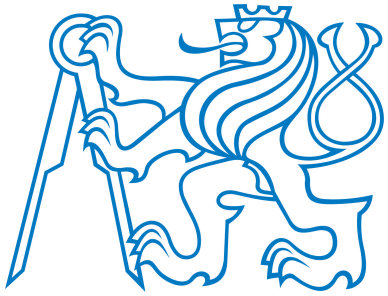
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Implementace USB rozhraní AVR mikrořadičem

Jan Smrž

Vedoucí práce: Ing. Pavel Kubalík

Studijní program: Elektrotechnika a informatika

Obor: Biomedicínské inženýrství

leden 2008

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra kybernetiky **Školní rok:** 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Jan Smrž

Obor: Biomedicínské inženýrství

Název tématu: Implementace USB rozhraní AVR mikrořadičem

Zásady pro vypracování:

1. Seznamte se s problematikou USB rozhraní realizovaného mikrořadičem AVR firmy Atmel. Prostudujte stávající způsoby realizace tohoto USB rozhraní.
2. Navrhněte a realizujte funkční zařízení, které bude umožňovat přenos dat pomocí USB rozhraní realizované mikrořadičem AVR.
3. Prostudujte existující řešení USB ovladačů. Na základě získaných informací vytvořte vlastní USB ovladač podporující navržené řešení.
4. Na jednoduché aplikaci demonstrujte funkci vytvořeného řešení. Výsledkem práce bude schéma zapojení, osazený plošný spoj, program pro mikrořadič AVR.

Seznam odborné literatury:

[1] Dokumentace výrobce mikrořadičů Atmel, www.atmel.com
[2] Specifikace USB 1.1, www.usb.org.

Vedoucí diplomové práce: Ing. Pavel Kubalík

Termín zadání diplomové práce: zimní semestr 2006/2007

Termín odevzdání diplomové práce: leden 2008

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

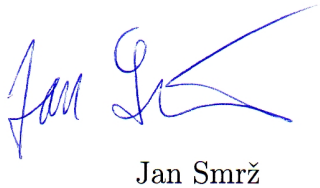
L.S.

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 19.02.2007

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.



V Praze dne 13. 1. 2008

Jan Smrž

Abstrakt

Univerzální sériové rozhraní (USB) je moderní prostředek vzájemného propojení elektronických zařízení. Nejčastěji se používá k připojení periferií k počítači. Mezi výhody této technologie patří její všeobecné rozšíření na různých platformách osobních počítačů, velký rozsah přenosových rychlostí, zdarma dostupná specifikace a podpora technologie Plug and Play. Při návrhu přenositelného elektronického zařízení připojitelného k počítači je USB mnohdy jedinou možnou volbou. Konstruktér má k dispozici řadu obvodů, které se starají o implementaci celé USB specifikace a na svém výstupu poskytují čistá data. Zpracování těchto dat se poté zpravidla děje v dalším programovatelném obvodu zajišťujícím samotnou funkčnost zařízení. Tato práce implementuje univerzální sériové rozhraní do mikrokontroléru, který jej nijak hardwarově nepodporuje, a tím realizuje protokolovou i funkční část zařízení pomocí jednoho programovatelného obvodu. Mikrokontroléry firmy Atmel z rodiny AVR mají pro tento účel vhodné předpoklady, především dostatečnou rychlost, nízkou spotřebu, široký výběr typů, cenovou dostupnost a softwarovou podporu.

Abstract

Universal Serial Bus (USB) is a modern means of communication between electronic devices. It is mainly used to connect peripherals to a personal computer. Multi-platform spread, wide range of transmission speeds, free specification and Plug and Play compliance are the main advantages of this technology. USB is frequently the only choice when designing a new portable electronic device with computer connectivity. Designer can choose from a variety of integrated circuits, which handle the whole USB specification and output the clean payload directly. Further data processing is usualy done in another programmable device that carries out a specific functionality. This diploma thesis implements Universal Serial Bus into a microcontroller that has no USB support, integrating USB protocol and specific functionality into a single programmable circuit. Atmel AVR family microcontrollers are suitable for this task offering high performance, low power consumption, device variety, reasonable price and software support.

Obsah

1 Úvod

1.1 Motivace

Valná většina současných USB periferií řeší implementaci specifikace hardwarově. Často se používají USB převodníky (například FTDI USB→serial [3]), které převádějí datový tok USB na jinou sběrnici (RS-232, RS-485, IEEE 1284) a neumožňují využít všech výhod USB sběrnice. Použití takových obvodů s sebou nese nutnost používat ovladače zařízení vydané výrobcem obvodu bez možnosti vlastní úpravy a zlepšení. Existují též hardwarové USB řadiče, které jsou součástí programovatelných jednočipových mikrokontrolérů. Řízení řadiče, jakožto i přístup k přijímaným a odesílaným datům se zajišťuje prováděním instrukcí mikropočítače. Pohodlné používání a efektivita takových obvodů je vyvážena vysokou cenou několikanásobně převyšující cenu ostatních programovatelných obvodů bez integrovaného USB řadiče. Aby bylo možné přistupovat k USB sběrnici i pomocí programovatelných obvodů, které nejsou pro takovou komunikaci nijak hardwarově vybaveny, musíme vytvořit software využívající pouze standardní vstupně-výstupní porty a nahrazující chybějící hardware efektivním používáním instrukční sady. V současné době je k dispozici kód [4], který implementuje USB rozhraní a funkčnost obvodu (přijímač dálkového ovládání - DO) plně softwarově, ale není modulární. Nelze tudíž oddělit kód USB řadiče od kódu zajišťujícího samotnou funkci přijímače DO. Jakákoliv úprava funkčnosti má za následek pracné přepisování téměř celého programu.

1.2 Cíl práce

Cílem této práce je překonat stávající stav softwarových USB řadičů vytvořením nového programového řešení, které bude při návrhu dodržovat následující kritéria: modularita funkčních celků, zapouzdřenost kódu, přenositelnost řešení uvnitř rodiny AVR, použití levného a dostupného hardware a minimalizace počtu externích elektronických součástek. V rámci dosažení takového cíle budou realizována 4 zařízení pracující na různých frekvencích a typech mikrokontrolérů, aby bylo zřetelné, v čem tkví výhody a jaké jsou hranice softwarového USB řadiče.

1.3 Univerzální sériová sběrnice - USB

Universal Serial Bus je standard organizace USB Implementers Forum. Definuje průmyslový standard Univerzální sériové sběrnice a popisuje její vlastnosti, protokol, typy přenosů, hospodaření s prostředky, potřebnou programovou podporu, elektrické a mechanické vlastnosti hardwaru. Existují tři verze specifikací: USB 1.0, USB 1.1 a USB

2.0. V této práci se budeme zabývat výhradně verzí USB 1.1, jejíž specifikace je volně k dispozici [1]. Mezi materiály, které vysvětlují USB 1.1 patří také dokument USB in a Nutshell [5], který je více didaktický a přehledně popisuje principy komunikace po USB.

Rozhraní je určeno pro komunikaci dvou elektronických přístrojů - hostitele a zařízení (host a device). Hostitel (většinou osobní počítač) může připojit až 127 zařízení. Komunikace je poloduplexní, vždy hostitelem iniciovaná, nešifrovaná, chráněná cyklickou redundantní kontrolou, přenášená po metalickém diferenciálním vedení. Velikou výhodou sběrnice je přítomnost napájecích vodičů, ze kterých lze čerpat 500mA při 5V. Modulační rychlost může být 1,5 Mbaud/s, 12 Mbaud/s nebo až 480 Mbaud/s. Sběrnice nativně podporuje technologii Plug and Play, uživatel se žádným způsobem neúčastní konfigurace zařízení, pouze je povinen dodat operačnímu systému potřebné ovladače zařízení.

Existují rozšíření USB specifikace, tzv. třídní specifikace, které blíže upravují povahu dat přenášených po USB sběrnici a definují třídy zařízení. Usnadňují tak návrhářům vývoj standardních zařízení, jako jsou myši, klávesnice, komunikační zařízení, úložiště dat a mnoho dalších. Odstraňuje se tak nutnost vymýšlet opakovaně již existující protokoly, programovat ovladače zařízení, knihovny i aplikační software. V této práci je částečně využita třída komunikačních zařízení (Communication Device Class - CDC), která poskytuje abstrakci sériové komunikace, v osobních počítačích rozšířené pod názvem COM port.

1.4 Mikrořadiče AVR

Firma Atmel vyrábí rodinu mikrokontrolérů AVR. Jedná se o jednočipové mikropočítače harwardovské architektury s redukovanou instrukční sadou (RISC). Obsahují tři typy pamětí, 8-bitovou sběrnici a množství paralelně pracujících periferií. Výkon rodiny AVR spočívá v efektivní instrukční sadě - většina instrukcí trvá 1 hodinový takt, přičemž maximální frekvence je 16 MHz až 20 MHz. Typově se dělí na obvody ATtiny a ATmega, hlavní rozdíl je v počtu vývodů a schopnostech periferií. Pouzdra, ve kterých se obvody dodávají, zahrnují klasické PDIP pouzdra i povrchově montované TQFP. Počty vývodů jsou od 8 do 100. Cenově se obvody pohybují kolem 50,- Kč. Vývojové prostředí je jednotné pro všechny typy AVR mikrokontrolérů a je k dispozici zdarma. Kompilátor jazyka C pro AVR (AVR GCC) je vydáván pod licencí GNU GPL.

1.5 Abstrakce zařízení a ovladače

Na elektronická zařízení lze nahlížet jako na zdroje či úložiště informace. Přestože existuje celá škála navzájem velice odlišných zařízení, je možné je při abstraktním

pojetí jejich funkce rozdělit do dvou skupin.

- **zařízení bloková** - realizují funkci paměti. Pro daný index (adresu) z omezeného definičního oboru dokáží uchovat určité množství dat. Požaduje se, aby pro $i \in (0, 1, 2, ..., n) : f(i) \rightarrow (0, 1, ..., 2^m)$, kde n je kapacita zařízení a m je šířka datového slova, byla funkce konstantní v čase. Podle toho, za jakých okolností lze funkci f měnit (zapisovat do paměti) se dělí na úložiště dočasná a trvalá, zapisovatelná nebo pouze pro čtení (zapisovatelná jednou).
- **zařízení znaková** - fungují jako zdroje/konzumenti datového toku. Data, která jsou ze/do zařízení přečtena/zapsána nemohou být vrácena zpět. Neexistuje zde též pojem adresy, protože z datové posloupnosti je k dispozici vždy jen jeden prvek. Typická znaková zařízení jsou klávesnice nebo terminál.

Operační systém, který pracuje s blokovými a znakovými zařízeními, poskytuje jednotné rozhraní pro oba typy. Zařízení se nazývá soubor. Jsou definovány funkce čtení, zápisu a řízení (kde je zahrnuto nastavení adresy pro bloková zařízení). Úkolem ovladače zařízení je specifickým způsobem realizovat funkce čtení, zápisu a řízení a tím vytvořit most mezi abstraktním pojetím souboru v rámci operačního systému a množinou reálných periferií, pro které neexistuje abstraktní model.

1.6 Hypotéza

Práce je založena na hypotéze, že pomocí AVR mikrořadiče a optimalizovaného kódu v Assembleru lze sestrojit softwarový USB řadič. Dále pomocí tohoto softwarového řadiče a doplňujícího programu v jazyce C lze vytvořit USB zařízení pro přenos uživatelských dat z osobního počítače do periferie.

2 Analýza

V této kapitole bude rozebrán současný stav a předvedena skupina nástrojů, kterou je možné použít pro nové řešení celé úlohy implementace USB řadiče.

2.1 Existujíci řešení USB řadičů

Současná řešení USB řadičů obsahují dedikované hardwarové řadiče (v samostatných pouzdrech), integrované hardwarové řadiče (periferie mikrokontroléru) nebo softwarové řadiče. Vzhledem k tomu, že se tato práce zabývá implementací softwarového řadiče, bude kladen důraz na odstranění nedostatků, kterými současná softwarová řešení trpí.

2.1.1 Obvody FTDI

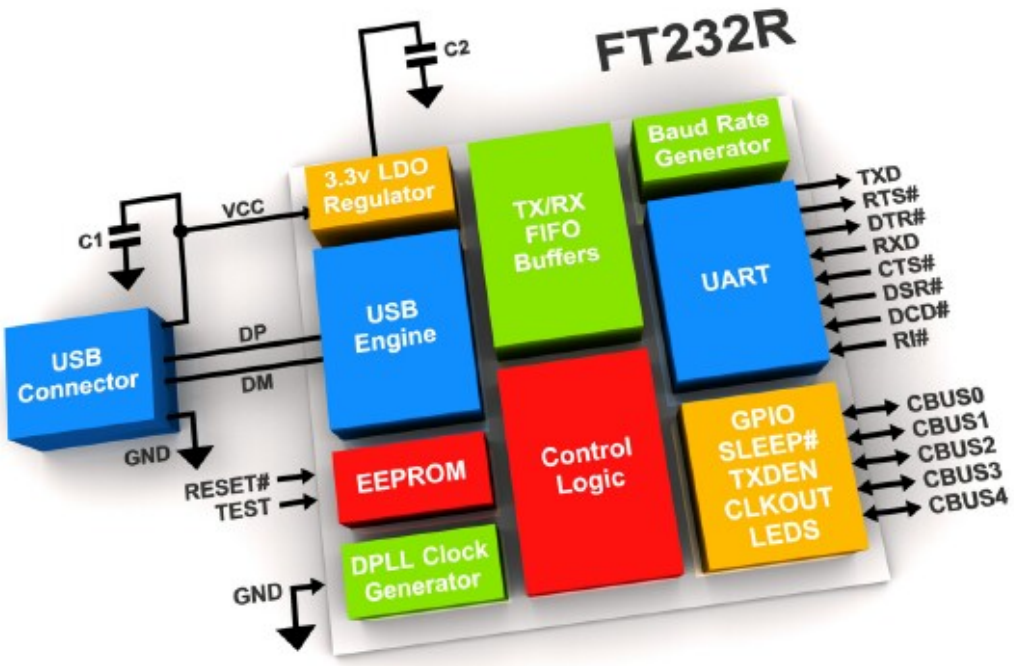
Britský výrobce Future Technology Devices International (FTDI) nabízí hardwarové USB řadiče realizující převod protokolu USB na jiné datové sběrnice (UART, parallel). Obvody nabízejí přenosovou rychlost full-speed na straně USB (12 MBaud/s). Pro svůj běh potřebují 6 MHz krystal, mohou být doplněny sériovou pamětí EEPROM (pro uložení deskriptorů). Blokové schéma je na obrázku 1. Výrobce nabízí buď ovladače, které fungují jako virtuální COM port, nebo jako obecné USB zařízení, ke kterému je nutné přistupovat voláním firemních knihovních funkcí.

obvod	**převod**	**popis**
FT232	USB→UART	výstup na sériovou linku
FT245	USB→FIFO	paralelní výstup
FT2232	USB→2xUART	dvojitý výstup na sériovou linku
FT8U100AX	USB→UART,PS2,KBD,hub	rozbočovač kombinovaný se sériovým výstupem a PS2

Tabulka 1: Přehled obvodů firmy FTDI

výhody	**nevýhody**
vysoká spolehlivost	obtížná upravitelnost
okamžitá použitelnost	nutnost používat ovladače výrobce
virtuální COM port	omezené použití pro zařízení jiné třídy
seriové čislo pro bezpečnost	vysoká cena (až 400,- Kč)
	latence přenosu

Tabulka 2: Bilance obvodů FTDI



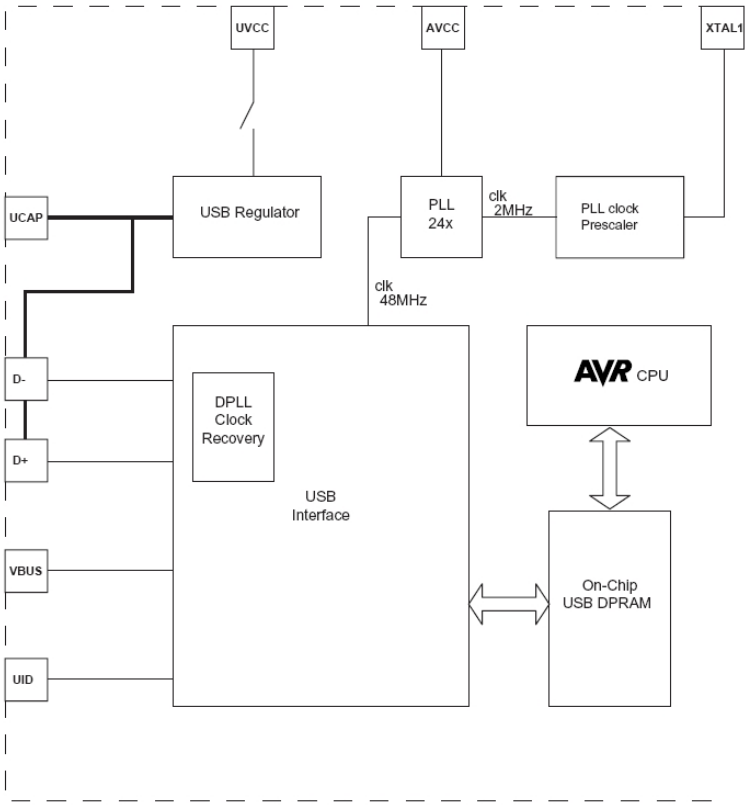
Obrázek 1: Blokové schéma obvodu FT232R firmy FTDI

Hardwarové dedikované USB řadiče jsou nabízeny i jinými výrobci a mají obdobné vlastnosti. Např. Atmel (AT76C712) nebo Silicon Labs (CT2102).

2.1.2 AVR AT90USB

Firma Atmel nabízí u šesti modelů mikrokontrolérů rodiny AVR integrovaný hardwarový USB řadič. Přístup k řadiči se děje pomocí I/O adresního rozsahu mikrokontroléru. Řadič je vybaven vlastní pamětí dvouportovou DPRAM pro ukládání přijatých/odesílaných dat. Události na USB sběrnici jsou transformovány do sady přerušení, díky čemuž lze vést komunikaci s minimálními časovými náklady. Periferie dokáže probudit zbytek kontroléru z úsporného režimu (na základě příchozího paketu). Řadič podporuje jak low-speed, tak full-speed rychlosti, obsahuje vlastní generátor hodinového signálu (48 MHz) a vlastní low-drop regulátor na 3,3 V.

Integrované USB řadiče se stávají standardní vstupně/výstupní periferií a mladší architektury mikrokontrolérů (ARM7 a výše) je často obsahují. Vytlačuje se tak použití externích řadičů s omezenými schopnostmi. Kromě mikrokontrolérů Atmel se USB ře-



Obrázek 2: Integrovaný USB řadič v AVR mikrokontroléru

výhody	**nevýhody**
spolehlivost	potřeba zkušeného návrháře
vše v jednom pouzdře	problematická kusová dostupnost
libovolná funkčnost	vysoká cena (500,- Kč)
možnost programování čipu přes USB	

Tabulka 3: Hodnocení obvodů AVR

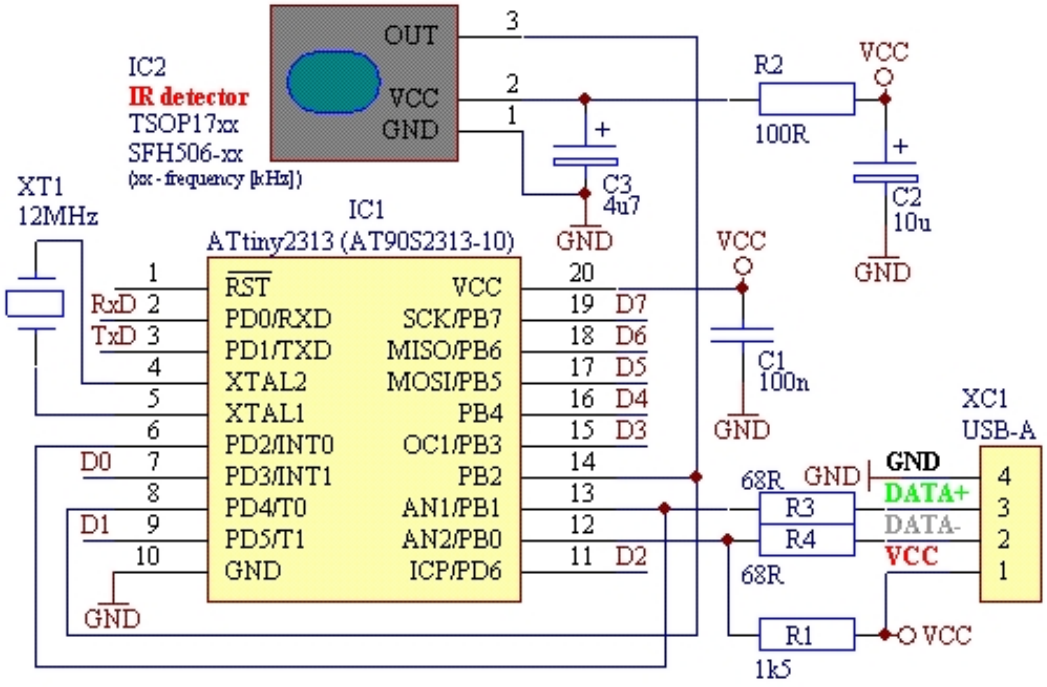
diče objevují i u firmy Microchip (např. model PIC16F745) a u řady jiných výrobců (Texas Instruments TUSB3210, Realtek, UbiCom).

obvod	Flash	RAM	E^2PROM	pouzdro	schopnosti
AT90USB1286	128 kB	8 kB	4kB	QFN 64	USB
AT90USB1287	128 kB	8 kB	4kB	TQFP 64, QFN 64	USB, On-the-Go
AT90USB162	16 kB	512 B	512 B	TQFP 32, QFN 32	USB
AT90USB646	64 kB	4 kB	2 kB	QFN 32	USB
AT90USB647	64 kB	4 kB	2 kB	TQFP 32, QFN 32	USB, On-the-Go
AT90USB82	8 kB	512 B	512 B	QFN 32	USB

Tabulka 4: Přehled obvodů AT90USB

2.1.3 IgorPlug Igora Češka

Nápad realizovat USB řadič zcela softwarově na mikrokontroléru, který USB hardwarově nepodporuje, byl uskutečněn Igorem Češkem, který v mikrokontroléru ATtiny2313 implementoval přijímač IR dálkového ovládání. Kód je psán plně v AVR assembleru.



Obrázek 3: Schéma přijímače IgorPlug

Autor používá systémové hodiny 12 MHz a získává tak 8 hodinových taktů na přijetí jednoho bitu USB low-speed signálu (1,5 MBaud/s). Přijímací rutiny pouze vzorkují USB sběrnici, deserializují bitový tok a po bytech jej ukládají do SRAM. Další operace (odstranění kanálového kódování, odstranění bit stuffingu a rozbor-parsování) jsou řešeny až po úplném přijetí paketu. Správné časování (1,5 MHz) je zajištěno vykonáváním

konstantního počtu instrukcí na přijetí jednoho bitu. Odesílací rutina je časově méně kritická a proto stihne v 8 taktech/bit přidat kanálové kódování i bit stuffing v reálném čase a není nutné paket kódováním opatřovat předem. Řešení je v hlavních rysech shodné s ukázkovým řešením pro ATtiny13 na 12MHz, kterou pro ilustraci prezentuje i tato práce.

výhody	**nevýhody**
nízká cena (40,- Kč)	vysoké nároky na zkušenost návrháře
bezproblémová dostupnost	porušení USB specifikace
malé fyzické rozměry	nezaručená spolehlivost
	programování vlastního ovladače

Tabulka 5: Hodnocení softwarového řešení USB řadiče IgorPlug

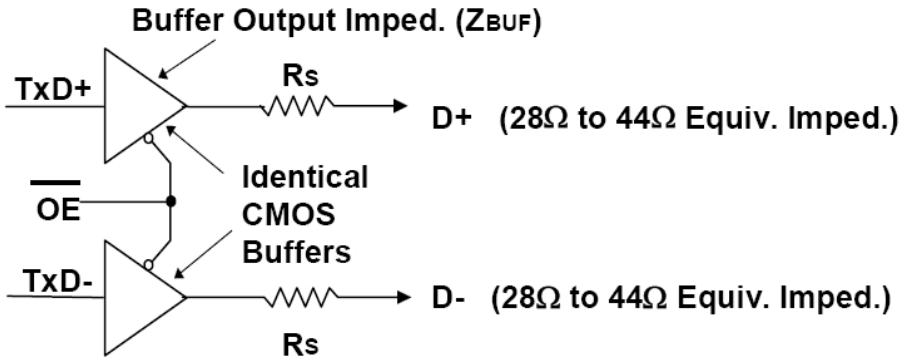
2.2 Rozbor vlastností USB

2.2.1 Souhrnné vlastnosti USB 1.1

- sériová poloduplexní sběrnice s přenosovou rychlostí 1,5 Mbaud (low speed) a 12 Mbaud (full speed)
- elektricky je signál přenášen po diferenciálním vedení a je galvanicky spojen s napájecím (Vbus) a zemnícím (GND) vodičem
- zařízení mohou být napájena přímo přes vodiče Vbus a GND, ale musejí mít omezenou spotřebu (<500 mA) a schopnost úsporného režimu.
- linková vrstva používá NRZI kódování a bit stuffing, pakety jsou chráněny CRC16 nebo CRC5
- de facto se nejedná o sběrnici, ale o propojení PTP (point-to-point) mezi hostitelem (host) nebo hubem a zařízením (device)
- síťová topologie je stromového typu, v každém uzlu je umístěn přepínač (switch, ale nazýván USB hub)
- z hlediska transportní vrstvy (roura mezi odesílatelem a příjemcem) je celé přenosové médium transparentní
- zařízení mohou obsahovat až 7 rour (pipe) a více konfigurací, zařízení se sama enumerují a splňují specifikaci Plug and Play
- maximální počet USB zařízení připojených k jednomu kořenovému přepínači je 127

2.2.2 Elektrická vrstva

Diferenciální vedení o impedanci 90 ohm slouží pro obousměrnou komunikaci. Přístup k médiu je deterministický a nemůže na něm dojít ke kolizi, protože komunikace je vždy iniciována hostitelem. Oba diferenciální vodiče jsou na obou stranách vedení buzeny CMOS hradly (obrázek 4) se sériově zapojenými rezistory (28 až 44 Ω), sloužícími jako impedanční přizpůsobení. Hostitel i zařízení mají možnost odpojení budicího hradla od linky pro případ, kdy přecházejí z vysílacího do přijímacího režimu.



Obrázek 4: Budiče USB diferenciálního vedení

Úrovně jsou na diferenciálních vodičích pro klidový stav (hostitel i zařízení jsou odpojené) na straně hostitele určeny dvěma pull-down rezistory a na straně zařízení jedním pull-up rezistorem. Připojením pull-up rezistoru k D+ nebo D- vodiči se zařízení identifikuje buď jako full-speed nebo jako low-speed (tabulka 6). S tím se mění také počáteční podmínka pro výpočet NRZI (tabulka 7).

	pull up D+	pull up D-
low-speed	-	1,5k
full-speed	1,5k	-

Tabulka 6: Volba rychlosti pomocí pull-up rezistorů

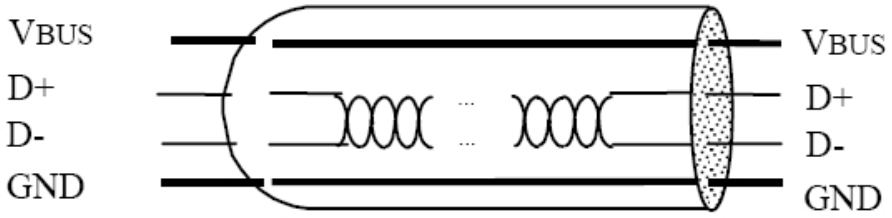
	D+	D-	počáteční stav D+
low-speed	0 V	3,3 V	0
full-speed	3,3 V	0 V	1

Tabulka 7: Klidové hodnoty na diferenciálních vodičích

Přítomnost společného potenciálu na obou diferenciálních vodičích po dobu alespoň dvou bitů znamená konec paketu a je označována jako SE0.

2.2.3 Mechanická část

Diferenciální vedení se realizují kroucenou dvojlinkou se zemněným pláštěm (obrázek 5).



Obrázek 5: Kabel USB

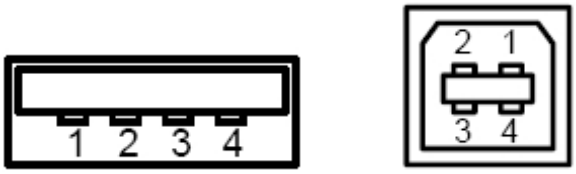
Konektory se na straně hostitele a na straně zařízení liší, takže nelze spojit dva hostitele nebo dvě zařízení k sobě. Typ A (samice) je na straně hositele a tyb B (samec) je na straně zařízení. U miniaturních zařízení se nemusí používat propojovací A-B kabel, ale zařízení je přímo opatřeno konektorem typu A (samec).

pin	**význam**	**barva vodiče**
1	Vcc	červená
2	D-	bílá
3	D+zelená	
4	GND	černá

Tabulka 8: Značení vodičů

2.2.4 Linková vrstva

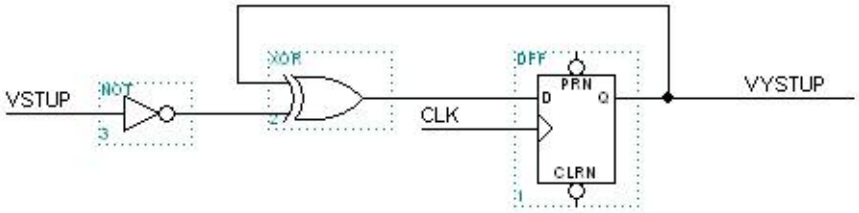
Přenášené elementární datové struktury se nazývají pakety. MTU (maximal transfer unit) neboli maximální počet bytů v paketu (bez hlaviček a CRC) je pro 0. endpoint



A (samec) B (samice)

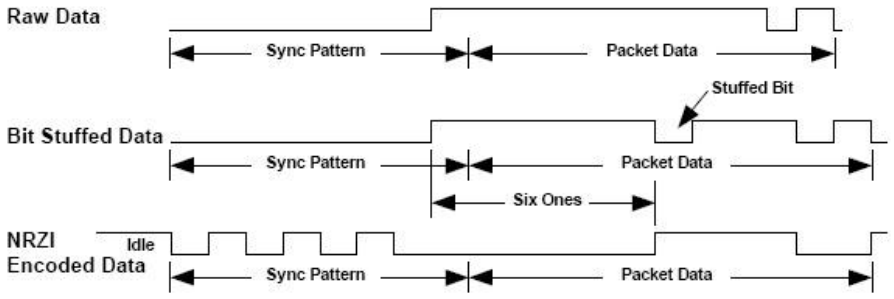
Obrázek 6: USB konektory

8,16,32 nebo 64 bytů, pro ostatní endpointy je volitelný v Endpoint descriptoru. Čistá data je tedy třeba rozdělit na více paketů, opatřit je hlavičkami a cyklickou redundantní kontrolou. Dále je nutné provést bit stuffing (za každou 6. jedničku musí být vložena nula), čímž může vzniknout neceločíselný počet bytů k přenesení. Poté se data zakódují kanálovým kódováním NRZI (Non-Return to Zero Invert), které nulu kóduje jako změnu úrovně a jedničku jako setrvání v úrovni. Pro bit v čase t platí line(t)=line(t-1) xor (!bit(t)). Realizace může proběhnout pomocí D klopného obvodu (DFF) na obrázku 7.



Obrázek 7: NRZI kodér

Procesy linkové vrstvy (bit stuffing a NRZI) jsou zobrazené na obrázku 8.



Obrázek 8: Příklad bit stuffingu a NRZI kódování

Všechny operace jsou bitově orientované a lze je tedy přepsat do kaskády proudových filtrů (obrázek 9), které mají za vstup bitový tok dat (LSb i LSB se posílají nejdříve). Skutečnost, že data se zpracovávají bit po bitu nahrává realizaci USB řadiče na hradlovém poli nebo pomocí bitového stavového automatu. Rozhodně však nelze data zpracovávat instrukcemi procesoru po celých bytech. Budeme-li například vstupní proud zpracovávat procesorem, bude se jednat o operace nad každým jedním bitem a teprve až po provedení všech dekódování a CRC výpočtů bude možné data deserializovat a skladovat po bytech.

Obrázek 9: Proudové filtry výstupní části řadiče

V případě, že odesílaná data jsou dopředu známa, je možné vyloučit z kaskády filtrů výpočet CRC16 a jeho předpočítanou hodnotu zařadit za data paketu.

2.2.5 Cyklická redundantní kontrola

Obsah paketů je doplněn 16-bitovým číslem, takzvanou cyklickou redundantní kontrolou (CRC). Odesílatel počítá CRC pro odeslanou zprávu a hodnotu CRC přikládá na konec paketu. Příjemce má dvě možnosti, jak zkontrolovat správnost paketu:

1. příjemce přečte paket, který se skládá z dat (payload) a z dvoubytového CRC. Příjemce pro data spočítá CRC a porovná ji s přijatou CRC. Pokud se obě čísla shodují, paket dorazil správně. Příjemce buď potřebuje vědět dopředu, jak dlouhá jsou data (to je na USB sběrnici nemožné), a nebo bude počítat CRC s dvoubytovým zpožděním a při konci paketu bude jisté, že výpočet proběhl pouze nad daty a ne nad přijímaným CRC.
2. příjemce přečte paket a CRC spočítá z celého jeho obsahu. Cyklická redundantní kontrola obecně má tu vlastnost, že přijímáme-li stejnou posloupnost bitů, jaká byla obsahem CRC registru, pak dostaneme vždy stejný výsledek (tzv. zbytek, residual). CRC spočítané z celého paketu tedy musí vyjít pro každý správně přijatý paket vždy stejně. Pro náš případ je zbytek 0x800D, zrcadlově 0x4FFE.

USB specifikace definuje pořadí odesílaných bitů jako LSb first, tedy nejméně významný bit je odesílán nejdříve. Pokud máme data uložena v registru, musíme provést pravou rotaci (rotate through carry right). Tím se vždy nejméně významný bit odebere a zpracuje. Po provedení sedmi rotací zbyde v registru nejvíce významný bit - a ke zpracování se tedy dostane jako poslední. Je výhodné, aby reprezentace všech čísel, která se chystáme po USB sběrnici odeslat, měla shodné pořadí bitů. Zvykem je, že binární čísla (a obecně čísla) se zapisují zleva doprava od MSb po LSb.

Výpočet CRC je možno provést dvěma způsoby. První je uveden v USB specifikaci a používá generující polynom $x^{16}+x^{15}+x^2+1$ s binárním vyjádřením 0x8005 a zbytkem 0x800D. Při generování se ale nepochopitelně používá levá rotace (přestože všechna data jsou při vysílání rotována doprava) a tím vzniká potřeba CRC před odesláním

zrcadlově převrátit. Druhý způsob výpočtu používá stejný generující polynom, ale protože používá správně pravou rotaci, musí být jeho binární vyjádření jiné, tedy 0xA001 a zbytek 0x4FFE. Rozdíl mezi metodami je kromě pořadí bitů také v počátečních hodnotách registrů.

1. metoda výpočtu CRC16:

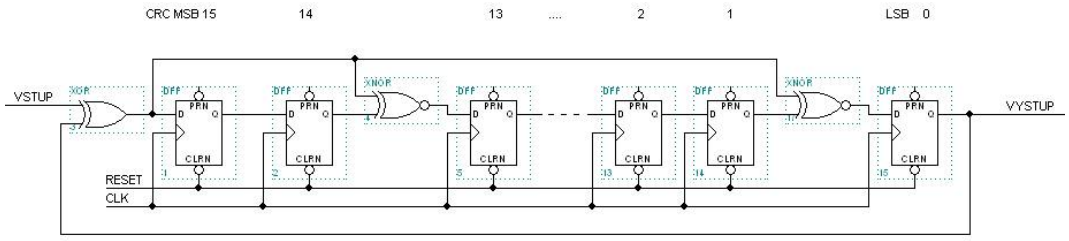
1. počáteční hodnota CRC je 0xFFFF
2. CRC=(CRC shl 1), přičemž vysunutý bit (MSb) je předmětem dalšího výpočtu
3. pokud je MSb různý od vstupního bitu (xor je 1), pak CRC=CRC xor 0x8005
4. pro všechny vstupní bity opakovat od bodu 2, jinak skončit
5. budeme-li CRC odesílat, musíme jej zrcadlově převrátit (15. bit na 0., 14. bit na 1., atd.) a znegovat (xor 0xFFFF)
6. byl-li CRC počítán pro kontrolu příchozího paketu, musí se pro bezchybný paket rovnat zbytku 0x800D

Medota 1. je nevhodná kvůli bodu 5), kdy je potřeba zrcadlově převracet a negovat CRC.

2. metoda výpočtu CRC16:

1. počáteční hodnota CRC je 0x0000
2. CRC=(CRC shr 1) or 0x8000, přičemž vysunutý bit (LSb) je předmětem dalšího výpočtu
3. pokud je LSb stejný jako vstupní bit (xor je 0), pak CRC=CRC xor 0xA001
4. pro všechny vstupní bity opakovat od bodu 2, jinak skončit
5. budeme-li CRC odesílat, můžeme jej použít bez jakékoliv úpravy
6. byl-li CRC počítán pro kontrolu příchozího paketu, musí se pro bezchybný paket rovnat zbytku 0x4FFE

2. metoda je ekvivalentní k 1., ale je zrcadlově převrácená od začátku a méně náročná na výpočet.



Obrázek 10: Výpočet CRC16 pomocí shift registru s DFF

Výpočet CRC je možné hardwarově realizovat shift registrem, jehož některé prvky mají možnost znegovat svůj obsah (funkce XOR - negace na masku). U takového CRC shift registru nemá smysl hovořit o levé nebo pravé rotaci, protože pořadí dat vstupu a výstupu je zaručeně stejné a nevzniká zde zmatek ohledně pořadí bitů. Shift registr pro počáteční hodnotu 0x0000 je uveden na obrázku 10.

Programově lze výpočet CRC nahradit voláním procedury **crcpipe** pro každý přijatý bit:

```
var crc:word; {pocatecni hodnota je 0}
procedure crcpipe(bit:byte);
var lsb:byte;
begin
 lsb:=crc and 1;
 crc:=(crc shr 1) or $8000;
 if lsb=bit then begin
  crc:=crc xor $A001;
 end;
end;
```

Obrázek 11: Výpočet CRC programově

Pro výpočty CRC a byl vyvinut program usbprep.exe, který jako parametr přijímá zprávu, která má být poslána po USB sběrnici, a opatří ji cyklickou redundantní kontrolou a bit stuffingem (obrázek 12).

2.2.6 Transportní vrstva

Na této úrovni řešíme vyměňování paketů mezi hostitelem a zařízením za účelem:

1. přenést užitečná data (payload) - rourový přístup
2. provést enumeraci (počáteční vyjednávání) speciální rourou (endpoint 0)
3. řídit stavy zařízení
4. korigovat chyby při přenosu, zařizovat handshaking

Existuje 16 typů paketů rozdělených do čtyřech skupin (detailněji v tabulce 9):

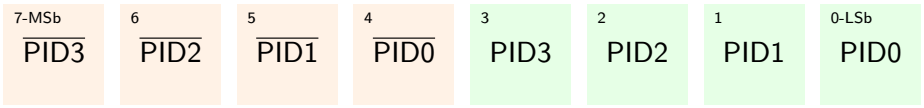
- potvrzovací (handshake) - ACK, NAK, STALL, NYET (No Response Yet)
- pověřovací (token) - IN, OUT, SETUP, SOF
- datové (data) - DATA0, DATA1, DATA2, MDATA
- speciální (special) - PREamble, ERR, Split, Ping

Identifikátor paketu (PID) je prvním bytem v paketu (po synchronizační sekvenci SYNC 0x80) a skládá se z čísla paketu (v dolní polovině bytu) a invertovaného čísla paketu (v horní polovině bytu), viz. obrázek 13.

```
c:\>usbprep 12 01 0100 00 00 00 08 065d 1031 0004 00 00 00 01

USB prep - rozdeli zpravu na pakety a opatri je CRC16 a bit stuffingem
Prevod zpravy:  12 01 0100 00 00 00 08 065d 1031 0004 00 00 00 01
Info: bitove toky jsou LSb (least significant bit) first, hex cisla LSB (least s
ignificant byte) first.
------
data1: 01001000100000000000000010000000000000000000000000000000010000
crc: 0xE713
00000001110100100100100010000000000000001000000000000000000000000000000010000
1100100011100111
delka paketu: 96 b
paket: 80 4B 12 01 00 01 00 00 00 08 13 E7
delka paketu s bit stuffingem: 96 b
delka paketu s bit stuffingem mod 8: 0
paket s bit stuffingem: 80 4B 12 01 00 01 00 00 00 08 13 E7
prvni byte, doplneno jednickami zpredu (LSB): 00000001
------
data0: 10111010011000001000110000010000010000000000000000000000000000
crc: 0x83D9
00000001110000111011010011000001000110000001000001000000000000000000000000000
1001101111000001
delka paketu: 96 b
paket: 80 C3 5D 06 31 10 04 00 00 00 D9 83
delka paketu s bit stuffingem: 96 b
delka paketu s bit stuffingem mod 8: 0
paket s bit stuffingem: 80 C3 5D 06 31 10 04 00 00 00 D9 83
prvni byte, doplneno jednickami zpredu (LSB): 00000001
------
data1: 0000000010000000
crc: 0x8F3F
00000001110100100000000010000000111111001111000 1
delka paketu: 48 b
paket: 80 4B 00 01 3F 8F
delka paketu s bit stuffingem: 49 b
delka paketu s bit stuffingem mod 8: 1
paket s bit stuffingem: 7F C0 25 80 80 1F 8F
prvni byte, doplneno jednickami zpredu (LSB): 11111110
```

Obrázek 12: Výstup programu usbprep.exe



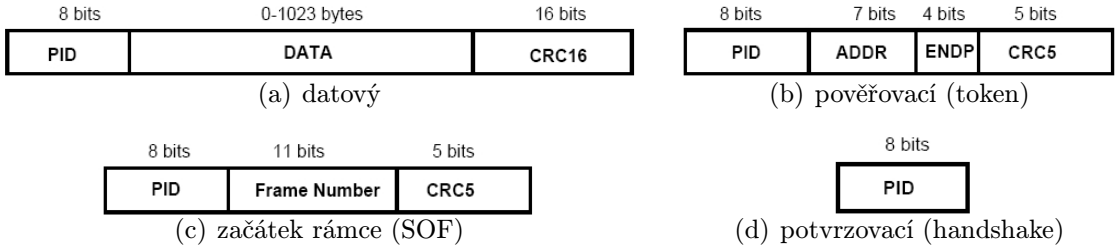
Obrázek 13: Bity v Packet IDentifikátoru

Formát paketu je závislý na jeho typu. Společné pro všechny pakety jsou úvodní synchronizace (SYNC), identifikace paketu (PID) a konec paketu (EOP, SE0 po dobu 2-3 bitů).

Na USB zařízení se stále snažíme dívat jako na rouru, která spojuje hostitele s koncovým bodem (endpointem). Přenos těchto užitečných dat se děje pomocí DATA paketů. Přenášený obsah v DATA paketech není sběrnicí interpretován a může obsahovat libo-

PID3 - PID0	PID	paket	typ
0001	0xE1	OUT Token	Token
1001	0x69	IN Token	
0101	0xA5	SOF Token	
1101	0x2D	SETUP Token	
0011	0xC3	DATA0	Data
1011	0x4B	DATA1	
0111	0x87	DATA2	
1111	0x0F	MDATA	
0010	0xD2	ACK	Handshake
1010	0x5A	NAK	
1110	0x1E	STALL	
0110	0x95	NYET	
1100	0x3C	PREamble	Special
1100	0x3C	ERR	
1000	0x78	Split	
0100	0xB4	Ping	

Tabulka 9: Typy paketů



(a) datový (b) pověřovací (token) (c) začátek rámce (SOF) (d) potvrzovací (handshake)

Obrázek 14: Formáty paketů

volná data. K řízení přenosu a k ovlivňování koncového bodu slouží jiné druhy paketů. Pověřovací pakety mají za úkol řídit směr komunikace (IN, OUT, SETUP) a potvrzovací (handshakové) pakety informovat o úspěšnosti přenosu. Specifikace vyžaduje, aby každé zařízení bylo schopno reagovat kromě datových paketů (s užitečným obsahem) také na požadavky sběrnice (requesty). Paket SETUP uvádí zařízení do režimu vyřizování požadavků. Po SETUP paketu přichází DATA paket, jehož obsah speficikuje úkon, který se od zařízení očekává. Zařízení poté v DATA paketech odesílá hostiteli výsledek požadavku. V průběhu úvodní identifikace zařízení (enumerace) patří mezi nejčastější požadavky čtení Device Descriptoru, Configuration Descriptoru a jiných datových struktur zařízení. USB specifikace umožňuje zařízení složitě se identifikovat, tvořit více konfigurací, podporovat různé jazyky a definovat násobné endpointy. Pro

naši úlohu je naopak potřeba vynechat všechny volitelné možnosti a zachovat pouze základní schopnosti zařízení.

Nutné schopnosti zařízení:

1. Odeslat 18 bytů Device Descriptoru
2. Odeslat 9 bytů Configuration Descriptoru
3. Odeslat 25 bytů descriptor komplexu (Configuration, Interface a Endpoint Descriptor)
4. Přijmout příkaz Set Address
5. Přijmout příkaz Set Configuration
6. Provést vlastní reset na základě SE0 trvajícího 10ms
7. Přijmout data pomocí posloupnosti OUT a DATA paketů

Pro získání představy, jak probíhá komunikace na USB sběrnici, je vhodné přímo změřit průběhy napětí na diferenciálních vodičích sběrnice. Je možné též použít softwarové monitory USB. Oba způsoby jsou vhodné k jinému účelu. Sledování osciloskopem umožňuje pochopit elektrickou vrstvu, kanálové kódování a doby trvání SE0. Softwarový monitoring podává obraz o vyměňovaných datech, ale z principu nedovede zobrazit jevy, o které se stará řadič USB (chyby přenosu, časování signálů atd.) Zařízení, která pasivně poslouchají na sběrnici a poskytují kompletní obraz o průbězích a datech, jsou ukázána na adrese *http://www.beyondlogic.org/usb/protocolanalysers.htm* [6].

2.3 Rozbor vlastností AVR

Mikrokontroléry AVR jsou vyráběny firmou Atmel. Mezi jejich základní vlastnosti patří:

- 8-bitová RISC architektura
- 130 instrukcí (120 pro ATtiny), většina jednotaktových
- rozsah frekvencí od 32kHz do 20MHz
- oddělená programová a datová paměť
- programová paměť flash přeprogramovatelná za běhu, 10 000 mazacích/zápisových cyklů
- operační paměť SRAM, dvoutaktový přístup
- integrovaná EEPROM, 100 000 mazacích/zápisových cyklů
- 32 pracovních registrů
- napájení 1,8 - 5,5V, nízká spotřeba, režim spánku
- množství digitálních i analogových periferií

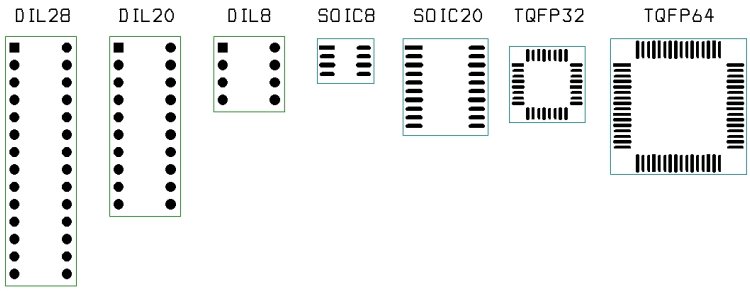
- jednoduché programování, software zdarma

AVR mikrokontroléry jsou škálovatelné, tedy je možné pořídit typy s redukovaným jádrem a omezeným počtem I/O funkcí (ATtiny) nebo plným jádrem a množstvím I/O (ATmega) či dokonce i s možností připojit externí RAM. Mezi nejmenší typ patří typy ATtiny15, 13, 12, mezi nejobsáhlejší ATmega128, ATmega2561.

Instrukční sada AVR procesorů obsahuje většinou instrukce, které trvají jeden hodinový takt. Přístupy do SRAM a skoky trvají dva takty. Provedeme porovnání čtyř typů mikrokontrolérů, které by bylo vhodné použít pro realizaci USB zařízení. Vzhledem k tomu, že tyto mikrokontroléry nepodporují hardwarově USB, nebudou hlavním měřítkem dovednosti periferií, ale pracovní frekvence procesoru a velikosti pamětí.

	ATtiny 13	ATtiny 2313	ATmega 8	ATmega 128
f_{max}	20 MHz	20 MHz	16 MHz	16 MHz
Flash	1 kB	2 kB	8 kB	128 kB
SRAM	64 B	128 B	1024 B	4096 B
EEPROM	64 B	128 B	512 B	4096 B
Pouzdro	DIL 8 SOIC 8	DIL 20 SOIC 20	TQFP 32 DIL 28	TQFP 64
Hodiny	externí vstup RC oscilátor	externí vstup RC oscilátor krystal	externí vstup RC oscilátor krystal	externí vstup RC oscilátor krystal

Tabulka 10: Srovnání 4 typů mikrokontrolérů AVR



Obrázek 15: Srovnání otisků pouzder

2.4 Operační frekvence

Rozsah frekvencí procesorů AVR určuje jedinou použitelnou modulační rychlost USB - a to 1,5 Mbaud. Aby mohl procesor generovat signál o takové modulační rychlosti, musí

být hodnota systémové frekvence celočíselným násobkem 1,5 MHz. Bez hlubší analýzy lze tedy vyloučit modulační rychlost USB 12 Mbaud, protože ani při přetaktování procesoru na 24 MHz by dva hodinové takty nestačily na zpracování jednoho bitu z linky.

Mezi použitelné násobky 1,5 MHz se řadí 12, 15, 18 a 19,5 MHz pro řadu ATtiny a 12 a 15 MHz pro řadu ATmega. Při frekvencích menších než 12 MHz vyvstává časový problém se zpracováním dat a pod 9 MHz je řadič technicky neřešitelný. Frekvence, typy hodin a ceny jsou shrnuty v tabulce 11.

frekv.	**hodiny**	**cena hodin**	**taktů/bit**	**AVR**	**cena AVR**
12 MHz	krystalový generátor	95 Kč	8	ATtiny13	33 Kč
15 MHz	krystal	10 Kč	10	ATmega8	32 Kč
18 MHz	krystal	20 Kč	12	ATtiny2313	44 Kč
19,5 MHz	prog. oscilátor	150 Kč	13	ATtiny13	33 Kč

Tabulka 11: Souhrn frekvencí, hodin a cen

2.5 Zdroje hodinového signálu

2.5.1 RC oscilátor

Kvůli minimalizaci počtu externích součástek nabízejí AVR mikrokontroléry možnost generovat hodinový signál vestavěným laditelným RC oscilátorem. Protože oscilátor není kalibrovaný, ale pouze laditelný, je nutné provést přesné měření frekvence. Program pro měření musí umět měnit hodnotu registru OSCCAL na základě stisku tlačítek. Jeho výstupem pak bude periodický signál o přesně dlouhém počtu taktů, který může na svém konci sériově vypsat hodnotu registru OSCCAL a jinak vypisovat střídající se nuly a jedničky. Takový signál pak lze zaznamenat kalibrovaným osciloskopem a pomocí kurzorů určit periodu signálu. Vydělením periody počtem taktů získáme trvání jednoho taktu. Pro procesor ATtiny13 s teplotou 22°C platí tabulka 12.

OSCCAL	perioda [us]	tiků	frekvence oscilátoru [MHz]	odchylka od 12MHz
121	193,593	2306	11,9159	$-0,70\%$
122	190,811	2306	12,0852	$+0,71\%$
123	186,226	2306	12,38	$+3,17\%$

Tabulka 12: Frekvence RC oscilátoru v závislosti na hodnotě registru OSCCAL

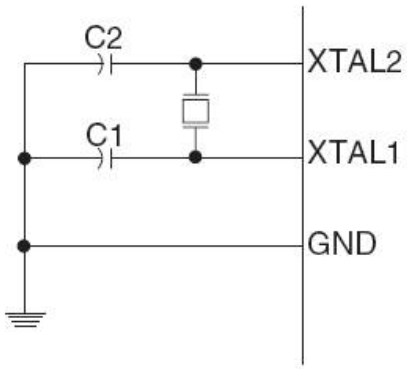
Frekvence 12MHz je osminásobkem základního signálového taktu 1,5Mb/s. USB specifikace stanovuje přesnost oscilátoru pro low-speed zařízení 15000ppm (1,5%), což je velice velkorysá hodnota. Možnost časovat mikrokontrolér RC oscilátorem je výhodná

hlavně kvůli úspoře prostoru - zařízení by nepotřebovalo téměř žádné externí součástky a vešlo by se do USB konektoru typu A. Avšak během testování RC oscilátoru se objevily skutečnosti, které naprosto vylučují jeho použití pro časování USB sběrnice.

Při měření frekvence systémových hodin (v tabulce 12) byly veličiny stabilní. Při zatížení výstupů mikrokontroléru však došlo ke zvýšenému proudovému odběru, ke vzniku úbytků napětí na vnitřních strukturách procesoru a ke změně frekvence oscilátoru o -5%. Tato změna nastávala korelovaně s měnícím se zatížením (vystavováním dat na linku), takže nešla korigovat. I přes snahu nastavit přesně jiné hodnoty frekvencí, které by také byly vhodné pro časování USB sběrnice (9 MHz nebo 10,5 MHz), se nepodařilo dodržet maximální odchylku 1,5%.

2.5.2 Krystal

Krystal poskytuje přesné časování obvodu s přibližnou tolerancí 30ppm. Jedná se o symetrickou součástku, která v zapojení s invertorem a dvěma kondenzátory (obrázek 16) C1 a C2 (od 12 pF do 22 pF) tvoří zpětnovazební filtr se jmenovitou rezonanční frekvencí. V extrémních případech se rezonátor nepodaří rozkmitat. Krystaly se běžně dodávají ve dvou pouzdrech - HC49U (vysoké) a HC49U/S (nízké). Pro naši úlohu bychom mohli mít zájem o řadu hodnot, které jsou celočíselným násobkem 1,5 MHz, avšak značná část těchto hodnot se nevyrábí (tabulka 13).



Obrázek 16: Zapojení krystalu

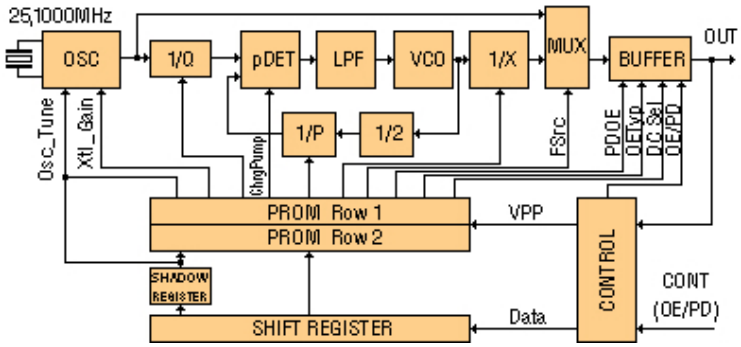
2.5.3 Krystalový oscilátor - generátor

Generátor je součástka obsahující řízený oscilátor se zesilovačem. Jeho výstupem je TTL signál o jmenovité frekvenci. Napájecí napětí je většinou 5V. V pouzdru DIL14 s přesností 100ppm je dodávána podobná řada, jako u krystalů. Pouzdra se prodávají jak kovová, tak plastová. Obě verze jsou svou velikostí nevhodné pro konstrukci

10,5 MHz	nevyrábí se
12,0 MHz	běžné, HC49U/S, HC49U/S
13,5 MHz	nevyrábí se
15,0 MHz	běžné, HC49U/S, HC49U
16,5 MHz	nevyrábí se
18,0 MHz	pouze pouzdro HC49U/S
19,5 MHz	nevyrábí se

Tabulka 13: Sourhn krystalů

miniaturních zařízení, avšak pro experimentování (například v kontaktním poli) jsou postačující. Existují i jiné typy krystalových oscilátorů, které jsou programovatelné a dodávají se v povrchově montovatelných pouzdrech. Nabídku těchto oscilátorů (SG8002 - obrázek 17) má na svých stránkách firma Spezial Electronic [7]. Cena programovatelných oscilátorů je 3x vyšší než samotného mikrokontrolérů, a proto jejich použití je spíše ukázkou miniaturizace, než návodem k reálnému využití. Velikost pouzdra je pro nejmenší typ 3,2 x 2,5 mm v ceně 175,- Kč a lze jej umístit i do omezeného prostoru USB konektoru. Jmenovitá frekvence je plně nastavitelná s přesností 50ppm.



Obrázek 17: Blokové schéma oscilátoru SG8002CE

2.6 Problémy softwarových řadičů

Řadiče USB jsou obecně dvousměrné převodníky a lze je rozložit na část vstupní a část výstupní. Kanálové kódování NRZI a bit stuffing způsobují, že přijímání paketu je výpočetně náročnější než jeho odesílání. Z časového hlediska velice náročnou operací je výpočet cyklické redundantní kontroly.

2.6.1 Kanálové kódování

Sériová asynchronní komunikace se vyznačuje absencí hodinového signálu, který by přesně stanovoval okamžik platnosti dat. Hodiny přijímače a vysílače nemají stejnou frekvenci a fázi, avšak je snahou tyto veličiny korigovat na základě přijímané datové sekvence. Korekce se děje zpravidla v obvodech fázového závěsu, které ale potřebují pro svůj běh dostatečný počet změn úrovně v přijímaném datovém signálu. Kanálové kódování NRZI kóduje přenášená data tak, aby obsahovala zaručený počet změn za čas. Statisticky lze potvrdit, že při přenosech se častěji přenáší binární nula než binární jednička. Kanálové kódování proto přiřazuje nule takové posloupnosti úrovní na lince, aby bylo dosaženo pokud možno co nejčastějšího střídání úrovní.

NRZI (Non-Return to Zero Invert) - kódování, při kterém změna na lince znamená logickou nulu. Odesílání posloupnosti nul se tedy zakóduje jako střídající se signál s frekvencí rovnou polovině modulační rychlosti (750 kHz pro low-speed).

Bit stuffing (bitová výplň) - úzce souvisí s NRZI kódováním. Není zaručeno, že signál kódovaný NRZI kódováním bude obsahovat dostatatečný počet změn. Při odesílání sekvence jedniček by byla linka neomezeně dlouho beze změny, čemuž je nutné zabránit vložením logické nuly vždy za šest po sobě následujících jedniček. Změna na lince tedy nastane nejméně každý 7. bit. Výplňové bity musí být při přijímání dat z linky odstraňovány.

Obecným problémem kanálového kódování je jeho bitová povaha. V procesorech jsou data uchovávána po bytech a pro výpočet kódování neexistují efektivní instrukce. Nezbývá tedy než data zpracovávát v cyklu bit po bitu. Nejvýhodnější je počítat kanálové kódování při odesílání dat. Při odesílání se data musí deserializovat a proto je nejlepší všechny bitově orientované operace provádět najednou po deserializaci. V tabulce 14 je uveden příklad, jaké operace se musejí provést, aby se data kódovala NRZI kódem přímo při odesílání dat.

Inicializace:

```
line=(1<<DM)
maska=(1<<DP) OR (1<<DM)
```

Odeslání 0:	Odeslání 1:	Příjem:

`line=line XOR maska`
`PORTB=line`
	`nop`
	`nop`
		`line=PORTB`
		`line=(line AND maska) XOR minule`
		`line!=0: minule=minule XOR maska`

Tabulka 14: Postup při programování kanálového kódování

Při odesílání jedničky lze následně využít dva volné takty (nop) pro počítání počtu souvislých jedniček a v případě nutnosti vložit výplňovou nulu. Tím lze původně ne-

symetrickou operaci (1 je kratší než 0) vyvážit, aby odeslání bitu trvalo vždy stejně dlouho.

2.6.2 Výpočet cyklické redundantní kontroly

USB specifikace opatřuje pakety dvěma typy CRC. Pětibitová CRC chrání pověřovací (token) pakety a paket začátku rámce. Zařízení ovšem takové pakety jenom přijímá a nikdy nevysílá. Je tedy možné se spokojit s tím, že CRC5 se u příchozích paketů nebude nikdy kontrolovat a tím odpadne problém s jejím výpočtem.

Obdobná možnost je u CRC16 v datových přenosech. Za cenu mírného snížení spolehlivosti se všechny přijímané datové pakety mohou považovat za správné. Při odesílání dat je ale nutné CRC16 počítat. U předem známých paketů (jako jsou například deskriptory zařízení) je možné CRC16 předpočítat a konstantní hodnotu přidat k datům. U proměnlivých dat (zpravidla užitečného obsahu) existuje více možností, jak CRC16 spočítat. Pokud zaručíme, že odesílaná data budou mít maximální délku 1 byte, tak lze jednoduše CRC16 předpočítat a hodnotu vyhledávat v tabulce (look-up table). V případě, že není dostatek paměti pro tabulku (512 B), lze voláním výpočtu (obrázek 11) zjistit hodnotu CRC16 programově, avšak pro více bytů výpočet zabere víc času, než je k dispozici mezi přijetím pověření a povinným odesláním dat. Pro vícebytové přenosy je vhodné používat tabulku a následující předpis pro bytově orientovaný výpočet CRC16:

```
CRC16 = (CRC16 << 8) XOR tabulka[ (CRC16 >> 8) XOR data ];
```

Počáteční hodnota CRC16 je 0xFFFF a výslednou hodnotu je nutné zrcadlit a negovat. Hodnoty tabulky lze získat bitově orientovaným výpočtem pro všechny hodnoty dat od 0 do 255.

2.6.3 Časové vytížení procesoru

Softwarový řadič se dělí na dvě části. První pracuje ve výhradním režimu, kdy jsou zakázána veškerá hardwarová přerušení, druhá pracuje v kooperativním multitaskingu s hlavní smyčkou procesoru. Část ve výhradním režimu je spuštěna externím přerušením (T0 nebo Pin Change Interrupt), zatímco kooperativní část je spuštěna voláním z hlavní smyčky procesoru.

Výhradní smyčka obsahuje:

- softwarové přerušení (4 takty), uschování PC (Program Counter)
- uložení vlajek a důležitých registrů na zásobník
- synchronizaci na polovinu bitu

- příjem paketu
- rozbor přijatého paketu
- příprava odpovědního paketu
- odeslání paketu
- změna stavu řadiče
- obnovení registrů a vlajek ze zásobníku, iret

Kooperativní smyčka zajišťuje:

- zpracování užitečných dat z přijímacího bufferu
- vyřízení vlastní funkčnosti (výpočty, komunikace jinými rozhraními)
- příprava užitečných dat do odesílacího bufferu
- výpočet CRC16 pro odesílaná data

Vyřizování požadavků enumerace se děje ve výhradní smyčce, neboť není nutné počítat CRC16 a sestavení odpovědního paketu je rychlé. Ušetří se tak RAM nutná pro buffery. U paketů do jiného endpointu než 0 je výhodné zavést jednostupňový vyrovnávací buffer. V případě, že by mělo dojít k přetečení vyrovnávacího bufferu, může výhradní smyčka rovnou zamítnout příjem/odeslání dat vysláním NAK potvrzení.

Kritické sekce hlavní smyčky - použitím globálního zákazu přerušení (cli) v hlavní smyčce může dojít k situaci, že příchod paketu bude zmeškán a obslužná rutina bude zavolána s několikabitovým zpožděním. Přijímací rutina nepotřebuje k synchronizaci celou sekvenci 0x80 (SYNC), ale pouze posledních 5 bitů. Pokud hlavní smyčka zajistí, že doby trvání kritických sekcí (mezi cli a sei instrukcemi) nepřesáhnou dobu trvání 3 bitů na USB lince, pak nehrozí ztráta paketů. V opačném případě je nutné zavést mechanismus, který zaručí, že nenastanou 3 ztráty paketů po sobě. USB hostitel se při ztrátě paketů pokusí 2x o opravu a poté přenos vzdá. Záleží také na tom, jak se k chybě postaví ovladač zařízení - zda se pokusí o další sérii přenosů. Rozhodně nelze takový postup doporučit, protože je nedeterministický a nelze zaručit a ani vyjádřit jeho spolehlivost.

3 Návrh řešení

Řešení se bude skládat ze čtyř přípravků, které budou prakticky demonstrovat klady a zápory softwarového USB řadiče. Přípravky využijí frekvence 12, 15, 18 a 19,5 MHz a budou realizovány na obvodech ATtiny a ATmega. Přípravek s ATmega8 bude programován v AVR GCC (jazyk C pro mikrokontroléry AVR), ostatní budou z úsporných důvodů psány v assembleru.

3.1 Přípravek 1. - Výstupní SPI port

Zařízení s výstupní bránou SPI rozhraní (Serial Peripheral Interface) může být použito jako budič posuvných registrů. Brána se bude skládat z datového a hodinového signálu. Použit bude obvod ATtiny13 v DIL8 pouzdře a krystalový generátor 12 MHz v plastovém DIL14 pouzdře. Vzhledem k malé programové paměti nebude obvod odpovídat žádné třídě zařízení a ovladač zařízení bude programován vlastní.

Navrhované vlastnosti zařízení:

- výstupní endpoint 1 s maximální délkou datového paketu 1 byte
- synchronní sériový bytově orientovaný výstup
- bez výpočtu a kontroly CRC
- bez dekódování NRZI a bit stuffingu
- společný konektor pro výstup dat a programování obvodu

3.2 Přípravek 2. - Regulátor s ADC a PWM

Zařízení pro realizaci regulátoru (s analogově-digitálním převodníkem a pulzně-šírkovou modulací) bude pracovat na nejvyšší možné frekvenci, a to 19,5 MHz. Použit bude obvod ATtiny13 v pouzdře SOIC8 a krystalový oscilátor bude mít miniaturní provedení (3,5x2,5 mm). Díky vyšší frekvenci bude zařízení schopno počítat CRC pro odeslaná jednobytová data. Malá programová paměť klade podobná omezení jako u předchozího zařízení (především limit na velikost deskriptorů) a proto bude nutné pro zařízení naprogramovat zvláštní ovladač.

Navrhované vlastnosti zařízení:

- jeden výstupní a jeden vstupní endpoint formují obousměrnou rouru s maximální velikostí paketu 1 byte
- výstupem bude šířkově modulovaný signál, který po zařazení dolnopropustního filtru může sloužit i jako analogový výstup v rozsahu 0 až 5V

- vstupem bude analogová veličina v rozsahu 0 až 5V, která bude převedena na číslo 0 až 255 (8 bitová přesnost).
- bez kontroly CRC přijatých paketů
- (de)kódování NRZI a bit stuffingu během příjmu/odesílání
- společný konektor pro výstup dat a programování obvodu
- aplikační software bude ukazovat použití v programu Simulink v Matlabu

3.3 Přípravek 3. - Sériové a paralelní vstupně-výstupní zařízení

Zařízení poskytující univerzální vstup-výstup pomocí 8-bitového portu a sériové linky. Realizován bude obvodem ATtiny2313 s 2 kB programovou pamětí a frekvencí 18 MHz generovanou krystalem. Přípravek se bude identifikovat jako zařízení třídy CDC (Communication Device Class) a bude pro něj použit generický ovladač usbser.sys, který je standardní součástí operačního systému Windows XP. Pro ostatní operační systémy existují podobné generické ovladače. Zařízení se bude ovládat shodně jako sériový COM port, čímž se velmi zjednoduší výroba aplikačního software. Přístup k zařízení je možný také z příkazové řádky standardními rourovými operacemi.

Navrhované vlastnosti zařízení:

- obousměrná roura skládající se ze 1 vstupního a 1 výstupního endpointu odpovídající specifikaci CDC
- výstupem a vstupem bude buď sériová asynchronní komunikace (UART) nebo paralelní 8-bitový IO
- bez kontroly CRC přijatých paketů
- (de)kódování NRZI a bit stuffingu během příjmu/odesílání

3.4 Přípravek 4. - Kompozitní USB zařízení

Pilotní přípravek diplomové práce bude realizován na obvodu ATmega8 v pouzdře TQFP32 s frekvencí 15 MHz generovanou krystalem HC49U/S. Kód mikrořadiče bude psán v jazyce C. Zařízení bude kompozitní, čili bude obsahovat 5 dílčích rozhraní, každé bude odpovídat třídě CDC. Každé dílčí rozhraní bude poskytovat data v jiném protokolu.

Navrhované vlastnosti zařízení:

- kompozitní zařízení obsahující 5 dílčích rozhraní odpovídajících specifikaci CDC
- data z USB linky budou konvertována do rozhraní SPI, I^2C, ADC, PWM, serial, parallel

- modulární kód jako knihovna pro AVR GCC
- (de)kódování NRZI a bit stuffingu během příjmu/odesílání, CRC řešeno tabulkou
- rezerva výkonu, paměti a výstupních pinů pro další rozšiřování schopností

4 Řešení

4.1 Návrh softwarového řadiče

4.1.1 Návrhové prostředí, jazyk

Návrh řadiče je vícestupňovou úlohou. U rutin příjmu a odesílání paketu je s výhodou použit jazyk Assembler pro jeho přímou vazbu na instrukce procesoru. Assembler jako jediný jazyk může plně využít všech prostředků ALU, registrů a vlajek. Cenou za úspory je velmi špatná čitelnost assemblerových programů. Naopak při programování ostatních části řadiče, které je zaměřeno na časté podmínky, větvení a výpočty, je použit vyšší programovací jazyk - C. Nespornými výhodami jazyka C je přehlednost programu, přenositelnost a snadná modifikace. Kompilátor C disponuje silnými optimalizacemi a rozdíl v efektivitě C a Assembleru je okrajovým problémem. Bohužel na mikrokontrolérech ATtiny s malou SRAM a flash pamětí nelze kompilátor použít a Assembler je pak jedinou možností.

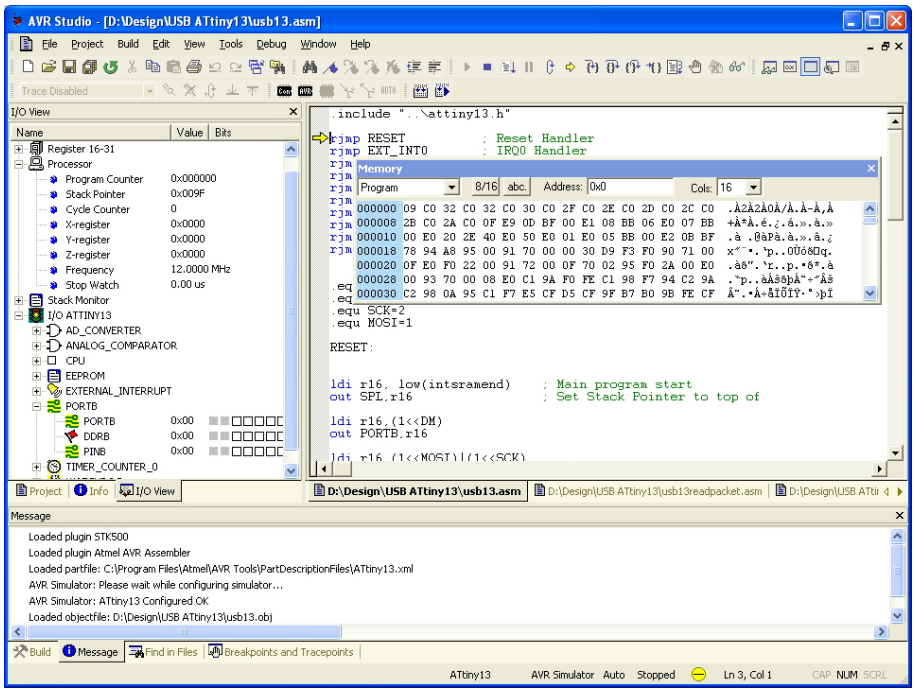
Pro vývoj programů v Assembleru bylo použito vývojové prostředí AVR Studio 4 [12], které je zdarma k dispozici na stránkách Atmelu. Prostředí se skládá z editoru, manažeru projektu, kompilátoru a debuggeru se simulátorem (obrázek 18). Software podporuje mnoho programátorů na bázi STK500 nebo JTAG ICE. Pokud nemá návrhář k dispozici žádné fyzické zařízení, může využít softwarového simulátoru mikrokontroléru (se zobrazením portů, registrů, vlajek a podrobností o procesoru). Překladač produkuje Intel Hex soubor, který obsahuje strojový kód a je čitelný jinými programátory, které nejsou AVR Studiem podporovány.

Jako kompilátor jazyka C byl použit AVR GCC z balíku WinAVR [13]. Projekt s licencí GNU GPL je orientovaný na příkazovou řádku, ale součástí balíku jsou i editory pro Windows. Použít lze však i textový editor TextPad 4.7.3 [19] s podporou spouštění programů a parsování chybových hlášek (obrázek 19). Pro automatizaci kompilace je vhodné buď naprogramovat `makefile` nebo jednodušším způsobem vytvořit dávkový soubor `.bat`.

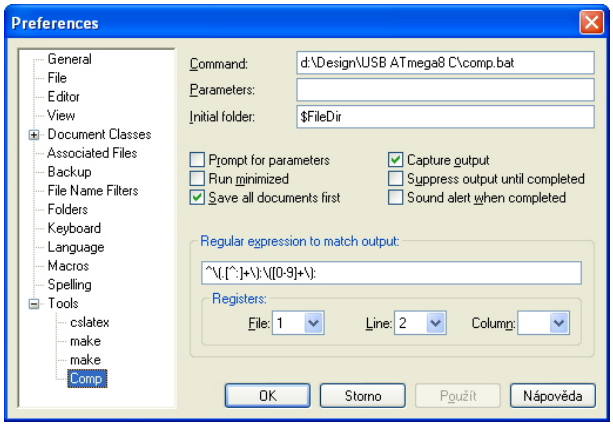
4.1.2 Operace zajišťované řadičem

Programový řadič lze dekomponovat na dílčí bloky, které budou realizovány jednotlivými rutinami. Řadič se kládá z procedur odeslání paketu, příjmu paketu, zpracování příchozího paketu, přípravy odchozího paketu a výpočtu CRC. Pro funkčnost obvodu je také potřeba naprogramovat specifické chování obsluhy datových toků ostatních rozhraní.

Softwarový řadič shodně s hardwarovým řadičem má úkol chovat se jako blok, jehož



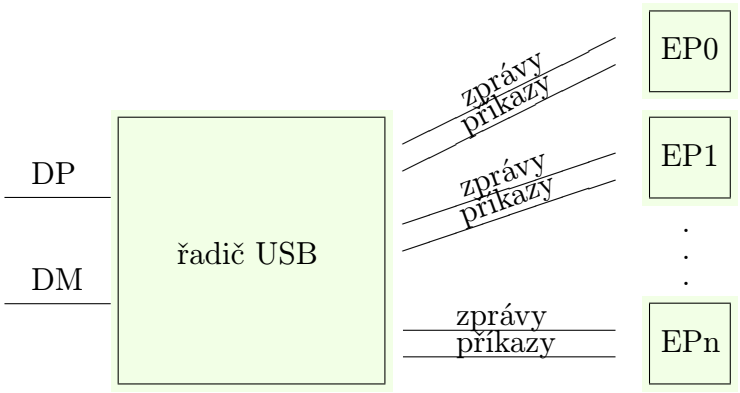
Obrázek 18: Vývojové prostředí AVR Studio 4 v režimu debuggeru



Obrázek 19: Nastavení TextPadu pro rozbor chybových hlášek AVR GCC

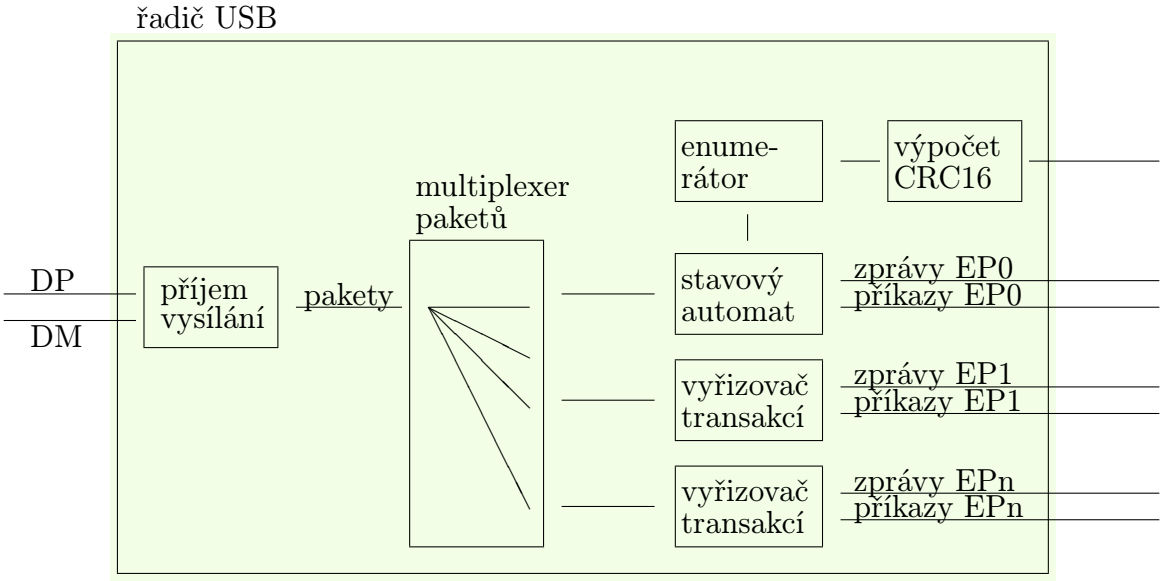
levá brána kontaktuje linkové vodiče DP a DM a pravá brána komunikuje s obsluhou v podobě zpráv nebo příkazů. Izoluje tak obsluhu od problematiky elektrické vrstvy, paketů, transakcí či enumerace. Z hlediska obsluhy je řadič černou skříňkou, ze které vystupují zprávy (posloupnosti znaků bez délkového omezení) vždy pro určitý endpoint nebo příkazy typu připrav data, restartuj se, uspi se a jiné, které mění vnitřní stavy obsluhy. Obsluha do skříňky posílá data a příkazy typu jsem připraven, potřebuji data,

nechci přijímat a podobně. Jiný mechanismus, než je posloupnost bytů a eventuelně čísel příkazů není pro obsluhu k dispozici. Obsluha nezná vnitřní funkci řadiče a snaží se chovat pouze jako konzument nebo producent datového toku.



Obrázek 20: Řadič jako blackbox

Řadič vystavuje dvě rozhraní, z nichž jedno je fyzické (vodiče DP a DM) a druhé je virtuální (zprávy). Rozhraní s dalšími zpracujícími bloky je řešeno pomocí paměťového prostoru, do kterého řadič ukládá přijatá data (zprávy) nebo čísla požadavků (příkazy) pro každý endpoint zvlášť. Konstrukce endpointů není stejná, protože endpoint 0 vykonává složitější operace než datové endpointy. Uvnitř bloku se také vyřizuje enumerace, takže pro endpointy je to operace skrytá.



Obrázek 21: Blokové schéma řadiče

Softwarově je nutné v řadiči ošetřit následující operace:

- detekce náběžné hrany na vodiči DP (řešeno externím přerušením INT0)
- přesné nalezení dalších hran (sekvenční dotazování na stav portu)
- cyklické přijímání bitů včetně odstranění NRZI kódování a bitové výplně a ukládáni do SRAM
- detekce konce paketu (testováním nulovosti obou vodičů)
- rozbor PID (identifikátor paketu)
- stanovení endpointu, pro který je paket určen
- rozbor dat, změna stavu automatu příslušného endpointu
- příprava odpovědi do paměti SRAM
- případný vypočet CRC16 a jeho uložení do paměti SRAM
- odvysílání připravené odpovědi včetně NRZI kódování a bitové vyplně
- po plnění koncové podmínky (odeslání daného počtu bytů) odvysílání konce paketu (EOP)
- návrat z přerušení (`iret`) k obnovou změněných registrů a vlajek (`pop` ze zásobníku)

Všechny operace, které se provádějí, nejsou časovány externím zdrojem, ale zcela přesně je navržen počet instrukcí a taktů. Dle toho je zcela přesně určeno i trvání bitů (logických stavů) na lince. Jediným nedeterministickým okamžikem je reakce procesoru na přicházející přerušení. Procesor má v závislosti na obsahu své pipeline prodlevu 4 až 8 taktů, než zavolá obsluhu přerušení. Obsluha tedy neví přesně, jaký čas uplynul od náběžné hrany a proto musí sledovat linku a hledat další hrany. Hledání je popsáno v kapitole 4.4.3. Správná synchronizace je nezbytná pro správný příjem paketu a spotřebuje 3 hrany synchronizační sekvence 0x80.

Samotný příjem paketu s dekódováním NRZI je časově náročná operace a správný návrh posloupnosti instrukcí je klíčovým problémem celého řadiče. Při návrhu byly použity metody časové optimalizace kódu, které šetří počet taktů procesoru za cenu zesložitění struktury programu, znečitelnění kódu a zvýšení spotřeby programové paměti. Časování smyček vyžaduje mnohdy použití instrukce `nop`, která pouze čeká 1 takt, a v běžných programech se nevyskytuje. Při čtení paketu je použit jediný čítač, a to na počítání posloupnosti jedniček pro odstranění bitové vyplně. Čítání počtu bitů je uskutečněno rotováním datového registru a celkový počet bytů je zahrnut do adresy v dvojregistru X. NRZI dekódování je řešeno střídavým čtením DM a DP podle toho, zda došlo ke změně či ne. Tato změna není držena v registru, ale je zabudována v kódu (rozdělením programu na dvě poloviny). U rychlejších frekvencí (přípravky 2. a 3.) je použita operace XOR (v AVR `eor`) a program se značně zjednoduší.

Detekce konce paketu se děje jako druhá operace po načtení hodnoty portu. Je realizovaná jednoduchou funkcí AND, která musí skončit se ZF=1 (Zero Flag). Díky persi-

stenci ZF, která není následnými operacemi přepsána, je možné provést skok `breq` až o několik instrukcí dále (nebo vůbec). Vzhledem k tomu, že operace příjmu a odesílání je nutné propracovávat na úrovni strojového kódu (důležitý je i počet bytů v programové paměti), je vždy použit jazyk Assembler. U knihovny v C je Assembler použit také, ale opatřen interfacem jazyka C pro další snadné používání.

Operace rozboru PID, stanovení cílového endpointu a další jsou řešeny běžným C kódem bez složitých myšlenkových pochodů. Výpočet CRC už ovšem triviální není. U metody, která v cyklu vypočítává bit po bitu je nutné urychlovat cykl vyvarováním se zbytečných skoků. U metody s tabulkou (Look-Up-Table), která je uložena v programové paměti, je použita adresace přes registr Z.

Odvysílání připravené odpovědi je stejně časově kritickou operací, jako příjem paketu. Z povahy NRZI (kde pro odvysílání jedničky není nutné provádět žádnou instrukci) jsou kódy kratší, ale přibývá trojité počítání (počtu bitů, počtu souvislých jedniček a počtu bytů), které časovou výhodu NRZI zcela vyrovná. Návrat z přerušení a obnova registrů je rutinní programátorskou operací.

4.1.3 Vztahy rutin softwarového řadiče

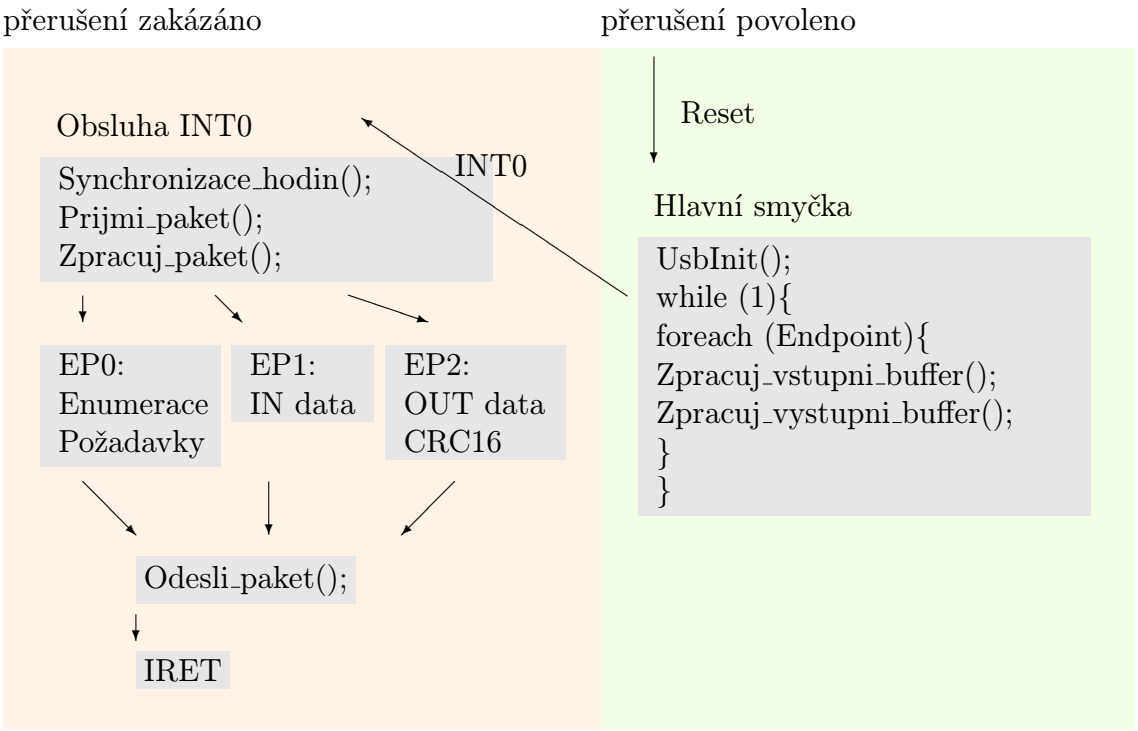
Vstupním bodem řadiče je obsluha přerušení vyvolaného změnou stavu na USB lince (externí INT0). Během obsluhy je přijímán paket a jsou zakázána všechna hardwarová přerušení, protože běh programu je přesně časově naplánován a nesmí dojít ke zdržení. Další operace, jako je zpracování paketu, příprava odpovědi a odeslání paketu jsou zavolány z této obsluhy a taktéž nesmějí být přerušeny. Příprava dat se děje v oddělené rutině, která buď běží v hlavní smyčce programu, nebo je z ní podle plánu volána. Časově není kritická a může být přerušena libovolnou událostí. Vztahy znázorňuje obrázek 22.

4.1.4 Rutiny řadiče a jazyk C

Pro dosažení přesného časování bylo nutné vytvořit rutiny čtení a zápisu v Assembleru. Aby mohl zbytek programu, který je psán v C, využít tyto rutiny, je nutné při linkování sdělit linkeru, kde se mají hledat nevyřešené vazby na assemblerovské funkce (obrázek 23).

Kompilaci je poté nutno provést v takovémto pořadí:

```
avr-gcc -mmcu=atmega8 -x assembler-with-cpp -c -O2 usb.c
avr-gcc -mmcu=atmega8 atmega8usb.c -c -O2
avr-gcc -mmcu=atmega8 -o atmega8usb.out atmega8usb.o usb.o
@del atmega8usb.asm
@avr-objdump -S atmega8usb.out >> atmega8usb.asm
@avr-objcopy -j .text -O ihex atmega8usb.out atmega8usb.hex
```

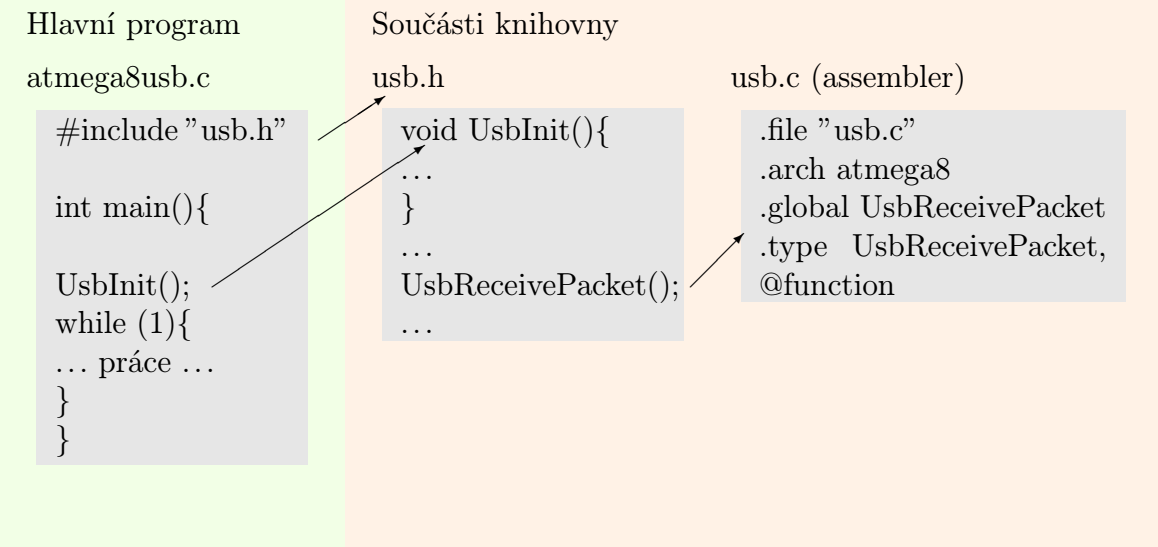


Obrázek 22: Vztahy rutin řadiče

4.1.5 Stavová tabulka zařízení

Program, který bude rozebírat přišlé pakety, rozhodovat se o jejich sdělení a odesílat zpět odpověď, bude pracovat podle schématu přijmi-vyhodnoť-odpověz. Rutiny tohoto analyzátoru (parseru) se zavolají jednou po příchodu paketu, provedou se a uvolní procesorový čas. Rutiny nebudou obsahovat žádné čekání. Vzhledem k omezeným možnostem procesorů AVR nebude nikde uložena celá zpráva, která je předmětem transakce. Každý blok (max. 8 bytů) bude do bufferu vložen těsně před odesláním a přijatá data budou ihned po přijetí zpracována. Parser bude tedy pracovat obdobně jako stavový automat, jehož vstupem je příchozí paket a výstupem odchozí paket a změna vnitřního stavu. Požadavky, které musí zařízení plnit:

0. užitečná data (rqData)
1. device descriptor (rqDevDesc)
2. configuration descriptor (rqConfDesc)
3. set address (rqSetAddr)



Obrázek 23: Knihovní vazby

Fáze transakce (x-action), kterými zařízení prochází:

0. klid, výchozí stav (xaNone)
1. požadavek na zápis (OUT paket) (xaOut)
2. požadavek na čtení (IN paket) (xaIn)
3. požadavek na nastavení (SETUP paket) (xaSetup)
4. ukončení zápisové transakce (IN paket přicházející při zápisové transakci) (xaWriteEnd)
5. ukončení čtecí transakce (OUT paket přicházející při čtecí transakci) (xaReadEnd)
6. odmítnutí ukončení čtecí transakce (OUT paket přicházející při čtecí transakci, aniž by byla potvrzena poslední data) (xaReadEndFail)
7. čekání na potvrzení datového paketu při čtecí transakci (xaInAckWait)

Stavy by měly být rozšířeny o počítadlo odeslaných bytů a o další pomocné proměnné (spánek, stav bez adresy, úsporný režim), avšak kvůli úspoře je možné některé stavy vynechat.

4.1.6 Deskriptory zařízení, enumerace

Enumerace je proces počátečního vyjednávání, kdy hostitel a host komunikují za účelem vzájemného poznávání a nastavování. Během enumerace posílá zařízení informace

Stav	Příchozí paket	Nový stav	Odchozí paket
cokoliv	SETUP	xaSetup	-
xaNone (klid)	OUT	xaOut	-
	IN	xaInWaitAck	DATA1
	ACK, NAK	xaNone	-
	DATAn	xaNone	NAK
xaIn	OUT	xaReadEnd	-
	IN	xaInWaitAck	DATA0 nebo DATA1
	ACK, NAK	xaNone (transakce zrušena)	-
	DATAn	xaIn	NAK
xaInWaitAck	OUT	xaReadEndFail	-
	IN	xaInWaitAck	DATAx znovu odeslat
	ACK	xaIn (+připravit další paket)	-
	NAK	xaIn	-
	DATAn	xaNone (transakce zrušena)	NAK
xaReadEndFail	OUT	xaOut (transakce zrušena)	-
	IN	xaInWaitAck (transakce zrušena)	DATA1
	ACK, NAK	xaNone (transakce zrušena)	-
	DATAn	xaNone (transakce zrušena)	NAK
xaOut	OUT	xaOut	-
	IN	xaWriteEnd	zero DATA
	ACK, NAK	xaNone (transakce zrušena)	-
	DATAn	xaOut (paket přijat v pořádku)	ACK
xaWriteEnd	OUT	xaOut (transakce zrušena)	-
	IN	xaWriteEnd	zero DATA
	ACK	xaNone (data odeslána v pořádku)	-
	NAK	xaNone (transakce zrušena)	-
	DATAn	xaNone (transakce zrušena)	NAK
xaReadEnd	OUT	xaOut (transakce zrušena)	-
	IN	xaInWaitAck (transakce zrušena)	DATA1
	ACK,NAK	xaNone (transakce zrušena)	-
	DATAn	xaNone (transakce potvrzena)	ACK
xaSetup	OUT	xaOut (setup zrušen)	-
	IN	xaInWaitAck (setup zrušen)	DATA1
	ACK, NAK	xaNone (setup zrušen)	-
	DATAn	xaOut nebo xaIn (dle obsahu dat)	ACK

Tabulka 15: Tabulka stavů zařízení

o svých hardwarových a softwarových prostředcích hostiteli, zatímco hostitel poskytuje zařízení adresu a mění vnitřní stavy zařízení.

Pokud budeme chápat komunikaci mezi hostitelem a zařízením jako obousměrnou rouru, po které proudí užitečná data, nutně dojdeme k potřebě odlišit data, která jsou předmětem enumerace a ostatní data, která jsou závislá na funkci zařízení, kvůli které bylo sestrojeno. USB specifikace řeší potřebu více komunikačních kanálů zavedením tzv. endpointů. Endpoint je funkční celek, který má k dispozici jednosměrnou rouru, po níž komunikuje s hostitelem. Pokud je vyžadována obousměrná komunikace, je potřeba zavést dva endpointy, každý v jednom směru. Výjimku tvoří nultý endpoint,

který je obousměrný a je určen pro vyřizování enumerace. Vlastnosti ostatních endpointů jsou předány hostiteli během enumerace ve speciálních datových strukturách (endpoint deskriptorech).

Offset	Popis	Hodnota
Device Descriptor		
0	bLength	0x12
1	bDescriptorType	0x01
2,3	bcdUSB	0x0100
4	bDeviceClass	0x00
5	bDeviceSubClass	0x00
6	bDeviceProtocol	0x00
7	bMaxPacketSize	0x08
8,9	idVendor	0x065d
10,11	idProduct	0x1031
12,13	bcdDevice	0x0004
14	iManufacturer	0x00
15	iProduct	0x00
16	iSerialNumber	0x00
17	bNumConfigurations	0x01

Offset	Popis	Hodnota
Configuration Descriptor		
0	bLength	0x09
1	bDescriptorType	0x02
2,3	wTotalLenght	0x0020
4	bNumInterfaces	0x01
5	bConfigurationValue	0x01
6	iConfiguration	0x00
7	bmAttributes	0x80
8	bMaxPower	0x32
Interface Descriptor		
9	bLength	0x09
10	bDescriptorType	0x04
11	bInterfaceNumber	0x00
12	bAlternateSetting	0x00
13	bNumEndpoints	0x02
14	bInterfaceClass	0x00
15	bInterfaceSubClass	0x00
16	bInterfaceProtocol	0x00
17	iInterface	0x00
Endpoint 1 Descriptor		
18	bLength	0x07
19	bDescriptionType	0x05
20	bEndpointAddress	0x01
21	bmAttributes	0x00
22,23	wMaxPacketSize	0x0001
24	bInterval	0x00
Endpoint 2 Descriptor		
25	bLength	0x07
26	bDescriptionType	0x05
27	bEndpointAddress	0x82
28	bmAttributes	0x00
29,30	wMaxPacketSize	0x0001
31	bInterval	0x00

Tabulka 16: Deskriptory

Pokud by se komunikace po rouře skládala pouze z užitečných dat (například výsledků měření), neměl by hostitel k dispozici žádný nástroj, kterým by měnil vnitřní stavy zařízení a nemohl by ovlivňovat jeho chování. Aby mohl hostitel posílat kromě dat také příkazy, zavádí USB specifikace zvláštní druh paketů (setup packet), které říkají, že data předaná v následné transakci nejsou určena pro funkčnost zařízení, ale pro řízení stavů zařízení.

Endpoint 0 musí reagovat na setup pakety a odesílat informace, které jsou po zařízení

požadovány. Abychom mohli navrhnout program, který bude požadavky vyřizovat, musíme si uvědomit, z jakých atomických operací se komunikace po USB skládá.

Transakce
Problém, který obecně nastává u datových přenosů, je konečná délka paketu. MTU (maximum transfer unit) udává počet bytů, které smějí být přeneseny v rámci jednoho paketu. MTU reflektuje hardwarová omezení, například nepřesnost synchronicity příjemce a odesílatele, velikosti vstupně/výstupních bufferů, účinnost CRC a podobně. Pokud je potřeba přenést data, která jsou delší než MTU, pak je nutné zavést fragmentaci dat (rozdělení na části) a vytvořit pro ni prostředky v protokolu. Stejně jako potvrzovací pakety (Handshake) sdělují úspěšnost přenosu paketu, pak musí existovat obdobný mechanismus, který potvrzuje úspěšné přesenení celé zprávy, která byla na rozdělena na dílčí pakety. Takový mechanismus je definován v rámci tzv. transakcí:

1. vstupní transakce - přenáší data ze zařízení k hostiteli
2. výstupní transakce - přenáší data od hostitele do zařízení
3. řídící transakce - posílá řídící data od hostitele do zařízení

Jednotlivé typy transakcí přenášejí pomocí posloupnosti datových paketů všechny části zprávy. Po přenesení všech fragmentů se směr komunikace obrátí a od konzumenta zprávy se očekává zaslání datového paketu s nulovou délkou. Zero Data paket je obdobou ACK paketu s tím rozdílem, že ACK potvrzuje správně přijatý fragment, zatímco ZERO data potvrzují správně přijatou celou zprávu.

V rámci enumerace je úkolem přenést zprávy od zařízení k hositeli. Jedná se o deskriptor zařízení (Device Descriptor) a konfigurační deskriptor (Configuration Descriptor), jejichž nejjednodušší tvar pro zařízení s obousměrnou rouru (EP1 a EP2) je uveden v tabulce 16. U každého zařízení se hodnoty v deskriptorech liší.

Deskriptory lze připravit na odesílání tak, že se data rozdělí na osmice a ty se opatří CRC16.

```
Device Descriptor:
DATA1 packet - CRC 0xe713 - Device Descriptor
80 4b 12 01 00 01 00 00 00 08 13 e7 eop
DATA0 packet - CRC 0x83d9 - Device Descriptor
80 c3 5d 06 31 10 04 00 00 00 d9 83 eop
DATA1 packet - CRC 0x8f3f - Device Descriptor
80 4b 00 01 3f 8f eop
```

Configuration Descriptor:

```
DATA1 packet - CRC 0xa20a - Configuration Descriptor
80 4b 09 02 20 00 01 01 00 80 0a a2 eop
DATA0 packet - CRC 0x7d04 - Configuration Descriptor a Interface Descriptor
80 c3 32 09 04 00 00 02 00 00 04 7d eop
DATA1 packet - CRC 0x2f72 - Interface Descriptor a Endpoint 1 Descriptor
80 4b 00 00 07 05 01 00 01 00 72 2f eop
DATA0 packet - CRC 0xbfe0 - Endpoint 1 Descriptor a Endpoint 2 Descriptor
80 c3 00 07 05 82 00 01 00 00 e0 bf eop
DATA1 packet - CRC 0x0000 - konec
80 4b 00 00 eop
```

V rámci enumerace je zařízení žádáno o zaslání konfiguračního deskriptoru pomocí požadavku ve kterém je specifikována přesná délka dat, kterou má zařízení vrátit. Při prvním čtení je z konfiguračního deskriptoru žádáno pouze 9 bytů. Z přijatých dat (položka `wTotalLength`) se hostitel dozví, jak velký je konfigurační deskriptor včetně poddeskriptorů (Interface, Endpoint, HID a jiné). Oproti přirozenému očekávání, že hostitel při druhém čtení zažádá o přesnou délku, však může přijít požadavek na délku větší. Běžně se jedná o 255 bytů, ale hodnota může být libovolná). V takovém případě se očekává, že zařízení pošle dat méně, přesně tolik, jaká je velikost deskriptoru. Předčasný konec hostitel rozpozná tím, že přišel paket, který je kratší než 8 bytů. U deskriptorů, jejichž velikost je násobkem 8, je z důvodu detekce konce přenosu nutné odeslat navíc nulový paket. Posílání více dat, než očekává hostitel, je zakázáno. U deskriptorů, které jsou větší než 255 bytů, nastává situace, kdy hostitel z neznámého důvodu ani podruhé nezažádá o plnou délku, ale pouze o 255 bytů (zřejmě z důvodu fixní velikosti bufferu vyjednávacího ovladače). Je nezbytně nutné odeslat přesný počet bytů (tzn. poslední paket se 7 byty). Až ve třetím čtení je zažádáno o deskriptor s plnou délkou, avšak nelze zaručit, že se nebude jednat o hodnotu, která bude jenom částí celkové délky deskriptoru. Uvedené případy jsou komplikací pro vyřizování enumerace. Odpovědní pakety s deskriptory jsou připraveny dopředu a CRC16 je spočítána pro přesné délky paketů. A protože nelze posílat více dat, než o kolik hostitel žádal, musí si zařízení připravit jinou odpověď na žádost o 9 bytů, jinou na žádost o 255 bytů a jinou na žádost o například 320 bytů (velikost deskriptoru 4. přípravku). Pokud ale hostitel požádá o 319 bytů a zařízení pošle 320 bytů, bude transakce zamítnuta a hostitel může zařízení odpojit. Pokud by tedy měla být enumerace za každých okolností spolehlivá, nesmělo by zařízení mít deskriptory připravené, a CRC16 by se muselo počítat za běhu.

U zvláštního druhu zařízení - tzv. kompozitních USB zařízení, je definován deskriptor, který se odesílá mezi konfiguračním a interface deskriptorem. Jedná se o Interface Association Descriptor (IAD) v tabulce 17, který sjednocuje více rozhraní do jednoho funkčního celku. Počet IAD deskriptorů odpovídá počtu dílčích zařízení, pro které bude zvlášť instalován ovladač. U přípravku 4. je použito 5 Interface Association deskriptorů. Čísla rozhraní (Interface) musí být pro jedno dílčí zařízení souvislá (po sobě jdoucí), protože položky IAD obsahují jen první interface (`bFirstInterface`) a počet interface (`bInterfaceCount`), takže neexistuje způsob, jak definovat zařízení s rozhraními

například 0, 1,4 a jiné s 3,5,6.

Offset	Popis	Hodnota
Interface Association Descriptor		
0	bLength	0x08
1	bDescriptorType	0x0B
2	bFirstInterface	0x00
3	bInterfaceCount	0x01
4	bFunctionClass	0x00
5	bFunctionSubClass	0x00
6	bFunctionProtocol	0x00
7	iFunction	0x00

Tabulka 17: Interface Association Descriptor

4.2 Časová optimalizace kódu

Návrh řadiče vychází z neoptimalizovaného kódu, který je nutné pomocí vhodných metod upravit tak, aby přijetí jednoho bitu trvalo přesný počet taktů. Soustřeďme se nyní pouze na real-time operaci příjmu paketu. Předpokládejme, že synchronizační sekvence je již přijata a že načtení vstupu (vzorkování linky) je třeba provést každý první takt v celkovém cyklu. Navrhneme smyčku, jejíž trvání bude vždy trvat konstantní počet hodinových taktů. Teprve poté budeme optimalizovat posloupnosti instrukcí tak, abychom vyhověli požadavkům na přesné trvání cyklu v délce 8 (pro 12 MHz), 10 (pro 15 MHz), 12 (pro 18 MHz) a 13 (pro 19,5 MHz) hodinových taktů.

Smyčka musí zvládnout:

1. načíst bit z portu B
2. provést dekódování NRZI
3. dekódovat bit stuffing (po šesti jedničkách vynechat cyklus)
4. deserializovat bity (vrotovat je do registru)
5. po 8 bitech zapsat data do SRAM
6. detekovat přítomnost SE0 (DP i DM v nule) a opustit smyčku

U rutiny odesílání platí v určité obměně totéž. K docílení přesně dlouhých cyklů pro každý bit použijeme metody časové optimalizace kódu.

4.2.1 Rozvinutí cyklu (unbundling)

Cyklus je opakované provádění části kódu. Počet provedených iterací se udržuje v pomocné proměnné (iterátoru). Celkový požadovaný počet iterací (průchodů) nemusí být

dopředu znám a zastavovací podmínka (podmínka pro ukončení provádění cyklu) nemusí být na hodnotu iterátoru vázána. Sledujme ale takové cykly, u kterých je předem znám počet iterací a jedná se o konstantu. V různých jazycích by se takový cyklus řešil konstruktem for.

```
ldi i,8
cyklus:
  cpi i,8
  breq ... začáteční akce
  cpi i,1
  breq ... závěrečná akce
  ; ... tělo cyklu
  dec i
brne cyklus
```

```
byte i;
for(i=8;i>0;i--){
  if (i==8){
    // ... akce na začátku
  }
  if (i==1){
    // ... akce na konci
  }
  // ... tělo cyklu
}
```

Doba provádění jednoho průchodu je zvýšena o kontrolu zastavovací podmínky. Rozvinutím se rozumí zopakování těla cyklu několikrát za sebou bez počítání iterátoru. Tím se ušetří jednak počítání (1 takt) a pak skok (2 takty). V našem případě ale není výhodné rozvinout cyklus úplně, protože bychom spotřebovali 8x více programové paměti. Unbundling 1. a 8. cyklu by vypadal takto:

```
rjmp  ... začáteční akce
; ... tělo cyklu
ldi i,6
cyklus:
  ; ... tělo cyklu
  dec i
rjmp cyklus
breq ... závěrečná akce
; ... tělo cyklu
```

```
unsigned char i;
// ... akce na začátku
// ... tělo cyklu
for(i=6;i>0;i--){
  // ... tělo cyklu
}
// ... akce na konci
// ... tělo cyklu
```

Kromě toho, že jsme ušetřili takty při počítání a kontrole iterátoru, zmizely nám také podmíněné skoky. U prvního a posledního rozvinutého průchodu totiž víme přesně, jaký podmíněný skok by se provedl a jaký ne. Můžeme tedy odstranit podmínky a skok provést rovnou. Časová úspora už je poměrně značná - takt z podmínky, která vyjde, a dva takty z podmínky, která nevyjde. Rozvíjení cyklu je speciální případ zabudování dat do programu - obecnějšího přístupu transformujícího proměnné na pozici programu.

4.2.2 Přeměna dat do pozice v programu

Procesor je stavový automat, který přechází mezi stavy na základě instrukcí. Počet stavů lze spočítat jako počet všech možností, které můžou paměťová místa procesoru (registry, paměti, výstupy) nabývat. Procesor s jedním 8-bitovým registrem a jedním 8-bitovým programovým ukazatelem (PC, program counter) má celkem 65536 stavů. Nahlížet na procesor jako na stavový automat může být výhodné - splývají zde totiž data, registry a pozice v programu do jediného pojmu - stavu procesoru. Představme

si, že potřebujeme uchovávat jednobitovou informaci, ale nemůžeme nebo nechceme ji držet v registru. Program, který hospodaří s touto informací nahrajeme do dolní poloviny paměti, jeho kopii do horní poloviny paměti. Prohlásíme, že pokud je programový ukazatel v dolní polovině, pak je naše proměnná nulová, pokud je programový ukazatel v horní polovině paměti, je proměnná rovna jedné.

Program Counter	Proměnná
dolní polovina paměti	0
horní polovina paměti	1

Místo operací, které by přiřazovaly naší proměnné hodnotu, budeme provádět skoky mezi oběma polovinami paměti. Protože jsou obě verze programu totožné, nebude mít skok žádný vliv na provádění programu, ale bude mít vliv na hodnotu proměnné. Naše proměnná nemá žádné konkrétní místo, kde by ležela. Je jistým způsobem zakódována pro Program Counteru. V tomto případě, kdy jsme programovou paměť rozdělili na poloviny, by se dalo říct, že naši proměnnou jsme přesunuli do nejvýznamnějšího bitu Program Counteru. Nemusíme ale lpět na takovémto rozdělení paměti. Můžeme si definovat různé oblasti programu, nespojité, prolnuté, ve kterých vždy bude jednoznačně definováno, jakou hodnotu má naše proměnná. Taková definice nemá žádné místo, kam by se zapisovala, je totiž zakódována v programu a jeho chování. Je například zbytečné (dokonce nemožné) provádět podmíněné skoky podle naší proměnné. Jednak neexistuje mechanismus, jak takovou podmínku zakódovat (naše proměnná neleží v registru ani vlajce), a hlavně taková podmínka je buď splněna vždy, nebo nikdy. Měl-li by se v horní části provést skok za podmínky, že naše proměnná je nulová, pak se může úplně vynechat, protože v horní polovině paměti je z definice naše proměnná rovna jedné vždy.

```
Blikač přes proměnnou a:

ldi i,8
ldi a,0
cyklus:
  neg a
  cpi a,0
  brne jedna
    cbi PINB,0      ; ukázková operace
    rjmp endif
  jedna:
    sbi PINB,0      ; ukázková operace
  endif:
  dec i
brne cyklus         ; přepneme 8x
```

```
Transformace:

ldi i,8
rjmp cyklus0        ; nahrazení ldi a,0

;zde platí a=0
cyklus0:
  rjmp cyklus1neg   ; náhrada za neg a
  cyklus0neg:
  cbi PINB,0        ; ukázková operace
  dec i
brne cyklus0        ; přepneme 8x

; zde platí a=1
cyklus1:
  rjmp cyklus0neg   ; náhrada za neg a
  cyklus1neg:
  sbi PINB,0        ; ukázková operace
  dec i
brne cyklus1        ; přepneme 8x
```

4.2.3 Změna směru čítání

Procesor disponuje vlajkami (flags) - jednobitovými hodnotami, ve kterých jsou uloženy informace o výsledku poslední operace. Jednou z vlajek je ZF (zero flag) která je nastavena v případě, že výsledekem poslední aritmetické operace byla nula. Jednoduché čítání od jedné do deseti se pak nechá úsporněji přepsat na čítání od deseti do jedné.

```
Od 1 do 10:

ldi i,1
cyklus:
  ; ... tělo cyklu
  inc i
  cpi i,11
brne cyklus
```

```
Od 10 do 1:

ldi i,10
cyklus:
  ; ... tělo cyklu
  dec i
brne cyklus
```

4.2.4 Čítání jako vedlejší produkt deserializace

Čítání nemusí být binární. Jako čítač může sloužit i posuvný registr. Naplníme-li posuvný registr hodnotou 0x80, pak je jisté, že právě po osmi pravých rotacích bude výstup z registru (vlajka carry) roven jedné. Přitom neklademe požadavky na hodnoty, které budou do posuvného registru zleva nasouvány. V našem případě můžeme využít tuto rotaci k deserializaci dat a přitom nepočítat klasickým způsobem počet bitů, které ještě musíme do registru vložit.

```
Binární čítání:

ldi bit,8
cyklus:
  in line,PINB    ; přijmeme data
  ror line        ; přesun LSb do CF
  ror data        ; rotace LSb first
  dec bit         ; už jich bylo 8?
brne cyklus       ; pokud ne, tak znovu
```

```
Posuvné čítání:

ldi data, 0x80
cyklus:
  in line,PINB    ; přijmeme data
  ror line        ; přesun LSb do CF
  ror data        ; CF do dat
brcc cyklus       ; už je CF jedna?
```

4.2.5 Přesuny větví programu

Skoky jsou časově náročné operace a proto je potřeba řadit za sebe jednotlivé části programu tak, aby co nejvíce navazovaly. Využívá se tím přirozeného postupného zpracování programu. Je například zbytečné umísťovat nepodmíněné skoky, pokud jde celá větev, kam se má skočit, přesunout pod skokovou instrukci. Za cenu mírného obsazení programové paměti získáme k dobru dva hodinové takty.

```
if-else přirozeně:

ldi bit,8
cyklus:
  ; ... práce
  cpi bit,4
  brne else
    ; ... zprac., bit=4
    rjmp ifend
  else:
    ; ... zprac., bit<>4
  ifend:
  dec bit
brne cyklus
```

```
Přesun větve:

ldi bit,8
cyklus:
  ; ... práce
  cpi bit,4
  breq ctyri
    ; ... zprac., bit<>4
    dec bit
    brne cyklus
    rjmp ... je hotovo
  ctyri:
    ; ... zprac., bit=4
    dec bit
brne cyklus
; není brne zbytečné?
; zde se totiž skočí vždy
```

```
Lepší přesun větve:

ldi bit,8
rjmp cyklus

ctyri:
  ; ... zprac., bit=4
  dec bit
cyklus:
  ; ... práce
  cpi bit,4
  breq ctyri
    ; ... zprac., bit<>4
  dec bit
brne cyklus
```

Vhodným přeskládáním programu se může docílit i toho, aby se sobě délky jednotlivých iterací rovnaly.

4.3 Hardware zařízení

4.3.1 Přehled funkcí

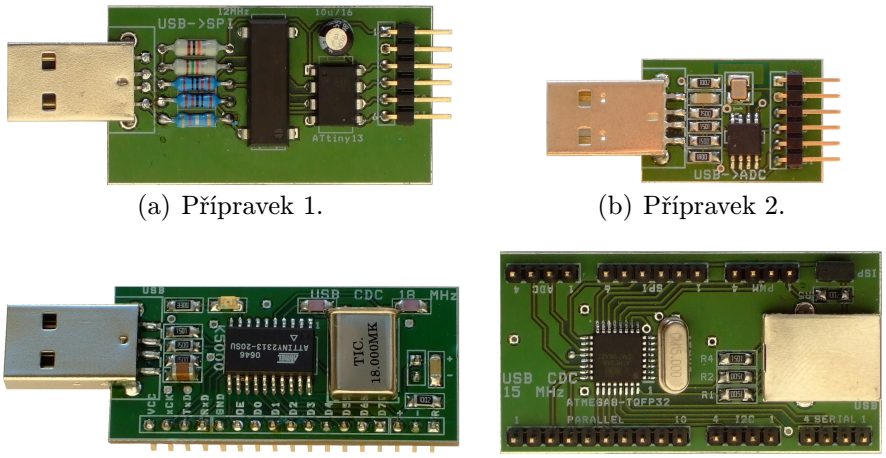
Čtyři přípravky, které jsou podstatou této práce, jsou výsledkem hledání nejlepšího řešení z hlediska prostoru, ceny a obtížnosti realizace. Byly vyzkoušeny ještě další možnosti, které nevedly k úspěchu.

Procesor	**Pouzdro**	**Frekv.**	**Hodiny**	**Kon.**	**USB**	**Funkce**
ATtiny13	DIL 8	12MHz	oscilátor	USB-A	1 endpoint	SPI rozhraní
ATtiny13	SOIC 8	19,5MHz	oscilátor	USB-A	2 endpointy	ADC+PWM
ATtiny2313	SOIC 20	18MHz	krystal	USB-A	4 endpointy	Serial+Parallel
ATmega8	TQFP 32	15MHz	krystal	USB-B	třída CDC	SPI, ADC, Serial, I2C, PWM, Parallel

Tabulka 18: Sourhn uskutečněných zařízení

Procesor	**Pouzdro**	**Frekvence**	**Typ hodin**	**Konektor**	**Důvod nefunkčnosti**
ATtiny13	SOIC 8	12 MHz	RC oscilátor	USB-A	nestabilní hodinový takt
ATtiny13	SOIC 8	10,5 MHz	RC oscilátor	USB-A	nestabilní hodinový takt
ATtiny13	SOIC 8	9 MHz	RC oscilátor	USB-A	nestabilní hodinový takt
ATmega128	TQFP64	15 MHz	krystal	USB-B	zdlouhavé programování

Tabulka 19: Sourhn neuskutečněných zařízení



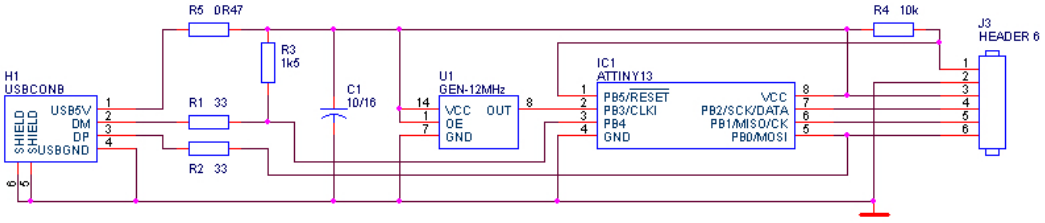
(a) Přípravek 1. (b) Přípravek 2.

(c) Přípravek 3. (d) Přípravek 4.

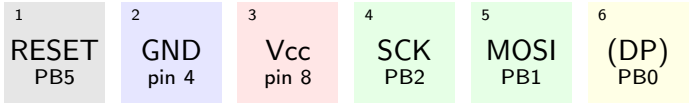
Obrázek 24: Uskutečněné přípravky

4.3.2 Přípravek 1. - výstupní SPI port

Deska plošných spojů je navržena na použití klasické montážní technologie (průchod pinů skrz desku) a je realizovatelná i v amatérských podmínkách. 6-pinový konektor je určen zároveň pro ISP programování obvodu i pro SPI výstup.



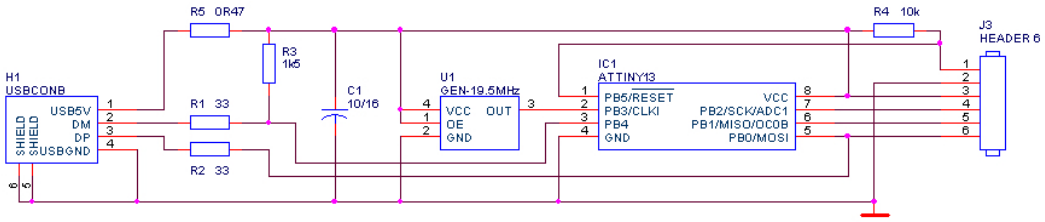
Obrázek 25: Schéma 1. přípravku - výstupní SPI port



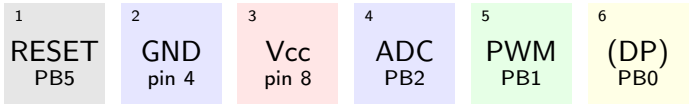
Obrázek 26: Pinout konektoru 1. přípravku

4.3.3 Přípravek 2. - regulátor s ADC a PWM

Pulzně-šířková modulace spolu s analogově-digitálním převodníkem umožňuje pomocí vhodné aplikace konstrukci regulátoru s rozsahem veličin 0-5V. Deska plošných spojů používá miniaturní pouzdra s technologií povrchové montáže. 6-pinový konektor je určen zároveň pro ISP programování obvodu i pro funkční vstup/výstup.



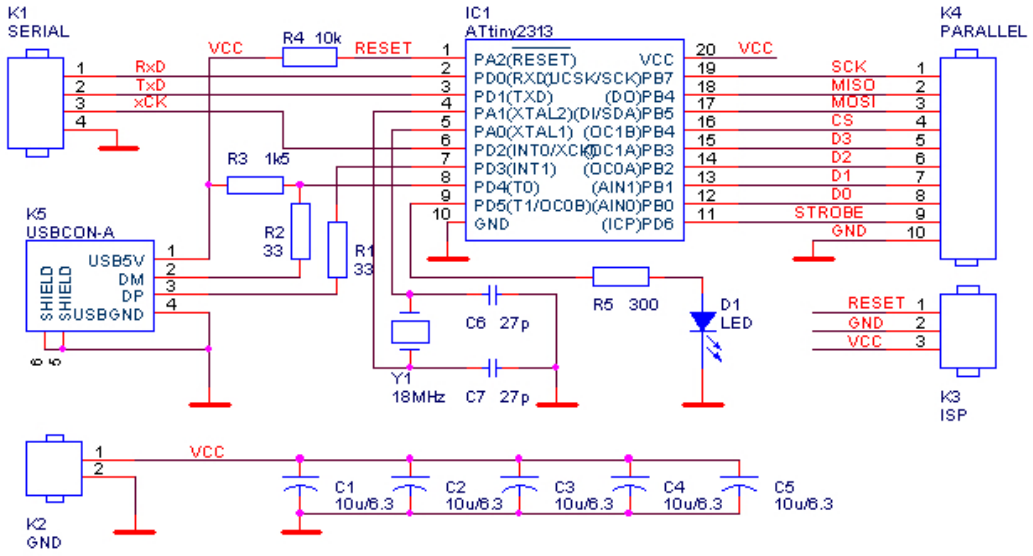
Obrázek 27: Schéma 2. přípravku - regulátor s ADC a PWM



Obrázek 28: Pinout konektoru 2. přípravku

4.3.4 Přípravek 3. - Sériové a paralelní vstupně-výstupní zařízení

Zařízení je na desku osazeno povrchovou montáží a konektory jsou uzpůsobeny k tomu, aby se deska mohla zasunout do nepájivého kontaktního pole pro experimentální účely. Schopnosti zařízení jsou přijímat a odesílat data po sériové lince a čtení a zápis dat do 8-bitového paralelního portu s nastavitelným směrem. Z uživatelské aplikace je tedy možné realizovat i obousměrnou komunikaci vhodným přepínáním mezi logickými stavy 0,1 a stavem vysoké impedance. Pomocný výstupní pin STROBE krátkým pulzem potvrzuje platnost dat a může být použit jako hodinový pulz pro paralelní posuvné registry.



Obrázek 29: Schéma 3. přípravku - sériové a paralelní I/O

Sériová část datového konektoru ATtiny2313:

1	2	3	4
RxD PD0	TxD PD1	xCK PD2	GND

Paralelní část datového konektoru ATtiny2313:

1	2	3	4	5	6	7	8	9	10
D7 PB7	D6 PB6	D5 PB5	D4 PB4	D3 PB3	D2 PB2	D1 PB1	D0 PB0	STROBE PD6	GND

LED (aktivita na USB lince):

LED
PD5

Obrázek 30: Pinout konektoru 3. přípravku

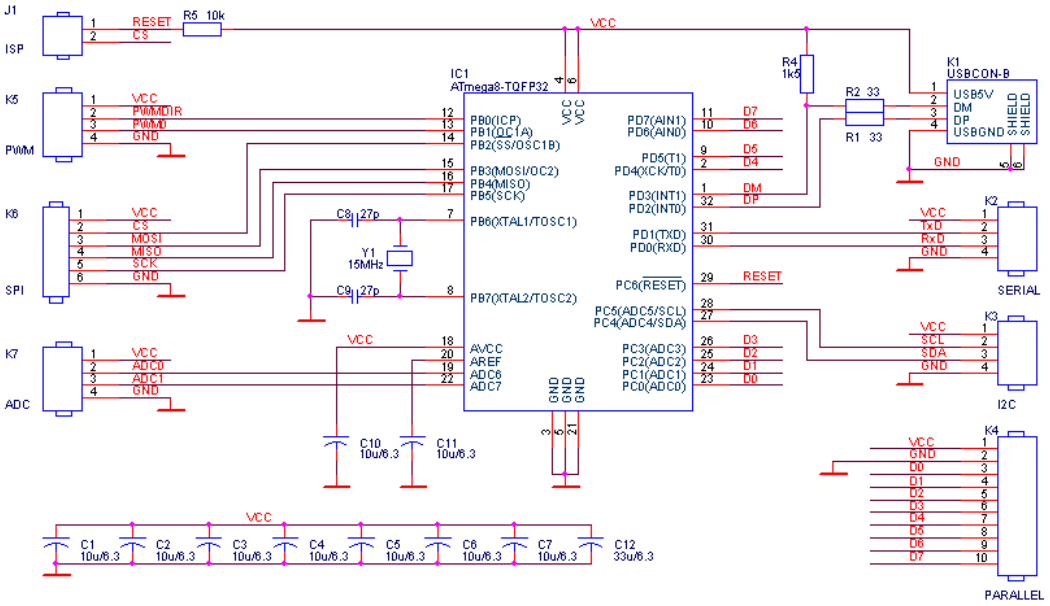
4.3.5 Přípravek 4. - Kompozitní USB zařízení

Zařízení poskytující více rozhraní je osazeno na desce s dvěma řadami headerů. Po připojení k hostiteli je rozpoznáno 5 dílčích USB zařízení, z nichž každé je členem třídy CDC. Pro každé dílčí zařízení (tabulka 20) je nainstalován generický ovladač a je mu rezervován port COM (např. COM10 až COM14). Rozhraní ADC je spojeno

s výstupem PWM, protože bylo nutné vytvořit maximálně 5 dílčích rozhraní (omezeno USB specifikací).

Rozhraní	Zkratka	Popis
MI00	SPI	Sériové Periferní Rozhraní
MI02	Serial	Sériová linka
MI04	Parallel	8-bitový paralelní výstup
MI06	I2C	Inter-Integrated Circuit bus
MI08	ADC+PWM	Analogově-digitální převodník na vstupu Pulzně-šířková modulace na výstupu

Tabulka 20: Části kompozitního zařízení



Obrázek 31: Schéma 4. přípravku - kompozitní USB zařízení

4.3.6 Plošné spoje

Pro návrh schémat obvodů byl použit program OrCAD 9.1 [17], ze kterého byl vyexportován netlist do programu Layout. Ke každé součástce náleží otisk (footprint), který definuje plochy potřebné v jednotlivých vrstvách desky. Knihovna Layoutu obsahuje množství běžných pouzder, ale USB konektory v knihovně zatím nejsou definovány. Proto jsem pomocí Library Manageru nakreslil vlastní footprinty USB konektorů A, A SMD a B (obrázek 33).

SPI port a programovací konektor USB zařízení s ATmega8:

1	2	3	4	5	6
Vcc	$\overline{\text{CS/RST}}$ PB2	MOSI PB3	MISO PB4	SCK PB5	GND

Sériová linka:

1	2	3	4
Vcc	TxD PD1	RxD PD0	GND

I2C:

1	2	3	4
Vcc	SCL PC5	SDA PC4	GND

PWM:

1	2	3	4
Vcc	PWMdir PB0	PWM0 PB1	GND

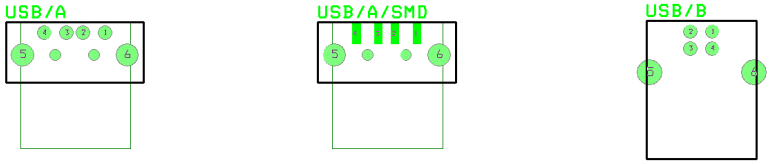
ADC:

1	2	3	4
Vcc	ADC0 ADC6	ADC1 ADC7	GND

Paralelní port:

1	2	3	4	5	6	7	8	9	10
Vcc	GND	D0 PC0	D1 PC1	D2 PC2	D3 PC3	D4 PD4	D5 PD5	D6 PD6	D7 PD7

Obrázek 32: Pinout konektorů 4. přípravku



(a) USB-A (b) USB-A SMD (c) USB-B

Obrázek 33: Motivy konektorů USB

Při kreslení čar jsem postupoval podle intuice s dodržováním základních pravidel, jako je dodržování přímosti a hladkosti spojů, nevytváření slepých ramen, souběh diferenciálních vodičů, vedení převážně na jedné straně (aby na druhé mohla být rozlita zem) či stromovitý rozvod napájení. Pro snížení vyzařování jsem desku opatřil oboustranně rozlitou zemí s prokovy. Při umísťování součástek jsem dbal na zachování rozteče 100 milů, především mezi pinovými hřebeny. Pomocí propojovacích kabelů s širšími konek-

tory (10 pinů) je možné z desky vyvést i více rozhraní najednou. U přípravku 3. jsou správné rozteče nutné kvůli připojení do nepájivého kontaktního pole. Tloušťky spojů odpovídají 4. konstrukční třídě, tj. tloušťka spojů, šířka izolačních mezer nebo rozdíl průměru plošky k vrtání je nejméně 0,3 mm (11,8 milů).

Po návrhu byl proveden export spodního a horního motivu, spodní a horní nepájivé masky a servisního potisku do formátu pro fotoplotter Extended-Gerber a data byla předána do výroby firmy Pragoboard s. r. o. Náklady na výrobu filmových podkladů činily 900,- a výroba 2 kusů desek (pro všechna zařízení) 400,- Kč. Desky jsou oboustranné s nepájivou maskou, servisním potiskem a HAL cínováním.

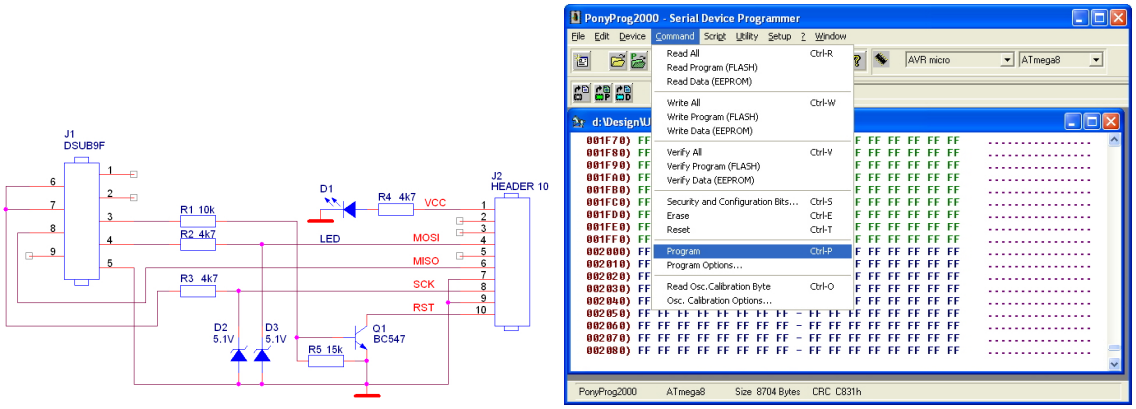
4.3.7 Programování paměti mikrokontroléru

Trvalé paměti mikrokontroléru jsou flash (programová) paměť, EEPROM paměť a sada přepalovacích propojek (Fuses). Nastavení těchto pamětí se děje pomocí speciálního hardware - programátoru. Existuje řada programátorů, které umějí zároveň komunikovat s debuggerem, avšak společným jmenovatelem těchto zařízení je vysoká cena (v řádu tisíců korun). Mikrokontrolér poskytuje několik způsobů, jak paměti naprogramovat. Jsou to tyto:

1. nízkonapěťové sériové programování - pomocí rozhraní SPI jsou během aktivního $\overline{\text{RESET}}$u nahrávány příkazy a data. Použity jsou vodiče $\overline{\text{RESET}}$, MOSI, MISO, SCK, VCC, GND. Resetovací funkci vstupu $\overline{\text{RESET}}$ lze změnit na obecný IO port, pak však nelze více použít nízkonapěťové programování.
2. vysokonapěťové sériové programování - sériovým synchronním rozhraním jsou do procesory nahrána data zároveň s příkazem. Použity jsou vodiče $\overline{\text{RESET}}$, SCI (hodiny), SDO (výstup dat), SDI (vstup dat), SII (příkaz), VCC, a GND. Připojením 12 V na pin $\overline{\text{RESET}}$ se aktivuje programovací režim. Všechny vodiče mohou být v normálním režimu použity pro běžnou funkci (i $\overline{\text{RESET}}$ může fungovat jako IO port). Tato programovací metoda je výhodou hlavně u 8-pinových provedení, protože oproti nízkonapěťovému programování nevyžaduje dedikovaný $\overline{\text{RESET}}$.
3. paralelní vysokonapěťové programování - u obvodů, které umožňují připojení 20 vodičů je toto programování použito jako náhrada za vysokonapěťové sériové programování. Programovací režim se aktivuje připojením 12 V na $\overline{\text{RESET}}$.
4. JTAG - programování sériovým synchronním rozhraním s vodiči TDI, TDO, TCK, TMS, VCC a GND. Je k dispozici například na ATmega128. Pomocí hodin TCK a dat TDI je nahrána do posuvného registru instrukce s daty a pomocí režimových hodin TMS a řídícího vstupu TDI je programování spouštěno. Pomocí rozhraní JTAG je možný i debugging obvodu.
5. samoprogramování - pomocí speciálního kódu umístěného na konci paměti (Bootloader) lze pomocí libovolného rozhraní, kterým mikrokontrolér disponuje, pře-

nést do SRAM data a ta použít k postupnému přeprogramovávání paměti. Ze všech metod je tato metoda nejrychlejší, ale vyžaduje prvotní nahrání bootloaderu jiným programátorem.

Nízkonapěťové sériové programování je možné zajistit pomocí levného přípravku SI Prog (schéma na obrázku 34(a)) a programu PonyProg (obrázek 34(b)). Pomocí sériové linky počítače jsou přes napěťové omezovače (Zenerovy diody) buzeny vstupy SCK a MOSI a bez změny logických úrovní (z TTL na RS-232) je zapojen vodič MISO. Díky specifickému chování budičů sériové linky jsou úrovně interpretovány správně, přestože jsou v rozporu se specifikacemi. Jedná se o přípravek, který pro svoje zprovoznění nepotřebuje jiný programátor (neobsahuje totiž žádný procesor). Je vhodný pro programování obvodů jen s malou programovou pamětí, protože pracuje pomalu. Naplnění větší paměti (např. 128 kB) trvá až minutu, ale mnohdy se nezdaří a je nutno proces opakovat. Program PonyProg dokáže otevřít 7 nejpoužívanějších formátů datových souborů, mimo jiné i Intel Hex nebo binární soubory. Paměti flash a EEPROM je možné oddělit nebo editovat, což je užitečné při plnění EEPROM ASCII řetězci.



(a) schéma SI Progu (b) software PonyProg

Obrázek 34: Levný programátor mikrokontrolérů AVR

4.4 Problémy hardware a jejich odstranění

4.4.1 Prodleva mezi pakety

Speficikací je stanoveno, že odesílání odpovědí na příchozí paket od hostitele musí být započato mezi dobou odpovídající 2 až 6,5 bitům. Pro low-speed je to 1,3 μs až 4,3 μs a na frekvenci mikrokontroléru 15 MHz to znamená 20 až 65 taktů volných pro přípravu odpovědi. Pakety odeslané po vypršení lhůty nemusí být hostitelem akceptovány.

Doba mezi pakety je určena výhradně pro rozpoznání obsahu paketu, změnu stavu

automatu a přípravu paketu (obvykle 12 bytů). Operace rozhodování se skládají ze série porovnávání a podmíněných skoků. V jazyce C je použita konstrukce `switch-case`. Časová náročnost takového porovnávání je lineární s počtem intervalů, do kterých chceme vstupní hodnotu přiřadit. Například u deskriptoru, který má 320 bytů a je odesílán ve 43 paketech, není možné uskutečnit 43 porovnání. Řešením je rozpad na strom a prohledávání metodou půlení intervalu, které nachází cíl v log(n) krocích, zde v přibližně 6 krocích.

Další zrychlení, kterým lze získat řádově 2-3 takty je `inline` direktiva překladače, která místo volání podprogramu zařídí jeho vložení. Pokud je podprogram volán jen jednou a v programu se vyskytuje pouze kvůli přehlednosti, nebude mít direktiva vliv na spotřebu programové paměti a ušetří se `rcall` a `ret` instrukce při volání a návratu.

Při návrhu je nutné neustále kontrolovat výsledek překladu a porovnávat programátorský záměr se strojovým kódem, který byl překladačem vytvořen. Balík WinAVR obsahuje i nástroje na zpětný rozbor .obj souboru (disassembly). Výstupem je kód v Assembleru, který je přehledně označen původními C příkazy a lze se v něm snadno orientovat. Často se stane, že interpretace některých složitých výrazů a datových typů je neúsporná a je lepší výraz rozdělit na více jednodušších a jednoznačnějších částí, popřípadě jej kódovat v Assembleru ručně.

4.4.2 Odraz na vedení

Diferenciální vedení mezi hostitelem a zařízením má impedanci 90 Ω, avšak impedanční přízpůsobení uzemněním vodiče přes 90 Ω odpor nelze použít. S odrazy na vedení se specifikace vypořádává přikázanou maximální délkou kabelů a omezovačem strmosti hran signálu. IO porty mikrokontroléru však nemají omezovač strmosti a proto nezbývá jiné řešení, než zvýšit vnitřní odpor zdroje zařazením sériového rezistoru 100 až 200 Ω na vodiče DP a DM. Odpor spolu s kapacitou vodiče formuje RC článek typu dolní propust a filtruje vysoké frekvence hranových přechodů. Odraz na vedení se výrazně zatlumí.

S odrazem na vedení souvisejí i logické úrovně signálů hostitele a zařízení. Specifikace povoluje napětí na signálových vodičích v rozsahu 0 - 3,3 V, avšak mikrokontrolér pracuje na napětí 5V. Konstrukce přijímače na straně hostitele obsahuje diferenciální zesilovač, který však nemusí správně reagovat na větší rozsah vstupních napětí. Pokud se vlivem nasčítání odrazů nebo použitím 5 V úrovní objeví na lince rozdílové napětí překračující povolenou mez, není zaručena správná reakce hostitele, natož bezchybné přijetí paketu. Pokud jsou již sériově s budiči zapojeny odpory v řádu stovek ohmů, je možné zapojením Zenerových diod (3,3 V) na oba vodiče DP i DM omezit napětí, které ze zařízení vystupuje. Zenerova dioda bude otevřená v závěrném směru a proud diodou poteče takový, aby úbytek na sériovém odporu činil rozdíl do 5 V, tj. 1,7 V. Tomu odpovídá proud přibližně 17 mA, což je již za místem ohybu voltampérové charakteristiky

Zenerovy diody.

4.4.3 Neurčitost poloviny bitu

Nedeterministickou veličinou je doba, která uplyne od příchodu náběžné hrany nového paketu do spuštění rutiny obsluhy externího přerušení INT0. Je to způsobeno různými reakčními dobami procesoru v závislosti na právě prováděných instrukcích. Aby bylo možné zjistit, kde přesně se nachází polovina bitu, je nutné nejdříve přesně určit okamžik hrany a pak počkat určitý počet taktů před odebráním vzorku z linky. K tomuto účelu slouží synchronizační sekvence 0x80, která je zasílána na začátku každého paketu, a v NRZI kódování vypadá jako posloupnost 10101011 (zleva doprava s rostoucím časem). Pro detekci hrany je výhodné použít funkci `sbic/sbis` (Skip if Bit in IO register is Cleared/Set), která při spotřebě dvou taktů zjistí hodnotu bitu a eventuelně přeskočí následující instrukci. Následující kód hledá s nepřesností 3 takty sestupnou hranu:

```
hledej:
  sbic PINB,DP
  rjmp hledej
```

Nepřesnost 3 takty je únosná i pro operační frekvenci 12 MHz. Bit trvá 8 taktů a vložením čekání lze zaručit, že vzorkování linky proběhne mezi 3-6 taktem, čili přibližně v polovině bitu. Je nutné též počítat se zpožděním latch obvodu na vstupu portu. Podle dokumentace mikrokontroléru je mezi skutečným okamžitem příchodu hrany a reakcí instrukce s portem pracující prodleva 0,5 až 1,5 taktu. K vysvětlení nepřesnosti 3 taktů slouží obrázek 35.

4	5	6	7	8	1	2	3	4	5	6
sbic	rjmp	rjmp	sbic	rjmp	rjmp	sbic	skip			
	sbic	rjmp	rjmp	sbic	rjmp	rjmp	sbic	skip		
		sbic	rjmp	rjmp	sbic	skip				
			sbic	rjmp	rjmp	sbic	skip			
				sbic	rjmp	rjmp	sbic	skip		
					sbic	skip				
						sbic	skip			
							sbic	skip		

Obrázek 35: Nepřesnost hledání hrany

4.4.4 Nedostatek vývodů

U obvodů ATtiny13 v pouzdře DIL8 je mírnou obtíží počet pinů volně použitelný pro funkčnost obvodu. Nutně jsou obsazeny vývody Vcc, GND, $\overline{\text{RESET}}$, CLKI, PB4 (DM) a DP0 (DP). Zbývají jen PB1 a PB2, což nestačí ani pro obousměrné rozhraní SPI. Při bližším pohledu je zřejmé, že vývod $\overline{\text{RESET}}$ neplní za běhu žádnou funkci, ale je nutný pro nízkonapěťové sériové programování. Pin je potřebný při programování PonyProgem, protože přípravek SI Prog je jednoduchý a neumí programovat vysokonapěťově.

Řešením je změna programátoru. Buď bude použit hotový výrobek profesionální firmy nebo může být programátor snadno sestrojen pomocí zde navrhovaného USB řadiče. ObvodATmega8 naprogramovaný jako CDC zařízení emulující sériový port může pomocí jednoduchého příkazu shellu `type program.hex >> COM9` přijímat po USB sběrnici .hex soubror s daty a na svém výstupu pomocí vodičů vysokonapěťového programování zapisovat paměť cílového mikrokontroléru. Napětí 12V lze získat nevýkonným miniaturním zvyšujícím zdrojem (například s MC34063).

Mezi řešení malého počtu vývodů bohužel nepatří změna zdroje hodin. V případě, že by se podařilo nastavit vnitřní RC oscilátor (popřípadě jej za běhu kalibrovat) tak, aby frekvence jím vytvořená byla stabilní (±1,5%), uvolnil by se vstup CLKI. RC oscilátor je ale velmi nestabilní a jeho použití není možné.

4.4.5 Rychlé přesuny bitů mezi registry

Při hledání řešení přijímacích rutin je jednou z klíčových úloh přesun bitů mezi registry. Konkrétně se jedná o přesun hodnoty bitu z registru do vlajky nebo z registru do registru. Nabízí se použití speciální vlaky T (Bit Copy Storage) a instrukcí `bld` a bst (Bit LoaD/STore). Pokud se jedná o první nebo poslední pozici v bytu, lze použít rotaci přes CF `ror` a `rol` (Rotate Through Carry Left/Right). Jednou z nejlepších metod je ale použití porovnání `cp` (Compare). Typicky se totiž jedná o rozhodování mezi stavy portu 01 nebo 10 (hodnoty diferenciálních vodičů). Pokud před porováváním vymaskujeme ostatní bity (logickým součinem), můžeme porovnáním naplnit vlajku CF. Záměnou pořadí operandů lze naplnit vlajku CF znegovanou hodnotou. Negování bitů u bitových přesunů jinak není možné bez skoků. Následující kód uskutečňuje NRZI dekódování, deserializaci a kontrolu koncové podmínky:

```
in line,PORTB                   ;line je r16
andi line,(1<<DP)|(1<<DM)       ;vymaskování neplatných bitů
breq koncova_podminka           ;v pripade nul na DP i DM konci prenos
cp line,hodnota                 ;hodnota je r17 a má hodnotu (1<<DP)
                                ;záměnou operandů lze obdržet negovanou hodnotu CF
brcs prijata_jednicka           ;rozskok mezi různými větvemi programu
rol data                        ;v CF je dekódovaný bit, stačí rotovat
```

Pokud je program rozdělen na několik větví, ve kterých platí různé předpoklady o minulých hodnotách linky, lze pomocí skoků úplně vyloučit operace, které by se daly

přirozeně očekávat (například XOR pro NRZI dekódování). Obecná pravidla lze však těžko formulovat, protože záleží vždy na konkrétním použití. Při prozkoumávání instrukční sady jsem opakovaně došel k potřebě instrukce NXOR (Negovaný Exkluzivní logický součet) a CPSNE (Compare, Skip if Not Equal). Bohužel v sadě jsou pouze instrukce opačného významu XOR (Exkluzívní logický součet) a CPSE (Compare, Skip if Equal).

4.4.6 Uchovávání deskriptorů a tabulek

Při hledání vhodného úložiště pro deskriptory a předpočítané hodnoty CRC16 je možné využít třech pamětí - flash, SRAM a EEPROM. Ukládání do programové paměti má tři varianty. První je přímé zabudování do kódu (`ldi r16,konstanta` a `st X+,r16`), čímž se rovnou plní buffer ve SRAM. Možnost je i založit tabulku v programové paměti a pak k ní přistupovat instrukcí `lpm r16,Z+` a `st X+,r16`). Extrémní je možnost definovat buffer pro odesílaný paket na adrese 0-11, což je v překryvu s obecnými registry r0-r11! Doba potřebná k uložení bytu pak trvá jen 2 takty `ldi r16,konstanta` a `mov r0,r16`). Pokud nastane případ, kdy i dva takty jsou hodně, může být buffer definován na rozsahu r17-r27 a byty zapisovány přes `ldi r17,konstanta`. Z horních registrů se tím vyloučí možnost používat registr X, zbydou jen Y, Z a r16.

Uchovávání dat v EEPROM je použito pouze u přípravku 2. s ATtiny13 na 19,5 MHz. Hodinový takt je dostatečně rychlý, aby časově náročné čtení z EEPROM bylo zvládnuto do limitu pro odeslání odpovědního paketu. Přeplněnost flash neumožňovala zabudování deskriptorů do kódu.

Poslední možnost je uchovávat připravené pakety ve SRAM. Je zřejmé, že SRAM není perzistentní pamětí a že je nutné ji po restartu inicializovat (většinou použitím části flash). Odesílací rutina pak musí pracovat s plovoucím bufferem, který je před odesláním předpřipraveného paketu nastaven na přesnou pozici ve SRAM. Například u obvodu ATmega8 je takové uchovávání paketů díky veliké SRAM výhodné, protože příprava paketu pak trvá konstantní čas.

4.5 Ovladače zařízení

4.5.1 Vlastní ovladače zařízení WDM

Pro 1. a 2. přípravek bylo nutné naprogramovat vlastní ovladače zařízení pro Windows XP technologií WDM (Windows Driver Model). K tomuto účelu je zdarma k dispozici vývojové prostředí DDK (Driver Development Kit) [14]. Programovacím jazykem je ANSI C. Ovladače programované technologií WDM musí transformovat standardní vstupně/výstupní požadavky (čtení, zápis, otevření, rušení přenosů, PnP a požadavky

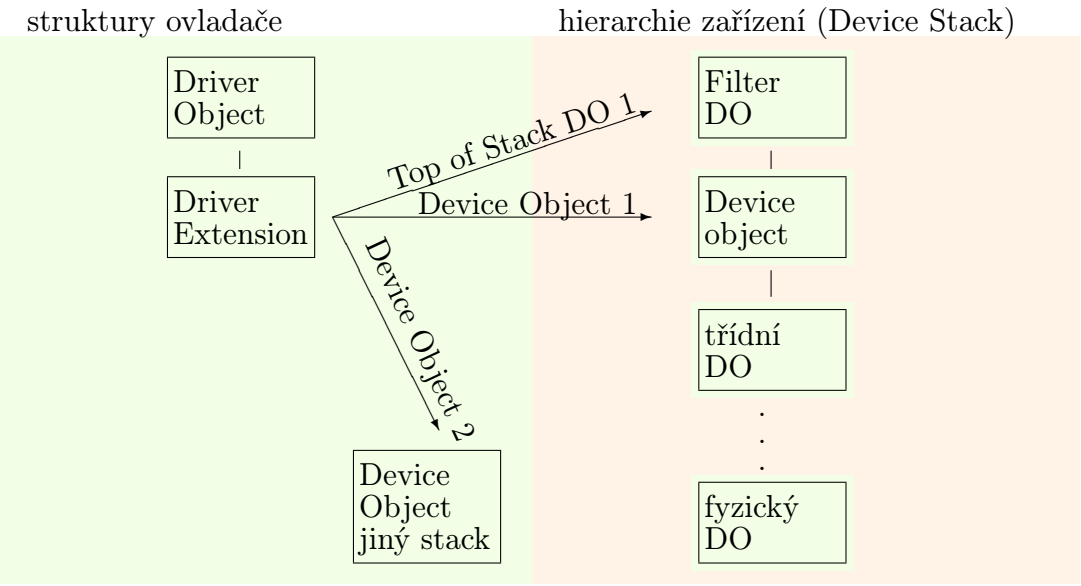
řízení spotřeby) na příkazy pro USB zařízení. Programování ovladačů je stížené skutečností, že kód běží v privilegovaném režimu a jakákoliv chyba vede na zhroucení systému. Během vývoje lze používat chybové výpisy a zobrazovat je programem DbgView [18].

Zcela zásadní pro vývoj ovladačů je pochopení principu, na kterém je Windows Driver Model založen, jaké datové struktury používá a jaké jsou zodpovědnosti jednotlivých ovladačů. Častou otázkou je alokace paměti, její uvolňování a plnění. Zdaleka ne vždy totiž platí, že všechny buffery jsou alokovány a inicializovány volajícím. Mnohdy je nezbytné vžít se do role systému a uvědomovat si, jakými prostředky disponuje. Windows Driver Model je velmi složitý a od Windows Vista byl nahrazen odlišným modelem WDF (Windows Driver Foundation). Zvláštní pozor je třeba dát na přístup do paměti, která nebyla alokována driverem. Je to případ bufferů, které byly vytvořeny v neprivilegovaném uživatelském režimu a mohou se tak stát předmětem swapování. Velkou hrozbou pro stabilitu je také režim spánku a úsporné režimy, jejichž implementace je v ovladačích mnohdy špatně zvládnuta a způsobuje havárie systému. Popis všech principů, datových struktur a systémových funkcí je možné získat v dokumentaci DDK [8] nebo na stránkách Microsoft Development Network [9].

Mezi tři skutečně základní datové struktury patří zásobník zařízení (Device Stack), tabulka požadavků IRP a spojový seznam zařízení obsluhovaných týmž driverem. Na obrázku 36 je zobrazen prostor ovladače, který udržuje ve svém spojovém seznamu ukazatele na všechny zařízení, pro něž vyřizuje požadavky (IRP - I/O Request Packet). Takový případ nastává například při zapojení dvou totožných zařízení do systému - jediný driver obdrží ve formě volání rutiny `AddDevice` příkaz zařadit nový Device Object (DO) do svého seznamu a hospodařit s ním. Každé ze zařízení, o které se ovladač stará, je ale i členem tzv. Device Stacku (zásobník zařízení). Device Stack vyjadřuje závislost podřízenosti a nadřízenosti jednotlivých Device Objectů. V Device Stacku je tedy na spodní úrovni objekt patřící ovladači sběrnice, zatímco na vrchu zásobníku jsou Device Objecty USB zařízení, popřípadě filtrovací či třídní objekty, které zpracovávají požadavky na větší úrovni abstrakce. Jako příklad může sloužit myš HID, která má nad svým Device Objectem zařazen objekt třídy HID a nad ním objekt polohovacího zařízení. Device Stack tedy vytváří závislosti mezi zařízení ve svislém směru, zatímco spojový seznam zařízení jednoho ovladače svazuje jednotlivé Device Objecty ve směru vodorovném.

Ovladač je reprezentován Driver Objectem - datovou strukturou se systémem definovanými položkami. Aby mohl uchovávat i informace, které jsou pro něj specifické, může alokovat Driver Extension dle vlastní definice, a do něj pak ukládat data, která se týkají obecných informací o ovladači (nikolik o zařízení). Teprve když systém zavolá `AddDevice`, může začít ovladač pracovat s konkrétním zařízením reprezentovaným Device Objectem (který je alokován systémem). Pro uložení specifických dat o jednotlivých zařízeních se používá Device Extension, datová struktura definovaná i alokovaná ovladačem. Je nutné, aby veškeré události související se zařízením byly zaznamenávány pouze do Device Extension a ne do Driver Extension. Například PnP stav zařízení, stav

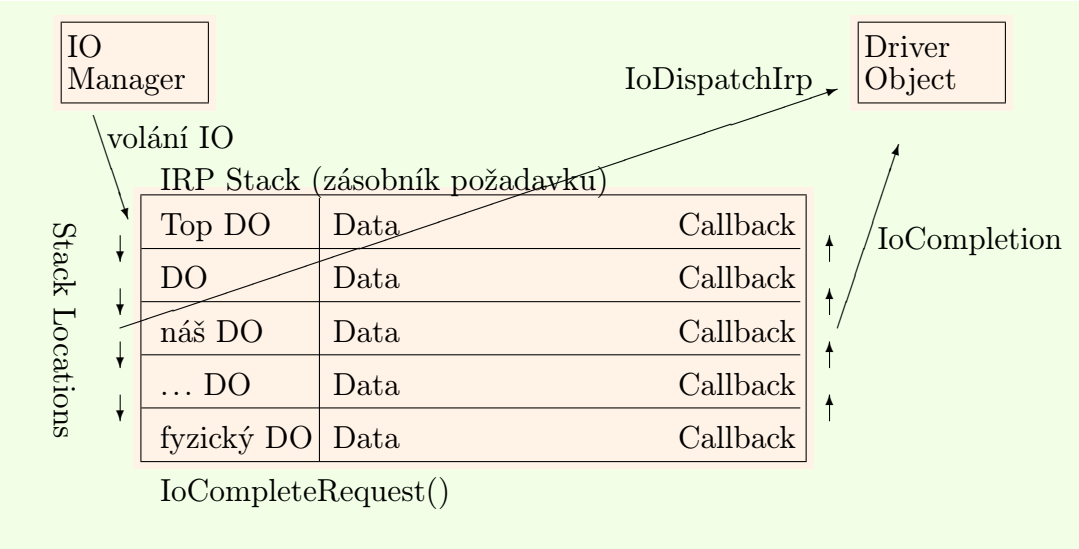
transakcí, cesty do registru apod. jsou vždy různé pro každé zařízení a není možné, aby například žádost o uspání jednoho zařízení ovlivnila transakce zařízení druhého.



Obrázek 36: Vztahy ovladače a zařízení

Všechny operace, které má ovladač nad zařízeními vykonávat, mají jednotnou formu - IRP. IRP je struktura s pevnými i variabilními částmi a ukazatel na tuto strukturu je ovladači předáván při volání `IoControl`. Jak je patrno z obrázku 37, požadavek, který byl předán ovladači vrchního Device Objectu, může být zkopírován a předán ovladači nižšího Device Objectu. Původní IRP je vlastněn stále stejným ovladačem a vzniká tak spojový seznam požadavků (IRP Stack). Přestože se z pohledu IO Manageru jedná o jediný požadavek (například o zápis), nižší ovladače mohou požadavek dekomponovat na podúlohy, které už vůbec nemusejí mít povahu zápisu. Každý nižší ovladač tedy vytvoří svou modifikaci IRP, kterou posílá ovladači svého podřízeného objektu. Zároveň ale chce mít možnost dozvědět se, zda požadavek poslaný dolů proběhl úspěšně. K tomu slouží návratové volání Callback, které je zavoláno systémem v okamžiku dokončení IRP nižšími ovladači. Ovladač se na základě výsledku z nižších IRP rozhodne, jaký chybový stav ohlásí svému nadřízenému objektu, jak naloží s daty, které byly výsledkem nižším IRP a podobně. Je pochopitelně možné místo jediné kopie IRP vytvořit kopií více, je též možné netvořit kopii žádnou a požadavek předat nižšímu ovladači nebo je možné požadavek vyřešit a žádné nižší volání neprovádět. Komplikovanost IRP je jedním z faktorů celkové neprůhlednosti WDM.

Programování ovladačů je ztíženo i o fakt, že příklady, které jsou součástí DDK, obsahují chyby. Špatná implementace požadavků Plug and Play u vzorových ovladačů



Obrázek 37: Průchod požadavku IRP stackem

USB zařízení způsobuje chybné chování při přechodech do režimu spánku. Není radno brát příklady z DDK jako normu, ale pouze jako doplněk k dokumentaci WDM. Při vývoji je téměř nezbytné postupovat podezřívavě a kontrolovat každý kus kódu, byť by se objevil na stránkách MSDN.

Pro vytvoření představy o rutinách ovladače uvádím deklarační část ovladače zařízení 1. přípravku:

```
#include <ntddk.h>
#include <initguid.h>
#include <usbdi.h>
#include <usbdlib.h>

// GUID tridy zarizeni
DEFINE_GUID(GUID_CLASS_UNIVERSAL_USBSPI, 0x873fdf, 0x61a8, 0x11d1, 0xaa, 0x5e, 0x00,
            0xc0, 0x4f, 0xb1, 0x72, 0x8b);

typedef struct _DEVICE_EXTENSION {
    // Functional Device Object
    PDEVICE_OBJECT pFDO;
    // Device object we call when submitting Urbs
    PDEVICE_OBJECT pTSDO;
    // The bus driver object
    PDEVICE_OBJECT pPDO;

    // Name buffer for our named Functional device object link
    // The name is generated based on the driver's class GUID
    UNICODE_STRING uInterfaceName;

    // Configuration Descriptor
```

```
    PUSB_CONFIGURATION_DESCRIPTOR UsbConfigurationDescriptor;
    // Interface Information structure
    PUSBD_INTERFACE_INFORMATION UsbInterface;
    int DosNameRegistered;
    int DeviceConfigured;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;

NTSTATUS JohnDriver_Create(PDEVICE_OBJECT DeviceObject, PIRP Irp);
NTSTATUS JohnDriver_Close(PDEVICE_OBJECT DeviceObject, PIRP Irp);
NTSTATUS JohnDriver_IoControl(PDEVICE_OBJECT DeviceObject, PIRP Irp);
NTSTATUS JohnDriver_WriteBufferedIO(PDEVICE_OBJECT DeviceObject, PIRP Irp);
NTSTATUS JohnDriver_ReadBufferedIO(PDEVICE_OBJECT DeviceObject, PIRP Irp);
NTSTATUS JohnDriver_UnSupportedFunction(PDEVICE_OBJECT DeviceObject, PIRP Irp);
NTSTATUS JohnDriver_DispatchPnP(PDEVICE_OBJECT DeviceObject, PIRP Irp);
NTSTATUS JohnDriver_AddDevice(IN PDRIVER_OBJECT DriverObject,
                              IN PDEVICE_OBJECT pPhysicalDeviceObject);
NTSTATUS ReadandSelectDescriptors(IN PDEVICE_OBJECT DeviceObject);
NTSTATUS CallUSBD(IN PDEVICE_OBJECT DeviceObject,IN PURB Urb);
NTSTATUS ConfigureDevice(IN PDEVICE_OBJECT DeviceObject);
NTSTATUS SelectInterfaces(IN PDEVICE_OBJECT DeviceObject,
                          IN PUSB_CONFIGURATION_DESCRIPTOR ConfigurationDescriptor);
NTSTATUS ReadWriteCompletion(IN PDEVICE_OBJECT DeviceObject,IN PIRP Irp,
                             IN PVOID Context);

#define DOS_DEVICE_NAME L"\\DosDevices\\USBSPI"
#define DEVICE_NAME L"\\Device\\USBSPI"
```

Pro samotný překlad jsem používal příkaz `build` z příkazové řádky, který přeloží zdrojové kódy z adresáře a vytvoří spustitelný systémový soubor (.sys). Zavedení ovladače do systému provede Správce zařízení při zapojení USB zařízení. K vyjádření vazby mezi ovladačem a zařízením se používá soubor s informacemi o zařízení (.inf). Oba tyto soubory jsou potřebné ke správnému chodu zařízení v systému. K vygenerování .inf souboru lze použít nástroj InfGen, který je součástí DDK.

4.5.2 Generické ovladače třídy komunikačních zařízení

Vzhledem k tomu, že programování ovladačů pod Windows XP je mnohem obtížnější než programování firmware mikrokontroléru, je mnohdy snazší upravit zařízení tak, aby vyhovovalo předdefinované třídě a poté používat generické ovladače zařízení, které jsou standardní součástí operačního systému. Nutné je pouze dodání souboru s informacemi o zařízení (.inf).

V praxi se nejčastěji používají tři druhy generických ovladačů. Třídy HID (Human Interface Device) a CDC (Communication Device Class) [10] potřebují .inf soubory a ovladače hid.sys a usbser.sys jsou instalovány za asistence uživatele. Třída MSD (Mass Storage Device) nepotřebuje ani .inf soubor a celá instalace proběhne bez zásahu uživatele.

4.5.3 Složená USB zařízení

V případě, že se USB zařízení skládá z více funkčních celků, z nichž každý by mohl být uvažován jako samostatné USB zařízení, je možné jej pomocí zvláštních deskriptorů IAD (Interface Association Descriptor) přihlásit k třídě složených USB zařízení. Příkladem složeného USB zařízení je klávesnice s myší, která je opatřena pouze jedním konektorem, ale funkčně ji lze rozdělit na dvě HID zařízení, každé s vlastním generickým ovladačem. V této práci je složené zařízení použito u přípravku 4. Každé z pěti rozhraní lze chápat jako samostatný komunikační port produkující a konzumující tok dat a vyhovující specifikaci komunikačních zařízení CDC (abstrakce sériové linky). Bohužel Windows XP nemají podporu pro složená zařízení třídy CDC a je nutné použít ovladače třetích stran.

4.5.4 Ovladače USB IO firmy Thesycon

V průběhu experimentování se zařízením je potřeba vyvolávat požadavky a posílat testovací data. Programování funkčně specifického ovladače je v takové fázi vývoje zařízení předčasné. S výhodami lze použít nespecifický ovladač USBIO firmy Thesycon. Tento ovladač se spojení s aplikačním software dokáže realizovat všechny druhy USB přenosů a požadavků. Neboť se každé zařízení prezentuje jako roura, je i ovladač konstruován jako obecné čtení/zápis do roury s podporou některých USB specifických operací (načtení deskriptoru, volba konfigurace a jiné).

Thesycon nabízí také ovladače třídy CDC, které jsou schopny pracovat s kompozitními USB zařízeními. Jejich neplacená verze neumožňuje ale plný rozsah funkcionality. Řešením v našem případě jsou obecné USB ovladače USBIO, které nemají funkční omezení, ale časové. Ovladač smí být v provozu nepřetržitě maximálně 4 hodiny, poté je požadován restart počítače.

4.6 Aplikační software

V programovacím jazyce Delphi 7 firmy Borland jsem naprogramoval pro přípraveky aplikační software, který ve stručnosti demonstruje funkci každého přípravku. Pomocí ovládacích prvků, jako jsou tlačítka nebo zaškrtávací boxy, lze rozsvěcet LED diody na přípravcích nebo zobrazovat průběh analogové veličiny na vstupu A/D převodníku. U 2. přípravku jsem vytvořil S-bloky v MEX C pro Simulink Matlabu a na jednoduchém schématu demostruji správnou funkci A/D převodníku a PWM budiče. Pro 1. přípravek jsou k dispozici i nástroje pro příkazovou řádku.

4.6.1 Standardní rozhraní datové roury

Každý přípravek je nutné nějakým způsobem adresovat. Operační systém poskytuje API funkce, které vypíší seznam zařízení přítomných v systému. V seznamu jsou uvedeny informace o zařízení, jednoznačná cesta k zařízení a cesta k uživatelským datům v systémovém registru. Komunikační zařízení třídy CDC jsou automaticky pojmenovány jako COMx, kde za x se postupně přidělují volná přirozená čísla. K takovému zařízení se přistupuje API funkcí CreateFile.

Ovladače, které byly programovány samostatně, exportují při svém zavedení do paměti veřejné rozhraní (interface). Pojmenování takového interface je lidsky srozumitelné a není proto nutné jeho systémovou adresu zjišťovat voláními API funkcí. Ovladače jsou pojmenovány USBSPI (pro přípravek 1.) a USBADC (pro přípravek 2.). Další komunikace se zařízením se děje pomocí standardních vstupně/výstupních volání (ReadFile, WriteFile). Následují ukázky přístupu k zařízení pomocí příkazové řádky, jazyka C a jazyka Pascal:

Zápis ze shellu:

```
echo B >> \\.\USBSPI
```

Zápis z C:

```
void main(){
  FILE * f;
  f = fopen ("\\\\.\\USBPIPE","wb");
  fputc('B',f);
  fclose(f);
}
```

Zápis z Pascalu:

```
var f:file of char;
begin
  assign(f,'\\.\USBPIPE');
  rewrite(f);
  write(f,'B');
  close(f);
end;
```

Čtení ze shellu:

```
piperead < \\.\USBADC
```

Čtení z C:

```
void main(){
  FILE * f;
  char a;
  f = fopen ("\\\\.\\USBADC","rb");
  a=fgetc(f);
  fclose(f);
  putchar(a);
}
```

Čtení z Pascalu:

```
var f:file of char;
    a:char;
begin
  assign(f,'\\.\USBADC');
  reset(f);
  read(f,a);
  close(f);
  writeln(a);
end;
```

4.6.2 Specifické rozhraní USB IO

Ovladače firmy Thesycon neexportují rozhraní srozumitelného názvu. Ke zjištění přesné cesty k zařízení je nutné použít sadu systémových funkcí (SetupDiGetClassDevs a obdobné), které vrátí informace o zařízeních instalovaných v systému. Všechna zařízení USBIO mají jednotný identifikátor (GUID), podle kterého lze zařízení poznat. Při zapojení více totožných zařízení do systému nastává problém s nejednoznačností, který není v systému Windows XP nikterak řešen, a většinou je na uživateli, aby rozhodl, která adresa patří kterému zařízení. K zařízením USBIO se přistupuje voláním knihovních funkcí (nikoliv standardním IO), které jsou zdarma k dispozici s ovladači.

5 Hodnocení navržených přípravků

Všechny 4 přípravky jsou funkční a k jejich provozu jsou nutné pouze pokročilé uživatelské dovednosti.

5.1 Fyzické rozměry

	šířka [mm]	**výška [mm]**	**určení**
1. přípravek	49	27	budič k displeji
2. přípravek	29	20	zabudování do konektoru
3. přípravek	50	24	zasunutí do kontaktního pole
4. přípravek	53	28	stolní přípravek

Tabulka 21: Rozměry přípravků

5.2 Datový tok

U všech přípravků lze dosáhnout přenosové rychlosti maximálně 8 kB/s. Přestože modulační rychlost USB low-speed je 1,5MBaud, po započítání režie na 1 byte dat lze teoreticky dojít k hodnotě propustnosti:

$$\text{propustnost} = \frac{\text{modulační rychlost}}{8 \cdot (\text{režijních bytů} + 1)} = \frac{1,5 \cdot 10^6}{8 \cdot (5 + 5 + 1)} = \frac{1,5 \cdot 10^6}{8 \cdot 11} = 17kB/s$$

Zařízení dosahuje přibližně 47% teoretické propustnosti při velikosti paketu 1 byte. Pokud by rychlost procesoru dovolila přenášet pakety s větším datovým obsahem (např. 16 bytů), pak by mohla propustnost dosáhnout až 100 kB/s.

5.3 Vytížení procesoru

Způsob návrhu softwarového řadiče, kdy přijímací a odesílací rutiny pracují se zakázaným přerušením, neposkytuje během komunikace volný strojový čas jiné úloze. Pouze v intervalech mezi přenosy jednotlivých paketů je možné procesorový čas věnovat ostatním úlohám. Garance volného času je ovšem nulová, protože počet příchozích paketů nelze ovlivnit. Z hlediska obsazení programové paměti jsou první 3 přípravky na hranicích možností. Přípravek s ATmega8 čerpá procentuelně nejméně - přibližně polovinu kapacity.

	Flash	**RAM**	**EEPROM**	**vývody**
1. přípravek	1024 B (100%)	19 B (30%)	0 (0%)	2 z 8
2. přípravek	1024 B (100%)	19 B (30%)	64 (100%)	2 z 8
3. přípravek	2048 B (100%)	25 B (20%)	0 (0%)	2 z 20
4. přípravek	4200 B (51%)	70 B (7%)	0 (0%)	2 z 32

Tabulka 22: Využití prostředků mikrokontroléru

U přípravku 4. přibližně platí, že pro přenosové rychlosti 1 kB/s je procesor vytížen z 10%. 90% strojového času je k dispozici pro funkcionalitu obvodu. Programová paměť je zaplněna z 51%, zbytek je volně k dispozici návrháři pro implementaci dalších funkcí.

5.4 Modularita kódu, upravitelnost

Přípravek číslo 4 s ATmega8 jako jediný používá při návrhu jazyk C a proto jako jediný splňuje požadavky na modularitu a další upravitelnost kódu. Přípravky s procesory ATtiny mají veškerý firmware psán v Assembleru a kód je z toho titulu nevhodný pro další úpravy.

Přípravek s mikrokontrolérem ATmega8 je ze všech mikrokontrolérů ATmega nejlevnější, přesto disponuje dostatečnými prostředky, aby zvládl běh programů kompilovaných z jazyka C. Ostatní přípravky pouze potvrzují, že přílišná snaha o miniaturizaci vede k vyšší ceně a menší upravitelnosti. Pouze v případech, kdy návrhář implementuje jednoduché a nenáročné USB zařízení, je možné používat přípravky bez rezerv (ATtiny), avšak stále to není důvod, proč rovnou nepoužít řešení s ATmega8. I přesto, že kód v C není drasticky optimalizován, stále jsou v procesoru ATmega8 k dispozici 4 kB programové paměti a návrhář nemusí mít obavy, že by se jím navržené dodatečné funkcionality do procesoru nevešly.

Pro návrháře, který bude stavět své USB zařízení na základě knihoven usb.c a usb.h, zde uvádím stručný seznam kroků, jak bez námahy modifikovat stávající kód a tím přizpůsobit zařízení k odlišné funkci. Návrhář nemusí porozumět USB specifikaci, ale měl by být seznámen se základy, které jsou didaktickým způsobem uvedeny na stránkách Beyond Logic Craiga Peacocka [5]. Návrhář musí vypracovat schéma toku dat od aplikace do funkcionality, na základě kterého určí počet a vlastnosti endpointů, konstrukci bufferů a blokové schéma obslužných rutin. Rozličnost zařízení je taková, že nelze obecně naprogramovat knihovnu, ze které by se jednoduchým způsobem odvozovalo libovolné zařízení. Obecnými prvky, které jsou součástí knihovny, jsou mechanismy potvrzování paketů a transakcí, přijímání pověření, vyřizování enumerace na endpointu 0 a volání dílčích obsluh pro každý endpoint zvlášť. Samotný kód obsluhy endpointů již pracuje nad daty, jejichž povahu nelze dopředu znát. Návrhář tedy musí naprogramovat rutiny,

které zpracovávají uživatelská data. Při enumeraci lze odpovídat i na požadavky, které nejsou ve specifikaci označeny jako povinné. V takovém případě je návrháři poskytnut rámec, do kterého přidá číslo specifického požadavku a data, která chce jako odpověď odeslat. Snadným způsobem lze tak implementovat požadavky na String deskriptory (znakové popisovače), HID deskriptory a jiné třídní struktury.

Doporučené kroky k správné modifikaci knihovny USB řadiče:

1. Připravit si Device, Configuration, Interface a Endpoint deskriptory a jiné specifické struktury, data rozdělit do paketů po 8 bytech a předpočítat 16-bitovou cyklickou redundantní kontrolu.
2. Vytvořit seznam požadavků, na které chce zařízení odpovídat, jejich čísel a formy dat.
3. V rutině `UsbAnalyzeStandardRequest` přidat do stromových podmínek čísla požadavků, které budou implementovány.
4. V rutině `UsbPrepareData` upravit nebo vytvořit podle okolního vzoru příkazy na odeslání deskriptorů.
5. V `UsbEp0ProcessIncomingData` reagovat na data přicházející jako parametr požadavků.
6. V rutinách `UsbServeEPn` naprogramovat využití užitečných dat přicházejících pro konkrétní endpoint.
7. V hlavním programu vložit `include "usb.h"`, mezi inicializační příkazy vložit `UsbInit();` a do hlavní smyčky umístit své vlastní algoritmy, které pracují s periferiemi a plní buffery endpointů.

USB řadič pracuje ve výhradním režimu se zakázanými přerušeními. Je nežádoucí zdržovat rutiny řadiče čekáním, výpočtem nebo voláním, které by mohlo být vykonáno mimo rutinu řadiče. U jednobytových paketů je na volbě návrháře, zda bude výpočet CRC16 pro odchozí pakety řešit uvnitř rutin (těsně před odesláním paketu) nebo vně rutin. U vícebytových přenosů je doporučeno CRC16 počítat dopředu a při odesílání paketu pouze načíst hotovou hodnotu z bufferu a paket bez prodlení odeslat.

5.5 Možnosti aplikace

Přípravek s ATmega8 je díky modulárnímu kódu v jazyce C vhodný pro nejrůznější aplikace, které by při realizaci s jinými obvody byly mnohonásobně dražší. S drobnými úpravami lze snadno implementovat například tato zařízení:

- levný ISP programátor mikrokontrolérů
- zařízení Mass Storage s přístupem k paměťové kartě Secure Digital
- datalogger sériové linky

- vzorkovač analogového signálu
- myš, klávesnice nebo jiné zařízení HID

5.6 Opakovatelnost

K výrobě přípravků byly použity pouze všeobecně dostupné produkty, které je možné objednávat jak kusově, tak v tisícových sériích. Výrobu desek plošných spojů profesionálním výrobcem je na základě filmových materiálů možné opakovat i po delší době. Použité mikrokontroléry, krystaly, rezistory či kondenzátory jsou k dispozici v bezolovnatých provedeních dle specifikace RoHS.

5.7 Ověření hypotézy

Funkční přípravky realizující přenos uživatelských dat mezi aplikací v osobním počítači a periferií jsou výsledkem experimentu, který potvrzuje platnost hypotézy stanovené v úvodu této práce. Výkonnostní parametry mikrokontrolérů AVR jsou dostatečné na to, aby spolu s optimalizovaným kódem zvládly realizovat softwarový USB řadič. USB zařízení vzniklé použitím navržené knihovny v jazyce C je funkční.

6 Závěr

Výsledkem práce jsou 4 funkční přípravky, ve kterých je softwarově implementováno Univerzální Sériové Rozhraní pomocí AVR mikrořadiče, který USB hardwarově nepodporuje. V souladu s cíli práce byl navržen modulární kód v jazyce C, který pomocí standardních vstupně/výstupních portů mikrokontroléru komunikuje po USB lince. Pro dva přípravky byl vyvinut vlastní ovladač zařízení WDM pro Windows XP a byla též předvedena možnost použití platformově nezávislých generických ovladačů. Aplikační software v jazyce Delphi umožňuje interaktivně vyzkoušet základní schopnosti všech přípravků. V porovnání s ostatními řadiči USB sběrnice je zde prezentovaný softwarový řadič výrazně levnější, flexibilnější, dostupnější a umožňuje realizaci USB zařízení pomocí jediného programovatelného obvodu s minimem externích součástek. Čitelný kód zabírající přibližně polovinu prostředků mikrořadiče je možné bez principielních úprav snadno rozšířit o další funkcionality.

7 Zdroje a literatura

[1] USB Implementers Forum. Universal Serial Bus Specification Revision 1.1 [online], c1998. 311 s. [cit. 2007-04-10]. URL: <*http://www.usb.org/developers*>.

[2] Atmel Corporation. AVR 8-Bit RISC Introduction [online], c2007. [cit. 2007-05-15].
URL: <*http://www.atmel.com/products/avr/overview.asp*>.

[3] Future Technology Devices International Limited. FTDI Products [online], c2007. [cit. 2007-05-16].
URL: <*http://www.ftdichip.com/FTProducts.htm*>.

[4] ČEŠKO, Igor. Dálkové ovládání přes USB [online], c2000-2007. [cit. 2007-05-21].
URL: <*http://www.cesko.host.sk/IgorPlugUSB/IgorPlug-USB%20(AVR).htm*>.

[5] PEACOCK, Craig. USB in a Nutshell [online], c2001-2007, upr. 2007-04-06, [cit. 2008-01-01].
URL: <*http://www.beyondlogic.org/usbnutshell/usb1.htm*>.

[6] PEACOCK, Craig. USB Protocol Analyzers [online], c2003-2005, upr. 2007-06-15, [cit. 2008-01-03].
URL: <*http://www.beyondlogic.org/usb/protocolanalysers.htm*>.

[7] Prezentace firmy Spezial Electronic [online],[cit. 2008-01-06].
URL: <*http://www.spezial.cz*>.

[8] Microsoft Corporation. Dokumentace ovladačů WDM jako součást Driver Development Kitu. [online], [cit 2008-01-12].
URL: <*http://www.microsoft.com/whdc/devtools/ddk/default.mspx*>.

[9] Microsoft Corporation. Microsoft Developer Network. [online], c2008, [cit 2008-01-12].
URL: <*http://msdn2.microsoft.com/cs-cz/default(en-us).aspx*>.

[10] USB Implementers Forum. Specifikace Communication Device Class verze 1.1 [online], c1999, upr. 1999-01-19, [cit. 2008-01-06].
URL: <*http://www.usb.org/developers/devclass_docs/usbcdc11.pdf*>.

V pro napsání této práce, pro výrobu hardware a vývoj programů byly použity tyto produkty:

[11] SCHENK, Christian. Programový balík MikTEX2.0 s publikačním systémem LATEX. [online], c2008, [cit 2008-01-12].
URL: <*http://miktex.org*>.

[12] Atmel Corporation. AVR Studio - vývojové prostředí pro simulaci programů v AVR Assembleru. [online], c2008, [cit 2008-01-12].
URL: <*http://www.atmel.com*>.

[13] Překladač AVR GCC pro překlad jazyka C pro mikrokontroléry AVR. [online], [cit 2008-01-12].
URL: <*http://winavr.sourceforge.net*>.

[14] Microsoft Corporation. Windows XP Driver Development Kit. [online], [cit 2008-01-12].
URL: <*http://www.microsoft.com/whdc/devtools/ddk/default.mspx*>.

[15] Lancos. PonyProg Serial Device Progammer. [online], c2007, [cit 2008-01-12].
URL: <*http://www.lancos.com/prog.html*>.

[16] Borland Software Corporation. Delphi 7 Enteprise - programovací jazyk s vývojovým prostředím. c2002.

[17] Cadence Design Systems, Inc. Orcad 9.2, c1985 - 2000.

[18] Microsoft Corporation. Sysinternals DebugView a IrpTracker. [online], c2007, [cit 2008-01-12].
URL: <*http://technet.microsoft.com/cs-cz/sysinternals/default(en-us).aspx*>.

[19] Helios Software Solutions. TextPad 4.7.3 textový editor. [online], c2007, [cit 2008-01-12].
URL: <*http://www.textpad.com*>.

[20] Microsoft Corporation. Operační systém Windows XP, Service Pack 2. c2002.

8 Seznam obrázků

Seznam obrázků

9 Seznam tabulek

Seznam tabulek

10 Obsah CD

Na kompaktním disku, který je přílohou této diplomové práce, je obsaženo:

- `ATtiny13spi` - složka pro 1. přípravek
- `ATtiny13adc` - složka pro 2. přípravek
- `ATtiny2313` - složka pro 3. přípravek
- `ATmega8` - složka pro 4. přípravek
- `Aplikace` - instalační soubory některých aplikací a nástrojů použitých k návrhu
- `Specifikace` - soubory specifikací a pomocné dokumenty
- `smrz.pdf` - tato diplomová práce ve formátu PDF

Jednotlivé složky obsahují ovladače zařízení, zdrojové kódy od softwarového USB řadiče, aplikační software a jeho zdrojové kódy a dále schémata a motivy desek plošných spojů. V každé složce jsou zavedeny tyto podsložky:

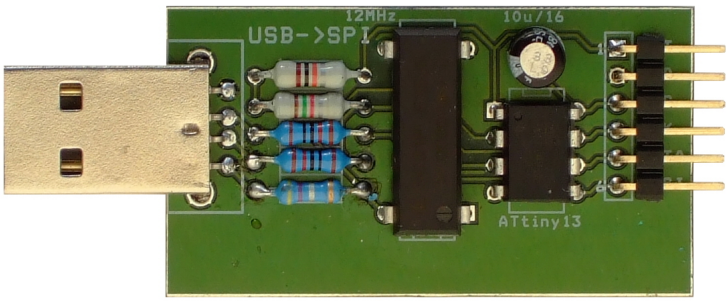
- `drv` - ovladače zařízení a soubory s informacemi o hardwaru
- `srcdrv` - zdrojové kódy ovladačů zařízení
- `apl` - aplikační software
- `srcapl` - zdrojové kódy aplikačního software
- `pcb` - schémata a motivy desek plošných spojů
- `src` - zdrojové kódy softwarového USB řadiče

Ve složce 4. zařízení je umístěn soubor `WinXPkb918365.exe` - hotfix operačního systému, který umožní správnou funkci kompozitního USB zařízení.

11 Příloha: návod k přípravkům

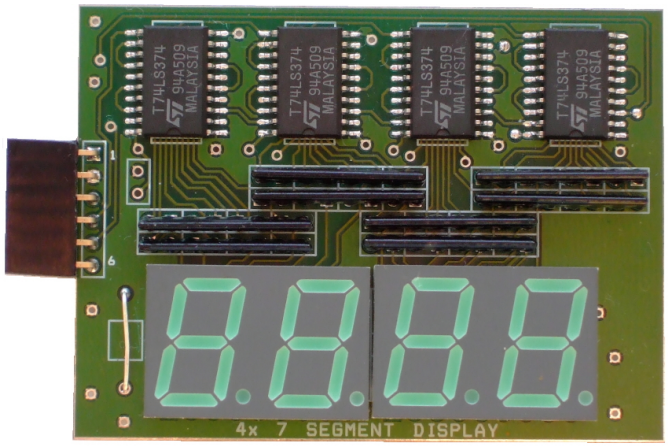
Přípravek 1. - výstupní SPI port

Přípravek zasuňte do volného USB portu nebo do prodlužovacího A-A kabelu. Systém nalezne nové USB zařízení a bude požadovat specifikování cesty k ovladači. Pokud je do mechaniky CD vložen disk, který je součástí této práce, měly by být soubory usbspi.inf a usbspi.sys nalezeny automaticky.

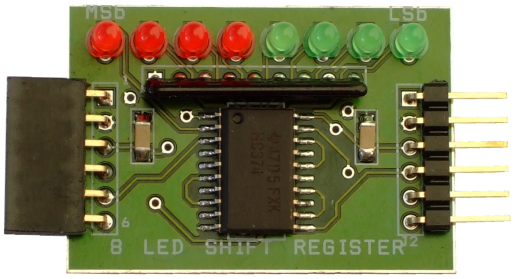


Obrázek 38: Přípravek 1.

K přípravku připojte některý z rozšiřujících modulů. Pro zobrazování čísel nebo znaků připojte modul čtyřmístného 7 segmentového displeje tak, aby piny konektorů označené tečkou byly spojeny spolu. Pro zobrazování binárních čísel je možné připojit blok 8xLED zobrazovače, popřípadě lze oba bloky zřetězit dohromady (obrázek 41.



Obrázek 39: Čtyřmístný 7 segmentový displej



Obrázek 40: 8xLED zobrazovač

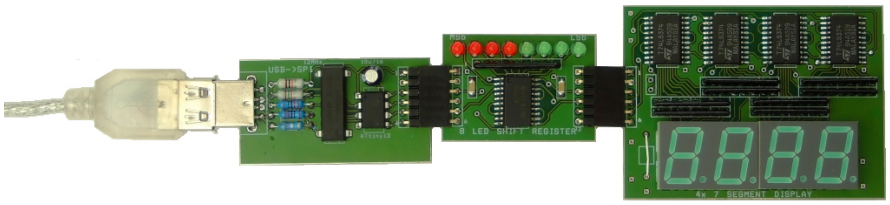
Zařízení funguje jako sériový synchronní výstupní port, který na datový pin (DATA) vystavuje postupně všechny bity přenášeného znaku od LSb po MSb a každý z nich potvrdí náběžnou hranou hodinového signálu (SCK).

Rozhraní zařízení bude ovladačem exportováno pod názvem USBSPI. Pro přístup k zařízení je možné použít buď ukázkovou aplikaci nebo přímo příkazovou řádku. Pro přímý přístup spusťte soubor zapis.bat nebo použitje příkaz:

```
C:\>echo A>> \\.\USBSPI
```

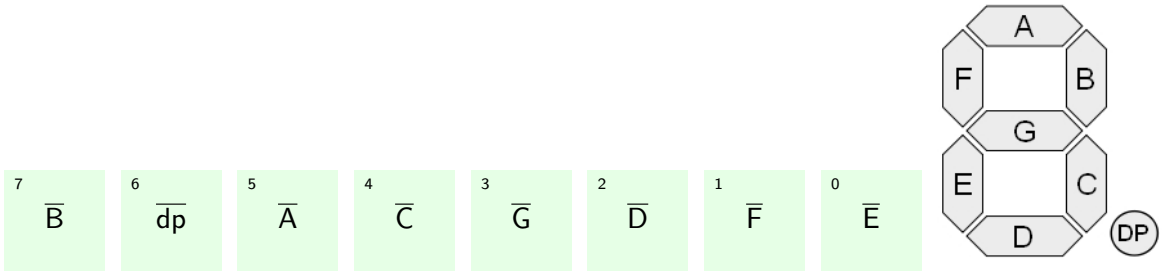
A na výstupu proběhnou znaky ASCII #65#13#10, které směrem zprava doleva nasunou na pozice displeje nebo LED. Pro zápis na 7 segmentový displej po znacích je možné použít program encode7.exe a výstup přesměrovat na zařízení:

```
E:\ATtiny13-12MHz>encode7 AHOJ >> \\.\USBSPI
```



Obrázek 41: Řetězení zobrazovačů

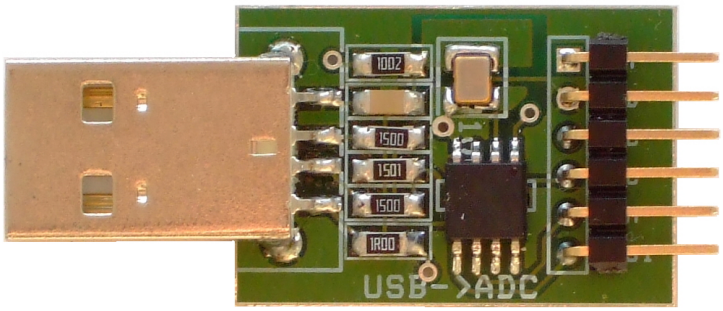
Každý bit dat, která jsou pomocí hodin odtaktována do modulu 7-segmentového displeje, rozsvítí určitý segment displeje (podle obrázku 42). Aby bylo možné na displeji zobrazovat rotující text, je každý nový znak přidáván zprava.



Obrázek 42: Mapování bitů na segmenty displeje

Přípravek 2. - regulátor s ADC a PWM

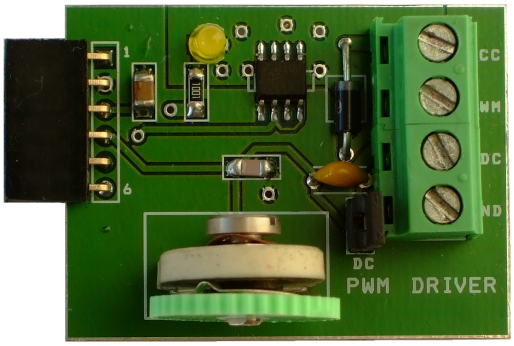
Přípravek zasuňte do volného USB portu nebo do prodlužovacího A-A kabelu. Systém nalezne nové USB zařízení a bude požadovat specifikování cesty k ovladači. Pokud je do mechaniky CD vložen disk, který je součástí této práce, měly by být soubory usbadc.inf a usbadc.sys nalezeny automaticky.



Obrázek 43: Přípravek 2.

K přípravku připojte rozšiřující modul, který obsahuje odporový trimr a budič s LED indikátorem. Trimr funguje jako nastavitelný dělič, tedy řiditelný zdroj napětí s vysokým vnitřním odporem, což lze také chápat jako zdroj analogového signálu. V případě, že chcete měřit napětí z externího zdroje, připojte jej mezi svorky ADC a GND. Zkratovací propojkou DC se určuje způsob vazby. Pokud je zkratovací propojka zapojena, napětí externího zdroje je přivedeno na měřicí vstup USB zařízení. Trimrový dělič je připojen ke stejnému uzlu, takže výsledné napětí je váženým průměrem napětí, kde vahou je převrácená hodnota vnitřního odporu zdroje. U dostatečně tvrdých zdrojů (vnitřní odpor pod 100 Ω) se hodnota trimru neprojeví. Pokud je ovšem zkratovací propojka odstraněna, je zvolena střídavá vazba. Z napětí přivedeného na ADC vstup je odfiltrována stejnosměrná složka a střídavé napětí sečteno s hodnotou trimrového děliče. Takovéto nastavení napěťového offsetu je nutné, protože napětí, které se takto přivádí na měřicí vstup USB zařízení musí být v rozsahu napětí 0 - 5 V. Střídavé signály

by bez offsetu nemohly být měřeny.



Obrázek 44: Modul regulátoru

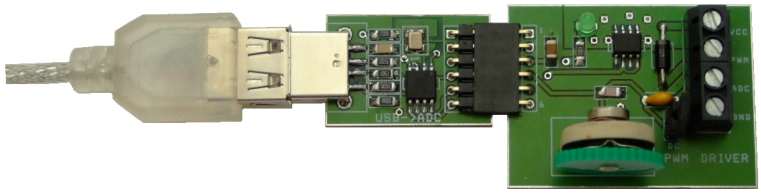
Rozhraní zařízení bude ovladačem exportováno pod názvem USBADC. Pro přístup k zařízení je možné použít buď ukázkovou aplikaci nebo přímo příkazovou řádku. Čtení hodnoty napětí v ASCII formě není užitečné, proto je lepší použít program piperead:

```
E:\ATtiny13adc>piperead -c 0 -d 100 -t -f "%d" <\\.\USBADC
```

Nekonečné čtení z roury lze přerušit stiskem Ctrl+C. Data budou vypisována jako decimální číslo každých 100ms. Zápis hodnoty do zařízení způsobí změnu komparační úrovně PWM obvodu. Pro číslo 0 je výstup v logické 0, pro číslo je stále v logické 1. Pro mezihodnoty platí přímá úměra mezi velikostí čísla a dobou setrvání pulzu v logické 1 dělenou periodou signálu (tzv. duty cycle).

Mnohem užitečnější je ale použití ukázkového schématu v Simulinku Matlabu. Pomocí s-souboru je definován nový blok Simulinku, který obsahuje jak vstupní bránu, tak výstupní bránu. Zde je vstup/výstup chápan z hlediska bloku Simulinku, tedy výstupem bloku je hodnota signálu na pinu ADC a vstupem bloku je požadovaná šířka pulzu na výstupu PWM.

Při jednoduchém propojení výstupu bloku se vstupem je velikost natočení trimru převedena na svit LED. Je možné realizovat i regulační obvod tak, že výstup PWM bude použit na buzení regulovaného systému a výstup z regulovaného systému bude připojen na pin ADC. Rychlosti dějů na výstupu regulovaného systému by měly být úměrné maximální přenosové rychlosti USB zařízení, kterou však nelze s jistotou určit. Doporučeno je čerpat 500 až 1000 vzorků za sekundu.

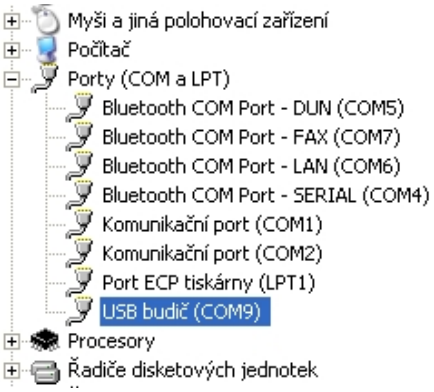


Obrázek 45: Přípravek 2. se zapojeným modulem

Přípravek 3. - sériové a paralelní vstupně-výstupní zařízení

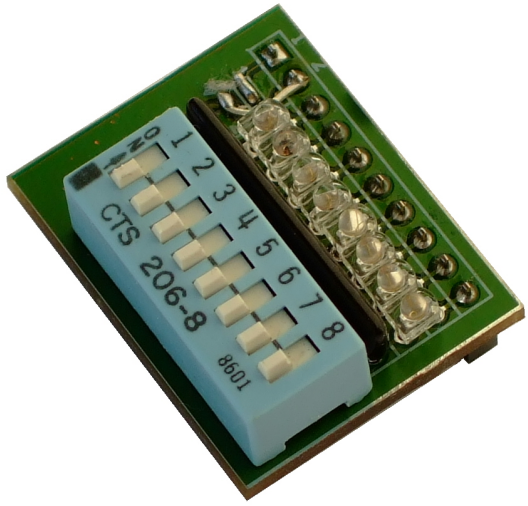
Přípravek zasuňte do volného USB portu nebo do prodlužovacího A-A kabelu. Systém nalezne nové USB zařízení a bude požadovat specifikování cesty k informacím o hardwaru. Pokud je do mechaniky CD vložen disk, který je součástí této práce, měly by být automaticky nalezen soubor usbparallel.inf, který sdělí systému, že má použít generický ovladač usbser.sys. Po úspěšné instalaci se ve Správci zařízení objeví ve skupině Porty nové zařízení ATtiny2313 s označením COMx (obrázek 47.

Obrázek 46: Přípravek 3.



Obrázek 47: Přípravek 3. ve správci zařízení

K přípravku připojte osminásobný spínač, který má správně zapojené zkratovací propojky (pin 1 je GND).



Obrázek 48: Přepínač

Spínač funguje tak, že při sepnutém stavu připojí k pinu pull-up resiztor 300 Ω. LED, které signalizují logickou úroveň jsou připojeny mezi pin a zem se sériovým odporem 10 kΩ. Pokud je port nastaven pro výstup a budiče jsou aktivní, nezávisle na nastavení přepínačů je na pinu logická úroveň daná binární hodnotou čísla zapsaného do USB zařízení. Při režimu čtení (budiče jsou ve stavu vysoké impedance) je napětí na pinu dáno děličem vytvořeným z pull-up rezistoru, sériového rezistoru a LED. Přepínače v zapnutém stavu pomocí pull-up rezistoru vytváření na pinu logickou 1 a LED svítí. Při vypnutí přepínače je napětí v logické 0 (přizemněno LED diodou a sériovým odporem) a LED nesvítí.

K zařízení je možné přistupovat jako k souboru se symbolickým názvem COMx pouze tehdy, když x je v rozsahu od 1 do 9. Pro COM10 a výše je nutné používat při rourových přesměrováních cestu \\.\COMx. Zařízení obsahuje dvě rozhraní: sériové a paralelní. Symbolický název COMx je však jen jeden a rozlišit mezi dvěma rozhraními nelze. Dále je potřeba nastavovat směr dat u paralelního rozhraní. Řešení bylo navrženo nestandardní - pro rozlišení rozhraní je použita hodnota modulační rychlosti (Baudrate). Pokud je modulační rychlost menší než 256, pak je zvoleno paralelní rozhraní a směr dat je určen hodnotou n-tého bitu v binární reprezentaci hodnoty modulační rychlosti. Pro 255 jsou aktivní všechny budiče, pro hodnotu 0 žádné a pro 15 jsou aktivní jen dolní 4 bity. Pokud je modulační rychlost větší než 255, pak je její použití standardní a je zvoleno sériové rozhraní. Všechny vstupně-výstupní operace jsou od změny modulační rychlosti směrovány na příslušné rozhraní do doby, než bude rychlost opět změněna.

Nastavení paralelního portu pro výstup:

```
C:\>mode COMx baud=255
```

Nastavení paralelního portu pro čtení:

```
C:\>mode COMx baud=0
```

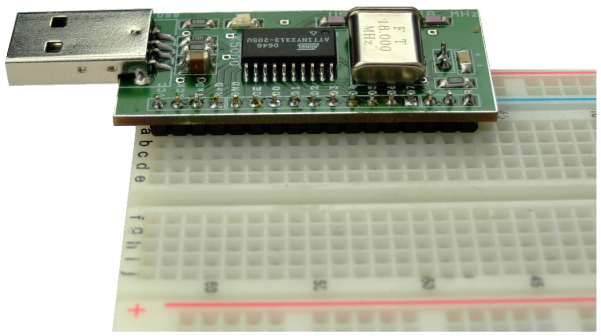
Nastavení sériového portu na rychlost 19200 baud:

```
C:\>mode COMx baud=19200
```

Zápis do zvoleného rozhraní:

```
C:\>echo A>>\\.\COMx
C:\>type ff.txt >> \\.\COMx
```

Zařízení je určeno jako náhrada klasických IO portů počítače (IEEE 1284 a RS232 porty), které postupem času mizí ze základního vybavení osobních počítačů. Port LPT (IEEE 1284) není pod Windows XP přístupný jako soubor a je nutné používat zvláštní knihovny. Vývody jsou na zařízení uspořádány tak, aby mohly být zasunuty do nepájivého kontaktního pole (obrázek 49) a rovnou použity pro experimentování. Celkem 17 vývodů směřuje dovnitř pole a dva vývody připojují napětí 5V z USB na napájecí rozvodné řady.

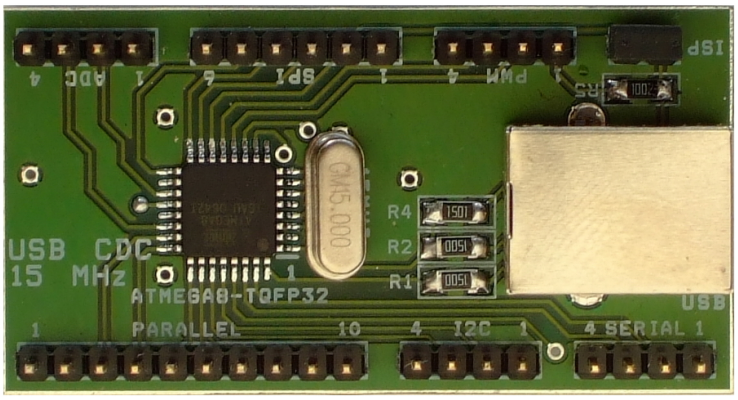


Obrázek 49: Zapojení přípravku 3. do kontaktního pole

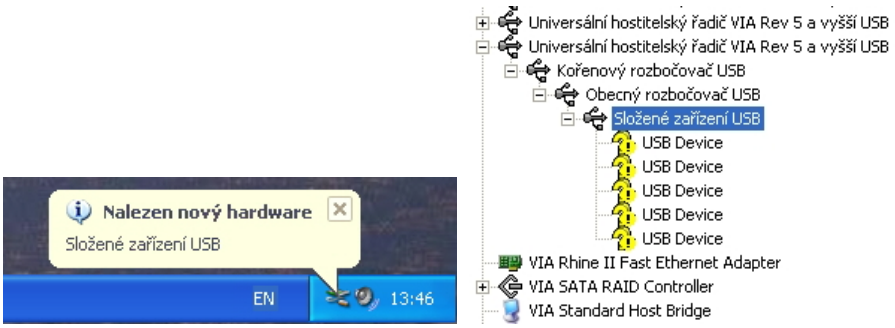
Přípravek 4. - kompozitní USB zařízení

Před zasunutím přípravku do USB portu nainstalujte záplatu operačního systému Windows XP KB918365, která je uložena na CD s touto prací. Záplata opravuje chybu operačního systému, kvůli které nedokázal správně pracovat se složenými USB zařízeními. Záplatu nelze z důvodu dalšího vývoje volně stáhnout, ale je vyžadováno zaslání žádosti, na základě které je záplata poskytnuta ke stažení.

Po nainstalování hotfixu pokračujte propojením přípravku s volným USB portem pomocí propojovacího A-B kabelu. Systém nalezne složené USB zařízení a poté objeví 5 nových zařízení (obrázek 51), pro která bude postupně vyžadovat ovladače. Pokud je do mechaniky CD vložen disk, který je součástí této práce, měly by být automaticky nalezeny soubory usbio.inf a usbio.sys. Po úspěšné instalaci se ve Správci zařízení objeví nová zařízení ve skupině USBIO Controlled Devices.



Obrázek 50: Přípravek 4.



Obrázek 51: Přípravek 4. ve správci zařízení

K přípravku připojte osminásobný spínač, který má správně zapojené zkratovací propojky (pin 2 je GND).

Ke kompozitnímu zařízení přistupujte z programů buď pomocí standardních API funkcí pro práci se soubory (CreateFile, ReadFile, WriteFile, CloseHandle) a pro zjištění cesty k zařízení použijte funkce SetupEnumDeviceInterfaces a přidružené funkce. Identifikátor dílčích zařízení (GUID) byl vygenerován firmou Thesycon:

```
USBIO_GUID = '{325ddf96-938c-11d3-9e34-0080c82727f4}';
```

Bližší návod, jak přistupovat k zařízením, je možné získat prostudováním knihoven, které jsou součástí ovladačového balíku USBIO, nebo konkrétně nahlédnutím do souboru usbio_i.pas, který byl použit pro vývoj demonstrační aplikace. Jednoduché názvy (pro DOS) nejsou ovladačem exportovány a proto nelze použít jednoduché cesty, jako tomu je u předchozích 3 přípravků.