UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY
A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2009 Martin Lauterbach

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Ovladač pro USB videokameru pro operační systém Linux

Autor práce: Martin Lauterbach

Vedoucí práce: Ing. Martin Hájek

Bakalářská práce

2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra elektrotechniky
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Martin LAUTERBACH**

Studijní program: **B2612 Elektrotechnika a informatika**

Studijní obor: **Komunikační a mikroprocesorová technika**

Název tématu: **Ovladač pro USB videokameru pro operační systém Linux**

Zásady pro vypracování:

Cílem bakalářské práce je vytvoření ovladače operačního systému Linux pro vývojový kit USB kamery firmy Omnivision s čipy OV538, OV7221. Ovladač realizujte jako modul jádra a navrhněte k němu vhodné softwarové rozhraní. Ovladač musí být schopný poskytnout snímek z kamery na žádost uživatelského softwaru. Tyto žádosti se opakují přibližně jednou za sekundu. V čase mezi žádostmi by mělo být vytížení systémových prostředků ovladačem co nejmenší.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

RICHARD, Stones, et al. Linux : Začínáme programovat. Praha : Computer Press, 2000. 897 s. ISBN 80-7226-307-2. JELÍNEK, Lukáš. Jádro systému Linux : Kompletní průvodce programátora. Praha : Computer Press, 2008. 688 s. ISBN 978-80-251-2084-2. Dokumentace k jádru operačního systému Linux. Katalogové listy, aplikační poznámky a další informace dosažitelné u výrobců použitých součástek na jejich www stránkách.

Vedoucí bakalářské práce: **Ing. Martin Hájek**
Katedra elektrotechniky

Datum zadání bakalářské práce: **15. ledna 2009**

Termín odevzdání bakalářské práce: **15. května 2009**

Univerzita Pardubice Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

Ing. Zdeněk Němec, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2009

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Kutné Hoře dne 13. srpna 2009

Martin Lauterbach

Poděkování:

Tímto bych rád poděkoval vedoucímu práce panu Ing. Martinu Hájkovi za odborné vedení a rady v průběhu práce.

Dále bych chtěl poděkovat firmě Steinel Technik, která iniciovala vytvoření této práce a umožnila mi její zpracování. Také mi umožnila podílet se na vývoji celého zařízení a získat tak cenné zkušenosti.

Souhrn

Tato bakalářská práce se zabývá tvorbou ovladače pro USB videokameru pro operační systém Linux. Popisuje možnosti operačního systému Linux a využití zpětného inženýrství při tvorbě tohoto ovladače.

Klíčová slova

ovladač, Linux, USB, video, izochronní přenos, obrazový senzor, V4L

Abstract

This bachelor thesis describes the development of driver for USB video camera for the Linux operating system. Describes the possibility of using Linux and reverse engineering to create this driver.

Keywords

driver, Linux, USB, video, isochronous transport, image sensor, V4L

Obsah

Seznam obrázků

Seznam zkratek

ABF	*Automatic Band Filter*, automatický pásmový filtr
ABLC	*Automatic Black Level Control*, automatické řízení úrovně černé
AEC	*Automatic Exposure Control*, automatické řízení expozice
AGC	*Automatic Gain Control*, automatické řízení zisku
AVI	*Audio Visual Interleaved*, souborový kontejner pro obrazová a zvuková data
AWB	*Automatic White Balance*, automatické vyvážení bílé
EEPROM	*Electronically Erasable Programmable Read-Only Memory* elektricky mazatelná přeprogramovatelná paměť
GCC	*GNU Compiler Collection*, překladač jazyka C
GNU	*GNU is Not Unix*, rekurzivní akronym označující projek s cílem vytvoření svobodného operačního systému
GPL	*General Public License*, licence pro šíření svobodného softwaru
IEC	*International Electrotechnical Commission*, Mezinárodní elektrotechnická komise
ISO	*International Organization for Standardization*, Mezinárodní organizace pro normalizaci
IEEE	*The Institute of Electrical and Electronics Engineers*, Institut pro elektrotechnické a elektronické inženýrství
LED	*Light Emitting Diode*, světlo vyzařující dioda
PID	*Product ID*, identifikátor produktu
POSIX	*Portable Operating System Interface*, standard pro přenositelné rozhraní operačních systémů

QVGA	*Quarter Video Graphics Array*, rozlišení 320x240 pixelů
RGB	barevný model, aditivní způsob míchání barev
SCCB	*Serial Camera Control Bus*, sběrnice pro řízení obrazových senzorů
URB	*USB Request Block*, blok požadavku USB
V4L	*Video for Linux*, linuxový subsystém pro práci s multimediálními zařízeními (videokamery, TV tunery atd.)
V4L2	*Video for Linux Two*, druhá generace V4L
VBI	*Vertical Blanking Interval*, svislý zatemňovací interval
VGA	*Video Graphics Array*, termín pro označení rozlišení 640x480 pixelů
VID	*Vendor ID*, identifikátor výrobce
X11	X Window System, umožňuje využít display pro grafické uživatelské prostředí
YCbCr	barevný model obrazu, Y je jas, Cb a Cr jsou modré a červené chrominační komponenty
YUVV	barevný model obrazu používaný v televizním vysílání

Kapitola 1

Úvod

Operační systém Linux je nasazován na stále větší množství zařízení, od velkých mainframe počítačů až po malá zařízení obsahující jednočipový mikrokontrolér. Je to umožněno jeho flexibilitou a tím, že lze Linux svobodně šířit a upravovat. Snadno můžeme studovat jeho zdrojové kódy a velké množství dostupné dokumentace.

Jedním z jeho plánovaných nasazení je i inteligentní videosenzor firmy Steinel, který má sloužit k zabezpečení prostor. Měl by rozpoznávat osoby a měl by například dokázat určit jejich počet v místnosti.

Základem inteligentního videosenzoru je modul osazený procesorem Intel Atom. Na tento základní modul jsou připojeny periferní součásti. Hlavní z nich je kamerový modul připojený na sběrnici USB.

Úkolem této bakalářské práce je vytvoření ovladače uvedeného kamerového modulu pro operační systém Linux. Kameru představuje vývojový kit od firmy Omnivision. Výrobce poskytl jen minimální dokumentaci, jednoduchý software a ovladač pro práci s kamerou pod operačním systémem Windows.

Je potřeba nalézt cestu, jak vytvořit ovladač pro operační systém Linux pro uvedenou kameru. Žádoucí je, aby vyvíjené zařízení bylo založeno právě na zmíněném kitu, protože jeho videosenzor má vysokou citlivost a jeho parametry jsou nejvhodnější. Díky nedostatku informací o jeho hardwaru je nutné zkoumat připojený kit v operačním systému Windows, zpětně zjistit jak ho nakonfigurovat a jak s ním komunikovat přes USB, aby bylo možné vytvořit ovladač pro Linux.

Kapitola 2

Operační systém Linux

Tato kapitola vychází především z literatury [13] a podle ní je Linux *monolitické jádro* (někdy bývá, vzhledem ke své architektuře, považováno též za jádro *hybridní*) operačního systému, šířené jako svobodný software (pod licencí GNU GPL verze 2). Dokáže využít nejmodernější počítačové technologie, ale současně umožňuje funkci starých programů s jen minimálními úpravami.

Linux je *flexibilní* - totožná verze (z hlediska zdrojových kódů) může pracovat na superpočítačích s mnoha procesory a obrovskou operační pamětí i v malém jednoúčelovém zařízení (modemu ADSL, strojku na jízdenky, síťové kartě atd.) s jednoduchým procesorem a minimální pamětí. Záleží jen na nastavení kompilace (a následně také systému za běhu), jak bude výsledné jádro vypadat.

Linux je *moderní* - nejnovější technologie, protokoly, algoritmy atd. se v linuxovém jádře objevují velmi brzy a s výbornou podporou. Jako příklad lze uvést třeba IP verze 6, hyperthreading, různé souborové systémy apod.

Vývoj Linuxu je *evoluční* - není řízen pevně definovaným návrhem, změny jednotliví vývojáři provádějí na základě požadavků uživatelů, zjištění z praxe a vývoje v oblasti hardwaru. Každá změna je podrobena diskuzi ve veřejném fóru, a pokud se ukáže jako přínosná, správce příslušné části jádra ji (po případném kratším či delším testování) zahrne do kódu jádra.

Linux je *svobodný* - vzhledem k licenci může kdokoliv studovat, jak jádro funguje, může si v něm cokoliv opravit nebo upravit, přidat nové funkce a upravené

jádro případně šířit dál. Tak mohou vznikat i speciální verze jádra, určené pro nějaké speciální účely. Příkladem lze uvést třeba projek *uClinux*, zabývající se vývojem jádra pro mikrořadiče.

2.1 POSIX

POSIX (*Portable Operating System Interface*) je přenositelné rozhraní pro operační systémy. Vzniklo podle rozhraní systému UNIX a popisuje jak základní systémovou funkcionalitu, tak i řadu různých volitelných součástí. Rozhraní POSIX je standardizováno normami IEEE 1003 a ISO/IEC 9945. První verze byla vydána v roce 1988, později byl standard aktualizován. Kvůli restriktivní licenční politice organizace IEEE vznikl později na základě rozhraní POSIX otevřený a volně dostupný standard *Single UNIX Specification*.

2.2 Licence GPL

Linuxové jádro je šířeno pod licencí GPL (*General Public License*). Tato licence umožňuje volné šíření a otevřenost Linuxu, ale zároveň nařizuje, že odvozená díla musejí být opět šířena jen pod touto licencí a zdrojový kód takového díla musí být dostupný. Plné znění licence lze nalézt v [1].

2.3 Programování jádra

Svět jádra je trochu jiný než svět aplikačních programů. Zatímco programátor jádra může využívat prakticky vše, co mu systém nabízí (kompletní instrukční sadu, celou operační paměť) a má přímý přístup k hardwaru, aplikační programátor si musí vystačit s omezenými prostředky a přímí přístup k zařízení (až na výjimky) nemá. Na druhou stranu aplikačnímu programátorovi projdou i všelijaké prohřešky při psaní programu, kdežto programátor jádra je svázán tuhou disciplínou. Programový kód

je v Linuxu prováděn ve dvou režimech úzce svázaných s použitou hardwarovou platformou. V uživatelkém režimu a v režimu jádra.

2.3.1 Uživatelský režim

Uživatelský režim je režim systému, v němž běží uživatelské procesy. Lze využívat jen omezenou instrukční sadu a platí i další omezení (ohledně operací s pamětí, přístupu k zařízením atd.).

2.3.2 Režim jádra

Režim jádra je režim systému, v němž se vykonává kód jádra - ať už v kontextu procesů či mimo něj. Lze používat plnou instrukční sadu a přistupovat v systému kamkoliv.

2.3.3 Systémová volání

Systémová volání jsou základní a nejčastěji používanou metodou komunikace aplikačních procesů s jádrem. Tvůrce programu je od nich obvykle oddělen vrstvou standardní knihovny.

Kapitola 3

Sběrnice USB

V této kapitole jsou některé zjednodušeně podané informace o USB. Čerpáno je z literatury [9],[10], [12], [13] a [16], kde lze nalézt podrobnější informace.

Sběrnice USB slouží ke spojení počítače a řady periferních zařízení. Byla vytvořena jako jednotná náhrada mnoha různých pomalých sběrnic (paralelní port, sériový port, připojení klávesnice). V současných počítačích se opravdu USB stalo nejpoužívanějším způsobem pro připojování periferií. Od specifikace USB verze 2.0, kde byla přidána podpora vysokorychlostních přenosů až do 480 Mb/s, je možné bezproblémově připojit zařízení vyžadující velké přenosové rychlosti (pevné disky, videokamery atd.).

USB je ve skutečnosti sběrnice jen logická. Z topologického hlediska není USB opravdovou sběrnicí, ale spíše stromovým uspořádáním dvoubodových spojení. Spojení jsou realizována datovým párem kroucených vodičů a dvěma vodiči pro napájení.

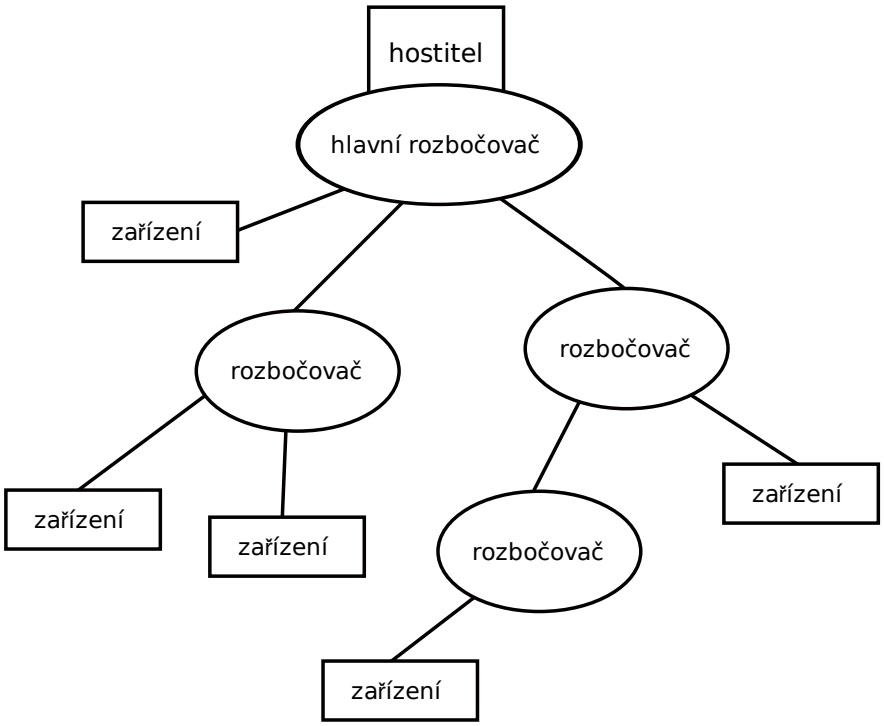
Sběrnice má jen jedenoho nadřízeného člena, kterým je tzv. *USB host controller*. Nadřízený člen se dotazuje postupně všech zařízeních na sběrnici zda nemají k odeslání nějaká data. Pokud ano, umožní jim je odeslat.

3.1 Topologie sběrnice

Podle specifikace [9] existují následující čtyři pohledy na topologii USB:

- hostitel a zařízení - základní kameny USB systému,

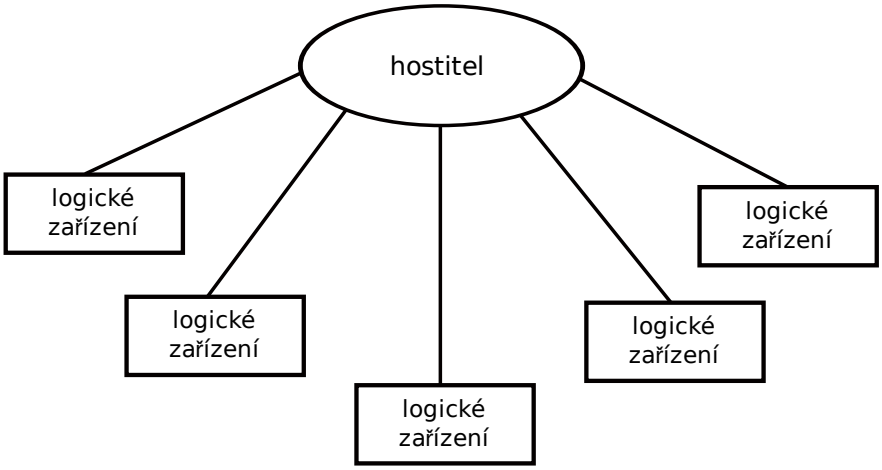
- fyzická topologie - skutečné propojení prvků systému (obr. 3.1),
- logická topologie - role jednotlivých prvků tak, jak se jeví z pohledu hostitele a zařízení (obr. 3.2),
- vztah softwaru a funkce zařízení - způsob, jakým na sebe vidí software klienta a rozhraní zařízení.



Obrázek 3.1: Fyzická topologie USB

3.2 Rychlosti přenosu dat

Následující popis rychlostí přenosu dat lze nalézt v literatuře [16]. Zařízení, která jsou připojena na sběrnici USB, mohou přenášet data několika rychlostmi. Původní norma, dnes označovaná jako USB 1.0, stanovovala dvě rychlosti. Základní rychlost 1,5 MB za sekundu (12 Mb za sekundu), neboli Full Speed, musí být podporována všemi rozbočovači a většinou i samotným kořenovým uzlem, nikoli však nutně koncovými



Obrázek 3.2: Logická topologie USB

zařízeními, protože některá zařízení, například klávesnice či joysticky, tak vysokou rychlost ke své činnosti nepotřebují. Taková pomalá zařízení pracují na rychlosti 187,5 kB za sekundu (1,5 Mb za sekundu), jež se označuje termínem Low Speed. Při této rychlosti se každý bit přenáší osmkrát pomaleji než při použití základní (plné) rychlosti. V roce 2001 byla v normě USB 2.0 stanovena i vyšší rychlost – High speed – při které lze teoreticky dosáhnout až hodnot 60 MB za sekundu (480 Mb za sekundu), ovšem ne všechna zařízení, která na sobě mají nálepku USB 2.0, skutečně až této rychlosti dosahují. Zařízení dovolující použití High speed může v případě potřeby přejít i na základní rychlost, což je nutné pro udržení zpětné kompatibility.

V současnosti se většinou můžeme setkat se základními deskami a zařízeními, které plně podporují USB 2.0, tedy i všechny tři výše popsané přenosové rychlosti. Ovšem poměrně nedávno, byla vydána specifikace USB 3.0, která mj. obsahuje i tzv. SuperSpeed, což je označení přenosové rychlosti, jejíž hodnota dosahuje 625 MB za sekundu (5,0 Gb za sekundu). V současnosti se se zařízeními, které by tuto rychlost podporovaly, pravděpodobně nesetkáme, ale firma Intel (jeden z autorů této specifikace) očekává, že by se již koncem tohoto roku mohly na trh dostat základní desky i externí zařízení odpovídající USB 3.0. SuperSpeed již umožňuje práci s videem či s rychlými externími pevnými disky, tj. jedná se o alternativu k externímu SCSI

či FireWire (v současnosti například pevné disky připojené přes FireWire pracují rychleji než srovnatelné disky připojené přes USB 2.0).

3.3 Přenosy přes USB

Komunikace po sběrnici USB odehrává v rámci tzv. *přenosů*. Není to tedy tak, že by měl ovladač celou sběrnici pro sebe - naopak, může se o ni dělit i poměrně velký počet ovladačů a USB řadič má za úkol přenosy poskládat tak, aby se co nejvíce vyhovělo požadavkům zařízení (resp. ovladačů).

Důležité právě je, o jaký druh přenosu se v jednotlivých případech jedná, protože se pak nároky značně liší. Pro některé přenosy řadič vyhrazuje pásmo, jiné se prostě musí spokojit s tím, co zbude.

Existují čtyři kategorie přenosů. Každá kategorie má své specifické vlastnosti, podle kterých si tvůrci fyzických zařízení vybírají, který druh přenosu pro ten který účel použijí. Jedná se o tyto kategorie:

- *Control* - řídicí přenosy. Používají se pro zjišťování stavu zařízení, jeho konfiguraci a ovládaní. Pro tyto přenosy se vždy vyhrazuje pásmo.

- *Interrupt* - periodické přenosy malých objemů dat. Vhodné pro zařízení, která často (a obvykle pravidelně) posílají malá množství dat, u kterých záleží na včasném doručení. Jsou to například myši, klávesnice, tablety apod. Řadič pro tyto přenosy vyhrazuje pásmo.

- *Bulk* - přenosy velkých objemů dat. Používají se pro takové situace, kdy se musí přenést nějaký objem, nicméně není příliš důležité, jak dlouho to bude trvat. Typické použití je pro úložná zařízení, tiskárny apod. Pásmo se nevyhrazuje, data mohou být přenášena po částech, doba trvání celého přenosu není určena.

- *Isochronous* - izochronní datové přenosy (pevný objem za určitou jednotku času). Má smysl tam, kde je potřeba přenést určitý (větší) objem včas a pozdní doručení je horší než nedoručení. Obvyklé použití je pro multimediální zařízení,

např. zvukové adaptéry nebo televizní tunery. Pásmo se nevyhrazuje - data, která se nepodařilo přenést včas, se jednoduše zahazují.

3.4 USB zařízení

USB zařízení jsou dělena do tříd, jako jsou např. rozbočovače, vstupní zařízení, tiskárny nebo paměti. Rozbočovače jsou speciální třídou USB zařízení, která poskytují další přípojné body. USB zařízení musí obsahovat informace o své konfiguraci a své identifikaci.

Ke každému USB zařízení je přistupováno pomocí USB adresy, která je zařízení přiřazena poté, co je zařízení připojeno a rozpoznáno během inventarizace sběrnice. Každé USB zařízení dále poskytuje jednu či více komunikačních rour, přes které komunikuje hostitelský systém se zařízením. Každé zařízení musí poskytovat speciální komunikační rouru ke koncovému bodu 0, která je použita pro řízení USB zařízení. Všechna USB zařízení mají stejný mechanismus přístupu k informacím přes tuto řídící rouru.

K řídící rouře jsou přiřazeny informace, které popisují dané USB zařízení. Tyto informace spadají do následujících kategorií:

- Standardní: Toto jsou informace, jejichž definice je společná všem USB zařízením, a obsahují položky jako např. identifikace výrobce, třída zařízení nebo řízení napájení.
- Informace závislé na třídě: Formát a definice těchto informací se různí podle třídy USB zařízení.
- Informace výrobce: Výrobce zařízení může do této kategorie vložit libovolné informace. Jejich formát není nijak specifikován.

Navíc každé USB zařízení nese informaci o svém stavu a své konfiguraci.

3.4.1 Koncové body

Koncový bod (endpoint) je jednoznačně identifikovatelná část USB zařízení, která stojí na jednom konci komunikačního toku mezi hostitelem a zařízením. Každé logické zařízení se skládá z několika nezávislých koncových bodů. Tomuto logickému zařízení je během připojování přiřazena jednoznačná identifikace v rámci sběrnice. Každý koncový bod v rámci zařízení má přiřazen výrobcem kód, nazývaný číslo bodu v rozsahu 0-15. Endpointy mají zároveň návrhem pevně daný směr komunikace (příjem/vysílání dat). Kombinace identifikátoru zařízení, čísla endpointu a směru komunikace dává dohromady jednoznačné určení endpointu v rámci USB sběrnice. V jednom zařízení se tedy mohou vyskytnout dva endpointy se stejným číslem a různým směrem přenosu dat.

Koncový bod má jisté charakteristiky, které určují typ přenosů mezi ním a klientským SW. Koncový bod sám sebe popisuje:

- požadavky na frekvenci a latenci přístupů ke sběrnici,
- požadavkem na šířku přenosového pásma,
- číslem koncového bodu,
- požadavky na chování obsluhy chyb,
- maximální velikostí paketu, kterou je schopen endpoint přijmout nebo odeslat,
- typem přenosu,
- směrem přenosu.

Endpointy jiné než endpoint 0 jsou po připojení v nedefinovaném stavu a nesmí k nim být přistupováno až do doby než je zařízení nakonfigurováno.

3.4.2 Rozhraní a konfigurace

Koncové body se sdružují do logických zařízení nazývaných rozhraní. Jediné fyzické zařízení může obsahovat větší počet rozhraní - např. tiskárna se čtečkou paměťových

karet bude mít jedno rozhraní pro tisk a druhé pro přístup na vkládané paměťové karty.

Fyzické rozhraní může mít více konfigurací. Konfigurace je sada rozhraní, v každé konfiguraci jsou tedy na tomtéž zařízení obsažena jiná rozhraní. V konkrétním okamžiku může být aktivní jen jedna konfigurace. Běžně se více konfigurací nepoužívá, nicméně se to obecně může vyskytnout.

3.4.3 Třídy a podtřídy USB zařízení

Na univerzální sériovou sběrnici je možné připojit velké množství různých zařízení, od pomalorychlostních klávesnic, myší a joysticků přes známé paměti Flash disk až po tiskárny, webové kamery či skenery. Každému zařízení je podle jeho funkce přiřazen kód nazývaný třída zařízení. Podle třídy zařízení, kterou je možné zjistit ihned po zapojení zařízení do sítě a jeho resetu, operační systém zvolí vhodný ovladač, přičemž – a to je docela důležité, především na Linuxu – zařízení, která spadají do stejné třídy, mohou používat shodný či velmi podobný ovladač, i když se může jednat o zařízení od různých výrobců. Některé třídy zařízení se dále dělí na specializované podtřídy a pod-podtřídy. V tabulce 3.1 jsou uvedeny všechny třídy.

Číslo třídy	**Popis třídy**	**Číslo třídy**	**Popis třídy**
00h	Určeno na úrovni rozhraní	0Bh	Čipová karta
01h	Zvuk	0Dh	Zabezpečení obsahu
02h	Komunikace - řízení	0Eh	Video
03h	Zařízení pro styk s člověkem	0Fh	Zdravotnické zařízení
05h	Fyzické zařízení	DCh	Diagnostika
06h	Obraz	E0h	Bezdrátové technologie
07h	Tiskárna	EFh	Různé
08h	Úložiště dat	FEh	Určeno aplikací
09h	Rozbočovač	FFh	Určeno výrobcem
0Ah	Komunikace - data		

Tabulka 3.1: Třídy USB zařízení

3.5 Rozbočovače

Vzhledem k tomu, že jedním kabelem lze propojit pouze dva uzly, musí existovat způsob, jak celkový počet uzlů (zařízení) zvýšit až na maximální hranici, která činí 127 uzlů připojitelných k jednomu kořenovému uzlu. Řešení problému připojení většího množství zařízení spočívá v použití takzvaných rozbočovačů. Každý rozbočovač, který může být představován buď specializovaným zařízením, jež slouží pouze pro připojení dalších uzlů, nebo má i jinou funkci (rozbočovače na monitorech, klávesnicích atd.) obsahuje vždy jeden konektor (většinou typu B) určený pro připojení buď přímo ke kořenu (počítači) nebo k dalšímu rozbočovači a prakticky libovolný počet konektorů (většinou typu A), na které lze připojit koncová zařízení. Díky tomuto uspořádání je již mechanicky zajištěno, že nevznikne zakázaná topologie obsahující cyklus. Na tomto místě je vhodné upozornit na to, že rozbočovač obsahuje poměrně složité obvody, které zajišťují řízení sběrnice, zesilování signálů, úpravu jejich hran a směrování paketů, tj. nejedná se o pouhé paralelní propojení několika konektorů. Norma stanovuje, že za sebou může být zapojeno maximálně pět rozbočovačů, ovšem díky tomu, že počet rozbočovačů na stejné úrovni již omezen není, lze snadno vybudovat i velmi rozsáhlé sítě, které jsou omezené především maximální povolenou délkou propojovacích kabelů (3m v případě USB 2.0).

3.6 Třída USB Still Image Class

Třída pro USB zařízení, která slouží pro zachycování nepohyblivého obrazu (*still image*), vychází se standardu PIMA 15740. Pro přenos obrazových dat je využit *bulk* koncový bod. Je zde specifikováno rozhraní zařízení a přenosový protokol obrazu. Podrobnější informace jsou uvedeny v dokumentu [8].

3.7 Třída USB Video Class

Třída *USB Video Class* (často zkráceně *UVC*) popisuje zařízení schopná streamovaného videa (webové kamery, digitální videokamery, transkodéry, televizní tunery atd.). Její specifikaci je možno nalézt v [11].

Kapitola 4

Ovladače v Linuxu

O tvorbě ovladačů v Linuxu je podrobně psáno v knihách [12], [13] a [14]. Z těchto knih vychází i tato kapitola.

Linux představuje jádro operačního systému, na které je napojeno velké množství ovladačů. Tyto ovladače jsou buď přímo součástí zkompilovaného jádra a nebo mohou být ve formě modulů, které je možné načítat nebo odstraňovat z paměti za běhu systému.

Většina ovladačů umožňuje operačnímu systému pracovat s hardwarem, ale existují ovladače jen na softwarové úrovni a jejich typickým zástupcem jsou ovladače souborových systémů.

Přednostně tedy ovladače zajišťují řízení hardwaru a zároveň poskytují obecnější rozhraní, které zajišťuje abstrakci zařízení. Například je pak jedno jestli pracujeme s blokovým zařízením pevného disku uvnitř počítače nebo paměti flash připojené na USB. Toto rozhraní periferií je možné rozdělit podle vlastností na znaková a bloková zařízení.

4.1 Znaková a bloková zařízení

Znaková zařízení komunikují prostřednictvím proudů znaků. Pro znaková zařízení platí, že nemusí být možné přistupovat někam jinam než na konec proudu. Typickým znakovým zařízením je sériový port.

U blokových zařízení probíhá komunikace pomocí bloků dat o pevné velikosti. Umožňují přístup kamkoliv pomocí adresace jednotlivých bloků. Zástupcem této kategorie jsou pevné disky, optické disky (CD, DVD atd.) a paměti flash.

4.2 Ladící zprávy

Programování ovladačů je velice specifické a je obtížné ladit běžící procesy jádra.

Nejjednodušším způsobem ladění jaderných modulů jsou ladící zprávy. K zápisu ladící zprávy z modulu slouží funkce `printk()`. Výstup těchto zpráv lze zobrazit příkazem `dmesg`.

Lze využít i složitější nástroje pro ladění, jako *KGDB*, ale u jednoduchých projektů je možné se bez nich obejít.

4.3 Struktura jednoduchého modulu

Modul jádra nemá narozdíl od běžného programu funkci `main()`, ale spíš jakýsi konstruktor a destruktor. Základem každého modulu je několik maker z vložených hlavičkových souborů. Zdrojový kód velmi jednoduchého modulu, který prakticky téměř nic nedělá, je následující:

```
#include <linux/module.h>
#include <linux/kernel.h>

//informace o~modulu
MODULE_DESCRIPTION("Jednoduchy modul");
MODULE_AUTHOR("Jmeno Autora");
MODULE_LICENSE("GPL");
MODULE_VERSION("1.0.1");

//funkce volana pri zavedeni
static int __init mujmodul_init(void)
{
    printk(KERN_INFO "mujmodul: modul inicializovan\n");
    return 0;
}

//funkce volana pri odstraneni
static void __exit mujmodul_exit(void)
```

```
{
    printk(KERN_INFO "mujmodul: modul uvolnen\n");
}

module_init(mujmodul_init);
module_exit(mujmodul_exit);
```

Na začátku jsou vloženy dva soubory obsahující některá nezbytná makra pro překlad modulu. Každý modul by měl obsahovat svůj popis. Není to nezbytné, ale licence by měla být uvedena (v podobě `MODULE_LICENSE`). Pokud lince nebude uvedena a nebo bude jiná než GPL, není modul takzvaně košér.

Následují dvě funkce, jedna volaná jádrem při zavedení modulu a druhá při jeho odstranění. Symbol `__init` říká, že funkce je volána jen při zavedení modulu a poté se může odstranit z paměti. Druhý symbol `__exit` označuje funkci volanou pouze při uvolnění modulu. Pokud se kompiluje přímo do jádra (ne do samostatného modul) tato funkce se vůbec nezkompiluje.

Na závěr pomocí maker `module_init()` a `module_exit()` řekneme, které funkce budou sloužit pro inicializaci a úklid.

4.4 Překlad modulu

Jádra řady 2.6 mají oproti dřívějším velkou výhodu v podobě kompilačního systému KBUILD, který kompilaci jádra nebo třeba jen jediného modulu velice ulehčuje. K překladu modulu jsou potřeba jen následující tři věci:

- zdrojové soubory jádra,
- kompilátor GCC,
- `Makefile`.

Pro zdrojový soubor `mujmodul.c` bude jednoduchý `Makefile` vypadat následovně:

```
obj-m += mujmodul.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

```
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Tento soubor přeloží modul pro jádro právě používané systémem. .

4.5 Zavedení modulu

Přeložený ovladač v podobě modulu jádra je v souboru s příponou `.ko`. Pro zavedení modulu zde slouží příkaz `insmod ovladac.ko`. Tímto příkazem zavedeme jen vytvořený modul bez řešení závislostí na ostatních modulech. Používané jádro musí být zkompilováno s podporou jaderných modulů. V běžně používaných distribucích taková jádra jsou.

Kapitola 5

Ovladače USB zařízení

Ovladače jednotlivých USB zařízení využívají vrstvu ovladačů *USB core*. Tato mezivrstva se stará o řadiče USB, které bývají často připojeny na sběrnici PCI, a řeší i fyzickou topologii sběrnice. Programátor pak využívá jen logickou topologii, o strukturu rozbočovačů se nemusí starat a vidí jen koncová zařízení.

5.1 Struktura `usb_driver`

Tato struktura slouží pro registraci ovladače vlastního zařízení do USB subsytému. Prototyp struktury je následující:

```
struct usb_driver {
  struct module * owner;
  const char * name;
  int (* probe) (struct usb_interface *intf,const struct usb_device_id *id);
  void (* disconnect) (struct usb_interface *intf);
  int (* ioctl) (struct usb_interface *intf, unsigned int code, void *buf);
  int (* suspend) (struct usb_interface *intf, pm_message_t message);
  int (* resume) (struct usb_interface *intf);
  const struct usb_device_id * id_table;
  struct device_driver driver;
};
```

USB ovladače musí poskytnout jméno (`name`), metody `probe` a `disconnect` a tabulku `id_table`. Další pole jsou nepovinná.

5.2 Struktura `file_operations`

Struktura `file_operations` je deklarována v hlavičkovém souboru linux/fs.h následovně:

```
struct file_operations {
  struct module *owner;
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
  int (*readdir) (struct file *, void *, filldir_t);
  unsigned int (*poll) (struct file *, struct poll_table_struct *);
  int (*ioctl) (struct inode *, struct file *, unsigned int,
     unsigned long);
  int (*mmap) (struct file *, struct vm_area_struct *);
  int (*open) (struct inode *, struct file *);
  int (*flush) (struct file *);
  int (*release) (struct inode *, struct file *);
  int (*fsync) (struct file *, struct dentry *, int datasync);
  int (*fasync) (int, struct file *, int);
  int (*lock) (struct file *, int, struct file_lock *);
  ssize_t (*readv) (struct file *, const struct iovec *, unsigned long,
     loff_t *);
  ssize_t (*writev) (struct file *, const struct iovec *, unsigned long,
     loff_t *);
};
```

Tato struktura je skupina ukazatelů na funkce, které jsou prováděny při některých systémových voláních nad souborem zařízení.

5.3 Registrace ovladače

Stejně jako všechny ovladače zařízení, i ty pro USB se musí odpovídajícím způsobem zaregistrovat. Nejdřív je ale potřeba připravit informace o tom, která zařízení ovladač podporuje. Slouží k tomu makra `USB_DEVICE`, `USB_DEVICE_VER`, `USB_DEVICE_INFO` a `USB_INTERFACE_INFO`. Pomocí maker se sestaví pole struktur `usb_device_id` a toto pole se zveřejní známým makrem `MODULE_DEVICE_TABLE`. Tím je zajištěno, že může program `depmod` informace o podporovaných zařízení přečíst a vytvořit z nich soubor `modules.usbmap` pro výběr správného modulu k načtení.

Pro vlastní registrace připravíme strukturu `usb_driver`, která kromě výše popsané tabulky zařízení a dalších informací obsahuje také ukazatele na funkce ovladače

(budou popsány později). V inicializační funkci modulu se pak pro danou strukturu zavolá `usb_register()`, čímž se ovladač stane plně použitelným. Při uvolňování se pak musí ovladač odregistrovat funkcí `usb_deregister()`

5.4 Blok požadavků (URB)

Pro jednotlivé přenosy přes USB se používají speciální datové struktury – bloky požadavků USB (USB Request Block, URB). Je to obdobné jako třeba u síťových ovladačů. Blok se vytvoří, nastaví se mu odpovídající parametry a pak se předá USB subsystému (ten se přes ovladač USB řadiče postará o provedení přenosu). Až se přenesou data, USB subsystém o tom vyrozumí ovladač. Přenos je asynchronní, požadavky se tedy řadí do fronty a během zpracování může ovladač dělat něco jiného.

Zpracovávaný požadavek lze stornovat. Může být stornován i automaticky, například v případě, že se USB subsystém od řadiče dozví, že bylo příslušné zařízení odebráno. I s touto eventualitou musí ovladač USB zařízení počítat. V některých jednodušších případech se URB nepoužívají (u přenosů typu BULK).

Ovladač je o dokončení nebo stornování přenosu vyrozuměn zavoláním obslužné funkce, která se v URB nastaví. Obslužná funkce musí být napsána podle pravidel pro obsluhu přerušení, protože může být takto zavolána – tedy co nejrychleji ven, žádné blokování atd. (v případě potřeby se musí pro „závadné“ operace použít odložené zpracování).

Kapitola 6

Video For Linux

V této kapitole je čerpáno z knihy [13], dále z dokumentace [7] a [15]. Dnes je používána druhá generace technologie Video For Linux (označováno V4L2). Tato technologie poskytuje v Linuxu abstraktní přístup k různým audiovizuálním zařízením připojitelným k systému. Cílem je, aby aplikace nemusely řešit, s jakým zařízením právě pracují, a všechno se ovládalo jednotným způsobem. Technologie nepřidává žádná nová systémová volání, vystačí si s těmi standardními.

V4L2 poskytuje aplikacím následující rozhraní:

- **záznam videa** (*video capture*),
- **překryvné video** (*video overlay*) - mechanizmus, kdy jdou data ze vstupního zařízení přímo do výstupního,
- **výstup videa** (*video output*),
- **data ze svislého zatemňovacího intervalu** (*vertical blanking interval, VBI*) - teletext nebo časové kódy,
- **rozhlasová zařízení**.

Ve starší generaci V4L bylo obsaženo ještě například zpracování teletextu nebo RDS, ale tyto věci jsou dnes řešeny v uživatelském prostoru, případně speciálními ovladači. Pod audiovizuální zařízení patří (z pohledu V4L) televizní a rozhlasové tunery, digitalizační karty, USB a IP kamery a další podobná zařízení.

V4L má dvě rozhraní. Jedno pro uživatelské aplikace. Druhé pro ovladače jednotlivých zařízení. Ovladače V4L jsou samostatným druhem ovladačů, existují mimo obvyklou kategorizaci (znaková, bloková a síťová zařízení).

6.1 Vnější a vnitřní rozhraní

Vnější rozhraní, tedy pro uživatelské aplikace, je založeno na běžných systémových voláních (`open()`, `read()`, `write()`, `ioctl()`, `mmap()` atd.). Ovladač exportuje virtuální zařízení, podle kterých pak *udev* vytváří příslušné soubory (`/dev/video0`, `/dev/vbi0` apod.).

Důležitým mechanizmem je mapování paměti. Přestože lze přenášet videodata přes klasické operace I/O, mnohem většího výkonu lze dosáhnout tak, že ovladač kopíruje data z/do paměťového bufferu. Ten může být vytvořen v aplikaci a poskytnut ovladači, ale vhodnější bývá jeho vytvoření v jádře, nebo v některých případech dokonce přímo v zařízení (odpadá tak zbytečné kopírování).

Rozhraní pro ovladače je poměrně složité. Je založeno na struktuře `video_device`, která obsahuje několik datových položek (strukturu `device`, název, typ, minor číslo atd.), strukturu souborových operací (`file_operations`) a potom obrovské množství funkcí zpětných volání (především `ioctl()`).

Některé činnosti není třeba implementovat zvlášť pro každý ovladač. V jádře existuje řada hotových univerzálních ovladačů (`tuner`, `video_buf`, `ir_common` atd.). Některé používané elektronické obvody mají také své ovladače a lze využít jejich služeb.

6.2 Ovladač GSPCA

Ovladač GSPCA byl vytvořen pro obsluhu webkamer připojených na USB. Jeho vnější rozhraní představuje Video for Linux. V současné době podporuje desítky webkamer různých výrobců. Od verze jádra 2.6.27 je jeho součástí.

Ovladač je rozdělen do dvou částí. Jedna část, ovladač `gspca_main`, tvoří jakousi mezivrstvu ovladače zařízení a rozhraní V4L. Tento kód je velmi podobný pro mnoho USB webkamer a je možné ho takto osamostatnit. Druhou část tvoří kód, který je specifický pro hardware webkamery. Tyto specifické ovladače jsou tvořeny moduly s názvem začínajícím `gspca_`.

Kapitola 7

Vývojový modul Omnivision

Základem kamerového modulu je videosenzor OV7221 a tzv. bridge OV538, který zpracovává dále obraz z videosenzoru a obrazová data zpřístupňuje na sběrnici USB.

Modul je tvořen dvěma deskami plošných spojů. Na jedné desce je osazen a zapojen bridge a na druhé videosenzor. Důvodem pro toto oddělení je to, že bridge dokáže spolupracoval s řadou různých videosenzorů.

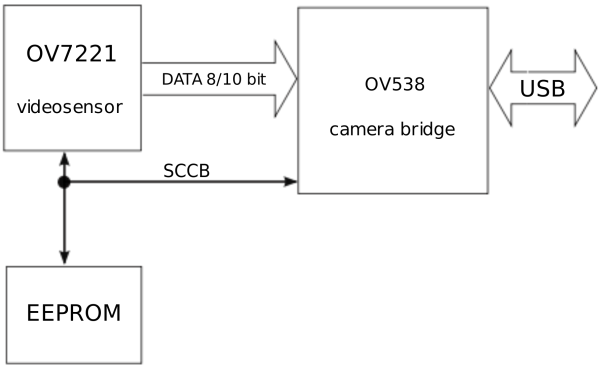
7.1 Zapojení

Blokové schéma kamery je na obrázku 7.1. Základem je kamerový můstek OV538, ke kterému je připojen videosenzor OV7221 a sériová paměť EEPROM. Můstek je řídícím obvodem celého zařízení a zároveň připojuje kameru na USB.

Sběrnice SCCB propojuje můstek, videosenzor a sériovou paměť EEPROM. Jejím úkolem je přenos parametrů nastavení z můstku do videosenzoru. V paměti EEPROM je pak uložen deskriptor USB zařízení a program pro můstek, který má roli nadřízeného obvodu na sběrnici. Videosenzor a EEPROM jsou podřízené.

7.2 USB bridge

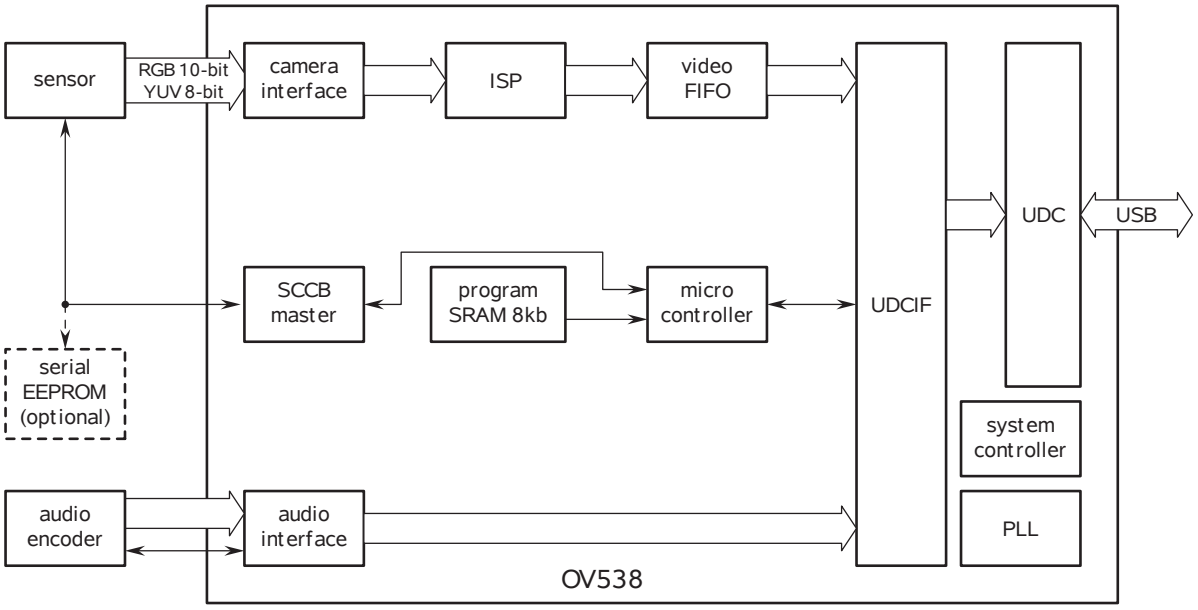
Můstek OV538 je jednočipový procesor pro aplikaci v kamerách využívajících USB 2.0. Podporuje obrazové senzory až do rozlišení 2 megapixely. Pro plnou funkčnost



Obrázek 7.1: Blokové schéma modulu

zařízení stačí k tomuto procesoru připojit obrazový senzor a žádné další součásti jako vyrovnávací paměť nebo obvody pro podporu USB nejsou potřeba. OV538 také nabízí jednokanálový zvukový vstup pro možnost rozšíření kamery o mikrofon.
Funkce obrazového procesoru:

- zpracování dat z RGB na YUV
- vylepšení hran
- úpravy odstínu a sytosti
- podvzorkování a možnost výběru okna z obrazu
- digitální zvětšení nebo zmenšení rozlišení obrazu
- kompenzace vlivu objektivu
- digitální efekty: odstíny šedi, negativ, sepia
- úprava ostrosti a světlosti
- oprava bílých pixelů



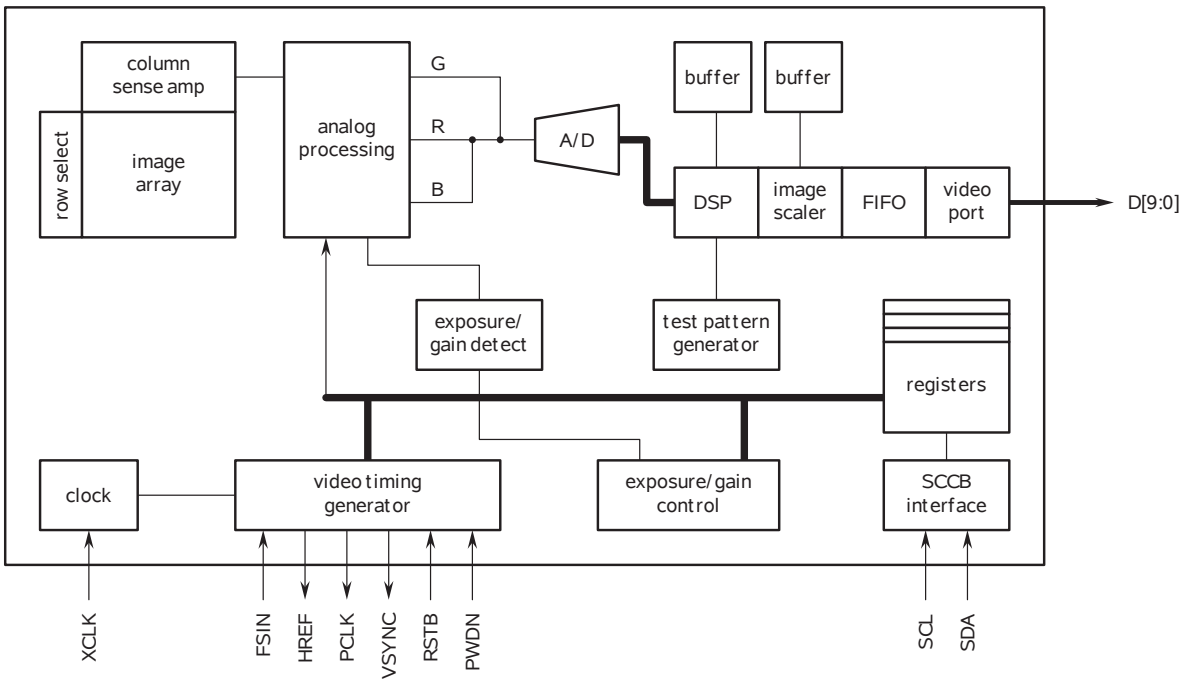
Obrázek 7.2: Blokové schéma USB můstku [5]

7.3 Videosenzor

Čip OV7221 je nízkonapěťový CMOS obrazový senzor, který poskytuje v malém pouzdře plnou funkčnost VGA kamery a obrazového procesoru. Nabízí plné rozlišení snímku, podvzorkovaný snímek nebo vybrané okno ze snímku. Výstup je 8 nebo 10 bitový ve velkém množství možných formátů. Pro jeho ovládání slouží sériové rozhraní označované SCCB a má následující vlastnosti:

- vysoká citlivost pro použití při špatném osvětlení,
- standardní řídicí rozhraní SCCB,
- výstupní formáty obrazu: RAW RGB, RGB (GRB 4:2:2, RGB565/555/444) a YCbCr (4:2:2),
- rozlišení obrazu: VGA,QVGA a jakákoliv velikost od CIF až po 40x30,
- automatické funkce zpracování obrazu: automatické řízení expozice (AEC), automatické řízení zisku (AGC), automatické vyvážení bílé (AWB), automatický pásmový filtr (ABF), automatická kalibrace úrovně černé (ABLC),

- řízení kvality obrazu zahrnující sytost, odstín, gamma korekci, ostrost (vylepšení hran) a antiblooming,
- vestavěný signálový procesor nabízí redukci šumu a řadu dalších vylepšení obrazu.



Obrázek 7.3: Blokové schéma obrazového senzoru [6]

7.4 Sběrnice SCCB

Sběrnice SCCB je v základu dvouvodičová sběrnice sloužící pro řízení obrazových senzorů. Je velmi podobná sběrnici I2C a prakticky je jen rozšířena o signál výběru (*chip select*) podřízeného zařízení a signál pro jeho uspání (*power down*). Podle specifikace [4] by na sběrnici mělo být jen jedno podřízené zařízení, ale v kamerovém modulu je připojen jak obrazový senzor tak paměť EEPROM.

7.5 Připojení obrazového senzoru

Obrazový senzor se připojuje pomocí paralelního synchronního rozhraní. Toto rozhraní je v podstatě standardem. Je možné takto připojit senzor zde k USB můstku nebo v jiných aplikacích třeba k signálovému procesoru (některé signálové procesory přímo toto rozhraní podporují).

Datové rozhraní je 10 nebo 8 bitové. Data se přenášejí jen směrem z obrazového senzoru. Pro synchronizaci slouží jednak signál `PCLK` (*pixel clock*) a dále signály horizontální a vertikální synchronizace.

7.6 Zpětné inženýrství

Zpětné inženýrství je při vývoji linuxových ovladačů velmi časté, protože řada tvůrců hardwaru vlastní ovladač pro Linux nenabízí a ani nechce poskytnout potřebné informace k jeho vytvoření.

Podobně tomu je i s vývojovým modulem, na kterém by měla být založena část vyvíjeného inteligentního videosenzoru. Výrobce poskytl jen velmi stručnou dokumentaci. V této dokumentaci jsou popsány jen základní vlastnosti a stručně registry zařízení. Rozhraní USB kamerového můstku nebylo popsáno vůbec.

7.6.1 Použitý software v Linuxu

Pro získání informací o kamerovém kitu byl v Linuxu použit program `lsusb`. Samotný příkaz bez parametrů vypíše všechna zařízení připojená na sběrnici. Pro detailnější výpis, který je uvede níže, jsou potřeba parametry `-v` (podrobný výpis) a `-d 05a9:a538` (specifikace zařízení, VID:PID). Pro plnou funkčnost je vhodné být přihlášen jako superuživatel.

7.6.2 Použitý software ve Windows

Ve Windows byl k dispozici ovladač, jednoduchý software pro nastavení kamery a pro přehrávání videa. Pro záznam komunikace po sběrnici USB byl použit software

USB Monitor [2]. Byla zaznamenána komunikace při fyzickém připojení zařízení do počítače, komunikace po spuštění obslužného softwaru a pak při přehrávání videa.

7.7 Deskriptor USB zařízení kamery

Podrobný výpis deskriptoru USB zařízení kamery získaný programem `lsusb` je následující:

```
Bus 002 Device 007: ID 05a9:a538 OmniVision Technologies, Inc.
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x05a9 OmniVision Technologies, Inc.
  idProduct          0xa538
  bcdDevice            1.00
  iManufacturer           1 OmniVision Application Team
  iProduct                2 OV538 USB camera-013008-1
  iSerial                 0
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           55
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0x80
      (Bus Powered)
    MaxPower              500mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           2
      bInterfaceClass       255 Vendor Specific Class
      bInterfaceSubClass      0
      bInterfaceProtocol      0
      iInterface              0
      Endpoint Descriptor:
        bLength                 7
```

```
    bDescriptorType         5
    bEndpointAddress     0x83  EP 3 IN
    bmAttributes            3
      Transfer Type            Interrupt
      Synch Type               None
      Usage Type               Data
    wMaxPacketSize     0x0004  1x 4 bytes
    bInterval              10
  Endpoint Descriptor:
    bLength                 7
    bDescriptorType         5
    bEndpointAddress     0x81  EP 1 IN
    bmAttributes            5
      Transfer Type            Isochronous
      Synch Type               Asynchronous
      Usage Type               Data
    wMaxPacketSize     0x0000  1x 0 bytes
    bInterval               1
Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        0
  bAlternateSetting       1
  bNumEndpoints           2
  bInterfaceClass       255 Vendor Specific Class
  bInterfaceSubClass      0
  bInterfaceProtocol      0
  iInterface              0
  Endpoint Descriptor:
    bLength                 7
    bDescriptorType         5
    bEndpointAddress     0x83  EP 3 IN
    bmAttributes            3
      Transfer Type            Interrupt
      Synch Type               None
      Usage Type               Data
    wMaxPacketSize     0x0004  1x 4 bytes
    bInterval              10
  Endpoint Descriptor:
    bLength                 7
    bDescriptorType         5
    bEndpointAddress     0x81  EP 1 IN
    bmAttributes            5
      Transfer Type            Isochronous
      Synch Type               Asynchronous
      Usage Type               Data
    wMaxPacketSize     0x13fc  3x 1020 bytes
    bInterval               1
```

```
Device Qualifier (for other device speed):
  bLength                10
  bDescriptorType         6
  bcdUSB               2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         8
  bNumConfigurations      1
Device Status:     0x0000
  (Bus Powered)
```

Hodnota parametru `bDeviceClass` je 00h a to znamená, že třída zařízení je specifikována na úrovni rozhraní. Zařízení obsahuje jedno rozhraní a dvě jeho alternativní konfigurace lišící se velikostí izochronního koncového bodu 1. Tento koncový bod představuje směr dat ze zařízení do počítače. Naneštěstí má parametr `bDeviceClass` specifikovaný u zařízení hodnotu 255 (FFh), jde o rozhraní specifikované výrobcem a opět nelze určit bližší popis. Situace by byla lepší, kdyby byla v deskriptoru specifikována některá třída USB zařízení (nejlépe *USB Video Class* nebo *USB Still Image Class*. Není tomu tak a je nutné hledat pomoc jinde.

Lze předpokládat, že obrazová data bude možné získat z jediného izochronního koncového bodu. Jelikož v prvním alternativním nastavení je jeho velikost nulová, bude nutné použít alternativní nastavení druhé (parametr `bAlternateSetting` roven 1, číslováno od 0).

7.8 Rozbor zachycených přenosů ve Windows

Ve Windows bylo pomocí programu *USB Monitor* [2] zachyceno velké množství dat. Program je umožňuje přehledně exportovat do formátu HTML, kde je možné je dále zkoumat. Zachytávání bylo zapnuto ještě před připojením kamery, aby bylo vidět celou inicializaci kterou provádí ovladač. Další blok dat byl zachycen při zapnutí dodaného programu s kamerou a nakonec poslední blok při spuštění přehrávání.

7.8.1 Data zachycená po připojení kamery

Po připojení kamery byl nejprve zachycen požadavek systému o deskriptor zařízení. Následovalo vyřízení této žádosti. Daleko zajímavější a důležitější jsou ihned potom zachycené kontrolní přenosy, které řídí již ovladač kamery ve Windows a slouží k inicializaci kamery. Jeden zachycený kontrolní přenes vypadá následovně:

```
000015: Control Transfer (UP), 15.06.2009 13:07:33.126 +0.0. Status: 0x00000000
Pipe Handle: 0x818e6020

 12

Setup Packet

 40 01 00 00 F2 00 01 00                          @...ò...


Recipient: Device
Request Type: Vendor
Vendor Direction: Host->Device
Request: 0x1 (Unknown)
Value: 0x0
Index: 0xf2
Length: 0x1
```

Uvedeným přenosem se zapíše hodnota 12h do registru na adrese F2h.

7.8.2 Data zachycená po zapnutí obslužného programu

Po zapnutí programu dodaného výrobcem k přehrávání videa z kamery dojde ke stejným přenosů jako po připojení kamery. Těsně před počátkem izochronních přenosů (než začne přenos videa) komunikuje kamera opět řadou kontrolních přenosů, kterými pravděpodobně nastavuje parametry důležité pro přenos.

7.8.3 Data zachycená při přenosu obrazu

Během přehrávání videa proudí velké množství dat, ze kterých nelze příliš mnoho vyčíst. Asi nejdůležitější poznatkem je, že pakety opravdu začínají bytem s hodnotou 12 a to odpovídá zpracování paketů linuxovým ovladačem pro čip OV534, jak je napsáno dále.

7.9 Průzkum podobných ovladačů

Velkou výhodou Linuxu je, že prakticky všechny jeho části jsou šířeny pod licencí GPL. Díky tomu je možné volně studovat a upravovat jeho součásti včetně ovladačů.

V jádře jsou už zahrnuty ovladače pro starší čipy firmy Omnivision (OV511, OV519, OV534). Ovladače jsou většinou pro více zařízení obsahující tyto čipy. Čip OV534 by dokonce měl být téměř totožný s čipem OV538 použitým v kameře.

Žádné zařízení s předchozími čipy nebylo k dispozici, ale studiem ovladačů šlo mnohé určit. Nejpodrobněji byl zkoumán ovladač pro OV534 (kvůli podobnosti s OV538). Podařilo se zjistit, že v ovladači je využíván přenos typu *bulk*, který vůbec není uveden v deskriptoru kamerového kitu. Z toho bylo usouzeno, že konfigurovatelný deskriptor kamery bude asi uložen v paměti EEPROM, která je připojena k USB můstku.

Ve zkoumaném ovladači byly i funkce pro zápis a čtení do registrů můstku a zprostředkovaně i na sběrnici SCCB. Tyto funkce využívají kontrolních přenosů a odpovídají zjištěním, která vznikla při zkoumání přenosů ve Windows. Tyto funkce jsou téměř bez změny využity dále. Způsob nastavování můstku i obrazového senzoru je tedy znám.

Otázkou zůstalo, jak získat obrazová data. Zkoumal se opět ovladač OV534, kde jsou zpracovávány pakety z USB přenosu tak, že se otestuje první byte v paketu. Nese-li tento byte hodnotu 12, pokračuje se ve zpracování dál. Ve snaze zjistit více informací došlo ke studiu tříd USB zařízení, které by mohly mít co do činění s obrazem (tj. třída *Video* a třída *Still Image*). V dokumentaci byla nalezena struktura hlavičky pro obrazové přenosy. V této struktuře první byte značí velikost hlavičky. V několika případech je tato velikost opravdu rovna 12. Této podobnosti je využito i dále při obsluze izochronních přenosů.

Kapitola 8

Ovladač s vlastním rozhraním

Důležitým krokem při tvorbě modulu nějakého ovladače je návrh vhodného mechanizmu pro řízení zařízení a přenosy většího množství dat, jako je to v tomto případě.

Na začátku tvorby tohoto ovladače nebylo známo, jak sestavit celý snímek z proudu dat izochronních přenosů a ani v jakém obrazovém formátu jsou tato data prezentována. Celé vlastní rozhraní je tomuto podřízeno a modul je postaven tak, aby bylo možné co nejvíce experimentovat.

8.1 Návrh rozhraní

Rozhraní modulu bylo nutné vytvořit tak, aby se snadno dalo použít v uživatelském softwaru, který je realizován jinou firmou.

V rámci úplného oddělení uživatelského softwaru zpracovávajíciho obraz bylo nutné vytvořit knihovnu, která by byla mezivrstvou uživatele a modulu jádra a poskytovala by potřebné funkce pro nastavení a získání obrazu.

Pro prací s nastavením kamery, čtení a zápis konfiguračních parametrů modulu byl vybrán mechanizmus `ioctl`. Je vhodný pro přenos jednoduchých parametrů nebo ukazatelů na složitější struktury mezi uživatelským prostorem a prostorem jádra. Zde byl použit pro řízení ovladače, čtení a zápis do registrů můstku a obrazového senzoru.

Pro získání obrazu byla do modulu začleněna funkce `read`. Po otevření souboru zařízení se přečte celý jeho obsah, který je tvořen obrazovými daty, a následně se opět zavře. Tímto se získá jeden snímek z kamery.

8.1.1 Uživatelská knihovna vlastního rozhraní

Pro úplné oddělení použité kamery a softwaru, který zpracovává obraz, byla vyžadována mezivrstva v podobě knihovny. Do této knihovny byla zamaskována některá volání `ioctl` do podoby čtení a nastavení potřebných parametrů kamery. Dále je zde funkce pro získání obrazu z kamery.

Příklad funkce z knihovny pro čtení obrazu:

```
int
cam_read_picture(unsigned short *picture,
         unsigned short *gain, unsigned short *expo)
{
    int fp,length,i;
    unsigned char tmp[614400];

    fp = open("/dev/ov538", 'r');
    if (fp < 0) {
        return -ERCAM_DRIVER;
    } else {
        length = read(fp,tmp,614400);

        for(i=0;i<307200;i++){
            picture[i]=tmp[2*i];
        }


        close(fp);
    }
    return ERCAM_NO;
}
```

8.2 Implementace

Modul jádra se skládá z inicializačních funkcí modulu, funkcí pro nastavování registrů obou čipů, funkcí pro řízení izochronního přenosu a následné sestavení obrazu z paketů.

8.2.1 Zavedení modulu

Při zavádění modulu je jádrem zavolána funkce `ov538_init`, která má následující kód:

```
static int __init ov538_init(void)
{
    int retval = 0;

    //registrace ovldace USB
    retval = usb_register(&ov538_driver);
    if (retval) {
        DEBUG("usb_register error %d",retval);
        return retval;
    }

    //vytvoreni uzlu zarizeni
    retval = register_chrdev(251, DEVICE_NAME, &fops);
    if (!retval) {
        DEBUG("Assigned major number %d",251);
    } else {
        DEBUG("FATAL: No major assigned");
        return retval;
    }


    return 0;
}
```

Pomocí funkce `usb_register()`, které se předá adresa vyplněné struktury `ov538_driver` se zaregistruje ovladač USB zařízení. Pokud je pak zařízení připojeno, systém mu podle jeho identifikačních čísel VID a PID přiřadí tento ovladač a je zavolána funkce `ov538_probe()`, která je popsána níže.

Jelikož se nepředpokládá, že by bylo připojeno k systému více videokamer, je hned při načtení modulu vytvořen uzel znakového zařízení s pevně daným hlavním číslem 251. Pro testovací účely tento způsob nevadí, ale pro produkci a širší využití ovladače to není příliš vhodné. Snadno takto může dojít ke kolizi hlavních čísel.

8.2.2 Odebrání modulu

Při odebrání modulu z běžícího jádra je zavolána funkce `ov538_exit()`, která odstraní přípojný uzel zařízení a zruší vazbu USB subsystému na ovladač.

```
static void __exit ov538_exit(void)
{
    unregister_chrdev(device_major, DEVICE_NAME);
    usb_deregister(&ov538_driver);

}
```

8.2.3 Připojení zařízení

```
static int
ov538_probe(struct usb_interface *interface, const struct usb_device_id *id)
{
    struct usb_host_interface *iface_desc;
    struct usb_endpoint_descriptor *endpoint;
    int i, retval;

    DEBUG("driver probe");
    ov538_udev = usb_get_dev(interface_to_usbdev(interface));
    ov538_iface = interface;

    retval = usb_set_interface(ov538_udev,0,1);
    if (!retval) {
        DEBUG("change alternate setting retval = %d",retval);
    }

    iface_desc = interface->cur_altsetting;

    for (i = 0; i < iface_desc->desc.bNumEndpoints; ++i) {
        endpoint = &iface_desc->endpoint[i].desc;

        if (usb_endpoint_is_isoc_in(endpoint)) {
            isoc_in_endpointAddr = endpoint->bEndpointAddress;
            DEBUG("found isoc in endpoint, size);

        }
    }
    transfer_buffer = kmalloc(16, GFP_KERNEL);
    if (!transfer_buffer) {
        DEBUG("could not allocate transfer buffer");
        return -ENOMEM;
    } else {
        DEBUG("transfer buffer created");
    }
    return 0;
}
```

8.2.4 Odpojení zařízení

```
static void
ov538_disconnect(struct usb_interface *intf)
{
    DEBUG("driver disconnect");
    destroy_urbs();
    kfree(transfer_buffer);
}
```

8.2.5 Zápis do registru obvodu OV538

Zápis do registru obvodu OV538 se provádí kontrolním přenosem po USB. Adresa registru je nastavena v parametru *offset* a hodnota je uložena v bufferu. Pro sestavení URB je zde využito makra `usb_control_msg`.

```
static void ov538_reg_write(u16 reg, u8 val)
{
    int ret;

    DEBUG("reg write: reg=0x%04x, val=0x%02x", reg, val);
    transfer_buffer[0] = val;
    ret = usb_control_msg(ov538_udev,
                  usb_sndctrlpipe(ov538_udev, 0),
                  0x1,
                  USB_DIR_OUT | USB_TYPE_VENDOR | USB_RECIP_DEVICE,
                  0x0, reg, transfer_buffer, 1, CTRL_TIMEOUT);
    if (ret < 0)
        DEBUG("reg write failed");
}
```

8.2.6 Čtení z registru obvodu OV538

Čtení z registrů je obdobné zápisu.

```
static u8 ov538_reg_read(u16 reg)
{
    int ret;

    ret = usb_control_msg(ov538_udev,
                  usb_rcvctrlpipe(ov538_udev, 0),
                  0x1,
                  USB_DIR_IN | USB_TYPE_VENDOR | USB_RECIP_DEVICE,
                  0x0, reg, transfer_buffer, 1, CTRL_TIMEOUT);
    DEBUG("reg read: reg=0x%04x, data=0x%02x", reg, transfer_buffer[0]);
    if (ret < 0)
        DEBUG("reg read failed");
```

```
    return transfer_buffer[0];
}
```

8.2.7 Ovládání LED na modulu

Ovládání LED se provádí změnou registru příslušejícího k sadě univerzálních vstupů a výstupů (GPIO).

```
static void ov538_LED(u16 val)
{
    u8 stat;
    DEBUG("LED: %d", val);

    stat = ov538_reg_read(0x21);
    stat |= 0x80;
    ov538_reg_write(0x21, stat);

    stat = ov538_reg_read(0x23);
    if (val)
        stat |= 0x80;
    else
        stat &= ~(0x80);

    ov538_reg_write(0x23, stat);
}
```

8.2.8 Kontrola stavu SCCB

Při použití sběrnice SCCB je potřeba kontrolovat registr jejího stavu v obvodu OV538, aby bylo zaručeno, že se operace úspěšně provedla.

```
static int sccb_check_status(void)
{
    u8 data;
    int i;

    for (i = 0; i < 5; i++) {
        data = ov538_reg_read(OV538_REG_STATUS);

        switch (data) {
        case 0x00:
            return 1;
        case 0x04:
            return 0;
        case 0x03:
            break;
        default:
```

```
            DEBUG("sccb status 0x%02x, attempt %d/5",data, i + 1);
        }
    }
    return 0;
}
```

8.2.9 Zápis do registru obrazového senzoru

Zápis do registru obrazového senzoru probíhá nepřímo nastavení několika registrů obvodu OV538.

```
static void sccb_reg_write(u16 reg, u8 val)
{
    DEBUG("sccb reg: 0x%04x, val: 0x%02x", reg, val);
    ov538_reg_write(OV538_REG_ADDRESS, 0x42);
    ov538_reg_write(OV538_REG_SUBADDR, reg);
    ov538_reg_write(OV538_REG_WRITE, val);
    ov538_reg_write(OV538_REG_OPERATION, OV538_OP_WRITE_3);

    if (!sccb_check_status())
        DEBUG("sccb_reg_write failed");
}
```

8.2.10 Čtení z registru obrazového senzoru

Čtení z registrů obrazového senzoru je obdobé zápisu.

```
static u8 sccb_reg_read(u16 reg)
{
    DEBUG("sccb reg: 0x%04x", reg);
    ov538_reg_write(OV538_REG_ADDRESS, 0x42);
    ov538_reg_write(OV538_REG_SUBADDR, reg);
    ov538_reg_write(OV538_REG_OPERATION, OV538_OP_WRITE_2);
    if (!sccb_check_status())
        DEBUG("sccb_reg_read failed 1");

    ov538_reg_write(OV538_REG_OPERATION, OV538_OP_READ_2);
    if (!sccb_check_status())
        DEBUG("sccb_reg_read failed 2");

    return ov538_reg_read(OV538_REG_READ);
}
```

8.2.11 Inicializace kamery

```
static void ov538_setup(void)
```

```
{
    int i;

    //inicializace USB mustku
    for (i = 0; i < ARRAY_SIZE(ov538_initdata); i++)
        ov538_reg_write(ov538_initdata[i][0], ov538_initdata[i][1]);

    DEBUG("sensor is ov%02x%02x", sccb_reg_read(0x0a), sccb_reg_read(0x0b));

    //inicializace obrazového senzoru
    for (i = 0; i < ARRAY_SIZE(sen_initdata); i++)
        sccb_reg_write(sen_initdata[i][0], sen_initdata[i][1]);
}
```

Vždy se jedná o dvojici bajtů (adresa registru v daném obvodu a hodnota).

8.2.12 Obsluha volání `ioctl`

Voláním `ioctl()` jsou předávány dva parametry z uživatelského prostoru. Jedním je příkaz a druhým je argument tohoto příkazu. Je možné takto předávat konkrétní hodnoty nebo ukazatele. Zde se předává číslo příkazu a parametr příkazu . Zápis nebo čtení dat registrů je prováděno dvojím voláním. Nejprve se nastaví data do pomocné proměnné a druhým voláním se provede zápis těchto dat do registru. Pro čtení je postup opačný.

Implementace volání je následující:

```
static int
fops_ioctl(struct inode *inode, struct file *file, uint cmd, ulong arg)
{
    DEBUG("file ioctl");

    switch(cmd){
    case OV538_IOCTL_REGR:
    {
        DEBUG("Ioctl: register read");
        return ov538_reg_read((u16)arg);
    }
    case OV538_IOCTL_REGW:
    {
        DEBUG("Ioctl: register write");
        ov538_reg_write(arg,ioctl_data);
        return 0;
```

```
    }
    case OV538_IOCTL_DATA:
    {
        DEBUG("Ioctl: set data");
        ioctl_data = (u8)arg;
        return 0;
    }
    case OV538_IOCTL_SCCB_REGW:
    {
        DEBUG("Ioctl: sccb register write");
        sccb_reg_write(arg,ioctl_data);
        return 0;
    }
    case OV538_IOCTL_SCCB_REGR:
    {
        DEBUG("Ioctl: sccb register read");
        return sccb_reg_read((u16)arg);
    }
    case OV538_IOCTL_SETUP:
    {
        DEBUG("Ioctl: setup");
        ov538_setup();
        return 0;
    }
    case OV538_IOCTL_LED:
    {
        DEBUG("Ioctl: LED %d",arg);
        ov538_LED(arg);
        return 0;
    }
    default:
        DEBUG("Ioctl: unsuported command %d", cmd);
        return -ENOIOCTLCMD;
    }
    return 0;
}
```

Pro snadnější použití je vytvořen hlavičkový soubor `ov538.h` s následujícím obsahem:

```
/* IOCTL operations*/

//precteni registru pres ioctl
#define OV538_IOCTL_REGR      1

//zapis do registru, nejprve je nutne nastavit data pres OV538_IOCTL_DATA
#define OV538_IOCTL_REGW      2
```

```
//prenos dat ovladaci pro dalsi pouziti
#define OV538_IOCTL_DATA      3

//cteni z~registru obrazoveho senzoru
#define OV538_IOCTL_SCCB_REGR      4

//zapis do registru CMOS senzoru,
//nejprve je nutne nastavit data pres OV538_IOCTL_DATA
#define OV538_IOCTL_SCCB_REGW      5

//nastaveni LED, 0 nesviti, 1 sviti
#define OV538_IOCTL_LED        20

#define OV538_IOCTL_SETUP      254
```

8.2.13 Obsluha přerušení izochronního přenosu

Při vyřízení požadavku URB izochronního přenosu je zavolána funkce `ov538_isoc_irq` zaregistrovanápři sestavení URB. Zde dojde k otestování paketu na jeho délku a pokud je nenulový, předá se jeho obsah na další zpracování.

```
static void ov538_isoc_irq(struct urb *urb)
{
  int i,n;
  unsigned char *data;
  DEBUG("isoc irq");

  //pro vsechny pakety v~URB:
  for (i = 0; i<urb->number_of_packets; i++){
    //zpracovava se kazdy pake s~nenulovou velikosti
    if (urb->iso_frame_desc[i].actual_length > 0) {
      data = urb->transfer_buffer + urb->iso_frame_desc[i].offset;
      packet_to_frame(data,urb->iso_frame_desc[i].actual_length);
    }

    //priprava pro dalsi odeslani
    urb->iso_frame_desc[i].actual_length = 0;
    urb->iso_frame_desc[i].status = 0;
  }

  if (usb_submit_urb(urb, GFP_ATOMIC)) {
            DEBUG("error submitting URB");
  }
}
```

8.2.14 Sestavení obrazu

Sestavení snímku probíha postupným předáváním obsahu paketů. Pakety mají hlavičku dlouhou 12 bajtů a její struktura je:

HL	HL							
BFH	EOH	ERR	STI	RES	SCR	PTS	EOF	FID
PTS	PTS [7:0]							
	PTS [15:8]							
	PTS [23:16]							
	PTS [31:24]							
SCR	SCR [7:0]							
	SCR [15:8]							
	SCR [23:16]							
	SCR [31:24]							
	SCR [39:32]							
	SCR [47:40]							

Tabulka 8.1: Hlavička obrazového USB paketu, řádky představují jeden byte

Popis jendotlivých polí:

- HL - *header length* - délka hlavičky,
- BFH - *bit field* - bitové příznaky,
  - EOH - *end of header* - značí konec pole příznakových bitů,
  - ERR - *error bit* - indekace chyby,
  - STI - *still image* - nastaveno pokud snímek není video,
  - RES - *reserved* - urřeno pro budoucí použití,
  - SCR - *source clock reference* - přítomnost pole SCR,
  - PTS - *presentation time stamp* - přítomnost pole PTS,
  - EOF - *end of frame* - indikace konce snímku, nese poslední paket ve snímku,
  - FID - *frame identifier* - s každým dalším snímkem se hodnota tohoto bitu mění, je konstantní po celou dobu trvání snímku,

- PTS - *presentation time stamp* - délka 4 bajty, po celou dobu snímku se nemění,
- SCR - *source clock reference* - toto pole má délku 6 bajtů a jeho přítomnost indikuje bit SCR.

Zpracování paketů provádí následující funkce:

```
static void packet_to_frame(unsigned char *data, int length)
{
  __u32 this_pts;
  __u32 this_fid;
  this_pts = (data[5] << 24) | (data[4] << 16) | (data[3] << 8) | data[2];
  this_fid = (data[1] & UVC_STREAM_FID) ? 1 : 0;

  //ladici vypis priznaku
  if(data[0]==12){
    DEBUG("EOH:%d, ERR:%d, STI:%d, RES:%d, SCR:%d, PTS:%d, EOF:%d, FID:%d",
                    (data[1] & UVC_STREAM_EOH) ? 1 : 0,
                    (data[1] & UVC_STREAM_ERR) ? 1 : 0,
                    (data[1] & UVC_STREAM_STI) ? 1 : 0,
                    (data[1] & UVC_STREAM_RES) ? 1 : 0,
                    (data[1] & UVC_STREAM_SCR) ? 1 : 0,
                    (data[1] & UVC_STREAM_PTS) ? 1 : 0,
                    (data[1] & UVC_STREAM_EOF) ? 1 : 0,
                    (data[1] & UVC_STREAM_FID) ? 1 : 0);

    //hlavicka zacina cislem 12
    if (data[0] != 12 || length < 12) {
        DEBUG("bad header");
        return;
    }

    // test na priznak chyby
    if (data[1] & UVC_STREAM_ERR) {
        DEBUG("payload error");
        return;
    }

    // Extract PTS and FID
    if (!(data[1] & UVC_STREAM_PTS)) {
        DEBUG("PTS not present");
        return;
    }

    //pokud se zmeni PTS nebo FID, zacina novy snimek
    if (this_pts != last_pts || this_fid != last_fid) {
        //pridat prvni paket
        last_pts = this_pts;
```

```
        last_fid = this_fid;
        frame_length = 0;
        DEBUG("start of frame");
    }


    //pridani obsahu paketu ke snimku
    if ((frame_length + length) <= IMG_SIZE)
      memcpy(tmp_data + frame_length,data+12,length-12);

    frame_length += length - 12;

    //konec snimku, zkopiruje se cely snimek
    if (data[1] & UVC_STREAM_EOF) {
        last_pts = 0;
        DEBUG("end of frame - length %d", frame_length);
        if(frame_length == 614400){
            memcpy(img_data,tmp_data,IMG_SIZE);
        }
    }
}
```

8.3 Použití

Rozhraní představuje jediný soubor znakového zařízení, který se musí ručně vytvořit příkazem `mknod`. Hlavní číslo nutné pro vytvoření je po načtení modulu v ladících zprávách dostupných příkazem `dmesg`.

Ovládání je provedeno přes systémové volání `ioctl()` a čtení obrazu pomocí `open()`, `read()` a `close()`.

8.3.1 Získání obrazu

Obraz je získán otevřením a přečtením celého souboru zařízení. Soubor je nutné zavřít a pro další čtení znovu otevřít, aby došlo k zachycení nového snímku z kamery.

8.3.2 Zobrazení obrazu

Pro zobrazení dat při testování byl vybrán software MATLAB, protože práce s obrazovými daty je v něm velmi snadná a obsahuje velké množství funkcí pro jeho zpracování.

Ve chvíli, kdy se podařilo získat z kamery první data, která už byla vhodná k zobrazení, potvrdila se vhodnost volby MATLABu. Formát těchto dat nebyl přesně znám a bylo nutné vyzkoušet je zobrazit jako několik možných formátů. Konečný kód pro zobrazení obrazu je následující:

```
fid = fopen('/dev/ov538','r');
F = fread(fid, 'uchar')';
fclose(fid);

A = zeros(480,640);
z = 1;
for i = 1:480
    for j = 1:640

       A(i,j) = F(z)/255;

       z = z + 2;

    end
end
imshow(A);
```

Kapitola 9

Ovladač s rozhraním V4L2

Přestože předchozí vlastní rozhraní dostačovalo, mělo určité nedostatky. Hlavním nedostatkem byla potřeba speciálního softwaru pro vizualizaci obrazu z kamery. Dále bylo potřeba při zavedení modulu ručně vytvořit soubor zařízení. Tato činnost byla sice automatizována po startu systému skriptem, ale neobešla se bez problémů a v závislosti na použité linuxové distribuci docházelo ke kolizím hlavních čísel zařízení. Problém s vytvářením souboru zařízení lze sice řešit pomocí systému *udev*, ale ostatní nedostatky zůstávají a lepší je využít mechanizmů V4L. Nevýhodou předchozího ovladače může být i to, že nelze připojit více stejných kamer k jednomu systému. U plánovaného videosenzoru na tom nezáleží, protože k němu nebude připojeno více kamer, ale obecně to nedostatek je.

Snahou bylo rozhraní nějakým způsobem standardizovat tak, aby se vývojový kit v Linuxu choval jako běžná webkamera. K tomuto postupu je v Linuxu určen subsystém Video for Linux. Protože už existující ovladač pro USB můstek OV538 existoval a využíval mezivrstvy GSPCA, byl i nově vznikající ovladač pro vývojový kit přepracován podobně. Rozdílem bylo použití izochronních přenosů, narozdíl od přenosu typu bulk u existujícího ovladače. Dalším zásadním rozdílem bylo nastavení a inicializace kamery, které se použily z prvního typu ovladače s vlastním rozhraním.

Tímto vznikl ovladač, který se automaticky zavedl po připojení kamery a vytvořil soubor zařízení `/dev/videoX`. Nyní lze k přehrávání videa z kamery použít běžné programy k tomu určené. Podrobnosti o získání obrazu jsou uvedeny dále.

9.1 Implementace

Obsluha hardwaru kamery je implementováno velmi podobně jako u vlastního rozhraní. Aby byl ovladač univerzální a použitelný pro více stejných zařízení, je společně s voláním každé funkce předáván ukazatel `gspca_dev`, který ukazuje na strukturu týkající se konkrétního zařízení. Tato struktura je vytvářena dynamicky při připojení zařízení.

9.1.1 Nastavení struktury `v4l2_pix_format`

Při využití technologii Video for Linux je nutné nastavit strukturu `v4l2_pix_format`, která nese parametry obrazu.

```
static const struct v4l2_pix_format vga_mode[] = {
    {640, 480, V4L2_PIX_FMT_YUYV, V4L2_FIELD_NONE,
     .bytesperline = 640 * 2,
     .sizeimage = 640 * 480 * 2,
     .colorspace = V4L2_COLORSPACE_JPEG,
     .priv = 0},
};
```

9.1.2 Struktura `sd_desc` ovladače GSPCA

Při použití ovladače GSPCA je nutné vyplnit strukturu `sd_desc`. Tato struktura vytváří rozhraní ovladače GSPCA s vlastní ovladačem hardwaru kamery. Zahrnuje ukazatele na funkce, které se mají provést při inicializaci kamery, při začátku a konci přehrávání atd.

```
static const struct sd_desc sd_desc = {
    .name     = MODULE_NAME,
    .ctrls    = sd_ctrls,
    .nctrls   = ARRAY_SIZE(sd_ctrls),
    .config   = sd_config,
    .init     = sd_init,
    .start    = sd_start,
    .stopN    = sd_stopN,
    .pkt_scan = sd_pkt_scan,
    .get_streamparm = sd_get_streamparm,
    .set_streamparm = sd_set_streamparm,
};
```

9.1.3 Funkce `sd_config()`

Další důležitou funkcí pro GSPCA je `sd_config`. Zde se nastavují parametry USB přenosu. Pokud je `bulk_size` rovno nule použije se izochronní přenos.

```
static int
sd_config(struct gspca_dev *gspca_dev,
              const struct usb_device_id *id)
{
    struct cam *cam;

    cam = &gspca_dev->cam;

    cam->epaddr = 0x81;
    cam->cam_mode = vga_mode;
    cam->nmodes = ARRAY_SIZE(vga_mode);

    cam->bulk_size = 0;
    cam->bulk_nurbs = 0;

    return 0;
}
```

9.2 Přehrávání videa z kamery

Díky rozhraní V4L se ovladač určitým způsobem standardizoval a je možné kameru používat nejen jednoúčelově se softwarem na rozpoznávání osob, ale lze ji použít jako jakoukoliv jinou webkameru fungující v Linuxu.

9.2.1 Program MPlayer

Program MPlayer [3] je filmový přehrávač pro Linux (spustitelný na mnoha platformách a CPU architekturách). Přehraje většinu multimediálních souborů s podporou mnoha nativních i binárních kodeků.

MPlayer podporuje velké množství výstupních audio a video rozhraní. Využívat je možné X11 nebo lze přehráva pouze s pomocí frame bufferu.

9.2.2 Využití MPlayeru pro webkameru

MPlayer lze použít i pro přehrávání ze zařízení na zachytávání videa (s rozhraním v4l2). Nastavení se provede volbou `-tv` s patřičnými parametry. Přehrát video z kamery je pak možné příkazem `mplayer -tv driver=v4l2:device=/dev/video tv://`.

Tohoto způsobu bylo použito pro otestování dlouhodobé stability v cílovém zařízení než bylo možné využít uživatelskou knihovnu s dalším softwarem na zpracování obrazu. Pro přehrávání pomocí MPlayeru s tímto ovladačem není potřeba uživatelská knihovna.

Kapitola 10

Závěr

Úkolem této bakalářské práce bylo vytvoření linuxového ovladače webkamery, kterou tvoří vývojový kit od společnosti Omnivision. Kamera založená na kitu má být součástí inteligentního senzoru pro rozpoznávaní osob a mimo jiné třeba případnému určení jejich počtu v místnosti.

Ovladač se podařilo vytvořit k dostačující funkci zařízení, ale v některých místech muselo dojít ke kompromisu. Snaha byla o monochromatický výstupní obraz bez komprese s hloubkou 10 bitů, kterého by mělo jít dosáhnout podle specifikací použitých integrovaných obvodů.

Největším problémem při tvorbě ovladače byl nedostatek informací o kitu kamery Omnivision. Bylo nutné zkoumat velké množství dat získaných pod operačním systémem Windows a následně testovat různé konfigurace v ovladači v Linuxu.

Výchozím bodem pro ovladač kamery Omnivision byl ovladač pro některé kamery s čipem OV534, který již byl součástí jádra. Čipy OV534 a OV538 jsou velmi podobné, ale nevýhodou byla konfigurace a programový kód uložený v paměti EEPROM kamerového kitu. Tím bylo značně změněno rozhraní kitu na sběrnici USB.

Nepodařilo se úplně dosáhnout požadované konfigurace videokamery s ohledem na formát video dat a je nutné spokojit se s formátem YUYV o hloubce 8 bitů. Tento formát trochu postrádá smysl, když je kamera osazena monochromatickým senzorem, a je vhodný spíš při použití barevného senzoru. Výsledný obraz i tak plně vyhovuje a je možné ho použít pro další zpracování.

Díky tomuto formátu je přenášeno dvojnásobné množství dat na jeden snímek, ale to nezpůsobuje potíže. Po příjetí snímku je jen nutné ho přetransformovat na monochromatický. Vytížení systému touto transformací, která zde spočívá ve vynechání každého druhého bajtu obrazové informace, je zanedbatelné.

Jako první bylo vytvořeno rozhraní ovladače podle vlastního návrhu. Tento postup vypadal snadnější a nevyžadoval znalost sytému v4l. Druhou výhodou v tomto případě bylo to, že byl ze začátku neznámý formát obrazu, který se podařilo z kamery získat. Testováním různých způsobů zobrazení získaných obrazových dat byl určen jejich formát (YUYV).

Při dalším testování ovladače na cílovém zařízení, kde nebyl software MATLAB, použitý v prvním případě k zobrazení dat, byla snaha ovladač více standardizovat a byl upraven na rozhraní V4L2. Tímto krokem se vylepšilo i dynamické vytvoření souboru zařízení při připojení kamery za běhu systému a bylo možné využít k přehrávání videa při testování program MPlayer.

V konečné fázi, když byl celý inteligentní senzor sestaven, se objevily velké problémy se stabilitou systému. Buď docházelo k problémům při čtení obrazu nebo celý systém zatuhl. Prvním problémem se ukázala vzrůstající teplota systému. Vysoké teplotě se zamezilo namontováním větráčku do zařízení. Problémy se stabilitou setrvaly a v podezření z nestability byl ovladač. Při testování docházelo v systému k odpojení a připojení USB zařízení kamery, přestože byla stále fyzicky připojena. V samotném ovladači se problém nepodařilo odstranit a vypadalo, že může být někde na úrovni ovladače `usb_core` nebo na úrovni hardwaru. Při kompletaci zařízení byl nahrazen klasický USB kabel typu A-B, kterým se kamera připojovala, krátkým párem kroucených vodičů a dvěma vodiči pro napájení. Všechny tyto vodiče byly na obou stranách připájeny a konektory na zařízeních nevyužity. Ve snaze odstranit problémy se stabilitou, byly nahrazeny uvedené vodiče zpět USB kabelem A-B zapojeným do konektorů. V tomto zapojení bylo schopno zařízení pracovat desítky hodin bezchybně. Problém mohl být způsoben nevyhovující impedancí nebo chybějícím stíněním použitých vodičů. Připojení kamery k základnímu modulu je předmětem dalších úprav a testování.

Literatura

[1] *GNU General Public License, version 2.* [online, cit. 1.8.2009].
URL `http://www.gnu.org/licenses/gpl-2.0.html`

[2] *HHD Software, USB Monitor.* [program, cit. 5.8.2009].
URL `http://www.hhdsoftware.com/Products/home/usb-monitor.html`

[3] *MPlayer manual page.* [online, cit. 18.7.2009].
URL `http://www.mplayerhq.hu/DOCS/man/cs/mplayer.1.html`

[4] *OmniVision Serial Camera Control Bus, Functional Specification.* [online, cit. 5.8.2009].
URL `http://www.ovt.com/products/sccbspec_an_2_1.pdf`

[5] *OV538 Advenced Information Datasheet.* [online, cit. 28.7.2009].
URL `http://www.yuanxinfeng.com/upload/OV538-B88.pdf`

[6] *OV7720 Product Brief.* [online, cit. 28.7.2009].
URL `http://www.ovt.com/uploads/parts/OV7720_PB%20(1.11)_web.pdf`

[7] *Overview of the V4L2 driver framework.* [zdrojové kódy jádra Linuxu, soubor Documentation/video4linux/v4l2-framework.txt].

[8] *Still Image Capture Device Definition 1.0.* [online, cit. 5.8.2009].
URL `http://www.usb.org/developers/devclass_docs/usb_still_img10.zip`

[9] *Universal Serial Bus Specification Revision 2.0.* [online,cit.28.7.2009].
URL `http://www.usb.org/developers/docs/usb_20_052709.zip`

[10] *USB 2.0 - díl 2.* [online, cit. 20.7.2009].
URL http://hw.cz/Rozhrani/ART1244-USB-2.0---dil-2.html

[11] *USB Device Class Definition for Video Devices, Revision 1.1.* [online, cit. 5.8.2009].
URL http://www.usb.org/developers/devclass_docs/USB_Video_Class_1_1.zip

[12] Corbet, J.: *Linux Device Drivers.* O'Reilly Media, 2005, ISBN 0-596-00590-3.
URL http://lwn.net/Kernel/LDD3/

[13] Jelínek, L.: *Jádro systému Linux.* Praha: Computer Press, 2008, ISBN 978-80-251-2084-2.

[14] Richard Stones, N. M.: *Beginning Linux Programming, Second Edition.* United States: Wrox Press Ltd., 2001, ISBN 1-861002-97-1.

[15] Schimek, M.: *Video for Linux Two API Specification.* [online, cit. 25.6.2009].
URL http://v4l2spec.bytesex.org/spec-single/v4l2.html

[16] Tišnovský, P.: *Universální sériová sběrnice (USB).* [online, cit. 1.8.2009].
URL http://www.root.cz/clanky/universalni-seriova-sbernice-usb/

Obsah CD

Obsah přiloženého CD je následující:

- soubor `bp.pdf` - technická zpráva ve formátu PDF
- adresář `tex` - zdrojové soubory technické zprávy
- adresář `driver1` - zdrojové soubory ovladače s vlastním rozhraním
- adresář `driver2` - zdrojové soubory ovladače s rozhraním V4L2
- adresář `capture` - zachycená komunikace s kamerou ve Windows