_Sector_Offset;
 unsigned char LUN_Index;
 unsigned char EOF;
 unsigned char Write_EOC;
 unsigned char Read_EOC;
 unsigned int Cluster;
 unsigned int Entry_Offset;
 unsigned int First_Cluster;
 unsigned int Read_Cluster;
 unsigned int EOF_Cluster;
 unsigned int Read_Data_Offset;
 unsigned int EOF_Data_Offset;
 unsigned int Read;
 unsigned long int Read_Pointer;
 unsigned long int File_Size;
 FILE_INFO Info;
} FILE;
```

Sector_Offset - Sektorový ofset v clusteru, ve kterém leží hlavička otevřeného souboru.

File_Name – Jméno souboru (textový řetězec zakončený nulou).
Status – Indikátor, zda je struktura momentálně používána.
Writed_To_File – Indikuje, zda bylo do souboru alespoň jednou zapisováno po jeho otevření.
Read_Sector_Offset – Čtecí ukazatel – sektorový ofset v posledním čteném clusteru.
EOF_Sector_Offset – Sektorový ofset v clusteru, ve kterém leží poslední bajt souboru.
LUN_Index – Index logické jednotky v poli *LUN*, ve které leží otevřený soubor.
EOF – Indikátor dosažení konce souboru při čtení (1 = konec).
Write_EOC – Indikátor se nastaví na 1 pokud poslední zapisovaný bajt funkcí *WRITE_FILE* je současně poslední bajt v clusteru.
Read_EOC – Indikátor se nastaví na 1 pokud při čtení ze souboru dosáhnou čtecí ukazatele konce posledního clusteru souboru.
Cluster – Číslo clusteru, ve kterém leží hlavička otevřeného souboru.
Entry_Offset – Bajtový ofset v sektoru , kde leží hlavička otevřeného souboru.
First_Cluster – Číslo prvního clusteru otevřeného souboru.
Read_Cluster – Čtecí ukazatel – číslo clusteru, do kterého ukazuje čtecí ukazatel *Read_Pointer*.
EOF_Cluster – Číslo posledního obsazeného clusteru souboru.
Read_Data_Offset – Čtecí ukazatel – bajtový ofset v sektoru kam ukazuje čtecí ukazatel *Read_Pointer*.
EOF_Data_Offset – Bajtový ofset na poslední obsazený bajt v posledním obsazeném sektoru.
Read – Skutečný počet bajtů přečtených ze souboru funkcí *READ_FILE*.
Read_Pointer – Čtecí ukazatel – obsahuje číslo následně čteného bajtu od začátku souboru.
File_Size – Velikost otevřeného souboru v bajtech.
Info – Struktura obsahující datum, čas a atributy otevřeného souboru.

Pracovní proměnná obsahuje po otevření souboru všechny informace o souboru a všechny parametry potřebné pro práci s ním. Důležitý je parametr *Status*, který indikuje zda je struktura momentálně používána. Slouží k zabránění případného poškození souborů na disku při neopatrném programování. Jedna pracovní proměnná smí být v jednom okamžiku použitá pro práci s jedním souborem! Před prvním použitím pracovní proměnné je nutné nastavit parametr *Status* na hodnotu 0, což znamená nepoužito (struktura je volná). Po použití funkce pro otevření nebo vytvoření souboru se hodnota parametru *Status* změní na 1. Funkce pro práci se soubory si stav před použitím samy kontrolují, nevyhovuje-li, podávají příslušný chybový kód. Funkce pro uzavření souboru *CLOSE_FILE* parametr *Status* nastavuje zpět na výchozí hodnotu 0.

Pokud soubor se zadaným jménem do funkce *REWRITE_FILE* v aktuálním adresáři již existuje, bude nejdříve smazán. Pokud v aktuálním adresáři není dostatek místa pro vytvoření hlavičky nového souboru, aktuální adresář bude funkcí rozšířen o nový cluster. Ve vytvořené hlavičce nového souboru není alokován žádný první cluster. Ten bude přiřazen až po prvním zápisu do souboru funkcí *WRITE_FILE* (do hlavičky bude číslo prvního clusteru zapsáno až funkcí *CLOSE_FILE*). Po vytvoření hlavičky budou všechny zadané parametry zkopírovány do pracovní proměnné souboru. Po vytvoření souboru funkcí *REWRITE_FILE* je soubor otevřen. Po ukončení práce se souborem je nutné provést jeho uzavření funkcí *CLOSE_FILE*, čímž se také uvolní jeho pracovní proměnná. Poté co je soubor otevřen lze z aktuálního adresáře odejít bez vlivu na funkce *WRITE_FILE*, *READ_FILE*, *FILE_POINTER* a *CLOSE_FILE*. Tyto funkce získávají pozici souboru z jeho pracovní proměnné.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_3 - není dostatek volného místa pro vytvoření souboru,
ERROR_4 - zadaná pracovní proměnná souboru je už používána,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

10.5.15 Funkce OPEN_FILE

Funkce *OPEN_FILE* slouží k otevření existujícího souboru pro čtení i zápis v aktuálním adresáři.

```
unsigned char OPEN_FILE(FILE *F, unsigned char *Name);
```

Funkce vrací parametr popsaný v kap. 10.5.6 a má dva vstupní parametry. Prvním parametrem je ukazatel *F* na jeho pracovní proměnnou, jejíž význam je popsaný v kap. 10.5.14. Druhý parametr je textový řetězec jména otevíraného souboru.

Funkce nalezne v aktuálním adresáři soubor *Name* a uloží všechny jeho parametry do pracovní proměnné přes ukazatel *F*. Aby bylo možné do souboru zapisovat, zjistí také umístění posledního bajtu souboru v logické jednotce (tyto informace budou rovněž uloženy do pracovní proměnné). Proto se doba otevírání souboru zvyšuje s jeho velikostí (funkce musí procházet záznam ve *FAT*). Otevřený soubor má ukazatele pro čtení nastavené na jeho začátek.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_1 - zadaný soubor v aktuálním adresáři neexistuje,
ERROR_4 - zadaná pracovní proměnná souboru je už používána,
ERROR_6 - chyba ve *FAT* otevíraného souboru,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

10.5.16 Funkce CLOSE_FILE

Funkce *CLOSE_FILE* slouží k uzavření otevřeného souboru funkcemi *REWRITE_FILE* nebo *OPEN_FILE*.

```
unsigned char CLOSE_FILE(FILE *F);
```

Funkce vrací parametr popsaný v kap. 10.5.6 a má jediný vstupní parametr ukazatel *F* na pracovní proměnnou uzavíraného souboru. Pokud nebylo do uzavíraného souboru od jeho otevření zapisováno, funkce *CLOSE_FILE* pouze nastaví parametr *Status* pracovní proměnné na hodnotu 0 (nepoužívána).

Bylo-li do souboru zapisováno, funkce *CLOSE_FILE* aktualizuje hlavičku souboru o datum a čas z globální struktury *File_Info* (struktura je popsána v kap. 10.5.22). Tyto parametry je tedy nutné před uzavřením souboru do globální struktury *File_Info* správně nastavit. V hlavičce je také aktualizována velikost souboru a číslo prvního clusteru. Při zapisování do souboru je hlavička aktualizována pouze při jeho uzavření touto funkcí.

Nastane-li havárie zařízení, data uložená do souboru před jeho uzavřením budou ztracená, protože nebude aktualizován údaj o velikosti souboru v hlavičce. Je to z důvodu, aby nebyl neustále přepisován tentýž sektor, kde je umístěna hlavička souboru, aby tím nevznikalo jeho opotřebení.

Funkce nevrací žádný chybový kód. Možný výsledek je pouze úspěch nebo chyba v komunikaci.

10.5.17 Funkce READ_FILE

Funkce *READ_FILE* slouží ke čtení dat z otevřeného souboru. Funkce čte data postupně od začátku souboru do jeho konce. Uživatelskou funkcí *FILE_POINTER* lze nastavit počátek čtení funkce *READ_FILE*.

```
unsigned char READ_FILE(FILE *F, unsigned char *Buffer, unsigned int Size);
```

Funkce vrací parametr popsaný v kap. 10.5.6. Má tři vstupní parametry. Stejně jako v předchozích případech je prvním parametrem ukazatel *F* na pracovní proměnnou otevřeného souboru. Druhý parametr je ukazatel *Buffer* na cílové datové pole typu *unsigned char* kam budou uložena přečtená data. Třetí parametr je počet čtených bajtů od současné pozice čtecího ukazatele souboru. Protože mikroprocesorový systém 8051 může mít datovou paměť o maximální velikosti 64kB, pro parametr *Size* stačí datový typ *unsigned int*. Počet čtených bajtů *Size* nesmí přesáhnout velikost cílového pole. Přečtená data jsou do cílového pole ukládána od začátku. Pro nejrychlejší funkci je vhodné číst celočíselné násobky velikosti sektoru logické jednotky.

Pokud je v souboru menší počet dat než zadaný *Size*, funkce přečte pouze dostupné množství. Počet skutečně přečtených bajtů je po ukončení funkce uložen v pracovní proměnné souboru v parametru *Read*. Pokud při čtení dosáhne funkce konce souboru, nastaví v pracovní proměnné parametr *EOF* na hodnotu 1.

Po ukončení funkce jsou o počet *Read* přenastaveny čtecí ukazatele pracovní proměnné souboru (*Read_Pointer*, *Read_Cluster*, *Read_Sector_Offset*, *Read_Data_Offset*).

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_5 - zadaná pracovní proměnná souboru není momentálně používána,
ERROR_6 - chyba ve *FAT* čteného souboru,
ERROR_7 - čtecí ukazatel již ukazuje na konec souboru.

Při úspěšném proběhnutí funkce jsou přečtená data uložena v cílovém poli. Pokud bude při čtení dosaženo konce souboru (*F->EOF* = 1) a poté budou do souboru zapsána další data funkcí *WRITE_FILE*, příznak *EOF* v pracovní proměnné je automaticky nastaven na hodnotu 0 a lze pokračovat ve čtení souboru.

10.5.18 Funkce WRITE_FILE

Funkce *WRITE_FILE* slouží k zápisu dat na konec otevřeného souboru.

```
unsigned char WRITE_FILE(FILE *F, unsigned char *Buffer, unsigned int
Size);
```

Funkce vrací parametr popsaný v kap. 10.5.6. Má tři vstupní parametry. Prvním parametrem je ukazatel *F* na pracovní proměnnou otevřeného souboru. Druhý parametr je ukazatel *Buffer* na zdrojové datové pole typu *unsigned char*, ze kterého budou uložena data do souboru. Třetí parametr *Size* je počet ukládaných bajtů od začátku zdrojového pole. Protože mikroprocesorový systém 8051 může mít datovou paměť o maximální velikosti 64kB, pro parametr *Size* stačí datový typ *unsigned int*. Počet ukládaných bajtů *Size* nesmí přesáhnout velikost zdrojového pole.

Funkce *WRITE_FILE* je optimalizována tak, aby provedla co nejméně čtecích a zapisovacích operací s flash diskem, což se týká především operací s tabulkou *FAT*. Funkci lze rozdělit na několik částí. Nejdříve je vypočten počet potřebných clusterů pro uložení zadaného množství dat. Pokud na disku není dostatek volného místa pro celé toto množství, funkce končí neúspěchem.

Aby funkce nemusela vyhledávat volné clustery ve *FAT* během samotného zápisu dat, čísla volných clusterů pro zápis zadaného množství bajtů jsou zjištěna na začátku funkce a uložena do pole *Free_Clusters*. To zabrání zbytečnému opakování čtení týchž sektorů *FAT*. Protože minimální možná velikost clusteru je 512 bajtů a maximální teoretický možný počet bajtů k zapsání do souboru najednou je 64kB, funkce si musí zapamatovat čísla až 128 volných clusterů.

Pozn.: V praxi je situace poněkud jiná. Velikost clusterů na flash discích se běžné pohybuje zhruba od 4kB do 16kB a zdrojové pole pro zápis nemůže zabírat celých 64kB datové paměti mikrokontroléru 8051, protože je předpoklad, že jsou v něm uloženy i jiná data popř. proměnné programu. Ale protože byl při vývoji k dispozici i flash disk o kapacitě pouhých 10MB, který měl cluster o velikost jen 512 bajtů, funkce počítá i s touto variantou. V praxi bude zřejmě většinou počet zapisovaných dat nižší než velikost jednoho clusteru, proto si funkce pravděpodobně nejčastěji alokuje jeden nebo vůbec žádný cluster.

Poté funkce provede samotný zápis dat ze zdrojového pole do vyhledaných volných clusterů. Nakonec je vytvořen záznam ve *FAT* tabulkách. Aby nemohlo dojít k poškození souboru v případě havárie zařízení (výpadek napájení apod.), záznam ve *FAT* je vytvářen od posledního (nového) clusteru a až nakonec je tento záznam napojen na původní záznam před voláním funkce.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_3 - není dostatek volného místa v logické jednotce,
ERROR_5 - zadaná pracovní proměnná souboru není momentálně používána.

Alokační záznam souboru je paralelně zapisován do všech dostupných kopií alokačních tabulek. Doba provádění funkce *WRITE_FILE* je silně závislá na aktuálním obsazení *FAT*. Pokud je ukazatel pro hledání volných clusterů ve *FAT* na jejím začátku, ale nejbližší volný cluster je až na jejím konci, funkce musí prohledat většinu *FAT*, což je v praxi 200 až 300 sektorů. Pro menší opotřebení datového pole flash disku je vhodné zapisovat minimální množství dat o velikosti jednoho sektoru, není to však podmínkou (lze zapisovat do souboru třeba i po jednom bajtu).

10.5.19 Funkce FILE_POINTER

Funkce *FILE_POINTER* slouží k nastavení čtecích ukazatelů pracovní proměnné otevřeného souboru. Umožňuje libovolně nastavit počátek čtení funkce *READ_FILE* od začátku souboru až po jeho konec.

```
unsigned char FILE_POINTER(FILE *F, unsigned long int Offset);
```

Funkce vrací parametr popsaný v kap. 10.5.6. Má dva vstupní parametry. Prvním je opět ukazatel *F* na pracovní proměnnou souboru, pro který bude nastavován čtecí ukazatel. Druhý parametr je bajtový ofset od začátku souboru. Protože soubor může mít velikost až 4GB, zadaný ofset musí být typu *unsigned long int*. Nulový ofset (*Offset* = 0) je nastavení na první bajt souboru.

Funkce ze zadaného ofsetu vypočte v kolikátém clusteru od začátku souboru ofset leží, vypočte také sektorový a bajtový ofset. Poté prochází záznam souboru ve *FAT*, aby bylo nalezeno číslo tohoto clusteru. Po tomto má funkce k dispozici všechny parametry pro uložení do čtecích ukazatelů.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_5 - zadaná pracovní proměnná souboru není momentálně používána,
ERROR_6 - chyba ve *FAT* souboru,
ERROR_8 - zadaný ofset je větší než velikost souboru.

10.5.20 Funkce RENAME

Funkce *RENAME* slouží k přejmenování souboru nebo adresáře ležícího v aktuálním adresáři. Funkce mění v hlavičce pouze název, všechny ostatní parametry zůstanou beze změny.

```
unsigned char RENAME(bit Dir_Or_File, unsigned char *Name, unsigned char
*New_Name);
```

Funkce vrací parametr popsaný v kap. 10.5.6. První vstupní parametr *Dir_Or_File* určuje, zda funkce přejmenovává adresář nebo soubor, protože v jednom adresáři mohou mít adresáře a soubory zcela shodné názvy.

Dir_Or_File = 0 - přejmenovává adresář,
Dir_Or_File = 1 - přejmenovává soubor.

Druhý parametr je jméno přejmenovávaného souboru nebo adresáře a poslední parametr je nové jméno.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_1 - zadaný soubor nebo adresář v aktuálním adresáři neexistuje,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

10.5.21 Funkce DLF

Funkce *DLF* slouží ke smazání zadaného souboru z aktuálního adresáře. Mazaný soubor nesmí obsahovat dlouhý název *LFN*. Ten by pak zůstal v adresáři zachován.

```
unsigned char DLF(unsigned char *Name);
```

Funkce vrací parametr popsaný v kap. 10.5.6. Jediným vstupním parametrem je jméno mazaného souboru. Funkce nejprve smaže hlavičku souboru z aktuálního adresáře tak, že na první pozici vstupu zapíše hodnotu **0xE5**. Druhým krokem je smazání záznamu souboru v tabulkách *FAT*.

Při selhání může funkce vracet tyto chybová hlášení:

ERROR_1 - zadaný soubor v aktuálním adresáři neexistuje,
ERROR_11 - chyba ve *FAT* aktuálního adresáře.

Doba vykonávání funkce závisí na velikosti mazaného souboru a také na způsobu jeho fragmentování. Čím je soubor větší, tím je větší i záznam ve *FAT* a mazání trvá delší dobu.

10.5.22 Funkce Set_File_Info

Funkce *Set_File_Info* slouží k rychlému uživatelskému nastavení parametrů do globální struktury *File_Info*. Samotná funkce není příliš důležitá, je určena spíše pro vývoj uživatelského programu.

```
void Set_File_Info(unsigned char Read_Only, unsigned char Hidden, unsigned
char System, unsigned char Archive, unsigned char Seconds, unsigned char
Minutes, unsigned char Hours, unsigned char Days, unsigned char Months,
unsigned char Years);
```

Funkce nevrací žádný parametr. Má deset vstupních parametrů:

Atributy: Read_Only, Hidden, System, Archive,
Čas: Seconds, Minutes, Hours,
Datum: Days, Months, Years.

Vstupní parametry funkce představují jednotlivé parametry struktury *File_Info*. Atributy nabývají dvou hodnot: **0** – nenastevn, **1** - nastaven. Čas a datum je zřejmý, pouze rok má stejný formát jaký je použit v hlavičkách souborů, tedy ROK – 1980 (např. pro rok 2008 musí být uložena hodnota 28).

Globální struktura *File_Info* je datový typ *FILE_INFO*. Struktura tohoto datového typu je následující.

```
xdata FILE_INFO        File_Info;

typedef struct FILE_INFO
{
 ATTRIBUTES Attributes;
 TIME       Time;
```

```
 DATE       Date;
} FILE_INFO;
```

Attributes – Struktura typu *ATTRIBUTES* obsahující atributy souboru nebo adresáře.
Time – Struktura typu *TIME* obsahující čas vytvoření nebo poslední změny souboru.
Date - Struktura typu *DATE* obsahující datum vytvoření nebo poslední změny souboru.

Datový typ *ATTRIBUTES* je popsán v kap. 10.5.12. Datové typy *TIME* a *DATE* mají následující strukturu.

```
typedef struct TIME
{
 unsigned char Second;
 unsigned char Minute;
 unsigned char Hour;
} TIME;

typedef struct DATE
{
 unsigned char Day;
 unsigned char Month;
 unsigned char Year;
} DATE;
```

Second – Sekundy.
Minute – Minuty.
Hour – Hodiny.

Day – Den.
Month - Měsíc
Year – Rok (-1980).

Ze struktury *File_Info* jsou brány parametry funkcemi pro vytvoření adresáře a souboru a funkcí pro uzavření souboru. Před použitím těchto funkcí by měla být struktura *File_Info* nastavena na nějaké platné parametry. Parametry lze do struktury *File_Info* ukládat přímo nebo pomocí funkce *Set_File_Info*. Funkce uloží její vstupní parametry přímo do jednotlivých parametrů struktury *File_Info*. Funkce *Set_File_Info* neplní atribut adresáře *Directory*. Tento atribut je pouze pro čtení např. ve struktuře *List* nebo v datovém typu *FILE*.

10.5.23 Ostatní funkce modulu file

Modul *file* obsahuje další jednodušší funkce, pomocí kterých realizuje funkce uživatelské. Pro přehled je zde zmíněn ve stručnosti jejich účel.

Check_Partition – Používána funkcí *Start_Flash_Disk*, naplní pole *LUN* z *FAT16BR* sektoru.
Get_Entries_Name – Načte z hlavičky souboru jeho jméno do textového řetězce.
Scan_Sector_Cluster – Hledá v načteném sektoru adresáře soubor nebo adresář daného názvu.
Scan_Sector_Name – Hledá v načteném sektoru adresáře název adresáře podle čísla clusteru.
Scan_Sector_Empty – Hledá v načteném sektoru adresáře volný vstup (*entry*).
Scan_Sector_Entry – Hledá v načteném sektoru adresáře platný vstup a uloží do struk. *List*.
Search_Entries – Používá funkce *Scan..*, hledá všechny potřebné typy vstupů v adresáři.
CD_Direct – Přímé vnoření do adresáře, umístění jeho hlavičky bere ze struktury *List*.

DLE_Direct – Přímé mazání souboru nebo adresáře, umístění hlavičky je ve struktuře *List*.
GET_EMPTY_ENTRY – Zjistí pozici volného vstupu v adresáři, případně ho rozšíří.
Set_Entries_Info – Vyplní hlavičku adresáře nebo souboru podle parametrů a struk. *File_Info*.
Get_Free_Cluster – Funkce vrací číslo volného clusteru ve *FAT*.
Expand_File – Rozšíří záznam souboru nebo adresáře ve *FAT* o jeden volný cluster.
Set_FAT – Nastaví zadanou buňku *FAT* na zadanou hodnotu.
Clear_Cluster – Vymaže obsah zadaného clusteru (na hodnoty 0).
Get_Sector – Vypočítá *LBA* adresu prvního sektoru zadaného clusteru.
Compare_Strings – Porovná obsah dvou zadaných textových řetězců.
Get_Cluster_From_FAT – Vrátí obsah buňky *FAT* zadané číslem clusteru.
Clean_FAT – Vymaže záznam souboru nebo adresáře z *FAT* od zadaného čísla clusteru.

10.6 Programový modul „board“

Programový modul *board* slouží k definici připojení jednoduchých periferií nebo obvodů k vývojové desce *SiLabs*. Modul obsahuje pouze definice připojení indikačních LED k vývojové desce a funkci pro dvojité bliknutí LED. V modulu není pro činnost komunikace s flash diskem žádná důležitá funkce, prakticky jen makra pro práci s připojenými LED, které lze z kompilace vynechat viz kap. 10.2. Makra pro nastavení LED vypadají následovně:

```
//sbit LED  = 0xA0;                       // LED na kitu
//sbit TL   = 0xA1;                       // Tlacitko na kitu
sbit RST    = 0x93;                       // P0.3 RST - Reset USB radice
sbit LED1   = 0xA5;                       // P0.4 zelena LED
sbit LED2   = 0xA6;                       // P0.5 cervena LED
sbit USB_SS = 0xA7;                       // P2.7 signal SS radice USB

// LED
#define LED_ON     0                      // led roznuta
#define LED_OFF    1                      // led zhasnuta

// zkracene ovladani LED - pouziva modul drv_usb_controller.c
#define LEDX1_ON    LED2 = LED_ON         // rozne LED2
#define LEDX1_OFF   LED2 = LED_OFF        // zhasne LED2
#define LEDX2_ON    LED1 = LED_ON         // rozne LED1
#define LEDX2_OFF   LED1 = LED_OFF        // zhasne LED1
```

Zakomentovaná makra *LED* a *TL* představují prvky umístěné na vývojové desce SiLabs, které v projektu nejsou použity. Signál *RST* je reset řadiče USB připojený na pin P1.3. *LED1* a *LED2* jsou dvě LED diody pro indikaci činnosti na sběrnici USB, jsou nepovinné a zde je možné nastavit jejich napojení na port mikrokontroléru C8051F120. Makra jsou nastaveny tak, že *LED1/2* svítí při LOG0. Poslední makra *LEDX1/2*.. slouží pro ovládání *LED1/2* modulem *drv_usb_controller*.

V modulu *board* je dále definována funkce pro dvojité bliknutí s *LED2*, kterou používá modul *drv_usb_controller* pro indikaci připojení a odpojení zařízení *FS* na sběrnici USB (opět uživatelsky volená kompilace).

```
void Blik();
```

10.7 Programový modul „init_up“

Programový modul *init_up* obsahuje počáteční nastavení mikrokontroléru C8051F120 a zdrojový soubor *init_up.c* je generován programem *Config2* od Silicon Laboratories. Pro tento projekt je důležité pouze nastavit externí přerušení *INT1* pro spouštění na hranu, povolit externí přerušení *INT1* a povolit sběrnici *SPI*.

Aby seděly jednotlivé přiřazení (*crossbar*) periferií na jednotlivé porty mikrokontroléru se zde popsaným modulem řadiče USB, musí být ještě povolen *UART0* a sběrnice *SMBus*. Naproti tomu nesmí být na porty mikrokontroléru přiřazen *UART1, PCA, CP0, CP1* a *Timer0*. Ostatní periferie mikrokontroléru nemají na funkci modulu MAX3421E vliv.

Hodinovou frekvenci oscilátoru je možné zvolit libovolně, je nutné ovšem v modulu *drv_usb_controller* definovat vhodné makro pro odpovídající frekvenci viz kap. 10.2.

Programový modul má jedinou výstupní funkci *Init_Device*, která je volána z hlavního modulu *main*, a která obsahuje funkce pro nastavení mikrokontroléru.

```
void Init_Device(void);
```

10.8 Soubor system.h a c8051f120.h

Soubor c8051f120.h obsahuje definice registrů mikrokontroléru C8051F120. Soubor je převzatý z databáze vývojového prostředí µVision3. Soubor system.h obsahuje obecná makra, které mohou používat všechny ostatní moduly projektu. Jedná se především o makra pro slovní vyjádření výsledků funkcí pro zvýšení čitelnosti programu. Jedinou výjimkou je makro *MAX_SECTOR_SIZE*.

```
#define MAX_SECTOR_SIZE          2048
```

Makro *MAX_SECTOR_SIZE* slouží pro uživatelské nastavení maximální velikosti sektoru, kterou program podpoří. Pokud je nastaveno např. na 2048, program podpoří logickou jednotku flash disku do velikosti sektoru 2048 bajtů. Vzhledem k činnosti modulů má smysl nastavovat toto makro pouze podle možných velikostí sektorů (512, 1024, 2048, 4096 atd.). Čím větší sektor program podpoří, tím kompilátor zabere větší část datové paměti – konkrétně dvojnásobek hodnoty *MAX_SECTOR_SIZE* pro vytvoření přenosových zásobníků. Pro většinu flash disků stačí kompilace pro sektor o velikosti 512 bajtů. Při vývoji programu byl však k dispozici i flash disk, který měl ve své jediné logické jednotce velikost sektoru 2048 bajtů.

10.9 Programový modul „main“

Programový modul *main* je modul obsahující hlavní funkci *main*. Uživatelské funkce pro práci s flash diskem mohou být samozřejmé používány v jakémkoliv jiném modulu, zde však bude práce s flash diskem a nastavení programu pro vysvětlení ukázána v hlavním modulu. Hlavní modul může vypadat např. takto:

```
#include "..\system\c8051f120.h"
#include "..\system\main.h"
#include "..\system\board.h"
#include "..\system\system.h"
#include "..\system\init_up.h"
#include "..\file\file.h"
#include "..\msc\msc.h"
#include "..\usb\usb.h"
#include "..\driver\drv_usb_controller.h"
#include "string.h"

/*****************************************************************************
Obsluha externiho preuseni INT1
*****************************************************************************/
void INT1() interrupt 2
{
 INT_USB_Controller();
}

/*****************************************************************************
MAIN
*****************************************************************************/
void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // detekce zarizeni na sbernici
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash diaku
 if (Number_of_LUNs > 0)      // test dostupnych logickych jednotek
  {
   Curr_LUN(0);               // prepnuti na první dostupnou LUN
   // operace s flash diskem

  }
 while(1);                    // konec programu
}
```

Na začátku jsou vloženy hlavičkové soubory jednotlivých modulů. Je zde také umístěna obsluha externího přerušení *INT1* což je přerušení od řadiče USB. Zde je nutné volat funkci *INT_USB_Controller* z modulu *drv_usb_controller*.

Ve funkci *main* je na počátku volána funkce *Init_Device* z modulu *init_up* pro provedení počáteční inicializace mikrokontroléru. Poté je zavolána funkce *USB_Controller_Init* z modulu *drv_usb_controller*, která provede počáteční nastavení řadiče USB. Po tento bod je všechno povinné. Od teď je řešení programu individuální a závisí na požadované funkci programu.

V tomto příkladu program pokračuje následovně. Je volána funkce *USB_Device_Detect* z modulu *drv_usb_controller*, která zjistí zda je momentálně připojeno na USB sběrnici *FS* zařízení a podle toho aktualizuje proměnnou *Inserted*. Pokud zařízení není připojeno (*Inserted = NO*), program se zastaví na následující podmínce, která čeká, až se hodnota proměnné *Inserted* změní na *YES* (= 1).

Pokud radič USB detekuje připojení *FS* zařízení, hodnota *Inserted* se aktualizuje na hodnotu *YES* v obsluze přerušení od řadiče USB (*INT1*) a hlavní program bude pokračovat. V tomto okamžiku některé MP3 přehrávače vyžadovaly malou prodlevu než se s nimi program pokusil komunikovat, proto je zde zavedeno časové zpoždění 1s. Funkce *Delay* je rovněž z modulu *drv_usb_controller*.

Poté hlavní program volá funkci *Start_Flash_Disk* z modulu *file* k celkové inicializaci flash disku. Pokud vše proběhlo v pořádku a je připojen na USB sběrnici flash disk, počet dostupných logických jednotek je větší než 0. Toto je indikováno v globální proměnné *Number_Of_LUNs*. Pokud není dostupná žádná logická jednotka (*Number_Of_LUNs* = 0), tak připojené zařízení není flash disk nebo flash disk nemá naformátovanou žádnou logickou jednotku na souborový systém *FAT16* nebo došlo k chybě při enumeraci. V každém případě s takovým zařízením program nebude umět komunikovat. Funkci *Start_Flash_Disk* lze volat i v případě, že není připojeno žádné zařízení na sběrnici. Funkce si přítomnost zařízení sama zjistí takže možností řešení čekacích a testovacích smyček je opravdu hodně.

Jeli počet logických jednotek jedna a více, v těle podmínky lze realizovat požadovanou operaci. Po proběhnutí funkce *Start_Flash_Disk* je program přepnut na první logickou jednotku s indexem 0 v poli LUN (*Current_LUN = 0*). Tedy volat uživatelskou funkci *Curr_LUN(0)* je v tomto případě zbytečné. Po skončení je program zastaven v nekonečné smyčce.

V konkrétní aplikaci bude program zřejmě řešen jinak. V aplikacích, které nepotřebují flash disk pro svou primární funkci, zřejmě nebude potřeba čekat na jeho připojení, bude stačit pouze test na začátku programu. V opačném případě lze vytvořit tělo hlavního programu tak, že na začátku bude realizováno čekání na připojení zařízení a v případě odpojení za chodu opětovný návrat na čekací smyčku pro připojení.

10.10 Uživatelská makra

Během popisu jednotlivých modulů byly zmíněny různá uživatelská makra v modulech projektu. V tab. 35 je uveden jejich přehled a stručný popis funkce. Tyto makra umožňují nastavit různé parametry programu při kompilaci podle požadavků uživatele.

Možné rozsahy hodnot maker jsou uvedeny v komentářích přímo v souborech programu. Výchozí nastavení maker bylo experimentálně ověřeno a vyhovuje širokému spektru zařízení. Před aplikací modulů je však třeba zkontrolovat, zda vyhovuje nastavení maker *PACKET_LED*, *CONNECT_LED*, *SYSTEM_CLOCK_X*, *WORK_WITH_ONE_LUN*, *MAX_LUNS* a *MAX_SECTOR_SIZE*.

Speciálním typem makra je makro *SPEED_RETARDER*. Při práci s některými MP3 přehrávači nastávala při rychlém čtení či zápisu chyba v komunikaci (zařízení přestala zcela reagovat). Dokonce s jedním konkrétním zařízením identifikoval řadič připojení zařízení během normální komunikace. Experimenty bylo zjištěno, že u těchto zařízení stačí vložit mezi jednotlivé přenosy protokolem *Bulk-Only* malé zpoždění o délce asi 5ms. Toto zpoždění je možné volit pomocí makra *SPEED_RETARDER*. Hodnota 10 znamená zpoždění 1ms. Např. s flash diskem *ISTICK220* tento problém nenastával a přenos byl vždy spolehlivý s nulovou hodnotou zpoždění. S vyšším zpožděním samozřejmě klesají přenosové rychlosti.

Tab. 35 Přehled uživatelských maker

Makro	Umístění	Funkce
PACKET_LED	drv_usb_controller.h	Je-li makro definováno, provede se kompilace s LEDX1 ze souboru board.h. LED indikuje přenos paketů po sběrnici USB.
CONNECT_LED	drv_usb_controller.h	Je-li makro definováno, provede se kompilace s LEDX2 ze souboru board.h. LED indikuje připojení a odpojení zařízení FS na sběrnici USB.
SYSTEM_CLOCK_1 SYSTEM_CLOCK_2 SYSTEM_CLOCK_3 SYSTEM_CLOCK_4 SYSTEM_CLOCK_5	drv_usb_controller.h	Musí být definováno jedno z těchto maker. Makro určuje konstanty pro zpožďovací funkci Delay podle hodinového kmitočtu mikrokontroléru. Podrobné vysvětlení je napsano v souboru drv_usb_controller.h.
WORK_WITH_ONE_LUN	file.h	Makro je možné definovat pouze pokud uživatelský program pracuje pouze s jednou LUN flash disku. Urychlí se v určitých situacích funkce pro práci se soubory.
MAX_LUNS	file.h	Určuje maximální počet LUN který program podpoří. Možný rozsah hodnot je 1 - 16.
MAX_BULK_ONLY_ERROR	msc.h	Určuje maximální počet opakování pokusů o provedení transportu Bulk-Only při chybách.
BULK_ONLY_DELAY	msc.h	Určuje dobu pro funkci Delay, kterou bude program čekat před opakováním transportu Bulk-Only při chybě.
MAX_PACKET_ERROR	usb.h	Určuje maximální počet opakování pokusů o přenos paketu na sběrnici USB při chybách nebo NAK.
MAX_REQUEST_ERROR	usb.h	Určuje maximální počet opakování pokusů o provedení USB řídícího požadavku při chybách.
NAK_DELAY	usb.h	Určuje dobu pro funkci Delay, kterou bude program čekat před opakováním paketu když brána vrátila paket NAK.
ERROR_DELAY	usb.h	Určuje dobu pro funkci Delay, kterou bude program čekat před opakováním paketu když nastala chyba v přenosu.
REQUEST_DELAY	usb.h	Určuje dobu pro funkci Delay, kterou bude program čekat před opakováním USB řídícího požadavku když nastala chyba v přenosu.
MAX_SECTOR_SIZE	system.h	Určuje maximální velikost sektoru logické jednotky, kterou program podpoří.
SPEED_RETARDER	msc.h	Umožňuje zpomalení přístupu na flash disk z důvodu zvýšení spolehlivosti.

10.11 Příklady realizace programů

Jsou zde uvedeny jednoduché konstrukce programů realizující různé operace pomocí uživatelských funkcí modulu *file*. Vždy je uvedena jen funkce *main*, obsluhu přerušení od řadiče USB a vložení hlavičkových souborů je třeba provést podle kap. 10.9. U většiny příkladů není pro přehlednost kontrolován výsledek jednotlivých funkcí, předpokládá se, že jsou vždy provedeny úspěšně.

10.11.1 Pohyb v adresářové struktuře logických jednotek

Příklad ukazuje práci s funkcemi pro pohyb v adresářové struktuře a také práci s více logickými jednotkami flash disku. V příkladu se program vnoří do adresáře LUN0:\DATA\DATA.A\TEMP\ a také do adresáře LUN1:\ZDROJ\TEPLOTA\HOD\INT\. Program předpokládá, že disk obsahuje dvě logické jednotky s *FAT16* a že obě cesty existují. Nakonec se program vynoří zpět do kořenových adresářů.

```
void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 1)      // musí byt dostupne alespon 2 LUN
 {
  Curr_LUN(0);          // prepnuti na LUN0
  CD("DATA");           // LUN0:/DATA/
  CD("DATA.A");         // LUN0:/DATA/DATA.A/
  CD("TEMP");           // LUN0:/DATA/DATA.A/TEMP/

  Curr_LUN(1);          // prepnuti na LUN1
  CD("ZDROJ");          // LUN1:/ZDROJ/
  CD("TEPLOTA");        // LUN1:/ZDROJ/TEPLOTA/
  CD("HOD");            // LUN1:/ZDROJ/TEPLOTA/HOD/
  CD("INT");            // LUN1:/ZDROJ/TEPLOTA/HOD/INT/

  //vynoreni do korenoveho adresare

  Curr_LUN(0);          // prepnuti na LUN0
  CD_UP();              // LUN0:/DATA/DATA.A/
  CD_UP();              // LUN0:/DATA/
  CD_UP();              // LUN0:/

  Curr_LUN(1);          // prepnuti na LUN1
  CD_ROOT();            // LUN1:/
 }
 while (1);             // konec
}
```

Na začátku programu je čekací smyčka na připojení flash disku. Pokud by program předpokládal, že flash disk je již připojený, funkci *USB_Device_Detect* a čekací smyčku testující proměnnou *Inserted* lze vynechat. Je ale vhodné před inicializací flash disku počkat několik vteřin, protože některé zařízení jako MP3 přehrávače obvykle komunikují až několik sekund po připojení napájecího napětí.

Ke vnoření do adresářů je použita funkce *CD*. Logické jednotky jsou přepínány funkcí *Curr_LUN*. V jednom případě je použito postupné vynoření pomocí opakovaných volání funkce *CD_UP*, ve druhém případě se program přesune pomocí funkce *CD_ROOT* přímo do kořenového adresáře.

10.11.2 Vytvoření a mazání adresářů, volné místo na disku

Následující příklad ukazuje způsob vytváření a mazání adresářů a také zjištění volného místa v logické jednotce. Program ověří, zda je na disku 100kB volného místa a pokud ano vytvoří adresář DATA v kořenu logické jednotky a v tomto novém adresáři vytvoří adresář VOLTAGE. Výsledek bude LUN0:\DATA\VOLTAGE\. Nakonec bude adresář DATA smazán. Program předpokládá, že na počátku adresář DATA v kořenu příslušné logické jednotky neexistuje.

```
void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 0)      // musí byt dostupna alespon 1 LUN
 {
  Curr_LUN(0);                // prepnuti na LUN0
  Get_Free_Space();           // zjisteni volneho mista LUN0
  if (Free_Space > 102400)    // > 100kB
  {
   Set_File_Info(0,0,0,0,38,17,11,16,4,28); // 11:17:38, 16.4.2008
   MKD("DATA");               // vytvoření adresare DATA
   CD("DATA");                // LUN0:/DATA/
   MKD("VOLTAGE");            // vytvoření adresare VOLTAGE

   // smazani adresare DATA

   CD_ROOT();                 // presun do korenoveho adresare
   DLD("DATA");               // smazani adresare DATA
  }
 }
 while (1);                   // konec
}
```

Pro zjištění volného místa logické jednotky je použita funkce *Get_Free_Space*. Pro vytvoření adresářů je použita funkce *MKD*. K nastavení struktury *File_Info*, ze které funkce *MKD* převezme některé parametry hlavičky, je použita funkce *Set_File_Info*. Je nastavena na čas vytvoření 11:17:38 a datum 16.4.2008, atributy nejsou nastaveny žádné. Oba adresáře budou vytvořeny se stejnými parametry. Ke smazání adresáře je použita funkce *DLD*. Smazán bude i jeho obsah, tedy adresář VOLTAGE.

10.11.3 Vytvoření seznamu souborů a adresářů

Zde je uveden příklad práce s uživatelskými funkcemi *LIST_RESET* a *LIST_NEXT*. Program najde všechny adresáře a soubory v adresáři LUN0:\ZDROJ\. Způsob zpracování nalezených adresářů a souborů příklad neukazuje.

```
void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
```

```
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 0)      // musí byt dostupna alespon 1 LUN
 {
  Curr_LUN(0);                // prepnuti na LUN0
  CD("ZDROJ");                // LUN0:/ZDROJ/
  LIST_RESET();               // nastaveni struk. List na zacatek adresare
  while (LIST_NEXT() == 0)    // vyhledavaci smycka
  {
   // zpracovani nalezene hlavicky
   // List.Entry_Name - jmeno souboru nebo adresare
   // List.Attributes.Directory - atribut adresare
  }
 }
 while (1);
}
```

Po vnoření do adresáře ZDROJ je volána funkce *LIST_RESET*, která nastaví počátek hledání na začátek aktuálního adresáře. Nyní je vykonávána ve smyčce funkce *LIST_NEXT*, která v každém cyklu nalezne od aktuální pozice nejbližší hlavičku souboru nebo adresáře. Až funkce *LIST_NEXT* dorazí na konec prohledávaného adresáře, zahlásí selhání (vrátí hodnotu 1). Selhání funkce může nastat také z důvodu chyby ve *FAT* prohledávaného adresáře, ale i v tomto případě nelze v hledání pokračovat.

V těle prohledávacího cyklu je nutné žádaným způsobem zpracovávat nalezené hlavičky, jejichž parametry funkce *LIST_NEXT* ukládá do globální struktury *List*, viz kap. 10.5.12. Filtrováním atributu adresáře lze zpracovávat jen adresáře nebo soubory. Uvedeným způsobem lze také ověřovat existenci adresáře nebo souboru v daném adresáři. Ověřovat existenci souboru lze však provést efektivněji pomocí funkce *OPEN_FILE*. Následující příklad cyklu ukazuje ověření existence adresáře DATA v prohledávaném adresáři.

```
bit Exist = NO;

while (LIST_NEXT() == 0)
{
 if ((List.Attributes.Directory == 1)
  && (Compare_Strings("DATA", &List.Entry_Name) == 1))
   {
    Exist = YES;
    break;
   }
}
```

Je deklarována proměnná *Exist*, která bude po skončení smyčky indikovat existenci adresáře DATA. Ve vyhledávací smyčce je testován atribut adresáře (zda nalezená hlavička je adresář), a porovnáváno jeho jméno s řetězcem DATA pomocí funkce *Compare_Strings* modulu *file*. Pokud platí současně obě podmínky, adresář DATA byl nalezen, je nastavena proměnná *Exist* a cyklus se ukončí.

10.11.4 Vytvoření souboru, ověření existence a mazání souboru

Příklad ukazuje způsob vytvoření nového souboru, způsob ověření existence souboru v adresáři a mazání souborů. V kořenovém adresáři bude vytvořen soubor TEPLOTA.DAT, a

v adresáři LUN0:\ZDROJ\ bude zjištěna existence souboru TEMP.TXT. Vytvořený soubor TEPLOTA.DAT bude nakonec smazán.

```
FILE Fa, Fb;
bit Exist = NO;

void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 0)      // musí byt dostupna alespon 1 LUN
 {
  Curr_LUN(0);                // prepnuti na LUN0
  Fa.Status = 0;              // pocatecni init Fa
  Fb.Status = 0;              // pocatecni init Fa

  Set_File_Info(0,0,0,0,20,54,12,16,4,28); // 12:54:20, 16.4.2008
  REWRITE_FILE(&Fa, "TEPLOTA.DAT"); // vytvori soubor

  CD("ZDROJ");                // LUN0:/ZDROJ/

  Error_Code = ERROR_0;
  OPEN_FILE(&Fb, "TEMP.TXT");// otevre soubor, pokud existuje
  if ((Error_Code == ERROR_0)||(Error_Code == ERROR_6)) Exist = YES;

  CLOSE_FILE(&Fa);            // uzavre soubor Fa
  CLOSE_FILE(&Fb);            // uzavre soubor Fb

  CD_ROOT();                  // LUN:/
  DLF("TEPLOTA.DAT");         // smaze soubor
 }
 while (1);                   // konec
}
```

Jsou deklarovány dvě pracovní proměnné typu *FILE* pro současnou práci se dvěma soubory. Proměnné typu *FILE* je třeba před prvním použitím vždy nastavit. Je třeba vynulovat parametr *Status*. Funkcí *Set_File_Info* je nastavena struktura *File_Info* na požadované parametry hlavičky souboru. K vytvoření souboru je použita funkce *REWRITE_FILE* a nově otevřenému souboru je přiřazena pracovní proměnná *Fa*. Soubor je zatím ponechán otevřený. Funkcí *CD* se program vnoří do adresáře ZDROJ, kde ověří existenci souboru TEMP.TXT. Ověření, zda soubor existuje, je provedeno pomocí funkce *OPEN_FILE*, která se pokusí soubor otevřít. Souboru bude přiřazena druhá deklarovaná pracovní proměnná *Fb*. Předtím je nastavena proměnná *Error_Code* na hodnotu *ERROR_0*, která bude indikovat příčinu selhání příkazu. Jeli po vykonání chybový kód roven *ERROR_0* (nenastala chyba, soubor je otevřen) nebo *ERROR_6* (chyba ve *FAT* otevíraného souboru, soubor není otevřen, ale existuje) soubor existuje a je nastavena proměnná *Exist*.

Poté jsou uzavřeny oba otevřené soubory funkcemi *CLOSE_FILE*, čímž budou obě proměnné *Fa* a *Fb* uvolněny. Zde je možné si všimnout, že lze uzavírat i soubor, který není otevřen, aniž by funkce *CLOSE_FILE* zhavarovala. Také je vhodné podotknout, že funkce *CLOSE_FILE*, *READ_FILE* a *WRITE_FILE* nejsou závislé na aktuálním adresáři ani na

aktuální logické jednotce. Vždy pracují pouze s pracovní proměnnou typu *FILE*, ve které je umístění souboru při jeho otevření uchováno.

Nakonec se program přesune do kořenového adresáře a pomocí funkce *DLF* smaže na začátku vytvořený soubor TEPLOTA.DAT. Pozn.: Nesmí se mazat otevřený soubor. Funkce proti této operaci nejsou chráněny. Také nesmí být smazán adresář, ve kterém leží otevřený soubor. Tyto zakázané operace by mohly vést k poškození dat na disku.

10.11.5 Čtení souboru a zápis do souboru

Následující příklad programu ukazuje způsob čtení a zápisu dat do souboru. Program otevře z kořenového adresáře soubor ZDROJ.DAT a přečte celý jeho obsah. Po jeho přečtení na jeho konec uloží textový řetězec. Textový řetězec bude uložen také do nově vytvořeného souboru TEXT.TXT. Program předpokládá existenci souboru ZDROJ.DAT.

```
FILE F;
unsigned char Buffer[512];

void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 0)      // musí byt dostupna alespon 1 LUN
 {
  Curr_LUN(0);                // prepnuti na LUN0
  F.Status = 0;               // init promenne F

  OPEN_FILE(&F, "ZDROJ.DAT"); // otevre soubor ZDROJ.DAT
  while (F.EOF == 0)          // delej dokud nedosahne cteni konce souboru
  {
   READ_FILE(&F, &Buffer, 512);     // precte 512 bajtu
    // zde bude zpracovani dat z pole Buffer
    // F.Read - pocet prectenych bajtu
  }
  strcpy(&Buffer, "Toto je pokus.");
  WRITE_FILE(&F, &Buffer, 14);      // zapise 14 bajtu
  Set_File_Info(0,0,0,0,46,57,15,4,16,28);
  CLOSE_FILE(&F);                   // uzavre soubor ZDROJ.DAT

  Set_File_Info(0,0,0,1,46,57,15,4,16,28);
  REWRITE_FILE(&F, "TEXT.TXT");     // vytvori soubor TEXT.TXT
  WRITE_FILE(&F, &Buffer, 14);      // zapise 14 bajtu
  CLOSE_FILE(&F);                   // uzavre soubor TEXT.TXT
 }
 while (1);
}
```

Na počátku je deklarována pracovní proměnná *F* typu *FILE* pro práci se soubory. Protože program pracuje najednou pouze s jedním souborem, vystačí si s deklarováním jedné proměnné typu *FILE*. Také je deklarováno pole *Buffer* pro datové přenosy ze souboru a do souboru.

Funkcí *OPEN_FILE* je otevřen soubor ZDROJ.DAT. Následuje cyklus, ve kterém je neustále načítáno 512 bajtů ze souboru ZDROJ.DAT do pole *Buffer*. Ve smyčce pouze chybí požadované zpracování načtených dat z pole *Buffer*. Počet přečtených dat funkcí *READ_FILE* do pole *Buffer* je po skončení funkce přístupné v pracovní proměnné *F.Read*. Až dojde funkce *READ_FILE* při čtení na konec souboru, nastaví parametr pracovní proměnné *F.EOF* na hodnotu 1 a cyklus se ukončí.

Do pole *Buffer* je uložen textový řetězec „Toto je pokus.“, který má 14 znaků. Funkcí *WRITE_FILE* je 14 bajtů z pole *Buffer* uloženo na konec souboru *ZDROJ.DAT*. Poté je naplněna struktura *File_Info* požadovaným časem a datumem, který bude uložen do hlavičky souboru při jeho uzavření funkcí *CLOSE_FILE*.

Nakonec je vytvořen soubor TEXT.TXT s nastaveným archivním atributem ve struktuře *File_Info*. Do souboru je uložen textový řetězec jako v předchozím případě a soubor je uzavřen.

Čas, datum a atributy ve struktuře *File_Info* by měly být aktualizovány před každým vytvořením adresáře nebo souboru a také před uzavřením souboru, pokud bylo do otevřeného souboru zapisováno. Při uzavření souboru není pouze brán ohled na atributy ve struktuře *File_Info*, ty zůstávají nezměněny.

10.11.6 Kopírování souborů

Následující zdrojový text ukazuje příklad algoritmu, který zkopíruje soubor z jedné logické jednotky na druhou. Tu samou operaci lze samozřejmě provádět na jedné logické jednotce. Soubor z cesty LUN0:\MANUAL.TXT bude zkopírován na druhou jednotku s cestou LUN1:\MANUAL.TXT. Program předpokládá existenci souboru MANUAL.TXT na jednotce s nižším indexem.

```
FILE zdroj, cil;
unsigned char Buffer[512];

void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 1)      // musí byt dostupne alespon 2 LUN
 {
  zdroj.Status = 0;
  cil.Status = 0;

  Curr_LUN(0);
  OPEN_FILE(&zdroj, "MANUAL.TXT");
  Curr_LUN(1);
  Set_File_Info(0,0,0,1,39,33,16,4,16,28);
  REWRITE_FILE(&cil, "MANUAL.TXT");

  while (zdroj.EOF == 0)
  {
   READ_FILE(&zdroj, &Buffer, 512);
```

```
   WRITE_FILE(&cil, &Buffer, zdroj.Read);
  }

  Set_File_Info(0,0,0,1,0,34,16,4,16,28);
  CLOSE_FILE(&zdroj);
  CLOSE_FILE(&cil);
 }
 while (1);
}
```

Jsou deklarovány dvě pracovní proměnné *zdroj* a *cil* typu *FILE*. V jednotce *LUN0* je otevřen soubor MANUAL.TXT a jemu přiřazena pracovní proměnná *zdroj*. V kořenu jednotky *LUN1* je vytvořen soubor MANUAL.TXT s nastaveným archivním atributem a je mu přiřazena pracovní proměnná *cil* (atributy je možné také zkopírovat z pracovní proměnné *zdroj* otevřeného souboru do struktury *File_Info*, ukázkový program to však neprovádí).

Následuje cyklus, který čte bloky dat (512 bajtů) ze zdrojového souboru a ukládá je do cílového souboru. Do cílového souboru je vždy uložen počet přečtených bajtů *zdroj.Read*, protože z největší pravděpodobnosti poslední čtená dávka bude menší než požadovaných 512 bajtů. Až dosáhne funkce *READ_FILE* konce souboru, bude nastaven parametr *zdroj.EOF* a cyklus skončí.

Nakonec je aktualizována struktura *File_Info* (pouze demonstrativně) na aktuální čas a datum a oba soubory jsou uzavřeny.

10.11.7 Nastavení pozice čtení ze souboru

Program ukazuje použití funkce pro nastavení čtecího ukazatele souboru při čtení dat ze souboru. Čtený soubor DOPIS.TXT má na začátku uložen textový řetězec „Ahoj, jmenuji se Petr.“ Program ze souboru přečte do pole *Buffer* pouze řetězec „Petr“.

```
FILE zdroj;
unsigned char Buffer[512];

void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 0)      // musí byt dostupna alespon 1 LUN
 {
  zdroj.Status = 0;
  Curr_LUN(0);
  OPEN_FILE(&zdroj, "DOPIS.TXT");
  FILE_POINTER(&zdroj, 17);
  READ_FILE(&zdroj, &Buffer, 4);
  CLOSE_FILE(&zdroj);
 }
 while (1);
}
```

Aby bylo možné ze souboru přečíst řetězec „Petr“, je nutné nastavit čtecí ukazatele pracovní proměnné *zdroj* na začátek tohoto řetězce. K tomu slouží funkce *FILE_POINTER*. Ofset pro první znak požadovaného řetězce je 17. Funkce *READ_FILE* načte 4 znaky od aktuální pozice ukazatele a v poli *Buffer* bude uložen řetězec „Petr“ (samozřejmě bez zakončení nulovým bajtem).

10.11.8 Přejmenování souboru a adresáře

Příkladu ukazuje použití funkce pro přejmenování souboru a adresáře. Soubor X.JPG program přejmenuje na X1.JPG a adresář ADR přejmenuje na ADR1. Program předpokládá, že soubor i adresář existují v kořenu první logické jednotky.

```
void main(void)
{
 Init_Device();               // init C8051F120
 USB_Controller_Init();       // init radice USB
 USB_Device_Detect();         // test zda je pripojeno FS zarizeni
 while (Inserted == NO);      // ceka na pripojeni flash disku
 Delay(TIME_1s);
 Start_Flash_Disk();          // init flash disku
 if (Number_of_LUNs > 0)      // musí byt dostupna alespon 1 LUN
 {
  Curr_LUN(0);
  RENAME(0, "ADR", "ADR1");   // prejmenuje adresar
  RENAME(1, "X.JPG", "X1.JPG");// prejmenuje soubor
 }
 while (1);                   // konec
}
```

K přejmenování souboru i adresáře je použita funkce *RENAME*. Činnost funkce je popsána v kap. 10.5.20. Funkce změní v hlavičce souboru nebo adresáře pouze název, ostatní parametry hlavičky budou nezměněny.

10.12 Identifikace připojeného flash disku

Pokud je potřeba identifikovat konkrétní připojený flash disk, lze použít zde popsané identifikátory. Každý flash disk by měl obsahovat USB deskriptor textového řetězce sériového čísla. Specifikace [15] uvádí, že by sériové číslo mělo mít minimálně 12 znaků. Po skončení inicializační funkce *Start_Flash_Disk* (a v případě, že enumerace proběhla úspěšně → *Enumerated* = 1), je prvních 16 znaků sériového čísla flash disku načteno v globální proměnné *Serial_Number*.

```
#define MAX_LENGTH_SERIAL        17

xdata unsigned char Serial_Number[MAX_LENGTH_SERIAL];
```

Řetězec *Serial_Number* je zakončen nulou, tedy lze používat funkce z modulu *string* pro práci s řetězci. Vždy má 16 znaků, před načtením sériového čísla je řetězec vyplněn *ASCII* znaky „0“. Zleva je pak do řetězce kopírováno sériové číslo. Pokud má sériové číslo menší počet než 16 znaků, ostatní znaky zůstanou nastaveny na „0“, má-li řetězec více než 16 znaků,

bude uložen pouze tento počet. Sériové číslo může obsahovat pouze *ASCII* znaky „0“ – „9“ a „A“ – „F“. Zde jsou uvedeny příklady načtených sériových čísel z konkrétních flash disků:

MSI MEGA STICK 128 `"0000E104E7CAC707"`
Pqi Inteligent Stick PRO220 `"0000000000002E700"`

V případě, že by flash disk neobsahoval deskriptor sériového čísla, řetězec *Serial_Number* by zůstal vyplněn znaky „0“. V takovém případě lze identifikovat flash disk podle identifikátorů *USB ID VENDOR* a *USB ID PRODUCT*. Tyto parametry jsou dostupné po úspěšné enumeraci v globální struktuře deskriptoru zařízení *Device_Descriptor*.

```
Device_Descriptor.idVendor
Device_Descriptor.idProduct
```

Oba parametry jsou datový typ *unsigned int*. Parametr *idVendor* obsahuje po enumeraci identifikátor výrobce flash disku a *idProduct* obsahuje identifikátor výrobku, který stanovuje výrobce. Nevýhodou je, že flash disky stejného typu budou mít tyto identifikátory stejné, tedy identifikace nebude v tomto případě zcela jednoznačná.

Pro spolehlivou identifikaci připojeného flash disku je vhodné kontrolovat USB identifikátory ve struktuře *Device_Descriptor* i sériové číslo v poli *Serial_Number*.

10.13 Pokyny pro první použití modulů

Při použití modulů této diplomové práce pro komunikaci s USB flash diskem je třeba provést několik úkonů a splnit několik podmínek, aby byl systém funkční a program mohl být správně zkompilován.

1. Modul *init_up* pouze nastavuje mikrokontrolér C8051F120 a tedy ho lze z projektu odstranit a nastavení mikrokontroléru provést libovolným jiným způsobem, podle požadavků uživatele.

2. Mikrokontrolér musí mít specificky nastaven *crossbar*, určený pro přiřazení signálů periferií k jednotlivým portům mikrokontroléru. Pro činnost je nutné, aby byla přiřazena sběrnice *SPI* a přerušení *INT1*. Aby tyto signály byly na odpovídajících portech mikrokontroléru, musí být přiřazen *UART0*, sběrnice *SPI*, sběrnice *SMBus*, vstup externího přerušení *INT0* a *INT1*. Naproti tomu nesmí být na porty mikrokontroléru přiřazen *UART1*, *PCA*, *CP0*, *CP1* a *Timer0*.

3. Externí přerušení *INT1* mikrokontroléru C8051F120 musí být povoleno a musí mít nastavenou reakci na hranu přerušovacího signálu. Sběrnice *SPI* musí být nastavena do režimu **4 wire single master mode**, musí být nastavena do režimu **master**, signály *SCK* a *MOSI* musejí být nastaveny na *Push-pull* výstup, *SCK* signál bude v klidu na log.0 a aktivní hrana bude první – režim *SPI(0,0)*. Frekvenci hodinového signálu *SPI* nastavit maximálně na 26MHz. Také port P2.7 pro aktivaci */SS* signálu řadiče USB a port P1.3 ovládající reset řadiče USB musejí být nastaveny rovněž na *Push-pull* výstup.

4. Kompilátor prostředí *µVision3* musí být nastaven na ***Memory model : Large***, ***Code ROM Size : Large*** a ***Operating system : None***.

5. Nastavit dle dřívějšího popisu uživatelská makra. Minimálně je vhodné prověřit nastavení maker *PACKET_LED*, *CONNECT_LED*, *SYSTEM_CLOCK_X*, *MAX_LUNS*, *MAX_SECTOR_SIZE*, *WORK_WITH_ONE_LUN*. Pokud bude definováno makro *PACKET_LED* nebo *CONNECT_LED* je třeba nastavit v modulu *board* (soubor *board.h*) přiřazení příslušné LED (*LED1* nebo *LED2*) na požadovaný port mikrokontroléru (např. `sbit LED1 = 0x84; sbit LED2 = 0x85;`). Pokud nebude deklarována *LED1*, v modulu *board* musí být zakomentovaná (nebo odstraněna) funkce *Blik*.

6. Definovat funkci pro obsluhu externího přerušení *INT1* (např. podle kap. 10.9). V této obsluze je nutné volat funkci *INT_USB_Controller* z modulu *drv_usb_controller*.

7. Napětí sběrnice USB musí být minimálně 5V. Některá USB zařízení (MP3 přehrávače) nepracovala s napájecím napětím již o dvě desetiny nižším. Proto je třeba vývojovou desku *Silicon Laboratories* (viz kap. 9) napájet napětím alespoň 5,5V (je použit *low-drop* stabilizátor).

10.14 Parametry komunikačního programu

Tab. 36 obsahuje shrnutí základních výsledných parametrů a vlastností komunikačního programu této diplomové práce. Maximální přenosové rychlosti jsou uvedeny pro hodinovou frekvenci mikrokontroléru vývojové desky 98 MHz.

Tab. 36 Parametry komunikačního programu

Parametr	Hodnota
Hostitelský řadič USB (Full-speed)	MAX3421E
Podporovaná USB zařízení	USB Flash drive (USB Flash disk), (tj. Mass Storage class, Bulk-Only Transport, SCSI command set, non-removable media)
Maximální podporovaný počet LUN	16
Podporované souborové systémy	FAT16
Maximální podporovaná velikost sektoru LUN	16kB
Možnost připojení USB Hubu	NE
Doba zjištění volného místa na disku	0,5-3s
Doba vytvoření nového adresáře nebo souboru	0,2-3s
Maximální možný počet najednou otevřených souborů uživatelským programem	omezeno velikostí RAM 8051, 45 bajtů RAM / soubor
Maximální rychlost čtení dat ze souboru	~170kB/s
Maximální rychlost zápisu dat do souboru	~60kB/s
Maximální teoretická velikost najednou čtených dat ze souboru funkcí READ_FILE	64kB
Maximální teor. velikost najednou zapisovaných dat do souboru funkcí WRITE_FILE	64kB
Minimální velikost alokované datové paměti (MAX_SECTOR_SIZE = 512)	~ 2kB
Možnost přerušovat probíhající komunikaci jinými funkcemi uživatelského programu	ANO, libovolně
Velikost komunikačních modulů po kompilaci	~24kB

Doby vykonávání operací a přenosové rychlosti jsou závislé na momentálním stavu připojeného flash disku. Nevhodné uspořádání dat v logické jednotce, přílišná fragmentace souborů nebo také relativní celkové zaplnění disku výrazně zpomaluje chod funkcí pro práci se souborovým systémem, protože se výrazně prodlužuje doba prohledávání alokační tabulky. Software je napsán tak, aby bylo možné všechny komunikační funkce libovolně přerušovat v jejich činnosti (přerušením procesoru).

Přenosové rychlosti jsou silně závislé na konkrétním připojeném flash disku. Výsledná rychlost závisí na tom jak rychle dokáže firmware flash disku reagovat na nové požadavky přenosu. Flash disky jsou obvykle optimalizovány pro čtení celých clusterů (nebo jejich násobků) v jednom přenosu, nežto moduly tohoto projektu vyčítají z flash disku každý sektor zahájením nového *Bulk-Only* přenosu, což se vzhledem k výsledné přenosové rychlosti ukázalo jako méně výhodné řešení.

11 Závěr

Dle zadání práce byly prostudovány způsoby komunikace po sběrnici USB, bylo z hlediska komunikačních protokolů nastudováno záznamové zařízení typu USB flash disk a pojednáno o použitých příkazech sady rozhraní *SCSI*, transportním protokolu *Bulk-Only* a souborovém systému *FAT16*. Z nalezených USB řadičů typu hostitel byl vybrán pro realizaci práce řadič MAX3421E firmy Maxim Integrated Products. S tímto řadičem byl navržen a realizován modul pro připojení k poskytnuté vývojové desce s mikrokontrolérem C8051F120 firmy Silicon Laboratories.

Na základě získaných poznatků o standardech a potřebných komunikačních protokolech byl realizován software pro mikrokontrolér C8051F120, který umožňuje prostřednictvím modulu s řadičem MAX3421E manipulovat se soubory na USB flash disku. Software umožňuje pohyb v adresářové struktuře, vytvářet adresáře i soubory včetně záznamu datumu a času vzniku či poslední změny, umožňuje zápis a čtení dat ze souborů včetně práce s několika soubory současně. Software rovněž umožňuje práci na více logických jednotkách flash disku. Dokáže zjistit volné místo na disku, přejmenovat soubor či adresář a také umožňuje mazaní celých souborů a adresářů včetně jejich obsahu. Také umožňuje vyčíst z flash disku sériové číslo a jeho ID znaky pro jednoznačnou identifikaci konkrétního připojeného flash disku.

Software byl vyvíjen pomocí různých zařízení spadajících do třídy *USB Mass Storage* používajících stejné komunikační protokoly jako USB flash disk a to převážně v součinnosti s USB flash diskem *Intelligent Stick PRO220* firmy *Power Quotient International* (pqi). Dále byl testován na kapesním MP3 přehrávači *MEGA STICK 128* firmy *Micro-Star International*, na fotoaparátu *DMC-LS65* firmy *Panasonic* a na MP3 přehrávači *Digital Audio Player X2* firmy *MEIZU Technology*. Se všemi zařízeními systém vykazoval spolehlivou funkci.

Vložením realizovaných programových modulů do stávajících programů pro mikrokontrolér C8051F120 se získá snadná možnost pohodlného záznamu nasbíraných dat zařízením na připojený USB flash disk a tyto posléze zpracovat např. na osobním počítači. Také v opačném případě může připojený flash disk sloužit jako zdroj dat pro dané zařízení.

Literatura

[1] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips. *Universal Serial Bus Specification - revision 2.0*. USB Implementers Forum, April 27, 2000. URL:<http://www.usb.org/developers/docs/usb_20_040908.zip> [cit. 2007-01-05].

[2] Compaq, Intel, Microsoft, NEC. *Universal Serial Bus Specification - revision 1.1*. USB Implementers Forum, September 23, 1998. URL:<http://www.scaramanga.co.uk/stuff/qemu-usb/usb11.pdf> [cit. 2006-10-01].

[3] *Silicon Laboratories C8051F120/1/2/3/4/5/6/7 C8051F130/1/2/3 Mixed Signal ISP Flash MCU Family*. Silicon Laboratories, December, 2005. URL:<http://www.silabs.com/public/documents/tpub_doc/dsheet/Microcontrollers/Precision_Mixed-Signal/en/C8051F12x-13x.pdf> [cit. 2007-01-30].

[4] *USB Peripheral/Host Controller with SPI Interface MAX3421E*. Maxim Integrated Products, June, 2006. URL:<http://www.maxim-ic.com/req_full_ds.cfm?action=request&id=3639> [cit. 2007-02-15].

[5] *MAX3421E Programming Guide*. Maxim Integrated Products, June, 2006. URL:<http://pdfserv.maxim-ic.com/en/an/AN3785.pdf> [cit. 2007-02-15].

[6] *Working Draft SCSI Block Commands – 2 (SBC-2) – revision 16*. Project T10/1417-D, November 13, 2004. URL:<http://www.t10.org/ftp/t10/drafts/sbc2/sbc2r16.pdf> [cit. 2007-07-10].

[7] *Working Draft SCSI Block Commands – 3 (SBC-3) – revision 10*. Project T10/1799-D, May 17, 2007. URL:<http://www.t10.org/ftp/t10/drafts/sbc3/sbc3r10.pdf> [cit. 2007-07-10].

[8] *dpANS SCSI Primary Commands – 2 (SPC-2) – revision 20*. Project T10/1236-D, July 18, 2001. URL:<http://www.t10.org/ftp/t10/drafts/spc2/spc2r20.pdf> [cit. 2007-07-10].

[9] *dpANS SCSI Primary Commands – 3 (SPC-3) – revision 23*. Project T10/1416-D, May 4, 2005. URL:<http://www.t10.org/ftp/t10/drafts/spc3/spc3r23.pdf> [cit. 2007-07-10].

[10] BURKHARD, K. *USB - Měření, řízení a regulace pomocí sběrnice USB*. BEN - technická literatura, Praha, 2004.

[11] *Beyond Logic - Universal Serial Bus* [online]. URL:<http://www.beyondlogic.org/> [cit. 2007-03-08].

[12] ŘEHÁK, J. *USB - Universal Serial Bus - Popis rozhraní* [online]. URL:<http://www.hw.cz/Teorie-a-praxe/Dokumentace/ART327-USB---Universal-serial-Bus---Popis-rozhrani.html> [cit. 2007-02-25].

[13] *Univerzální sériová sběrnice Universal Serial Bus – USB* [online]. URL:<http://home.zcu.cz/~eckhardt/popis.html> [cit. 2007-02-25].

[14] *Universal Serial Bus Mass Storage Class Specification Overview – revision 1.2*. USB Implementers Forum, June 23, 2003. URL:<http://www.usb.org/developers/devclass_docs/usb_msc_overview_1.2.pdf> [cit. 2007-06-28].

[15] *Universal Serial Bus Mass Storage Class Bulk-Only Transport – revision 1.0*. USB Implementers Forum, September 31, 1999. URL:<http://www.usb.org/developers/devclass_docs/usbmassbulk_10.pdf> [cit. 2007-06-28].

[16] AXELSON, J. *USB Mass Storage Designing and Programming Devices and Embedded Hosts*. Madison WI: Lakeview Research LCC, 2006.

[17] DOBIASH, J. *FAT16 Structure Information* [online]. URL:<http://home.teleport.com/~brainy/fat16.htm> [cit. 2007-10-02].

[18] *Long Filename Specifications* [online]. vinDaci, January 6, 1998 URL:<http://home.teleport.com/~brainy/lfn.htm> [cit. 2007-10-02].

[19] *USB 2.0 Full-Speed Hos/Function Processor AT43USB370*. Atmel Corporation, December, 2003. URL:<http://www.atmel.com/dyn/resources/prod_documents/doc3340.pdf> [cit. 2007-02-15].

[20] *Cypress SL811HS Embedded USB Host/Slave Controller*. Cypress Semiconductor Corporation, February 2, 2007. URL:<http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/sl811hst_8.pdf> [cit. 2008-01-02].

[21] *CY7C67300 EZ-Host™ Programmable Embedded USB Host/Peripheral Controller with Automotive AEC Grade Support*. Cypress Semiconductor Corporation, November 8, 2006. URL:<http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cy7c67300_8.pdf> [cit. 2007-02-18].

[22] *ISP1160 Embedded Universal Serial Bus Host Controller*. Philips Semiconductors, December 24, 2004, URL:<http://www.nxp.com/acrobat_download/datasheets/ISP1160-05.pdf> [cit. 2007-02-19].

[23] *ISP1362 Single-chip Universal Serial On-The-Go controller*. Philips Semiconductors, December 24, 2004, URL:<http://www1.cs.columbia.edu/~sedwards/classes/2008/4840/Philips-ISP1362-USB-controller.pdf> [cit. 2007-02-19].

[24] *Vinculum VNC1L Embedded USB Host Controller I.C.* Future Technology Devices International, September, 2006, URL:<http://www.vinculum.com/documents/datasheets/DS_VNC1L-1A.pdf> [cit. 2007-02-28].

[25] *Universal Serial Bus (USB) Language Identifiers (LANGIDs)*. USB Implementers Forum, March 26, 2000. URL:<http://www.usb.org/developers/docs/USB_LANGIDs.pdf> [cit. 2007-04-02].

[26] *Technical Committee T10 – SCSI Storage Interfaces* [online]. URL:<http://www.t10.org/index.html>.