VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

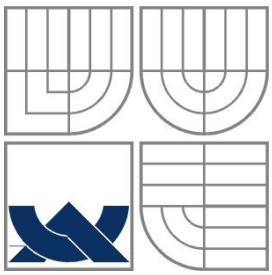
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TVORBA OVLADAČŮ ZAŘÍZENÍ POD SYSTÉMY LINUX A WINDOWS

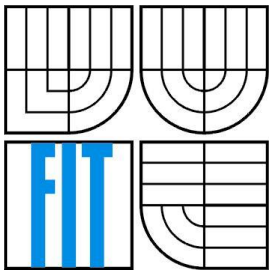
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAKUB SZNAPKA
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TVORBA OVLADAČŮ ZAŘÍZENÍ POD SYSTÉMY LINUX A WINDOWS

WRITING DEVICE DRIVERS UNDER LINUX AND WINDOWS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
JAKUB SZNAPKA

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. JOSEF STRNADEL, Ph.D.

BRNO 2015

Abstrakt

Práce se zabývá návrhem a implementací USB ovladače a klíče třídy HID pro mikrokontrolér MC9S08JM60. Popisuje se zde jak teoretická část nezbytná pro návrh, tak i implementace s ukázky kódu a obrázky pro lepší ilustraci. Jsou zde zahrnuty i výpisy testovacích programů pro ověření funkčnosti.

Abstract

This thesis is focused on design and implementation of USB driver and key of HID class for microcontrollers MC9S08JM60. It describes theoretical part necessary for design, as well as implementation with shown lines of code and pictures. Outputs of test programs are also included.

Klíčová slova

Mikrokontrolér, MC9S08JM60, USB, USB rozhraní, USB klíč, koncový bod, buffer, ovladač, protokol, specifikace, human interface device.

Keywords

Microcontroller, MC9S08JM60, USB, USB interface, USB key, endpoint, buffer, driver, protocol, specification, human interface device.

Citace

Jakub Sznapka: Tvorba ovladačů zařízení pod systémy Linux a Windows, bakalářská práce, Brno, FIT VUT v Brně, rok 2015

Tvorba ovladačů zařízení pod systémy Linux a Windows

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana
Ing. Josefa Strnadela, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jakub Sznapka
5. 3. 2015

Poděkování

Děkuji panu Ing. Josefovi Strnadelovi, Ph.D. za odborné rady a pomoc při řešení bakalářské práce.
Také bych chtěl poděkovat své rodině za podporu a pomoc při práci na technické zprávě.

1 Obsah

1.1 Seznam obrázků

1.2 Seznam tabulek

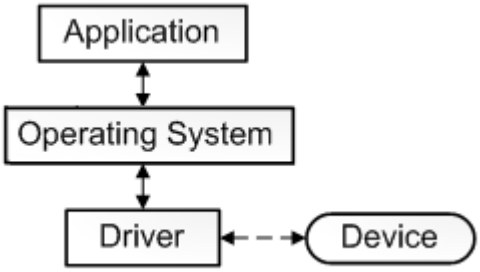
2 Úvod

Ovladače jsou jedny z nejdůležitějších programů v počítači. Umožňují operačnímu systému a ostatním programům komunikaci se zařízením, pro které je ovladač vyvinut. Slouží tedy jako komunikační rozhraní mezi programy a zařízením tak, že převádí požadavky programů, aby jim zařízení rozumělo, a data od zařízení převádí do formy srozumitelné pro program. S rychlostí, jakou je dnes vyvíjen hardware zaručuje, že ovladače budou velmi potřeba i v blízké budoucnosti. Tato práce se zabývá problematikou vývoje ovladačů pro různá zařízení a operační systémy.
V kapitole 3 je určeno, jakým směrem se bakalářská práce bude ubírat a proč. Kapitola 4 obsahuje nezbytné pojmy pro pochopení práce. Také je v ní obsažen popis zvolené platformy, pro kterou bude ovladač vyvíjen a nakonec obsahuje programové vybavení a popis softwaru, využitý při tvorbě ovladače. Kapitola 5 už blíže specifikuje fungování ovladačů, nachází se v ní princip činnosti zařízení a ovladače. Je v ní také důkladně popsána implementace ovladače.

3 Rozbor tématu

3.1 Problematika návrhu ovladačů

Ovladače jsou programy, které ovládají zařízení, připojené k počítači. Tyto programy jsou softwarové instrukce, které umožňují komunikaci zařízení s počítačem. Komunikace probíhá přes sběrnici nebo komunikační podsystém. Funkce ovladačů spočívá v poskytnutí rozhraní mezi počítačem a zařízením. Hlavní účel ovladače je v přeložení požadavků od jiných programů popř. operačního systému tak, aby jim zařízení rozumělo, a následně překládá zpět odpověď od zařízení. Ovladače jsou závislé na hardwaru a operačním systému.



Obrázek 1 – ovladač

Programování ovladačů vyžaduje úplnou znalost hardwaru zařízení a funkčnosti softwaru pro danou platformu. Obvykle je totiž požadováno zpracování přerušení a asynchronních požadavků. Každé připojené zařízení vyžaduje ovladač, aby rozumělo požadavkům programů. Některá zařízení však mají společné funkce a mohou využívat společných ovladačů.

3.2 Přehled dostupných OS

Vzhledem k rozšířenosti jsou uvedeny pouze 3 nejpoužívanější operační systémy:
MAC OS – operační systém pro počítače firmy Apple. Používá hybridní jádro (XNU) a grafické uživatelské rozhraní Aqua. Aktuální verze systému je OS X Yosemite 10.10.3. Pro vývoj ovladačů pod tímto systémem byl vydán open-source Framework I/O Kit, který lze použít k vytváření ovladačů pro systémy OS X a iOS.
Microsoft Windows - operační systém firmy Microsoft s jádrem Windows NT. Pro vývoj ovladačů pro tento operační systém existuje sada Windows Driver Kit (WDK). Tato sada obsahuje dokumentaci, příklady, prostředí a nástroje pro vývoj ovladačů. Hlavní částí této sady je Windows Driver Frameworks(WDF). WDF se skládá z Kernel Mode Driver Framework(KMDF) a User Mode Driver Framework(UMDF). Hlavním cílem této sady je umožnit vývojářům použití jednoho konceptu pro více typů ovladačů.
Linux – open-source operační systém rodiny Unix. Ovladače pro Linux jsou uloženy v modulech. Moduly jsou části kódu, které mohou být vloženy do kernelu (a rozšířit tak jeho funkčnost) za běhu a mohou být za běhu i odebrány. O vložení modulu do kernelu se stará program *insmod* a o odebírání *rmmod*.

3.3 Problematika výběru zařízení

Ovladače jsou velmi závislé na hardwaru. Rozhraní mezi zařízením a operačním systémem je nezbytné jak pro periferie, tak i pro sběrnice kterými se připojují. Příklady zařízení:

- Tiskárny
- Video adaptéry
- Síťové karty
- Zvukové karty
- Sběrnice
- I/O zařízení (myš, klávesnice, joystick)
- Úložiště dat (hard disk, blu-ray)
- Skenery a kamery

Speciálním typem jsou virtuální zařízení. Ovladače pro virtuální zařízení emulují skutečné hardwarové zařízení, takže místo funkce rozhraní ovladač emuluje části hardwaru a umožňuje tak operačnímu systému iluzi komunikace se skutečným hardwarem. Ovladače pro virtuální zařízení také mohou simulovat různé události, např. přerušení apod.

3.4 Zvolené řešení

Pro zvolení řešení je potřeba zohlednit několik faktorů, které mají vliv na vývoj ovladačů:
Dokumentace – jeden z nejdůležitějších faktorů. Vývoj ovladačů vyžaduje úplnou znalost platformy, pro kterou je ovladač vyvíjen a znalost operačního systému, na kterém bude pracovat. Vzhledem k faktu, že pro fungování ovladače je nutné znát podrobně oba konce rozhraní, je tedy nezbytné aby existovala potřebná dokumentace, jinak je vývoj velmi komplikovaný.

- **Dostupnost** – stejně důležitá je i dostupnost jak vývojových sad pro zvolený operační systém, tak i zařízení, aby bylo možné ovladač otestovat a popř. opravit chyby v rozhraní.
- **Rozsáhlost** – velmi komplikované zařízení, i když je dobře zdokumentované, může mít velmi složitou implementaci.
- **Aktuálnost –** příliš nové prostředky nemusí být řádně zdokumentovány pro námi zvolený systém, a příliš staré nemusí být systémem vůbec podporovány

Jako řešení tedy byl zvolen ovladač pro sběrnici USB, jelikož se jedná o nejrozšířenější sběrnici, která je dobře zdokumentována a veškeré podklady lze nalézt na internetu. Zvolená platforma, na které se USB nachází, je výukový kit DEMOJM s mikrokontrolérem MC9S08JM60. Tento kit je totiž volně k zapůjčení na VUT a tak je možné ovladač řádně otestovat, navíc se jedná o výukový kit, takže dokumentace je velmi podrobná. Ovladač vyvíjen na operačním systému Windows 7 a testován bude jak na Windows tak Linux.

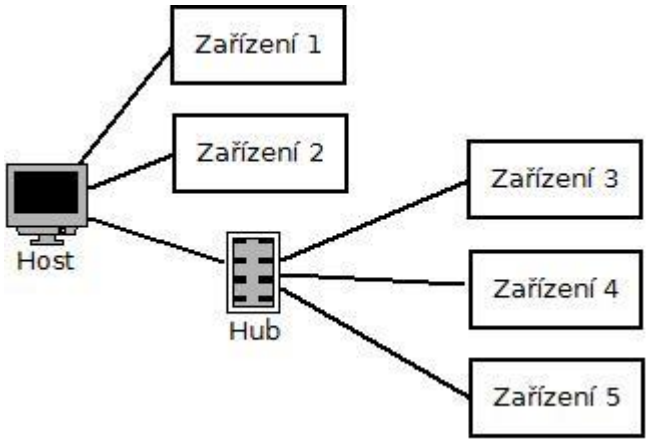
4 Popis prostředků zvoleného řešení

4.1 Principy USB a konstrukce USB zařízení

Standard USB definuje konektory, kabel a komunikační protokoly pro připojení, komunikaci a napájení mezí počítačem a elektronickým zařízením. USB bylo navrhnuto, aby sjednotilo propojení mezi počítačovými periferiemi (např. klávesnice, myš, tiskárna atd.) a počítačem.
Zatím existují 3 verze USB, které se se liší rychlostí přenosu dat: USB 1.0 definuje datový přenos "Low Speed" 1.5 Mbit/s, který měl sloužit pro připojení nízko-rychlostních zařízení, nebo "Full Speed" 12 Mbit/s pro připojení disků. Specifikace USB 2.0 zaznamenala 40ti násobný nárůst rychlosti přenosu dat, tedy 480 Mbit/s. Nejnovější verzí je USB 3.0, která dosahuje rychlosti až 5 Gbit/s a je zpětně kompatibilní s USB 2.0.

4.1.1 Architektura

USB je univerzální sériová asynchronní sběrnice. Její topologie je navrhnuta jako asymetrická, skládající se z hosta a periferních zařízení. Až 127 zařízení může být připojeno k jednomu hostu, včetně USB hubů, které můžou zvýšit počet zařízení. Architektura je typu master-slave, kdy master řídí připojování zařízení, jejich konfiguraci a veškerou komunikaci.
Host je řídící prvek USB sběrnice a je v něm obsažen kořenový hub. Vyzývá připojené zařízení a rozbočovače na komunikaci, ale v jeden okamžik může vyzývat pouze jedno zařízení.



Obrázek 2 - USB systém

4.1.2 USB třídy

Standard USB definuje třídy, které se používají pro identifikaci funkcí zařízení. Kromě identifikace slouží také k sloučení zařízení se stejnými požadavky na přenos.

Příklady USB tříd:

Kód třídy	Název třídy	Popis
01h	Audio	Reproduktory, mikrofon
03h	Human interface device	Myš, klávesnice
05h	Physical interface device	Joystick
08h	Mass storage	USB flash disk, externí disk
0Eh	Video	Webkamera

Tabulka 1 - USB třídy

4.1.3 Princip rozhraní

Přenos informací mezi zařízením a hostem probíhá přes koncové body (endpointy). Každé USB zařízení obsahuje několik nezávislých endpointů.

Endpoint je adresovatelný buffer, který slouží jako zdroj nebo úložiště dat. Existují 2 druhy endpointů podle směru dat.

1) **Endpoint IN** - slouží pro přenos ze zařízení do počítače. Data zde jsou uložena, dokud si je nevyžádá host.
2) **Endpoint OUT** - pro přenos z počítače do zařízení

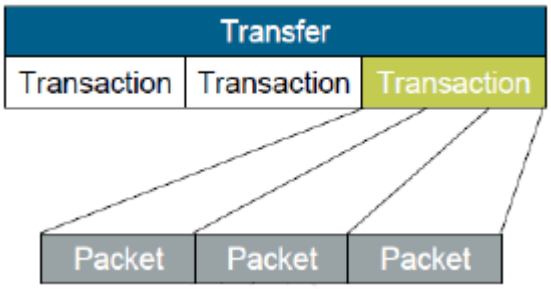
Každý endpoint je popsán vlastnostmi, které jsou: číslo, směr přenosu, velikost bufferu a přenosové pásmo. Dva endpointy můžou mít stejné číslo ale rozdílný směr přenosu. Zařízení může obsahovat maximálně 16 endpointů.
Každé USB zařízení musí podporovat endpoint 0 pro enumeraci, řídící a stavové procesy. Endpoint 0 je obousměrný a inicializuje se jako první. Kromě toho má každé zařízení ustanovenou rouru (pipe) pro komunikaci mezi počítačem a endpointem 0.

Roura je logický kanál mezi hostitelem a koncovým bodem a komunikuje s koncovým bodem pomocí jeho adresy. Roura pro endpoint je otevřena, když je zařízení nakonfigurováno. Roura je pouze softwarově vytvořená. USB definuje 2 typy rour:

1) **Stream roura** - nemá žádný definovaný USB formát a mohou být kontrolovány buď hostem, nebo zařízením. Jednosměrná.
2) **Message roura** – má definovaný USB formát. Je kontrolována hostem a směr dat je uveden v požadavku. Obousměrná.

4.1.4 Komunikace

Komunikace probíhá pomocí přenosu, což je proces zajišťující kompletní vykonání příkazu od počítače USB zařízením. Komunikaci inicializuje host odesláním token paketu s adresou zařízení a adresou endpointu. Je tvořen posloupností transakcí, které se skládají z paketů. První paket obsahuje popis komunikace, zařízení, se kterým se bude pracovat, koncový bod a směr.



Obrázek 3 - Formát transakce [8]

Pakety se skládají z následujících polí

1) **Sync –** Všechny pakety musí začínat polem sync. Délka je 8 bit pro low speed a 32 bitů pro high speed. Hlavní účel sync pole je synchronizace časovače přijímače s časovačem vysílače. Poslední 2 bity značí začátek PID.
2) **PID –** Identifikační číslo paketu. Slouží pro identifikaci typu paketu. Číslo je dlouhé 4 bity, ale pro zajištění správnosti doručení je PID zopakován a pole je tedy dlouhé 8 bitů.

Typ	PID	Popis
Token	0001	OUT token
	1001	IN token
	0101	SOF token
	1101	SETUP token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK
	1010	NAK
	1110	STALL
	0110	NYET

Tabulka 2 - Typy paketu podle PID

3) **ADDR –** Obsahuje adresu zařízení, pro které je paket určen. Tímto polem může být adresováno až 127 zařízení, avšak adresa 0 je neplatná a pokud paket obsahuje adresu 0, všechna zařízení, kterým nebyla adresa přidělena, na tento paket odpoví.
4) **CRC –** slouží pro detekci chyb během přenosu. Token pakety mají 5 bitů CRC a data pakety 16bitů CRC
5) **EOP –** Konec paketu, který je signalizován pomocí SE0 ve dvou bitech.

Pakety mají předem definovaný formát, jenž je dán specifikací USB.

1) **Token packet –** používá se pro přístup ke správné adrese a endpointu.
   - In – host bude data číst
   - Out – host bude data odesílat
   - Setup – zahájení řídícího přenosu

Obrázek 4 - Token packet [8]

2) **Data packet –** slouží k přenášení dat. Data pakety jsou označeny buď Data0 nebo Data1 a mohou přenášet až 1024 bytů dat.

Obrázek 5 - Data packet [8]

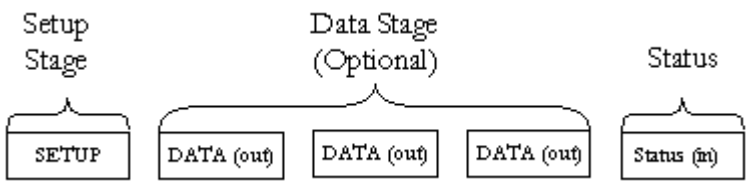
3) **Handshake packet –** jsou požívány pro potvrzení proběhlé transakce popř. chyby
   - ACK – paket byl úspěšně přijat
   - NAK – zařízení dočasně nemůže přijímat ani vysílat data
   - STALL – zařízení čeká na zákrok hosta

Obrázek 6 - Handshake packet [8]

4.1.5 Typy přenosů

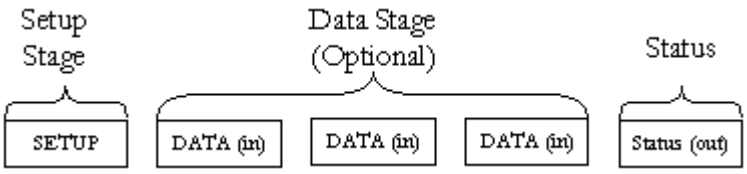
Řídící přenos – obousměrný, počítačem řízený přenos příkazů, stavových informací atd. Řídící přenos pokaždé začíná setup fází, následována libovolným počtem datových transakcí, které obsahují specifické informace pro požadovanou operaci. Směr přenosu v datové fázi je specifikován požadavkem v setup fázi. Nakonec stavová transakce vrátí stav hostovi. Směr stavové fáze je opačný od datové fáze. Existují 3 varianty řídícího přenosu:

1) **Control write** – host zapisuje data do zařízení



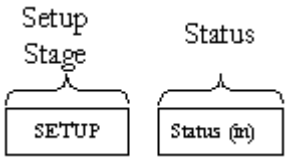
Obrázek 7 - Control write [10]

2) **Control read –** host chce přečíst data ze zařízení



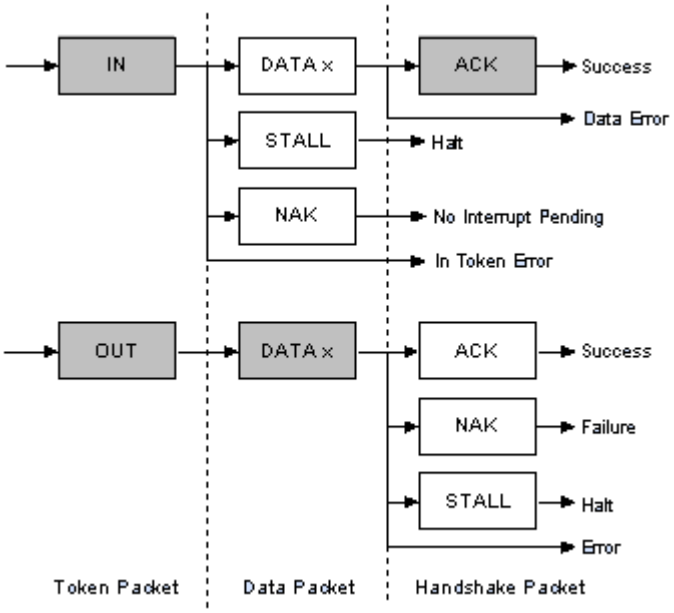
Obrázek 8 - Control read [10]

3) **No-data control –** řídící přenost bez přenosu dat



Obrázek 9 - No-data control [10]

Přerušení – je nejvíce využíváno zařízeními, které přijímají nebo odesílají data v asynchronním časovém rámci. Jedná se o jednosměrný přenos s garantovaným zpožděním a detekcí chyb. Maximální velikost dat pro low-speed zařízení je 8 bytů, pro full speed 64 bytů a pro high speed 1024 bytů.



Obrázek 10 - Formát transakce přerušení [8]

Přerušení je rozděleno na 2 směry:

1) **IN** – host periodicky vyzývá interrupt endpoint. Míra vyzývání je specifikována v endpoint deskriptoru. Při každé výzvě zasílá host IN token, na který mu zařízení odpoví datovým paketem, nebo v případě poškození tokenu bude token ignorovat.
2) **OUT** – pokud chce host zaslat zařízení interrupt data, odešle OUT token následovaný datovým paketem obsahující interrupt data. Pokud byl OUT token nebo data paket poškozen, zařízení je ignoruje. V opačném případě odešle odpověď.

Isochronní přenos – je používán při proudových přenosech (např. audio nebo video stream). Obsahuje detekci chyb pomocí CRC, ale negarantuje bezchybnost. Maximální velikost dat je specifikována v endpoint deskriptoru a může být 1023 bytů pro full-speed a 1024 bytů pro high-speed. Isochronní transakce nemají fázi handshake.

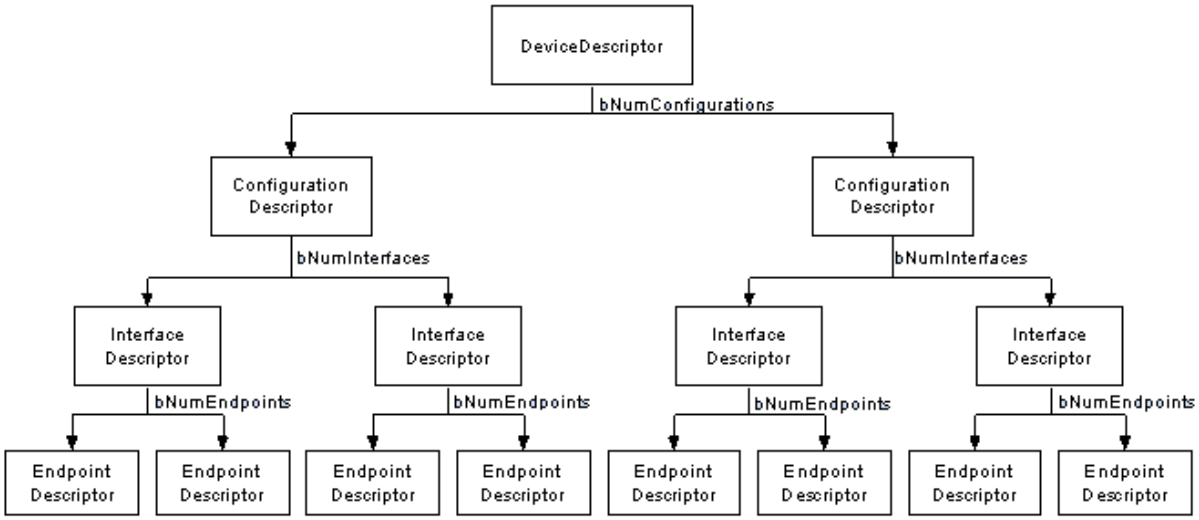
Dávkový přenos – je nejrychlejší ze všech přenosů. Dávkový přenos poskytuje detekci a opravu chyb pomocí CRC16, takže garantuje bezchybný přenos. Používá pouze šířku pásma sběrnice, která zbude po alokování ostatních transakcí. Dávkový přenos by měl být použít pouze pro časově nezávislou komunikaci, protože negarantuje zpoždění.

4.1.6 USB deskriptory

Všechny USB zařízení obsahují deskriptory, což jsou datové struktury s definovaným formátem, které hostu popisují např. o jaké zařízení se jedná, jakou verzi USB podporuje, počet endpointů atd. Nejčastější USB deskriptory jsou:

1) Device descriptor
2) Configuration descriptor
3) Interface descriptor
4) Endpoint descriptor
5) String descriptor

USB zařízení mohou mít pouze jeden **device descriptor**. Ten obsahuje informace o Product a Vendor identifikátorech, typ zařízení a počet konfigurací, který určuje počet configuration descriptorů.
Configuration descriptor specifikuje např. napětí, jaké daná konfigurace potřebuje, zda je napájení ze sběrnice a počet rozhraní. Při enumeraci si host přečte device descriptor a zvolí, která konfigurace bude použita. Aktivní může být jen jedna.
Interface descriptor popisuje endpointy pro jednotlivé funkce daného zařízení. Zařízení může mít v aktuálním čase povoleno více rozhraní.



Obrázek 11 - Hierarchie deskriptorů [8]

Device descriptor

Device descriptor reprezentuje celé USB zařízení. Tento deskriptor je pouze jeden, platí pro všechny konfigurace a specifikuje základní, ale důležité informace o zařízení: velikost deskriptoru, typ deskriptoru, verze USB, třídu USB, podtřídu USB, protokol USB, vendor ID, product ID, číslo zařízení, číslo výrobce a produkt, velikost paketu a počet konfigurací.

Configuration descriptor

USB zařízení může mít více rozdílných konfigurací. Host může zvolit konfiguraci zasláním příkazu SetConfiguration s číslem požadované konfigurace. Obsahuje: délku a typ deskriptoru, počet rozhraní, identifikační číslo konfigurace, popis konfigurace, atributy a maximální spotřebu.

Interface descriptor

Slouží pro popis endpointů kromě endpointu 0. Ten je nakonfigurován ještě dřív, než je vůbec nějaký deskriptor požadován. Host si z tohoto deskriptoru zjišťuje požadavky na přenosové pásmo. Informace: délka a typ deskriptoru, adresy endpointů, atributy endpointů, maximální velikost paketu, interval vyzývání

String descriptor

Nepovinný deskriptor, který slouží pro poskytnutí lehce čitelných informací. Tyto deskriptory začínají identifikačním číslem jazyka.

4.1.7 Enumerace

Enumerace je proces inicializace, kdy se nejprve detekuje připojení či odpojení zařízení. Každý hub má endpoint vyhrazený pro přerušení pro detekci tohoto jevu. Host je informován o připojených zařízeních a následně je vyzývá ke komunikaci. Po ustanovení komunikace si host vyžádá informace o zařízení a přidělí jim adresu.
USB automat má následující stavy

1) Powered: USB je pouze připojeno i napájeno.
2) Attached: USB zařízení je připojeno i napájeno, ale enumerace ještě nezačala.
3) Default: zařízení bylo resetováno hostem.
4) Address pending: host odešle nastavení adresy.
5) Addressed: adresa byla úspěšně přidělena.
6) Configured: zařízení přijalo a provedlo svoji konfiguraci
7) Suspend: zařízení je ve stavu suspend.

V případě připojení USB zařízení do napájeného portu, jsou spuštěny následující akce:

1. Hub informuje hosta o nastalé situaci. Zařízení je v Powered stavu a port, ke kterému je připojen, je zakázán.
2. Host zjistí dotazováním, o jakou situaci se jedná.
3. Poté, co se host dozví, na jaký port je zařízení připojeno, počká 100ms pro dokončení procesu připojení a ustálení napětí na zařízení. Poté je port povolen a je odeslán požadavek na reset.
4. Hub provede reset na daném portu. USB zařízení je teď v Default stavu a nemůže odebírat více než 100mA. Všechny registry byly vyresetovány a zařízení má výchozí adresu.
5. Host přidělí unikátní adresu zařízení. To se nyní nachází v Addressed stavu.
6. Host přečte device descriptor aby zjistil, jaká je maximální velikost přenosu přes rouru.
7. Host přečte deskriptor konfigurace.
8. Host přiřadí konfiguraci k zařízení, podle způsobu, jak se zařízení bude používat. Zařízení se nyní nachází v Configured stavu a jsou nakonfigurovány i všechny endpointy.

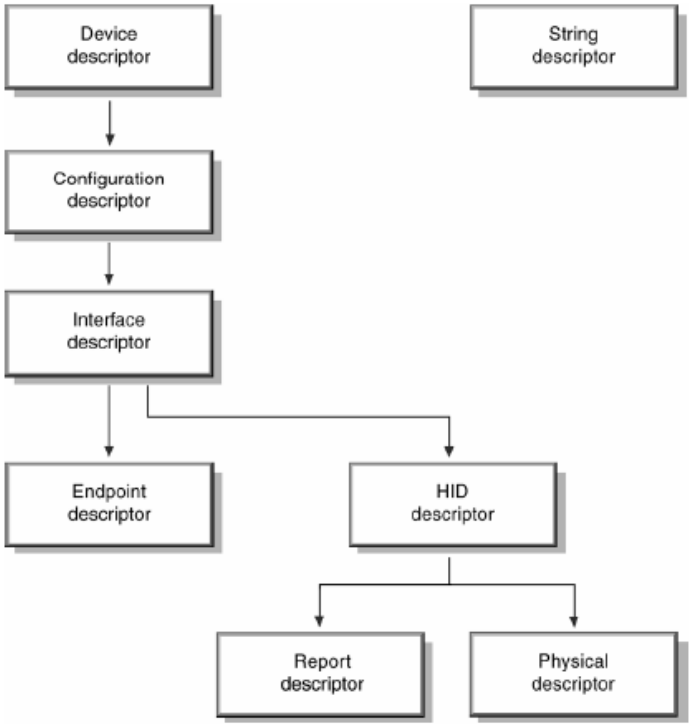
4.1.8 Třída HID

Do třídy HID patří zejména zařízení, které jsou používány uživateli pro ovládání počítačů. Příklady zařízení, které do této třídy patří:

- Klávesnice, joystick, myš
- Tlačítka, posouvače, přepínače
- Volant a pedály pro počítač, ovladače, různá herní a simulační zařízení
- Zařízení, které poskytují stejná data jako HID, např. čtečky kódu, voltmetry atd.

HID device descriptor identifikuje, které ostatní HID deskriptory jsou definovány, a ukazuje jejich velikost.

Report descriptor popisuje data generovaná zařízením a co znamenají. Report může například definovat pozici a stav tlačítka. Informace obsažené v reportu jsou použity pro směrování vstupu (poslat vstup na rozhraní myši nebo joysticku) a umožnují přiřadit funkcionalitu danému vstupu.
Fyzický deskriptor je volitelný a poskytuje informace o částech těla, sloužící pro ovládání zařízení.

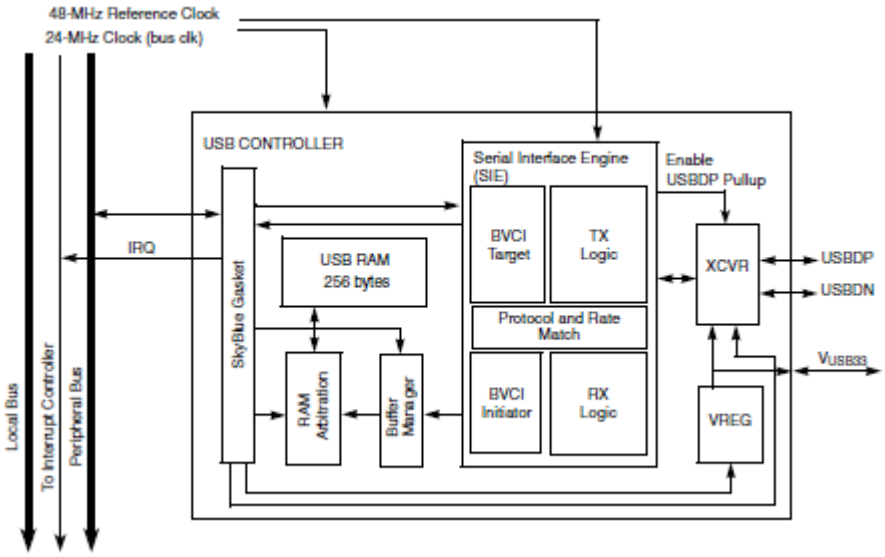


Obrázek 12 - Hierarchie HID deskriptorů [12]

HID zařízení komunikuje s ovladačem pomocí řídícího přenosu nebo přerušení. Řídící přenos se používá pro přijímání a odesílání požadavků, vysílání dat na výzvu HID ovladače a přijímání dat. Přerušení se používá pro asynchronní (nevyžádaná) data ze zařízení a odesílaní dat do zařízení.

4.2 Zvolená platforma

Mikrokontrolér MC9S08JM60 obsahuje modul, který je založen na USB 2.0. Poskytuje full-speed(12Mbit/s) a obsahuje napěťový regulátor 3.3V, seriál interface engine (SIE), 7 endpointů a 256 bytů RAM pro datový přenos.



Obrázek 13 - USB modul na MC9S08JM60 [7]

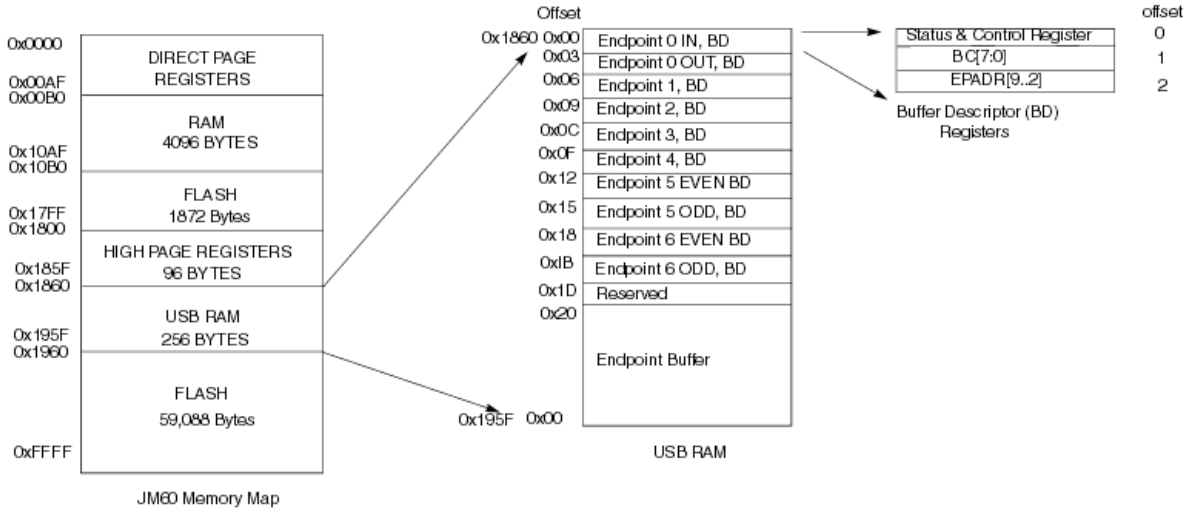
USB modul se, stejně jako klasické USB zařízení, dělí na 4 vrstvy.

- **Fyzická vrstva** zahrnuje data signalizaci, start of packet(SOP) a resume signály.
- **Protokolová vrstva** se nachází nad fyzickou. Veškeré transakce jsou řízeny touto vrstvou a probíhá zde i zpracování packetů.
- **Procesor příkazů a data report handler** obsahuje firmware. Enumerace, konfigurace, zpracování požadavků a protokolů se provádí zde.
- **Aplikační vrstva** je zaměřena na aplikace a komunikaci

Komunikace s USB modulem probíhá přes přenosy, složené z transakcí. Transakce jako data se ukládají na jednotlivé buffery endpointů, které jsou adresovány pomocí jejich adresy. Buffery se nachází na USB RAM.

4.2.1 USB RAM

USB RAM na MC9S08JM60 poskytuje 256 bytů pro buffer descriptor registry a buffery. Pokud je USB modul vypnutý nebo na RAM zbylo místo, může byt použita jako obecná RAM paměť.



Obrázek 14 - USB RAM a buffer descriptor table [7]

Buffer descriptor table se nachází na začátku USB RAM. Buffer deskriptory (BD) pro 7 endpointů jsou organizovány za sebou od EP0 IN po EP6 OUT. BDT celkem obsahuje 10 buffer deskriptorů dohromady o velikosti 30 bytů, které jsou naalokovány od 0x1860 do 0x187D.
První byte BD jsou **stavové a řídící registry**. Obsahují 4 bity, které CPU čte nebo zapisuje, aby umožnil USB komunikaci.

4.2.2 USB endpoint

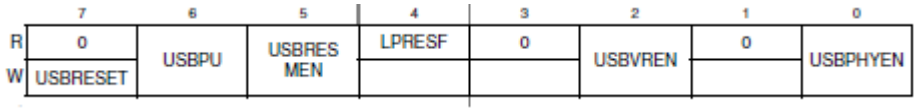
MC9S08JM60 má 7 endpointů, které se můžou nakonfigurovat pro komunikaci přes roury.

Endpoint 0 je obousměrný koncový bod, sloužící pro řídící přenos.
Endpointy 1-6 mohou být nakonfigurovány pro přenos jedním směrem.

Každý z endpointů 5 a 6 obsahují dva buffery. Jeden může být řízen CPU a druhý může sloužit pro komunikaci.
Libovolný endpoint může být povolen či zakázán v registru EPCTLn. V tomto registru lze nastavit i směr daného endpointu. Řídící přenos, bulk přenos, isochronní přenos a interrupt přenos jsou modulem podporovány.

4.2.3 Registry

USB Control Register 0



Obrázek 15 - USB Control Register 0 [7]

- USBRESET – pro generování resetu USB modulu
- USBPU – zjištění zdroje pullup rezistoru
- USBRESMEN – pokud je nastaven tento bit, umožní asynchronní probuzení modulu
- LPRESF – nastaven v suspend módu a je detekován resume signál
- USBVREN – povolení 3.3V USB napěťového regulátoru
- USBPHYEN – povolení či zakázání XCVR(USB přijímač a vysílač)

Interrupt Status Register

Obrázek 16 - Interrupt status register [7]

- STALLF – nastaven v případě odeslání STALL signálu
- RESUMEF – použit jako indikátor vzdáleného probuzení
- SLEEPF – pokud jsou detekovány 3 ms nečinnosti na sběrnici a modul vstupuje do suspend módu
- TOKDNEF – nastaven v případě dokončení aktuální transakce
- SOFTOKF – nastaven pokud modul obdrží SOF(start of frame) token
- ERRORF – v případě nastalé chyby v ERRSTAT registru
- USBRSTF – bit nastaven v případě USB resetu

Ostatní registry

Jednotlivé endpointy se ovládají pomocí EPTCLx registrů. Spolu s přerušením v INTSTAT a INTENB registrech se sledují i ERRSTAT a ERRENB registry pro zjištění chyb.

4.2.4 Serial interface engine (SIE)

USB SIE obsahuje logiku pro příjem a vysílání transakcí. Příjem a vysílání jsou řízeny pomocí protokolů, které spravují tok paketů z/do USB modulu.

SIE Transmitter logic

Má 2 základní funkce, a to správu formátu data paketů uložených v bufferu endpointu a odesílaní paketů.

Správa formátu paketů zahrnuje:

- NRZI kódování
- CRC
- pole SYNC
- pole EOP

Transmitter se také používá pro generování odpovědi na přijatý paket od hosta. V případě přijetí paketu, transmitter odpoví signály ACK, NAK nebo STALL.

SIE Receiver logic

Tato logika slouží pro příjem a uložení USB paketů. Pakety se ukládají do USB RAM a čekají na zpracování softwarem nebo CPU.
Zpracování paketů obsahuje následující operace:

- NRZI dekódování
- detekce synchronizace
- CRC
- detekce EOP
- identifikace podle čísla PID

SIE poskytuje i detekci chyb jako např. špatné CRC a timeout. Pokud je přijat paket ve správném formátu, je potvrzen hostovi. V případě chyby je však paket ignorován, tím vyprší timeout, a host paket znovu odešle.

4.3 Programové vybavení a OS

Jak Windows, tak Linux nabízejí jednoduché způsoby odchytávání komunikace na USB sběrnici. Zatímco Linux už má v sobě nějaké předinstalované, na Windows je potřeba si je stáhnout. Každý software pro zachytávání komunikace má své výhody a nevýhody, valná většina z nich však podporuje základní funkčnost, a to: sledování enumerace, informace o zařízení a deskriptorech atd.

4.3.1 Softwarové prostředky Windows

Wireshark – samostatně je implementován pouze pro zachycení síťové komunikace, s přídavnou knihovnou USBPcap však umožní zachytávat komunikace na USB sběrnici. Operační systém konvertuje USB pakety na síťové. Wireshark je možno použít i na Linuxu.

USBLyzer – nástroj pro monitorování provozu na USB sběrnici a analyzování aktivity USB zařízení. Kromě toho však umožňuje monitorování při připojení zařízení. Analyzér funguje jako ovladač a zachycuje, dekóduje a zobrazuje veškeré informace uživateli.

Další programy: SniffUSB, Portmon, QEMU, KVM, USBSnoop, USBView

4.3.2 Softwarové prostředky Linux

lsusb – nástroj pro zobrazování informací o USB sběrnicích a zařízeních.
usbview – zobrazí grafický přehled o připojených USB zařízeních. Umožnuje zobrazení pomocí stromové struktury.
usbfs – souborový systém USB. Jedná se se o dynamicky generovaný souborový systém a obsahuje informace o připojených zařízeních.

- /proc/bus/usb/devices – textový soubor, který obsahuje informace o zařízeních a jejich konfigurace, které jsou známé kernelu.

usbmon – část kernelu používaná pro sbírání dat o komunikaci na sběrnici.
Příklad spuštění:

```
mount –t debugfs none /sys/kernel/debug
modprobe usbmon
cat /sys/kernel/debug/usbmon/3t
```

Tato posloupnost příkazů zobrazí komunikaci na USB sběrnici 3, číslo odpovídá sběrnici v souboru /proc/bus/usb/devices
dmesg | grep usb - vypíše veškeré zprávy z kernelu, týkající se USB

4.3.3 Vlastní testovací aplikace

Testovací aplikace
Aplikace je naprogramována v jazyce C a využívá knihovny <*linux/usb.h*> práci s USB zařízením. Jedná se o modul pro systém Linux, který očekává zařízení s danými Vendor a Product identifikátory a v případě připojení takové zařízení vypíše jeho informace.

Instalace aplikace

1) Použitím Makefile se vytvoří modul "info.ko", který lze nahrát do systému Linux.
2) Příkazem "`sudo insmod info.ko`" se modul nahraje do systému
3) Při připojení zařízení už pro něj systém najde ovladač a provede tělo aplikace
4) Příkazem "`sudo rmmod info.ko`" lze modul opět odebrat.

Popis implementace aplikace
Aplikace ke správné funkčnosti musí znát Vendor_ID a Product_ID zařízení, které očekává. V případě připojení takového zařízeni si ho ve funkci `pen_init()` zaregistruje. Poté je proveden sběr informací o zařízení ve funkci `pen_probe()`. V této funkci zjistí aktuální rozhraní a poté se vypíšou informace o endpointech v tomto rozhraní.
V momentě kdy se zařízení odpojí, odregistruje ho a ukončí se.

4.3.4 Codewarrior

Codewarrior je integrované vývojové prostředí (IDE) pro operační systémy Windows, Linux, Mac a další. Podporovanými programovacími jazyky jsou C, C++ a assembler. Ovladače jsou na doporučení vyvíjeny ve verzi 6.3, nejnovější verzí pro Windows je však 10.6. Codewarrior pro Linux, vzhledem k malé žádanosti, přestal být podporován a momentálně poslední verzí je 10.2. Ovladače byly vyvíjeny v OS Windows 7 64bit, čímž nastal problém, protože codewarrior není kompatibilní s 64bitovými verzemi operačních systémů. Existuje několik možností, jak přesto codewarrior na 64bitových OS nainstalovat.

- Použít XP mód
- Nainstalovat Windows XP na virtuální počítače (VMWare/VirtualBox)

Poslední možností, která zároveň byla použita pro tento projekt, je odstranit kontrolu operačního systému před začátkem instalace. Jsou k tomu potřeba Codewarrior Special Edition V6.3 (soubor CW_MCU_V6_3_SE.exe) a speciální skript RemoveOSCheck.vbs. Pro nainstalování codewarrioru je potřeba držet se následujících kroků.

1. Rozbalit instalátor Codewarrior
2. Spustit skript RemoveOSCheck.vbs s parametrem CW_MCUs_V6_3.msi. Skript i soubor je potřeba mít ve stejné složce.
3. Spustit soubor setup.exe v rozbalené složce.

Po nainstalování však Codewarrior nebude detekovat žádná připojená zařízení. Je to způsobeno tím, že knihovny CW neumí komunikovat s 64 bitovými ovladači. Pro detekování připojených zařízení je nutno následující:

1. Stáhnout a nainstalovat PE Micro Multilink drivers v11 (http://www.pemicro.com/downloads/download_file.cfm?download_id=301)
2. Stáhnout přiložený soubor wdapi1110_32.dll
3. Vložit soubor do složky Freescale\CodeWarrior for Microcontrollers V6.3\prog
4. Přejmenovat soubor na wdapi900.dll
5. Stáhnout a nainstalovat aktualizaci (http://www.pemicro.com/faqs/faq_view.cfm?id=211)

5 Realizace zařízení a jejich ovladačů

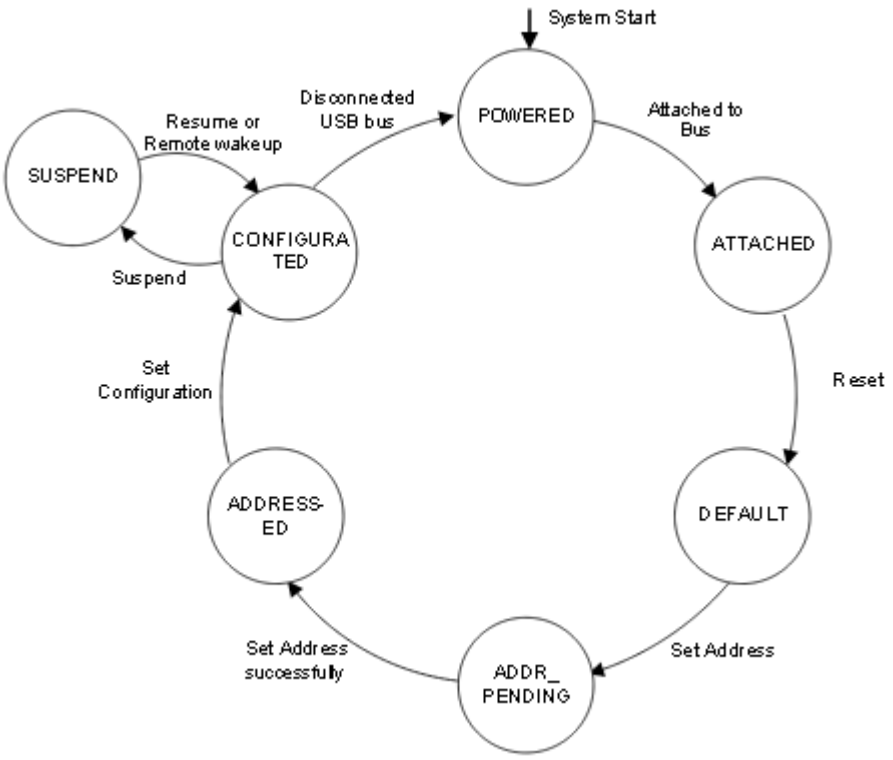
5.1 Princip činnosti zařízení

5.1.1 Nastavení rour a enumerace

USB zařízení musí na začátek nastavit alespoň jednu řídící rouru. Řídící roura se nachází na endpointu 0, využívá řídící přenos a podporuje komunikaci oběma směry. Tato řídící roura je používána softwarem pro identifikaci a konfiguraci zařízení.

Enumerace – proces identifikace a konfigurace USB zařízení. Tento proces je podobný pro všechny USB zařízení. Veškerá komunikace pří enumeraci probíhá přes endpoint 0.
USB automat má následující stavy

1) Powered: USB je pouze připojeno i napájeno.
2) Attached: USB zařízení je připojeno i napájeno, ale enumerace ještě nezačala.
3) Default: zařízení bylo resetováno hostem.
4) Address pending: host odešle nastavení adresy.
5) Addressed: adresa byla úspěšně přidělena.
6) Configured: zařízení přijalo a provedlo svoji konfiguraci
7) Suspend: zařízení je ve stavu suspend.



Obrázek 17 - USB automat [5]

Funkce aplikace – následuje popis kroků inicializace MCU a USB modulu.

Inicializace MCU - jako první krok aplikace provede inicializace MCU a MCG modulu pro generování hodinových signálů.

Inicializace USB – na začátku se zařízení nachází ve stavu POWERED_STATE. Nejdřív je proveden reset modulu, následně se inicializují EP0 v obou směrech a povolí se v registrech.

Zpracování přerušení - aplikace zpracovává přerušení typu 7(USB), a podle nastavení registrů se vykoná akce.

Transakce – podle aktuálního stavu automatu jsou vykonány akce, které nakonfigurují zařízení.

Setup – v případě přijetí setup transakce se pomocí automatu obslouží příslušný požadavek

5.1.2 Řídící přenos

Datový paket pro řídící přenos začíná na DATA0 a překlápí se v setup a datové fázi přenosu. Podle směru přenosu v různých fázích se rozlišují 3 stavy:

1) Je očekáván setup token: WAIT_SETUP_TOKEN
2) Budou se odesílat data: CTL_TRF_DATA_TX
3) Příjem dat: CTL_TRF_DATA_RX

V případě dokončení transakce se nastaví TOKDNEF bit. Pokud se transakce objevila na endpointu 0, je zkontrolován směr transakce. Při směru IN se ověří, zda se jedná o SETUP token. Pokud je požadavek neznámý, je nastaven STALL bit v EPCTL registru a modul se připraví na další přenos.

5.1.3 Standardní požadavky

Požadavky zasílané hostem jsou přenášeny v setup fázi. Každý požadavek začíná 8 bytů dlouhým setup paketem, který má následující formát

Pole	Počet bytů	Popis
bmRequestType	1	Určuje směr přenosu, typ a příjemce
bRequest	1	Hodnota požadavku
wValue	2	Hodnota
wIndex	2	Index
wLength	2	Počet bytů pro přenos v datové fázi

Tabulka 3 - Formát požadavku

V každém zařízení musí být implementovány tyto požadavky

bmRequestType	bRequest	Data
0x80	GET_STATUS(0x00)	Device status
0x00	CLEAR_FEATURE(0x01)	-
0x00	SET_FEATURE(0x03)	-
0x00	SET_ADDRESS(0x05)	-
0x80	GET_DESCRIPTOR(0x06)	Descriptor
0x00	GET_DESCRIPTOR(0x07)	Descriptor
0x80	GET_CONFIGURATION(0x08)	Configuration value
0x00	SET_CONFIGURATION(0x09)	-

Tabulka 4 - Standardní požadavky

5.1.4 Zpracování příkazů

V případě, že USB modul příjme nebo odešle data, zvolí následující postup:

1. Vypočítá si adresu v BDT na základě čísla endpointu, směru přenosu, zvoleného bufferu a poté přečte buffer descriptor.
2. Po přečtení dat a v případě, že má USB modul výhradní přístup k buffer descriptoru, SIE přenese data z/do bufferu, na který ukazuje adresa. Po aktualizaci BDT předá USB modul vlastnictví bufferu SIE.
3. Aktualizuje se STAT registr a nastaví se TOKDNE přerušení. Poté co mikrokontrolér zpracuje přerušení, přečte si stavový registr a informuje se jak zpracovat endpoint. Následuje další USB přenos.

5.1.5 Zpracování paketů a datových rour

Zpracování paketů se skládá z řízení bufferů pro IN a OUT transakce. Stejně tak zpracování datových rour je v podstatě řízení bufferů. Firmware je zodpovědný za řízení buffer RAM tak, aby byl buffer vždy přístupný pro modul, když jej potřebuje. Zařízení alokuje buffer ve sdílené RAM, nastaví buffer descriptor a čeká na přerušení. Po příchozím přerušení se zjistí ze stavového registru, který endpoint byl zvolen, a přečte se BDT záznam jaká akce se následně provede. Při zpracování datových paketů je firmware zodpovědný za kontrolu velikosti paketů, aby odpovídaly USB specifikaci, a fyzických omezení modulu.

5.1.6 Zpracování požadavků

Většina příkazů na USB zařízení je směrováno na endpoint 0. Host použije standardní požadavky pro enumeraci a konfiguraci zařízení. Ovladač specifikuje třídu zařízení.

USB požadavky vždy odpovídají specifickému formátu:

- Host odešle setup token, následován 8bytovým setup paketem.
- Následuje volitelná datový fáze, kde se může přenést až 64KB dat mezi hostem a zařízením.
- Požadavek je považován za zpracovaný po stavové fázi.

Firmware sleduje stavové registry, endpoint 0 a setup paket, aby správně vykonal hostovy požadavky. Zpracování požadavků na endpoint 0 se řídí daným postupem:

1. Alokuje se 8 bytů pro endpoint 0 OUT.
2. Vytvoří se záznam v tabulce BDT pro endpoint 0 OUT a předá se přístup k BD USB modulu.
3. Čeká se na přerušení.
4. Přečte se stavový registr. Registr musí ukazovat na endpoint 0 a příjem.
5. Přečte se buffer descriptor endpointu 0. Typ tokenu musí být setup token.
6. Zpracuje se setup paket. Zde záleží na směru přenosu, jaký setup paket určuje. V případě, že se jedná o OUT přenos, zpracují se data o velikosti specifikované v wLength poli. Pokud se jedná o IN přenos, odesílaná data budou mít maximální velikost specifikovanou ve wLength poli.
7. Po zpracování datové fáze se vytvoří stavová transakce.

5.1.7 Stavy zařízení

Suspend

USB host může dát zařízení nebo celou sběrnici do suspend stavu. Do tohoto stavu zařízení vstoupí po nečinnosti delší než 3ms. Vstup do suspend stavu je ohlášen nastavením příslušného bitu ve stavovém registru. Zařízení v suspend stavu je povinno odebírat méně než 500μA. Po 3ms nečinnosti zařízení tohoto nízkého odběru dosáhne do 7 ms. USB modul nemůže vstoupit do suspend módu pokud host odesílá start-of-frame pakety.

Resume

Po přijetí resume signálu, USB modul generuje asynchronní přerušení, probudí zařízení a hodiny ze stop módu. Detekce resume signálu je aktivována nepřetržitě, i když jsou vypnuty hodiny. Existují 3 způsoby jak probudit modul ze suspend módu.

Host initiated resume – host musí signalizovat resume alespoň 20ms následovaný EOP signálem. Délka signálu zajistí probuzení všech zařízení. Po probuzení je potřeba, aby byl obnoven provoz na sběrnici během 3ms jinak zařízení opět vstoupí do suspend módu.

Reset signál – reset probudí zařízení ze suspend módu.

Remote wakeup – USB zařízení může odeslat hostovi resume signál nastavením příslušného bitu. Firmware musí bit nastavit na 1 na určitou dobu a poté vynulovat.

Reset

Modul podporuje více způsobů resetu, buď reset sběrnice od hosta nebo reset modulu od MCU.
Reset sběrnice – reset sběrnice může host provést v jakýkoliv čas. Tento reset je definován jako signál single ended zero (SE0) generovaný alespoň 2,5 μs. Když zařízení detekuje reset signál, resetuje se do nekonfigurovaného stavu a nastaví adresu USB na nulu. Tento reset je využíván pro přechod zařízení do známého stavu před enumerací. USB modul odpoví na reset signál nastavením příslušeného bitu ve stavovém registru. Software je povinen přerušení obsloužit pro správné fungování USB.
Reset modulu – v průběhu resetu modulu je modul nakonfigurován do defaultního stavu. Reset modulu může být vnucen i nastavením USBRESET bitu v USBCTL0 registru.
Defaultní stav nastaví:

- Maskování přerušení
- USB hodiny povoleny
- USB regulátor napětí zakázán
- USB přijímač a vysílač zakázán
- USBDP pullup zakázán
- Endpointy zakázány
- USB adresa nastavena na nulu

5.2 Implementace

USB ovladač pro mikrokontrolér je napsán v jazyce C. Zpracovává standardní požadavky popsané v kapitole 5.3. Kromě toho implementuje proces enumerace, reset a odesílání deskriptorů.

5.2.1 Inicializace zařízení

Ze všeho nejdřív je potřeba zařízení po připojení ke sběrnici inicializovat. Inicializuje se ve dvou krocích. V prvním kroku se inicializují hodiny a MCU. USB modul vyžaduje dva zdroje hodin (24 MHz a 48MHz). Hodiny jsou implementovány ve funkci `MCG_Init()`. Následně se inicializuje MCU ve funkci `MCU_Init()`, nastavením příslušných bitů v registrech Systém Options a Systém Power Management Status and Control. Tím se docílí, aby byla povolena instrukce STOP pro přechod do režimu nízké spotřeby.
Dalším krokem je inicializace USB modulu. Zařízení je připojeno na sběrnici a napájeno, proto se nachází ve stavu POWERED. Poté je vynulována konfigurace pro případ, že byla nějaká aktivní a nastaví se bit pro reset v USBCTL0 registru:

```
USBCTL0_USBRESET = 1
```

Zařízení čeká, než proběhne reset a poté inicializuje buffer deskriptory pro endpoint 0 v obou směrech:
EP0 OUT:

```
Bdtmap.ep0Bo.Stat._byte = _SIE|_DATA0|_DTS;
Bdtmap.ep0Bo.Cnt = EP0_BUFF_SIZE;
Bdtmap.ep0Bo.Addr = 0x0C;
```

EP0 IN:

```
Bdtmap.ep0Bi.Stat._byte = _CPU;
Bdtmap.ep0Bi.Cnt = 0x00;
Bdtmap.ep0Bi.Addr = 0x08;
```

Po nastavení EP0 je potřeba jej povolit v registru a povolit přerušení

```
EPCTL0 = 0x0D
INTENB = 0x01
```

Nyní se zařízení nachází v ATTACHED stavu. Program se nyní nachází v hlavní smyčce. Je to nekonečná smyčka, kontrolující v jakém stavu se modul nachází a podle toho určí následující akce. Po připojení nebo resetu nastavuje registry pro povolení endpointu 0 pro příchozí a odchozí komunikaci, maskuje přerušení a překlápí USB automat. Pokud nastala žádost na přechod do režimu nízké spotřeby, spustí funkci pro zpracování tohoto stavu.

Zpracování přerušení

Ovladač je řízen přerušeními. Podle nastavených bitů v registru INTSTAT je vykonána příslušná akce. Při události přerušení se spustí funkce `USB_ISR()`, přečte se INTSTAT registr a spustí se funkce dle vyvolaného přerušení.
Implementovány jsou obsluhy přerušení u těchto příznaků:

a) **Low-power resume příznak** (USBCTL0_LPRESF a suspend mód) – byl detekován stav signalizující resume, když je zařízení v low-power stop3 módu. Tento příznak je spuštěn asynchronním přerušením. Pro vynulování toho příznaku je potřeba vynulovat USBCTL0_USBRESMEN bit.

b) **Resume příznak** (INTSTAT_RESUMEF a INTENB_RESUME) – probuzení a obnova aktivity. Nejdřív je signalizován resume a poté je vypnuto resume přerušení. Nakonec je v registru CTL vynulován bit TSUSPEND, čímž se opět povolí přijímaní a odesílaní transakcí.

c) **Reset** (INTSTAT_USBRSTF a INTENB_USBRST) – reset je obsloužen funkcí `USB_Reset()`. V prvním kroku se odstraní příznaky v registrech ERRSTAT a INSTAT, povolí se přerušení v registrech ERRENB a INTENB (kromě resume přerušení), poté se vymaže adresa a nastaví se na nulovou. Veškerá nastavení endpointů kromě EP0 jsou vymazána vynulováním kontrolních endpoint registrů EPCTLx (pro x 1-6). Pro endpoint 0 je nastaven řídící přenos a povolen handshake. Následně jsou pro všechny čekající, neobsloužené transakce nastaven příznak INSTAT_TOKDNEF, který zruší všechny transakce jejich nastavením jako kompletní. Jediný typ paketu, který je možné po resetu přijmout je setup, proto se nastaví endpoint 0 pro příjem, povolí se transakce a automat

zpracování přenosů je nastaven na WAIT_SETUP_TOKEN. Nakonec je zakázán remote wake-up, nastavení aktuální konfigurace je vymazáno a zařízení se nachází v DEFAULT stavu.

d) **SOF paket** (INTSTAT_SOFTOKF a INTENB_SOFTOK) – po přijetí Start-of-frame paketu následuje pouze potvrzení jeho přijetí.

e) **STALL handshake** (INTSTAT_STALLF a INTENB_STALL) **–** v případě že SIE zaslalo STALL paket, ověří se, že paket byl přijat na endpointu 0. Poté se endpoint 0 připraví pro další přenos a automat zpracování přenosů je nastaven na WAIT_SETUP_TOKEN. Nakonec je potvrzen STALL handshake.

f) **Suspend –** v případě zjištění neaktivity na sběrnici delší než 3ms, zařízení přejde do SUSPEND stavu a USB automat se nachází ve stavu USB_SUSPEND. V SUSPEND stavu se bude zařízení nacházet dokud nepřijde signál probuzení. Tento signál se neustále testuje ve funkci `USB_WakeUp()`, ve které se zjišťuje i zdroj probuzení. Probuzení může být způsobeno buď "remote wake-up" signálem nebo "resume" signálem. Při remote wake-up signálu, který se zpracovává ve funkci `USB_Remote_Wakeup()`, se nejdřív nastaví detekce resume příznaku v registru INTSTAT a zakáže se další resume přerušení příznakem INTENB_RESUME. Poté proběhne kontrola, zda byl dostatečně dlouhý (musí mít délku mezi 1-15ms). V případě resume signálu se počká na odeznění přechodového jevu a zařízení se vrací do normálního stavu.

g) **Dokončení transakce** (INTSTAT_TOKDNEF a INTENB_TOKDNE) **–** po přijetí příznaku dokončení tokenu je spuštěna funkce `USB_transaction()`. Podrobnější popis zpracování transakce níže.

Dokončení transakce

Zpracování transakcí probíhá ve funkci `USB_Transaction()`. Nejdříve se ze stavového registru STAT zjistí, o jaký endpoint se jedná. Pokud se jedná o endpoint 0, zjišťuje se směr transakce.
OUT – Přijatá transakce se může zpracovat 2 způsoby. Buď se jedná o setup (popis zpracování v 5.2.2 Setup) nebo jde o data. Data jsou zpracována, jen když se automat zpracování přenosu nachází ve stavu CTL_TRF_DATA_RX. V tomto stavu se nejdřív uloží velikost dat, poté čítač a nakonec data. Pak už se jen nastaví inicializační parametry stavového registru buffer descriptoru a určí se data toggle synchronizace.
IN – Zde záleží na stavu USB automatu. Jako první je tedy nutné testovat, zda se automat nenachází ve stavu ADD_PENDING_STATE. Tento stav totiž určuje, že v příští transakci je očekávána adresa. Pokud je očekávána, nastaví se adresa ze setup paketu a USB automat se překlopí do stavu ADDRESSED_STATE. Následuje přenos dat. To se děje ve funkci `USB_control_INdata()`. Nejdřív je potřeba otestovat, zda velikost dat nepřesahuje velikost bufferu. Pokud velikost přesahuje, jsou uložena data pouze o maximální velikosti bufferu.

Příprava na další příjem a odesílání
Ať už se jednalo o IN nebo OUT transakci, připraví se EP0 na další.

```
Bdtmap.ep0Bo.Cnt = EP0_BUFF_SIZE
Bdtmap.ep0Bo.Addr = 0x0C
Bdtmap.ep0Bo.Stat._byte = _SIE|_DATA0|_DTS
Bdtmap.ep0Bi.Stat._byte = _CPU
```

Deskriptory jsou struktury (popsány v kap. 4.1.6), ve kterých jsou uložené veškeré informace o zařízení. Všechny struktury obsahují na prvním místě informaci o svojí délce a typu.

Device descriptor je uložen ve struktuře `Device_Dsc`.
Configuration, interface a endpoint descriptor se nachází ve struktuře `Cfg_01`. Configuration descriptor je definován pouze jeden. Konfigurace se ukládá do proměnné `Usb_Active_Cfg` a aktivní může být pouze jedna. Interface descriptor je definován také jen jeden. Rozhraní se ukládají do pole `Usb_Alt_Intf[]`, které může obsahovat více rozhraní. Následují deskriptory endpointů a ty jsou definovány čtyři.
String descriptor je uložen ve struktuře `Str_Des`. Tato struktura obsahuje celkem tři string deskriptory uložené ve strukturách sd000, sd001 a sd002, které obsahují název výrobce zařízení a ovladače.

5.2.2 Ovladač

Řídící přenosy – jsou zpracovávány automaticky společně se standardními požadavky. Stavy, podle kterých se řídí jednoduchý konečný automat:

WAIT_SETUP_TOKEN – jako následující transakce je očekáván setup, jehož zpracování se provede ve funkci `USB_control_setup()`.
CTL_TRF_DATA_TX – odeslání dat hostovi v následujícím přenosu.
CTL_TRF_DATA_RX – příjem dat od hosta

Pokud automat zpracování tokenu ve stavu, kdy se očekává setup token, spustí se obsluha tohoto tokenu.

Struktura setup tokenu a jeho obsah:

`byte bmRequestType` – směr, typ a příjemce dat
`byte bRequest` - požadavek
`word wValue` - hodnota
`word wIndex` - index
`word wLength` – délka dat v datové fázi

Nejdřív se zkontroluje typ požadavku, čímž se určí, který algoritmus setup paket obslouží. Pokud se jedná o standardní požadavek, je očekáván některý ze standardních požadavků uvedených v tabulce (kapitola 5.1.3).

- **Get_descriptor** – požadavek je implementován ve funkci `USB_get_descritptor()`. Podle hodnoty proměnné `bDscType`, která určuje typ deskriptoru, je do bufferu vložen daný deskriptor.
- **Set_configuration** – nejdříve jsou vymazány nastavení všech endpointů kromě 0 vymazáním jejich kontrolních registrů. Vymazáno je i pole rozhraní. Následně je uložena požadovaná konfigurace jako aktivní a USB automat se nachází ve stavu CONFIGURED_STATE. Pokud žádná konfigurace nebyla zvolena, automat se nachází ve stavu ADDRESSED_STATE.
- **Set_address** – požadavek na nastavení adresy pouze překlopí automat do stavu ADDR_PENDING_STATE. Nastavení adresy proběhne ve zpracování transakce.
- **Get_configuration** – požadavek na konfiguraci uloží do bufferu aktuální konfiguraci, která se nachází v `Usb_Active_Cfg`.
- **Get_status** – tento požadavek rozlišuje příjemce a podle toho zasílá odpověď. Pokud je příjemce zařízení, uvede informace o příznacích self-power a remote wakeup. Pro příjemce endpoint je odeslán stall bit.
- **Set_feature –** nastavení vlastností podle příjemce a typu vlastnosti. Pro zařízení je nastaveno povolení nebo zakázání remote wakeup. Pro endpoint povolí nebo zakáže stall.
- **Get_interface –** nejdřív je zkontrolováno, zda rozhraní, o které je žádáno, vůbec existuje (číslo rozhraní je menší než maximální počet rozhraní). Pokud existuje, je uloženo do bufferu.
- **Set_interface –** aktuální rozhraní je nastaveno na požadované.
- **Set_description –** neimplementováno.

Po zpracování požadavků následuje funkce `USB_control_end()`. Tato funkce dokončuje odesílání a příjem nastavením endpointu 0. Způsob nastavení a dokončení přenosu závisí na typu požadavků, které byly obsluhovány. Pro standardní požadavky se určí, zda se jednalo o control read či control write. V případě control read se uloží data do bufferu, automat zpracování přenosu se nastaví na CTL_TRF_DATA_TX a endpoint 0 se nastaví podle následujícího:

```
Bdtmap.ep0Bo.Cnt = EP0_BUFF_SIZE;
Bdtmap.ep0Bo.Addr = 0x0C;
Bdtmap.ep0Bo.Stat._byte = _SIE;
Bdtmap.ep0Bi.Addr = 0x08;
Bdtmap.ep0Bi.Stat._byte = _SIE|_DATA1|_DTS;
```

Control write pouze překlopí automat do stavu CTL_TRF_DATA_RX a nastaví vlastnosti endpointu 0 na:

```
Bdtmap.ep0Bi.Cnt = 0;
Bdtmap.ep0Bi.Addr = 0x08;
Bdtmap.ep0Bi.Stat._byte = _SIE|_DATA1|_DTS;
Bdtmap.ep0Bo.Cnt = EP0_BUFF_SIZE;
Bdtmap.ep0Bo.Addr = 0x0C;
Bdtmap.ep0Bo.Stat._byte = _SIE|_DATA1|_DTS;
```

Nakonec se pouze povolí přijímání a odesílání transakcí.

5.2.3 USB klíč

USB klíč je zařízení třídy HID, i když nevyžaduje přítomnost ani nekomunikuje s uživatelem. Do této třídy se řadí kvůli podobné komunikaci zařízení s hostem a zpracováním požadavků. Stejně jako ostatní USB zařízení musí umět zpracovat standardní požadavky, avšak jako zařízení třídy HID musí umět zpracovat i další požadavky specifické pro tuto třídu. Jako klíč je zařízení považováno, protože obsahuje řetězec, který se chová jako heslo. Určitá aplikace tedy může být naprogramována tak, že nebude reagovat, dokud v USB portu nebude připojeno zařízení s požadovaným heslem.

Jak už bylo uvedeno i USB klíč musí zpracovat standardní požadavky. Děje se tomu tak ve funkci `USB_control_setup()`. Ovšem v této funkci je nyní implementováno i zpracování požadavků třídy HID. V případě, že byl zadán požadavek a program zjistí, že se nejedná o standardní, spustí se procedura `USB_classReq_handler()`. Procedura má za úkol zjistit, o jaký požadavek se jedná a podle toho odešle odpověď.

Odpovědi na požadavky jsou následující:

- **Get_descriptor –** třída HID obsahuje rozšířenou verzi požadavku, kdy je nutné implementovat další deskriptory. Jak už je naznačeno v hierarchii HID deskriptorů (kap. 4.1.8), jsou jimi HID descriptor a report descriptor.
    - **HID_descriptor –** tento požadavek vrací část konfiguračního deskriptoru, protože se v něm nachází HID descriptor.
    - **Report_descriptor –** vrací HID report a jeho velikost. HID report pro USB klíč obsahuje pole bytů, popisující data pakety.
- **Get_interface –** odešle číslo právě aktivního rozhraní. Vzhledem k tomu, že implementováno je pouze jedno, vždy vrátí pouze nulté rozhraní.
- **Get_protocol/Set_protocol –** vrací číslo právě aktivního protokolu, popř. nastaví právě aktivní protokol. Neimplementováno.
- **Get_report –** požadavek obsluhující odeslání klíče. Po příchodu tohoto požadavku je klíč uložen do bufferu a odeslán aby mohl být zkontrolován žádající aplikací.

5.2.4 Inicializace endpointů

Endpointy jiné než 0 mohou být potřebné pro přenos různých dat. V tomto projektu jsou inicializovány 4 endpointy kromě endpointu 0, dva pro směr IN a dva pro směr OUT. Nastavení směru probíhá v EPCTLx registrech. Pro směr OUT je nutno zadat i velikost bufferu endpointu. Následně je uložena adresa endpointu, která se odvíjí od velikosti bufferu.

Název endpointu	EPCTLx nastavení	Adresa	Velikost bufferu	Stavový registr
EP1	OUT	0x08	16	SIE | DATA0 | DTS
EP2	IN	0x0C		CPU | DATA1
EP3	OUT	0x10	32	SIE | DATA0 | DTS
EP4	IN	0x18		CPU | DATA1

Tabulka 5 - Inicializace endpointů 1 až 4

Endpointy 1 a 3 jsou tedy OUT, 2 a 4 jsou IN. Všechny endpointy mají povolen v EPCTLx registrech handshake paket. Endpointy 1 a 3 vlastní SIE (seriál interface engine) a je na nich očekáván datový paket DATA0. Buffery endpointů 2 a 4 vlastní CPU a příští datový paket je očekáván DATA1.

5.3 Struktura ovladače

Usb_Ep0_Handler.c – zpracování řídících přenosů a standardních požadavků
Usb_Drv.c – inicializace modulu, zpracování přerušení
Usb_Descriptor.c – deskriptory
MCU.c – inicializace a nastavení hodin
Usb_Bdt.h – buffer descriptor table
Usb_Config.h – stavy USB automatu
Usb_Descriptor.h, Usb_Drv.h, Usb_Ep0_handler.h – pomocné datové struktury

Různé pomocné datové struktury a deskriptory byly převzaty z oficiální dokumentace Freescale [5].

6 Závěr

Cílem práce byl návrh a implementace ovladače pro různé operační systémy. Kromě ovladače práce obsahuje i pár testovacích programů, kterými je možné ověřit správnou funkčnost. Ovladač je použitelný pro vývoj vestavěných aplikací. Tato funkčnost je rozšířena o USB klíč, který slouží pro ověření zařízení a zajišťuje tak větší bezpečnost. Kromě ovladače byla implementována i inicializace ostatních endpointů a tak je možné ovladač použít pro více funkcí. Ovladač byl vyvíjen v operačním systému Windows 7, ale v práci je obsažen i testovací modul pro Linux, který ověří správnou inicializaci zařízení a nastavení deskriptorů.

Do budoucna může práce obsahovat více příkladů použití, jelikož byla vyvíjena pro kit DEMOJM, který má mnoho funkcí a ovladač je vyvíjen pouze pro USB modul. Jako rozšíření je možné implementovat myš díky vestavěnému akcelerometru, nebo úložiště dat. V případě zaměření na modul pro operační systém Linux, implementoval by ovladač více konfigurací popř. rozhraní, mezi kterými by modul přepínal.

Implementovaný ovladač je použitelný pouze pro výukový kit, avšak jeho vývojem je možné zjistit, jak ovladače fungují, jakou mají roli a jak postupovat při budoucím vývoji ovladačů pro jiná zařízení. Modul pro Linux se dá použít pro jakékoliv zařízení, stačí jen znát Vendor ID a Product ID. Implementovaný modul je pouze informační, knihovna usb.h však umožňuje rozsáhlé rozšíření funkčnosti modulu. Kromě funkcí získávajících informace o zařízení, obsahuje i funkce pro řízení, a tak je možné například měnit konfigurace a rozhraní, resetovat zařízení, posílat signály typu resume a suspend apod.

7 Literatura

[1] J. Corbet, A. Rubini a G. Kroah-Hartman, „Linux Device Drivers, Third Edition," 27 Leden 2005. [Online]. Dostupné z: http://lwn.net/Kernel/LDD3/.

[2] „Writing device drivers in Linux: A brief tutorial," 24 Prosinec 2006. [Online]. Dostupné z: http://www.freesoftwaremagazine.com/articles/drivers_linux.

[3] G. Kroah-Hartman, „How to Write a Linux USB Device Driver," Linux Journal, 1 Říjen 2001. [Online]. Dostupné z: http://www.linuxjournal.com/article/4786.

[4] M. Opdenacker, „Linux USB drivers," 15 Září 2009. [Online]. Dostupné z: http://free-electrons.com/doc/linux-usb.pdf.

[5] Liu, Derek; Freescale Semiconductor Inc., „USB Device Development with the MC9S08JM60," Červenec 2008. [Online]. Dostupné z: http://cache.freescale.com/files/microcontrollers/doc/app_note/AN3560.pdf.

[6] Freescale Semiconductor Inc., „Freescale USB Device Stack," Květen 2012. [Online]. Dostupné z: http://cache.freescale.com/files/microcontrollers/doc/user_guide/USBUG.pdf?fasp=1.

[7] Freescale Semicondictor Inc., „MC9S08JM60 MC9S08JM32 Data Sheet," Říjen 2014. [Online]. Dostupné z: http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC9S08JM60.pdf.

[8] C. Peacock, „USB in a NutShell," Beyond Logic, 17 Září 2010. [Online]. Dostupné z: http://www.beyondlogic.org/usbnutshell/usb1.shtml.

[9] „Developing Windows client drivers for USB devices," Windows Dev Center, [Online]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/hardware/hh406260%28v=vs.85%29.aspx.

[10] Jungo Ltd., „USB v8.11 User's Manual" 2006. [Online]. Dostupné z: http://www.jungo.com/st/support/documentation/windriver/

[11] M. Tsegaye, R. Foss, „A comprasion of the Linux and Windows Device Driver Architecture" [Online]. Dostupné z: http://www.cs.ru.ac.za/research/oscomposr.pdf

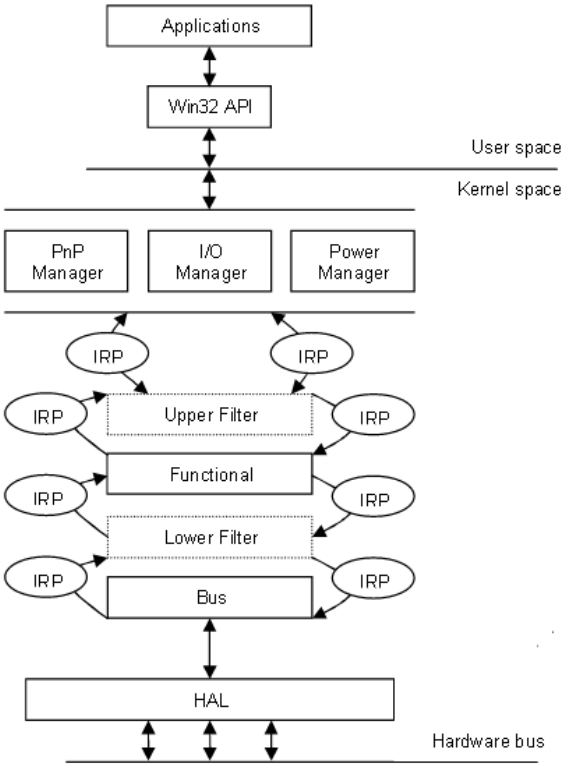
[12] USB Implementers' forum, „Device Class Definition for Human Interface Devices" 2001. [Online]. Dostupné z: http://www.usb.org/developers/hidpage/HID1_11.pdf

Přílohy

1 Návod k tvorbě ovladačů

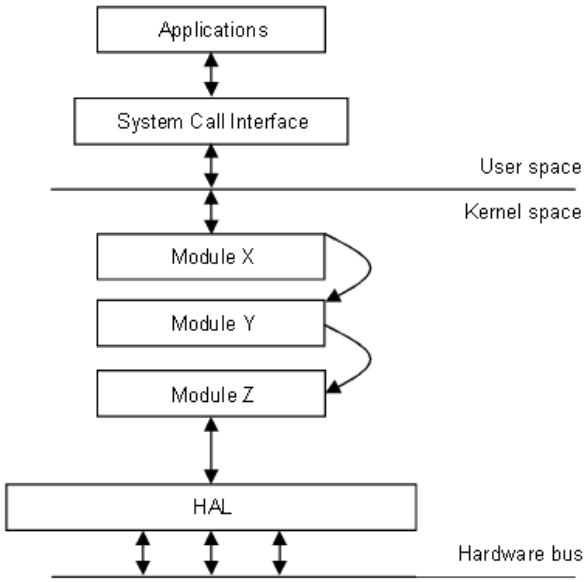
Tento návod shrnuje potřebné předpoklady pro tvorbu ovladačů pro systémy Windows a Linux. Nejdříve jsou shrnuty rozdíly v tvorbě ovladačů pro oba systémy a následně je uvedena posloupnost činností pro její tvorbu.

Porovnání architektury Linux a Windows ovladačů



Obrázek 18 - Model ovladače Windows [11]

Ovladače na MS Windows jsou vyvíjeny pomocí Windows Driver Frameworks(WDF) nebo Windows Driver Model (WDM). Místo přímé komunikace s hardwarem, ovladač zasílá svoje požadavky na HAL(Hardware Abstraction Layer), což je vrstva, která zasílá obdržené požadavky na hardware. Komunikace mezi vrstvami ovladače ve Windows probíhá pomocí paketů s požadavky (I/O Request Packets nebo IRP), které jsou poskytnuty jako argumenty funkcím. Windows má oddělené části kernelu, které spravují vstup/výstupní operace (I/O) a odesílají zprávy ovladačům ve formě IRP.



Obrázek 19 - Model ovladače Linux [11]

V Linuxu, stejně jako ve Windows, existuje prostor pro uživatelské programy a pro programy kernelu. Ovladače jsou reprezentovány moduly, které rozšiřují funkcionalitu kernelu. Tyto moduly (viz. Obrázek 19 - Model ovladače Linux) mohou byt vrstveny a komunikace mezi nimi je dosažena pomocí volání funkcí. Tyto funkce musí ovladač zveřejnit při vkládáni do kernelu a jsou viditelné pro všechny moduly. Argumenty při volání funkci v Linuxu jsou však přizpůsobené danému ovladači.

1.1 Posloupnost činností pro tvorbu ovladačů

Požadavky: znalost programování v jazyce C/C++ a programování mikroprocesorů (přerušení a paměť)

Následuje posloupnost kroků, které je potřeba splnit pro vývoj ovladačů pro operační systémy Windows a Linux. První kroky jsou společné pro oba systémy.

1) USB specifikace – jako první krok je nezbytné nastudovat USB specifikaci (2.0 nebo 3.0). Ta obsahuje technické detaily pro pochopení požadavků. Je důležité rozumět toku dat, jak host a zařízení komunikují a formát požadavků, kterým zařízení rozumí.

2) USB zařízení – nyní je potřeba určit zařízení, pro které bude ovladač vyvíjen. Buď může být zvoleno standardní zařízení, které se běžně používá, nebo vývojový kit. Vývojové kity obsahují zpravidla více různých modulů, pro které může být ovladač určen. Mezi běžně dostupné vývojové kity patří např. DEMOJM společnosti Freescale, nebo kity pro Windows: OSR USB FX2 a Microsoft USB Test Tool.

3) Hardwarová specifikace – ke zvolenému zařízení je nutná hardwarová specifikace. Ta popisuje možnosti hardwaru a podporované příkazy. Slouží pro určení funkcionality zařízení. Také je potřeba znát dispozice zařízení a jeho možnosti, které jsou definovány v konfiguracích, rozhraních a koncových bodech. Zobrazení těchto vlastností lze docílit i externími programy, které umožňují procházet dispozice zařízení a zobrazit veškeré požadované vlastnosti.

Nyní se činnost dělí podle operačního systému

A) Windows

4) Architektura ovladačů ve Windows – poznat architekturu ovladačů a jejich konceptů ve Windows. Pro vývoj ovladačů ve Windows je nutné porozumět základům jejich funkčnosti, a jak jsou spouštěny. V tomto bodu je taky dobré si projít, jak Windows organizuje Plug and Play zařízení, kde se ve stromu zařízení nachází a kde se nachází ovladače. Také je nutné rozlišit architektury ovladačů, a to mezi user mode a kernel mode. Veškeré podklady se dají nalézt na stránkách Windows. Architektura ovladačů se nachází zde a umístění ovladačů a zařízení zde.

5) Příprava prostředí – nyní už vývoj začíná být specifický. Je dobré si určit, zda bude vyvíjen user mode nebo kernel mode ovladač. V případě user mode je slouží k debuggování prostředí Microsoft Visual Studio a pro kernel mode je nutné nakonfigurovat cílový počítač. Windows poskytuje i kit pro vývoj ovladačů, který je nutné si nainstalovat.

6) Vývoj ovladače – implementace ovladače. V této fázi se nachází vývoj, kompilace a instalace ovladače. Pro začátek je dobré použít předpřipravené šablony, nacházející se v Microsoft Visual Studiu. Zde už je nutné rozumět ovladačům, zařízením a jejich komunikace. V dalších fázích je možné ovladač rozšířit o přenos řídících požadavků a datových přenosů.

B) Linux

4) Kernel – je nutné poznat rozdíl mezi uživatelským prostorem a jádrem (kernel). Veškeré programy jsou částí uživatelského prostoru, ovšem ovladače a moduly patří do kernelu. Proto je dobré vědět, jak mezi sebou komunikují, jaké mají omezení a jaké oprávnění je potřeba pro jejich spuštění.

5) Komunikace s hardwarem – uživatelské aplikace využívají funkce kernelu pro komunikaci se zařízením. Pro pochopení této komunikace je nutné znát jak spolu kernel a hardware komunikují.

6) Implementace – ovladače v Linuxu jsou moduly rozšiřující funkčnost kernelu. Pro vývoj ovladače je tedy potřeba umět tento modul naprogramovat, přeložit pomocí makefilu a nahrát do kernelu. Jako modul můžete použít svůj vlastní nebo ten, který je zde přiložen. Makefile, ačkoliv jeho obsah bude zpravidla stejný, se bude spouštět s jinými argumenty pro každou verzi kernelu. Nakonec se přeložený modul nahraje do kernelu pomocí příkazu `insmod`. Jako ukázku je možné požít přiložený příklad, který stačí přeložit a nahrát do kernelu.

2 Zdrojové kódy

- **src/usb/***: adresář obsahující projekt v Codewarrioru. Tento projekt tvoří hlavní část této bakalářské práce a obsahuje USB ovladač pro DEMOJM a USB klíč.

- **src/cw/***: obsahuje skript nezbytný pro spuštění instalace Codewarrioru a soubor, který je nutné vložit do instalace pro rozpoznání připojeného vývojového kitu

- **src/testapp/*:** zde se nachází testovací aplikace ve formě ovladače pro Linux a Makefile, který obsahuje vše potřebné pro spuštění nezávisle na verzi Linuxu

- **src/manualapp/***: jednoduchý ovladač pro Linux, zmíněný v Příloze 1.1 – Posloupnost činností pro tvorbu ovladačů