České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Možnosti použití operačního systému GNU/Linux v zabezpečovací kameře EYE-02

Praha, 2009 **Jiří Vít**

Prohlášení

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

V Praze dne 15.5.2009

Jiří Vít
podpis

Poděkování

Děkuji vedoucímu své diplomové práce Ing. Pavlu Píšovi za ochotu ujmout se jejího vedení a za připomínky k práci, dále firmě JABLOCOM a jejím zaměstnancům Ing. Jiřímu Holingerovi a Ing. Jiřímu Bažantovi, PhD., za to, že mi umožnili podílet se na vývoji jejich výrobku a za jejich užitečné rady a Bc. Janu Breuerovi za jeho rady ohledně detailů operačního systému Linux.

Abstrakt

Diplomová práce se věnuje použití operačního systému Linux v bezpečnostní kameře JABLOCOM EYE-02 vybavené mikroprocesorem AT91SAM9260 firmy Atmel. Na začátku je stručně naznačen postup, jakým byl operační systém připraven pro použití v zařízení. Následuje popis významu ovladačů v systému Linux, jejich obvyklá struktura a postupy při vývoji.

V dalších kapitolách je popsán vývoj ovladačů pro některá zařízení přítomná na kameře, jmenovitě CMOS videosnímač a PIR detektor pohybu. Protože je videosnímač připojen k mikroprocesoru prostřednictvím specializovaného rozhraní ISI, je kromě ovladače pro samotný snímač prezentován také ovladač pro toto rozhraní, které zatím v Linuxu není podporováno.

V závěru práce jsou zhodnoceny dosažené výsledky a nastíněn směr dalšího vývoje.

Abstract

This diploma thesis focuses on usability of the Linux operation system in security camera JABLOCOM EYE-02. This camera is equipped with Atmel AT91SAM9260 microcontroller.

In the first part is shortly described the way how the operation system has been prepared for the use in this device. This is followed by some introduction to Linux device drivers, their usual structure and development methods.

Next chapters describe driver development for some devices used in the camera, this is CMOS video sensor and PIR motion sensor. The video sensor is connected to the microprocessor using specialised ISI interface, so the thesis presents two drivers – one for the video sensor a one for the ISI, which is in the kernel not yet supported.

The conclusion describes reached results and possibilities of future development.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Jiří Vít**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu: **Možnosti použití OS GNU/Linux v zabezpečovací kameře EYE-02**

Pokyny pro vypracování:

1. Prostudujte dokumentaci ke kameře EYE-02 a k mikroprocesoru AT91SAM9260, který je v kameře použit.
2. Prostudujte možnosti provozu operačního systému GNU/Linux na tomto mikroprocesoru.
3. Vytvořte linuxové drivery pro CMOS video senzor, PIR a pro USB mass storage.
4. Uveďte příklady a popis rozhranní pro přístupk driverům.
5. Drivery vyvíjete pod open source licencí.

Seznam odborné literatury:

1. Lukáš Jelínek, Jádro systému Linux - Kompletní průvodce programátora, Computer Press 2008
2. Embedded Linux kernel and driver development, http://free-electrons.com/training/drivers

Vedoucí: Ing. Pavel Píša

Platnost zadání: do konce zimního semestru 2009/10

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 27. 2. 2009

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato práce se zabývá použitím operačního systému Linux v bezpečnostní kameře EYE-02 firmy JABLOCOM v Jablonci nad Nisou. Kamera je v současnosti provozována bez operačního systému, proto jsem byl požádán, abych prozkoumal možnosti a přínos využití Linuxu.

Úlohu přenesení Linuxu do kamery jsem řešil v rámci týmového projektu v roce 2008 se svým kolegou Janem Breuerem. Při řešení jsme narazili na problém s příliš malou pamětí Data Flash, kterou byla kamera vybavena. Do této paměti se nevešel zkompilovaný binární obraz jádra systému. Kamera však obsahuje slot pro Micro SD kartu, kterou jsme pro bootování s úspěchem využili. Nyní je v paměti Data Flash umístěn pouze bootlader, který umí číst Micro SD kartu. Obraz jádra je na kartě uložen na přesně definované pozici, odkud ho bootloader zkopíruje do RAM a následně spustí, čímž dojde k nastartování systému.

Nyní je mým úkolem vytvořit drivery pro hardwarové prvky kamery – CMOS videosenzor, PIR detektor pohybu a USB velkokapacitní zařízení. První dva drivery budu řešit sám, třetí byl již napsán a je v jádře k dispozici.

Mou prvotní ambicí je proniknout do problematiky vývoje driverů. Na vývoji driverů budu pokračovat i po odevzdání diplomové práce a budu je (podle zadání) vyvíjet pod licencí open-source. Driver pro videosenzor (resp. pro rozhraní ISI použitého mikroprocesoru, k němuž je videosenzor připojen) ještě nebyl pro konkrétní mikroprocesor vyvinut, proto bych ho po dokončení rád dostal do hlavní větve linuxového jádra.

Kapitola 2

Kamera EYE-02 a operační systém Linux

2.1 Kamera EYE-02

Kamera EYE-02 je zabezpečovací kamera, kterou vyvíjí firma JABLOCOM v Jablonci nad Nisou. Na obr. 2.1 je znázorněno blokové schéma.

Její prvotní funkcí je sledování vymezeného prostoru a v případě detekce pohybu sejmutí obrázku a upozornění uživatele prostřednictvím SMS nebo MMS. Pro tento účel je kamera vybavena CMOS videosnímačem, detektorem pohybu a GSM modulem. Uživatelské rozhraní představují dvě světelné diody (červená a zelená) a tlačítko. Kromě LED s viditelným světlem můžeme na kameře nalézt také sadu infračervených LED, které umožňují noční vidění. Kamera dále obsahuje USB rozhraní a slot na Micro SD kartu. Pomocí USB rozhraní je možné připojit kameru k PC, provést konfiguraci a stáhnout snímky. MicroSD karta slouží pro uchování snímků a dalších potřebných dat.

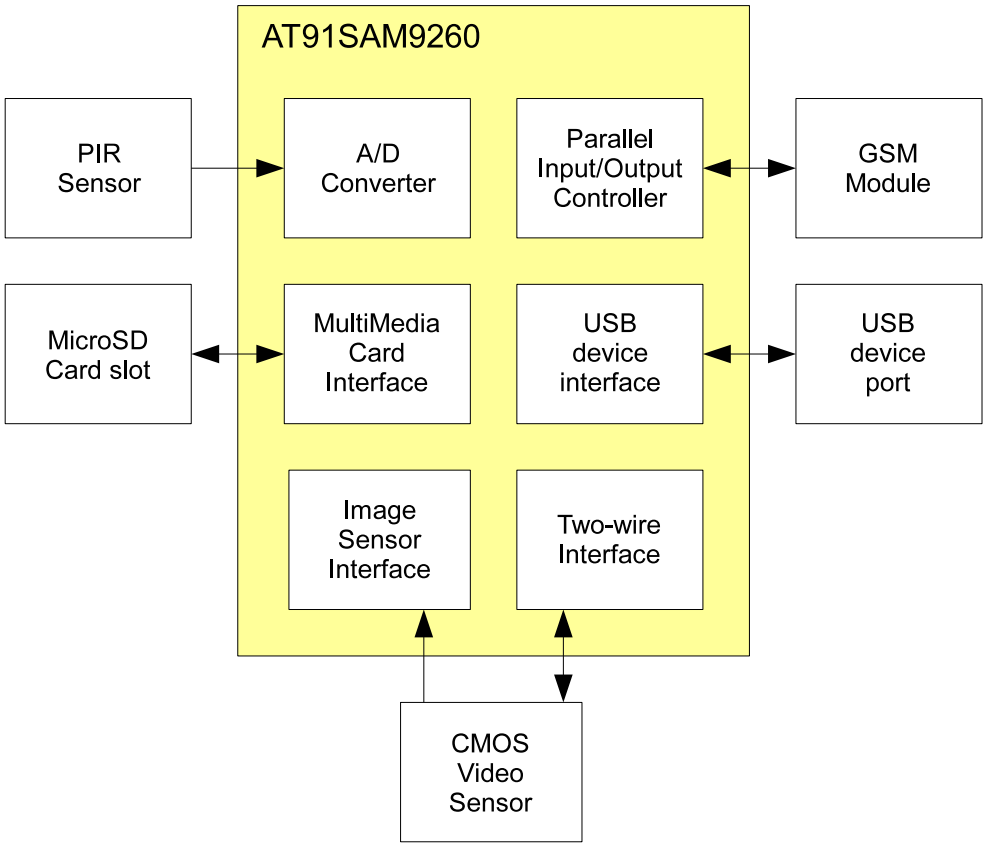
V kameře je použit mikroprocesor AT91SAM9260 od firmy Atmel a CMOS videosnímač MT9V011 od firmy Micron. Mikroprocesor je vybaven rozhraním ISI (Image Sensor Interface) pro připojení videosnímače, schopným přijatá data okamžitě zpracovat.

2.2 Provoz OS Linux v kameře

Linux je operační systém unixového typu. Jeho jádro je šiřitelné podle licence GNU GPL, díky čemuž ho lze přizpůsobit i jiným architekturám, než je IBM PC i386, pro kterou byl původně napsán. Komunita dobrovolníků, která se kolem Linuxu vytvořila, už portovala jádro i na mikroprocesory třídy ARM.

Před provozem Linuxu na kameře EYE-02 bylo tedy nutné provést následující kroky:

- **Získání zdrojového kódu jádra** – zdrojové kódy jsou dostupné na internetové stránce *The Linux Kernel Archives* na adrese `http://www.kernel.org`.
- **Aplikace patche pro konkrétní architekturu** – patch pro mikroprocesory AT91 je k dispozici na adrese `http://maxim.org.za/at91_26.html`. Tento krok ovšem nemusí být nezbytný – pokud jsou změny v patchi důkladně otestované a osvědčí se, jsou zahrnuty do příští verze. V době psaní této diplomové práce (květen 2009) je nejnovější verze jádra 2.6.29.3 a ta pro použití v kameře postačuje i bez patche.



Obrázek 2.1: Blokové schéma kamery EYE-02

- **Přizpůsobení jádra hardwaru** – jádro je sice připravené pro použití s mikroprocesorem AT91, ale musí se přizpůsobit ještě zapojení procesoru v konkrétní aplikaci.
- **Konfigurace jádra** – rozhodujeme se, které části kódu do překladu zahrnout a které nikoliv, popř. s jakými parametry překlad provádět. Zdrojové kódy jádra jsou velice univerzální a obsahují věci pro různé architektury. My např. pro naši aplikaci nepotřebujeme drivery pro LCD nebo myš, zato ale užijeme podporu pro komunikaci rozhraním I2C.
- **Překlad jádra** – pro tento krok potřebujeme překladač, který umí překládat pro cílovou platformu ARM. Vhodný je např. překladač *GCC*, který je v tomto ohledu velmi univerzální.
- **Zavedení binárního obrazu jádra** - obraz (vytvořený překladem) musíme předat mikroprocesoru, aby ho mohl spustit. V kameře EYE-02 se po startu mikroprocesoru spustí bootloader, který obraz jádra zkopíruje z Micro SD karty do RAM a následně spustí. Po překladu tedy obraz jádra musíme umístit na Micro SD kartu.

- **Příprava souborového systému** - když jádro naběhne a provede inicializaci, potřebuje kořenový souborový systém. Ten musíme mít připravený dopředu na blokovém zařízení (pro nás Micro SD karta), které jádru specifikujeme při konfiguraci. Souborový systém obsahuje hlavně konfigurační skripty a jaderné moduly.

Micro SD karta je rozdělena do tří oddílů. První oddíl má velikost cca 4MB a je určen pro binární obraz jádra, tam ho hledá bootloader při startu procesoru. Není na něm tedy zaveden žádný souborový systém, obsahuje pouze binární data. Máme-li obraz jádra připraven v souboru `zImage` a první oddíl na kartě je reprezentován souborem `/dev/mmcblk0p1`, dostaneme obraz jádra na kartu tímto příkazem:

```
dd if=zImage of=/dev/mmcblk0p1
```

Druhý oddíl má velikost cca 20MB a je na něm zaveden souborový systém *ext2*, který slouží operačnímu systému jako kořenový souborový systém. Třetí oddíl je formátován jako *FAT32* a slouží pro uchování dat, která budou zpřístupněna po rozhraní USB pomocí driveru *USB file storage*. Tento prostor je určen především pro nasnímané fotografie.

S kamerou lze komunikovat dvěma způsoby:

(a) Rozhraním UART, které slouží pro debug,

(b) rozhraním USB, které při použití modulu *g_ether* emuluje ethernetové spojení. V kameře běží telnetový démon.

Kapitola 3

Drivery v Linuxu

3.1 Význam a rozdělení driverů

Driver je program, který zprostředkuje přístup uživatele k hardwarovému zařízení. Komunikace s hardwarem je poměrně náročná záležitost, je nutné znát strukturu konkrétního zařízení a princip jeho práce. Driver tyto detaily ukryje a nabídne uživateli rozhraní s přesně definovanou množinou operací (pro všechny drivery shodnou).

Zařízení členíme na tři typy: *znaková*, *bloková* a *síťová*.

- **Znaková zařízení** přijímají a poskytují data ve formě sekvencí bajtů. Driver, který takové zařízení ovládá, obvykle poskytuje minimální množinu operací *open*, *close*, *read* a *write*. Zařízení jsou přístupná v souborovém systému v adresáři `/dev` a patří mezi ně např. sériový port nebo speciální zařízení `zero`, které při každém čtení vrátí nulu.

- **Bloková zařízení** jsou taková, na nichž může existovat souborový systém. Jedná se především o záznamová zařízení (disky, CD-ROM). Také jsou reprezentována soubory v adresáři `/dev` a v Linuxu (narozdíl od většiny ostatních unixových systémů) s nimi lze zacházet jako se znakovými. Spolupráce s jádrem je ovšem zcela odlišná.

- **Síťová rozhraní** slouží pro výměnu dat s jinými počítači. Nejsou v souborovém systému, tudíž na ně nelze aplikovat metody pro znaková a bloková zařízení.

Soubory v adresáři `/dev` se nevytvoří samy od sebe. K tomu je nutné použít utilitu `mknod` ve tvaru

```
mknod <filename> <device_type> <major_number> <minor_number>
```

Zde `filename` je název souboru, kterým chceme zařízení reprezentovat, `device_type` udává druh zařízení, tedy jestli je znakové (`c`) nebo blokové (`b`) a `major` a `minor` jsou čísla zařízení.

Číslo `major` v rozmezí 0–255 zařazuje zařízení do skupiny. V zdrojových kódech v souboru `devices.txt` v adresáři `Documentation` jsou vypsané skupiny zařízení a jim přiřazená `major` čísla. Číslo `minor` upřesňuje typ zařízení v rámci skupiny.

Číslo zařízení se využívá právě k založení souboru v adresáři `/dev`. Je přiřazeno v inicializační funkci driveru. Pokud driver patří do nějaké existující skupiny, může si přiřadit její `major` číslo, popř. může nechat jádro, aby mu samo přiřadilo nějaké volné. Seznam aktivních ovladačů najdeme v souboru `/proc/devices` spolu s čísly zařízení.

3.2 Jaderné moduly

Drivery vůbec nemusejí být součástí jádra, mohou se vyvíjet mimo jako *jaderné moduly*. To jsou programy, které sice běží v jaderném prostoru, ale mohou se do něj zavést až při jeho chodu. Je to mnohem pohodlnější než při každé změně v kódu driveru znovu překládat a zavádět jádro.

Moduly je možné dostat do jádra třemi způsoby:

(a) Modul se vyvíjí zcela mimo jádro, překládá se nezávisle na něm. Zavedení je možné příkazem `insmod`, bude popsán níže.

(b) Modul je součástí jádra, překládá se s ním, ale stále je to modul, může být zaveden až při běhu. Toto je nutné specifikovat v konfiguračních souborech `Kconfig`, aby si uživatel při konfiguraci jádra mohl vybrat, že chce tuto součást překládat jako modul. Při překladu jádra se potom voláním

```
make modules_install
```

přeložené moduly nainstalují do zadaného adresáře, kde je bude jádro hledat.

(c) Modul je součástí jádra a zavádí se automaticky při jeho startu. Opět se specifikuje při konfiguraci (a pokud si napíšeme takovou součást my, musíme ji sami přidat do `Makefile` a `Kconfig`).

Pro práci s moduly nabízí Linux několik utilit:

- `insmod` – zavede modul do jádra. Moduly mohou na sobě navzájem záviset (jeden exportuje symboly, které druhý využívá) a pokud se pokoušíme zavést modul, kterému chybí závislosti, tak tato funkce vrátí chybu.
- `rmmod` – odstraní modul z jádra. Pokud ovšem není zrovna někým využíván.
- `lsmod` – vypíše všechny aktivní moduly v jádře, jejich velikost a u každého modulu počet a seznam jiných modulů, které tento využívají. Tato funkce pouze vypíše v pohlednější formě obsah souboru `/proc/modules`.
- `modprobe` – chytřejší varianta programu `insmod`. Jednak hledá moduly v adresáři, kde jsou nainstalované (takže se může volat odkudkoliv) a jednak řeší chybějící závislosti. Pokud zjistí, že zaváděnému modulu nějaká taková chybí, tak si prohlédne tabulku symbolů (která se vytvoří při kompilaci jádra) a pokud tam požadovaný symbol najde, zavede kromě zadaného modulu ještě ty, které exportují potřebné symboly.

Pro psaní a překlad modulů platí poněkud jiná pravidla než pro psaní uživatelských aplikací. Každý modul musí definovat funkce:

```
int  module_init(void);
void module_exit(void);
```

Funkce se nemusí jmenovat zrovna takto, stačí když zachovají prototyp, ale potom je nutné pomocí maker (o stejných jménech jako výše uvedené funkce) definovat, která funkce slouží jako `init` a `exit`:

```
int  isi_init(void) { ... }
void isi_exit(void) { ... }

module_init(isi_init);
module_exit(isi_exit);
```

Dále je vhodné pomocí jiných maker definovat název modulu, autora a licenci (pokud ta není definovaná, tak jádro při zavádění nadává).

Ani překlad neprobíhá stejně jako u uživatelských aplikací. Máme-li zdrojový soubor `isi.c` a chceme z něj udělat modul, musíme do `Makefile` umístit řádku:

```
obj-m := isi.o
```

Pokud ovšem na tento `Makefile` pustíme obyčejný `make`, dostaneme vynadáno. Pro překlad modulů se totiž musí volat překládací systém jádra. K tomu použijeme `make` ve tvaru:

```
make -C /usr/src/linux-2.6.29.3 M=/home/vitj2/c/atmel-isi modules
```

Za parametr `C` patří cesta k zdrojovému kódu jádra a za `M` adresář s modulem.

Je nutné zmínit, že k překladu modulů je potřeba přeložené jádro s kompletními zdrojovými kódy.

Překladem se vytvoří modul se jménem `isi.ko`, který zavedeme příkazem `insmod`.

3.3 Standardní souborové operace na zařízení

Driver komunikuje s uživatelem pomocí vymezené množiny operací, které se provádějí na souboru zařízení v adresáři `/dev`. Tuto množinu definuje struktura `file_operations`. Definuje jich mnoho, my si probereme jen ty nejdůležitější. Popis čerpám z literatury [1].

- `int (*open)(struct inode *, struct file *);`
  První funkce, která se volá při přístupu k souboru zařízení. Obvykle v ní provádíme přípravu zařízení k práci. Také se kontroluje počet přístupů – pokud si nepřejeme, aby k souboru přistupovalo více procesů, zavedeme si příznak otevřeného zařízení a zde ho kontrolujeme. Pokud už bylo otevřeno, vracíme chybovou hlášku.

- `ssize_t (*read)(struct file *, char __user *, size_t, loff_t *);`
  Operace čtení dat ze znakového zařízení. Úkolem této funkce je předat data do uživatelského prostoru. Parametr typu `size_t` nám říká, kolik dat se vejde do uživatelského bufferu. Pokud jich máme k přenosu víc, bude se funkce volat několikrát a my musíme zajistit, aby data správně navazovala.

  Zde je dobré zdůraznit, že přesun z jaderného prostoru (v němž běží modul) do uživatelského nelze realizovat prostým použitím pointeru. K tomu jsou připraveny speciální funkce v includovaném souboru `<asm/uaccess.h>`, které je nutné zavolat.

Návratová hodnota je buď množství přenesených dat (kladné číslo), konec souboru dat (nula) nebo záporné chybové číslo. V případě, že přeneseme všechna data, je nutné vrátit nulu, protože podle toho uživatelská aplikace pozná, že má s čtením přestat (pokud nečte omezený počet znaků).

- `ssize_t (*write)(struct file *, const char __user *, size_t, loff_t *);`
  Zápis dat do zařízení. Funguje podobně jako `read()` a stejná omezení platí i pro přenos dat mezi jaderným a uživatelským prostorem. Vrací množství přijatých dat.

- `int (*ioctl)(struct inode *, struct file *, unsigned int, unsigned long)`
  Speciální příkaz pro zařízení. Používá se, když chceme se zařízením provádět i něco jiného, než čtení a zápis dat. Např. konfigurace. Příkazy se specifikují formou celočíselné konstanty. Driver definuje množinu příkazů, které podporuje. Lze dodat také pointer na datovou strukturu pro výměnu dat.

- `int (*mmap)(struct file *, struct vm_area_struct *)`
  Mapování paměti zařízení do adresového prostoru procesu. Ušetří se tím čas pro přesun dat funkcí `read()`.

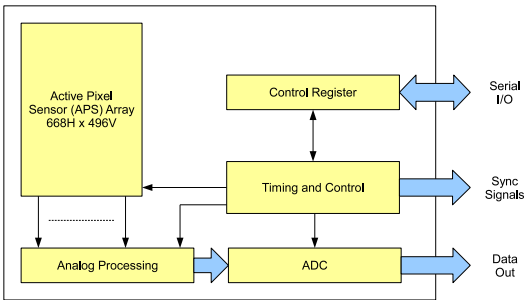
- `int (*release)(struct inode *, struct file *)`
  Uvolnění zařízení. Podobně jako `open()`, tato operace se volá jako poslední, těsně před tím, než proces soubor zařízení uvolní. V této funkci bychom měli uvolnit všechny zabrané zdroje, které zařízen nepotřebuje ke své činnosti.

Kapitola 4

Driver pro CMOS videosnímač

4.1 CMOS videosnímač MT9V011

V kameře EYE-02 je pro snímání obrazu použit CMOS snímač MT9V011 od firmy Micron. Jedná se čtvrtpalcový snímač s rozlišením 640 × 480 aktivních pixelů s Bayerovou RGB maskou. Na obr. 4.1 je znázorněno blokové schéma snímače. Toto schéma je převzaté z katalogového listu snímače [4].



Obrázek 4.1: Blokové schéma snímače MT9V011

Komunikace se senzorem probíhá prostřednictvím několika druhů signálů:

- **Sériové rozhraní** – dvouvodičová komunikace s protokolem SMBus (I2C), slouží pro zápis a čtení ovládacích registrů senzoru. Těmito registry se dá spustit nebo zastavit snímání a nastavit různé parametry, např. doba expozice (neboli integrace). Seznam registrů a jejich použití lze nalézt v katalogovém listu [4].
- **Datový výstup** – desetibitový paralelní výstup pro přenos sejmutých dat.
- **Časová synchronizace** – hodinový signál CLKIN, který musí poskytnout master zařízení a který je pro chod senzoru nezbytný, dále hodinový signál PIXCLK, který generuje senzor, a synchronizační signály LINE_VALID a FRAME_VALID, které podávají informaci o sejmutém řádku a snímku.

Snímač neumožňuje vyčítat data v jiném formátu, než je Bayer.

4.2 Rozhraní Atmel ISI

ISI (Image Sensor Interface) je speciální rozhraní, kterým je vybaven mikroprocesor AT91SAM9260 a které slouží pro připojení CMOS snímače. Je téměř dokonalá protistrana pro použitý snímač MT9V011, pouze datová sběrnice je o dva bity širší než ta u snímače, jinak hodinové a synchronizační signály si odpovídají.

Kromě samotného příjmu dat nabízí ISI i užitečné funkce pro jejich zpracování, jako je např. převzorkování na jinou velikost, ořez nebo přepočet formátu. Je schopné přijímat barevný i šedotónový obraz. Barevný obraz snímá ve formátech RGB nebo YCbCr.

Kombinace ISI a použitého senzoru bohužel neumožňuje plně využít funkcí ISI. Popsané možnosti zpracování obrazu se totiž týkají jen barevného obrazu ve zmíněných formátech. Snímač MT9V011 však produkuje data přímo z Bayerovy masky, která žádnému podporovanému barevnému formátu neodpovídají, takže lze použít pouze šedotónový režim. V tomto režimu ISI data pouze přenese z datové sběrnice prostřednictvím DMA do paměti a nijak je nezpracuje. Debayerizace se musí provést softwarově.

4.3 API Video4Linux2

Pro usnadnění tvorby ovladačů pro multimediální zařízení nabízí linuxové jádro vrstvu Video4Linux2 (dvojka v názvu značí, že se jedná o následníka Video4Linux). Tato vrstva podporuje širokou škálu zařízení, mezi něž patří např. zařízení pro zachytávání videa (kamery, videosnímače), pro výstup videa nebo rádiová zařízení.

Výhodou použití Video4Linux2 je jednotné API pro uživatelské aplikace a usnadnění práce vývojáře ovladačů, protože vrstva provádí mnoho operací, které by jinak vývojář musel řešit sám.

Pro zařízení pro zachytávání videa nabízí V4L2 následující funkce:

- **zjištění možností** – uživatelská aplikace se zeptá zařízení, co všechno dovede, jaké formáty podporuje apod.
- **konfigurace** – nastavení velikosti okna, ořezu obrazu, počtu snímků
- **příjem dat** – příjem sejmutých obrazových dat ze zařízení a jejich uložení do paměti

K třetímu bodu je vhodné poznamenat, že specifikace V4L2 [8] doporučuje, aby ovladač pouze přenesl data do paměti v tom tvaru, v jakém je získal ze zařízení, a neprováděl žádné zpracování nebo převody formátů. To by měla zajistit až aplikace.

Tyto tři body se realizují pomocí obvyklých souborových operací se zařízením. Zjištění možností a konfigurace se provádí systémovým voláním *ioctl()*, zatímco přenos dat může probíhat buď voláním *read()*, nebo lépe voláním *mmap()*, kdy se ušetří čas za přesun dat do uživatelského prostoru a aplikace může data ihned zpracovat.

Hlavní strukturou, která reprezentuje videozařízení, je struktura `video_device`. V ní jsou nejdůležitější tři skupiny členů:

- identifikace zařízení – název, vlastnící modul
- souborové operace – struktura s ukazateli na funkce, které realizují volání jako *open()*, *read()*, *mmap()*

- operace IOCTL – struktura s ukazateli na funkce, které realizují jednotlivé příkazy z volání *ioctl()*

Právě volání *ioctl()* vrstva V4L2 velmi usnadňuje. Bývá zvykem, že funkce obsluhující toto volání obsahuje dlouhou konstrukci *switch...case* pro každý *ioctl* příkaz a je velmi nepřehledná. Vrstva V4L2 definuje množinu příkazů a pro každou z nich je definován funkční prototyp. Programátor pro každý podporovaný *ioctl* příkaz definuje funkci odpovídající příslušnému prototypu a vrstva se už postará o zavolání této funkce a přenos dat mezi jaderným a uživatelským prostorem.

Nejdůležitější *ioctl* příkazy jsou tyto (každý příkaz je uveden konstantou, která tento příkaz reprezentuje):

- `VIDIOC_QUERYCAP` – dotaz na identifikaci a možnosti zařízení. Aplikace při volání předá ukazatel na strukturu `v4l2_capability`, kterou driver vyplní. Struktura obsahuje jméno ovladače a zařízení, verzi ovladače a bitové pole, v němž jsou vypsané funkce podporované ovladačem. Pro bitové pole V4L2 definuje konstanty. Náš ovladač pro ISI bude podporovat vlastnosti `VIDEO_CAPTURE` (je schopen zachytávat video) a `READWRITE` (podporuje souborové operace *read()* a/nebo *write()*). Ovladač pro ISI podporuje pouze operaci *read()*, protože zapisovat data do snímače nemá smysl.

- `VIDIOC_CROPCAP` – dotaz na možnosti ořezu a převzorkování obrazu. Pomocí struktury `v4l2_rect` je definováno největší okno, které lze z obrazu vybrat.

- `VIDIOC_G_CROP, VIDIOC_S_CROP` – zjištění / nastavení ořezu. Opět je využita struktura `v4l2_rect`.

- `VIDIOC_G_FMT, VIDIOC_S_FMT` – zjištění / nastavení formátu dat. V4L2 definuje množství formátů, kterými mohou kódována obrazová data. Protože podle doporučení by měl ovladač předávat data v podobě, v jaké je přijme ze snímače, závisí podporované formáty na konkrétním typu snímače. Snímač MT9V011 podporuje jenom jeden formát – Bayerovu masku s rozlišením 16 bitů. Podporované formáty zařízení lze zjistit příkazem `VIDIOC_ENUM_FMT`.

- `VIDIOC_QUERYCTRL, VIDIOC_S_CTRL, VIDIOC_G_CTRL` – zjištění / nastavení jiného parametru snímání, který není pokryt ostatními *ioctl*. V4L2 definuje množinu takových parametrů (např. doba expozice) a tyto funkce je obsluhují. Výčet možných parametrů lze získat použitím příkazu `VIDIOC_QUERYCTRL`

4.4 Struktura ovladače

Rozhraní ISI je určeno pro připojení mnoha typů snímačů, nejenom toho jednoho použitého v kameře EYE-02. Z toho důvodu je vhodné rozdělit kód na část týkající se ISI a část týkající se konkrétního videosnímače.

- **Ovladač ISI** bude komunikovat s aplikací a zprostředkuje jí přístup k celému snímacímu systému. Tento ovladač bude zcela nezávislý na použitém typu snímače, aby bylo možné použít ho pro všechna zařízení, která se k ISI dají připojit.

- **Ovladač snímače** bude přizpůsoben konkrétnímu typu snímače MT9V011. Snímače se mezi sebou liší ve způsobu ovládání a konfigurace. Snímač MT9V011 se ovládá a konfiguruje po sériové lince podle protokolu I2C. Ovladač proto bude schopen tuto komunikaci realizovat a bude zaměřený na množinu registrů snímače MT9V011. K tomuto ovladači nebude mít přístup aplikace, ale pouze ovladač ISI.

Aby bylo možné ovladače takto koncipovat, je potřeba definovat jednotné rozhraní mezi nimi. Takové, které ovladač pro ISI bude využívat a které ovladač snímače bude implementovat. Pokud se např. ovladač ISI rozhodne, že je potřeba zkrátit expozici, tak musí mít k dispozici prostředek, kterým tento požadavek sdělí ovladači pro snímač, aby to tento mohl změnit v registrech.

Takovéto rozhraní je realizováno strukturou `image_sensor`, definovanou v souboru `camera.h`. Tato struktura reprezentuje snímač. Kromě několika formalit, jako je vlastník struktury nebo název snímače, obsahuje tato struktura především ukazatele na funkce, kterými se snímač konfiguruje a ovládá. Namátkou jsou to např. tyto:

```
struct image_sensor {
    int (*set_format)    (struct image_sensor *cam,
                          struct image_sensor_format *fmt);
    int (*cropcap)       (struct image_sensor *cam,
                          struct v4l2_cropcap *cropcap);
    int (*start_capture) (struct image_sensor *cam);
};
```

Interakce mezi oběma ovladači je následující: ovladač pro ISI definuje a exportuje tyto funkce:

```
int  register_camera   (struct image_sensor *cam);
void unregister_camera (struct image_sensor *cam);
```

Ovladač pro snímač si definuje strukturu `cam`, definuje všechny ovládací a konfigurační funkce, přiřadí pointery a strukturu zaregistruje. Ovladač pro ISI si pointer na strukturu uloží a následně s ním pracuje. Když potřebuje konfigurovat snímač, zavolá příslušný funkční ukazatel v struktuře `cam` a ovladač snímače tuto funkci provede.

Konfigurace je potom věc specifická pro každý snímač (např. pro náš MT9V011 probíhá pomocí protokolu I2C) a pro ovladač ISI je zcela transparentní.

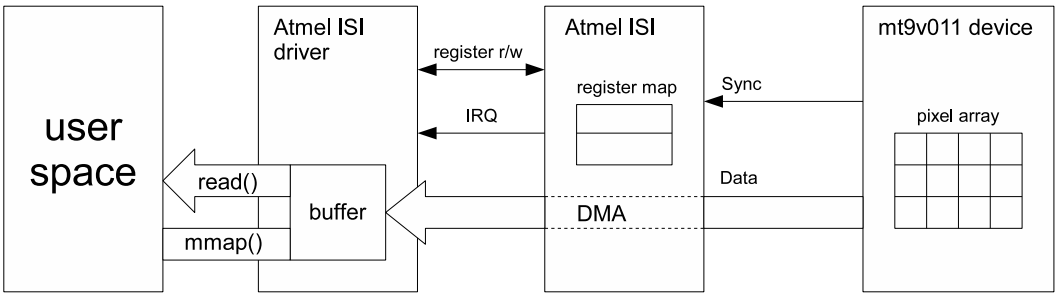
4.5 Ovladač ISI

Ovladač pro rozhraní ISI, který jsem nazval *atmel-isi*, má dva základní úkoly:

- **Konfigurace zařízení** – nastavení velikosti obrázku, doby expozice. Provádí se *ioctl* voláním.

- **Přenos dat** – sejmutí obrazových dat ze snímače a jejich zprostředkování aplikaci. Může být realizováno voláním *read()*, kdy budou data po sejmutí přesunuta do uživatelského prostoru, nebo voláním *mmap()*, kdy bude do uživatelského prostoru namapován buffer, do něhož byla data přenesena z ISI funkcí DMA.

Význam ovladače *atmel-isi* je znázorněn na obr. 4.2.



Obrázek 4.2: Význam ovladače `atmel-isi`

Ovladač jsem nevymyslel celý sám, ale (jak už to tak u open-source programů chodí) inspiroval jsem se u podobných kódů. Nejužitečnější mi byl ovladač pro podobné zařízení na procesoru AVR32.

4.5.1 Úpravy linuxového jádra

Protože pro zařízení ISI zatím neexistuje podpora v linuxovém jádře, musel jsem ji doplnit. Týká se následujících souborů:

- `arch/arm/mach-at91/include/mach/board.h` – doplněna definice struktury `at91_isi_data`, která má následující tvar:

  ```
  struct at91_isi_data {
      unsigned int clock_rate;
      unsigned int nr_data_pins;
  };
  ```

  Struktura slouží pro specifikaci frekvence hodinového signálu CLKIN, který vstupuje do snímače (hodnota se udává v Hz) a počtu datových pinů, které se využívají při přenosu dat (pro případ, že má snímač užší datovou sběrnici než ISI).

  Dále byl doplněn funkční prototyp:

  ```
  extern void __init at91_add_device_isi(struct at91_isi_data *data)
  ```

  Ten slouží pro registraci zařízení ISI v jádře a byla jím nahrazena původní verze bez vstupních parametrů, která sloužila jen jako výplň a byla připravena pro doplnění.

- `arch/arm/mach-at91/at91sam9260_devices.c` – v tomto souboru jsou definovány funkce, které registrují a inicializují jednotlivá zařízení na mikroprocesoru. Mezi tato zařízení patří také ISI, proto bylo nutné doplnit i podporu pro něj.

  Jednak bylo doplněno pole `isi_resources`, které je složeno ze struktur `resource`. V tomto poli byly pro zařízení ISI definovány dva zdroje:

(a) paměťová oblast, která pokrývá ovládací registry modulu ISI

(b) přerušení od modulu ISI

Dále byla doplněna struktura `at91sam9260_isi_device` typu `platform_device`, která definuje zařízení ISI (pod názvem `atmel_isi`), aby mu následně bylo možné přiřadit *platform driver*.

Nakonec byla definována funkce `at91_add_device_isi` (vyhovující prototypu v souboru `board.h`), která provede přípravu pro použití modulu ISI a jeho registraci jako *platform device*. Příprava spočívá ve dvou částech:

(a) Nastavení funkce pinů, které využívá modul ISI. Je nutné oznámit procesoru, že tyto piny budou užívány periferií a nikoliv jako paralelní port.

(b) Nastavení hodinového signálu CLKIN pro videosnímač. Tento signál snímač nezbytně potřebuje, aby vůbec komunikoval. Bez něj nereaguje ani na komunikaci po I2C.

V jádře jsou v adresáři `mach-at91` dále přítomny soubory typu `board-xxx.c`, které jsou specifické pro jednotlivé aplikace s mikroprocesorem. Takovou aplikací je i kamera EYE-02, proto byl doplněn soubor `board-eye02.c`, který popisuje konkrétní architekturu kamery. V tomto souboru je definována struktura `eye_isi_data` typu `at91_isi_data`, v ní je zadána frekvence hodinového signálu a počet použitých datových vodičů na ISI. Dále je v inicializační funkci aplikace volána funkce `at91_add_device_isi`, která zaregistruje ISI jako *platform device*. Od této chvíle je v jádře přítomna struktura typu `platform_device`, která reprezentuje modul ISI a je možné přiřadit jí *platform driver*, což bude provádět už samotný ovladač.

Každá aplikace s tímto mikroprocesorem by měla mít svůj vlastní definiční soubor a z něho volat registrační funkci zařízení ISI, aby bylo následně možné přiřadit driver.

4.5.2 Inicializace driveru

V předchozí podkapitole jsem uvedl, že modul ISI je nyní v jádře zaveden jako *platform device*. V inicializaci jaderného modulu `atmel-isi` se tedy provádí registrace *platform driveru*, zatímco v *cleanup* funkci se tato registrace ruší. K tomu jsou užity tyto metody:

```
platform_driver_register(&atmel_isi_driver);
platform_driver_unregister(&atmel_isi_driver);
```

Struktura `atmel_isi_driver` je typu `platform_driver`. Má uvedeno jméno

```
.driver.name = "atmel_isi";
```

Díky tomu ho po registraci jádro spojí s definovaným stejnojmenným *platform device* a zavolá metodu `.probe`.

V metodě `atmel_isi_probe()` se provedou následující akce:

- **Alokace hlavní struktury** – tedy struktury `atmel_isi`, která v sobě nese všechny potřebné informace o driveru, aby k nim ostatní funkce měly snadný přístup.

- **Nastavení hodinového signálu** – potřebujeme dva hodinové signály. Jeden, který budeme pouštět do snímače (vstup CLKIN), ten nám v procesoru zajistí programovatelný zdroj hodin PCK1. Dále potřebujeme signál ze skupiny *peripheral clock*, bez kterého se neobejde modul ISI na procesoru. V inicializaci oba tyto signály povolíme.
- **Příprava video_device** – zadání jména a zpětných volání pro souborové operace.
- **Přemapování paměti** – fyzické do virtuálního prostoru, abychom mohli přistupovat k ovládacím registrům ISI a do SRAM, kam budeme ukládat adresy bufferů pro záznam obrazu.
- **Registrace přerušení** – ISI generuje mnoho přerušení, která se nám hodí, např. oznámení o dokončeném resetu nebo o dokončeném snímání.
- **Alokace zachycovacího bufferu** – do něj bude modul ISI pomocí DMA přenášet data ze snímače.

Poslední bod si zaslouží trochu rozepsat, proto jsem mu věnoval samostatnou podkapitolu.

4.5.3 Buffer pro zachycený obraz

Použitý senzor je schopný sejmout obraz o velikosti 640×480 pixelů (ve skutečnosti ještě o něco větší, ale část z něho už by zabíraly černé kalibrační zóny). Protože senzor obsahuje desetibitový A/D převodník, budeme pro každý pixel potřebovat dva bajty. Jednoduchým výpočtem získáme velikost potřebného prostoru

$$640 \times 480 \times 2 = 614400\,\mathrm{B}.$$

Toto je velmi rozsáhlý prostor, funkce `kalloc()` ovšem umí alokovat jen max. 128 kB. Mohla by se alokovat paměť v uživatelském prostoru, ale ta nejspíš nebude souvislá. To by nevadilo, kdyby DMA kanál podporoval funkci *scatter-gather* (pozastavení přenosu, v mezičase lze přepsat cílovou adresu DMA přenosu na jinou stránku), jenže ISI tuto funkci nepodporuje.

Řešením je patch jménem *bigphysarea*. Ten doplní do jádra kód, který při bootování a rozdělování paměti zabere souvislý úsek pro sebe. Zároveň exportuje funkce `bmalloc()` a `bfree()`, které mají k tomuto úseku přístup.

Já jsem se ovšem použití tohoto patche vyhnul, protože jsem k alokaci použil funkci

```
__get_free_pages(GFP_USER|GFP_DMA, get_order(size));
```

Tento nápad nepochází z mojí hlavy, přečetl jsem si ho v jiném kódu, kde byl doplněn komentářem, že tato funkce dostala během bootování spoustu paměti a ta je stále k dispozici. Tento postup funguje, proto jsem ho dále nerozebíral a používal. V případě problémů je tu stále možnost použití *bigphysarea*.

Protože ovladač (zatím) snímá jeden obrázek na jeden požadavek, vystačil jsem si s jedním bufferem. Pokud by přišla potřeba většího množství obrázků, je možné alokovat více bufferů za sebou a přenášet do nich data cyklicky.

4.5.4 Souborové operace `open()` a `release()`

Operace `open()` se volá v okamžiku otevření souboru zařízení v adresáři `/dev`.

Funkce nejprve zkontroluje, jestli už zařízení není otevřené. Není žádoucí, aby bylo zařízení využíváno několika procesy, na to driver není připraven. Tato kontrola se provádí ověřením členu struktury `atmel_isi.users`. Tento člen je inkrementován na konci funkce `open()` a na konci funkce `release()` je dekrementován.

Dále funkce ověří, jestli byl příkazem `register_camera()` registrován nějaký snímač. Pokud ne, funkce vrátí chybovou hodnotu `ENODEV` (zařízení není k dispozici). Nemůžeme snímat obraz, když nemáme odkud.

Důležitá činnost, kterou tato funkce provádí, je zápis členu `private_data` struktury `file`. Pokud se totiž volají souborové operace jako `read()`, nepředají se jim jako parametr žádná naše data, ale pouze ukazatel na strukturu `file`, která je ta samá, co byla předána funkci `open()`. My bychom ovšem ve funkci `read()` potřebovali minimálně přístup k hlavní struktuře `atmel_isi`. Jenže ta je dynamicky alokovaná a neexistuje na ni žádný globální ukazatel. Proto jediný způsob, jak se k ní ve funkci `read()` dostaneme, je takový, že referenci na ni uložíme do pole `private_data` struktury `file`. Potom při volání funkce `read()` můžeme z předané struktury `file` toto pole přečíst a máme přístup k naší struktuře `atmel_isi`.

V našem driveru máme pro zapouzdření všech potřebných *private dat* definovánu strukturu `atmel_isi_fh`:

```
struct atmel_isi_fh {
    struct atmel_isi   *isi;
    unsigned int        read_off;
};
```

Parametr `isi` je odkaz na hlavní strukturu driveru, zatímco parametr `read_off` využijeme při operaci `read()`. Využití je popsáno v její samostatné podkapitole.

Nakonec se provede reset rozhraní ISI (zápisem do ovládacího registru).

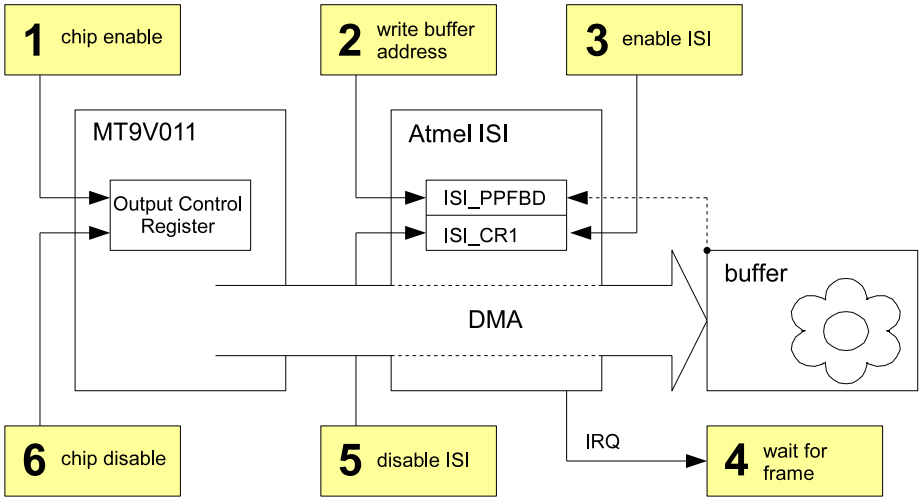
Operace `release()` se provádí ve chvíli, kdy je soubor zařízení uzavřen. V tento okamžik se přeruší snímání, zastaví se činnost ISI a sníží se počet přístupů k souboru (`atmel_isi.users`).

4.5.5 Proces sejmutí obrazu

Postup při snímání obrazových dat je znázorněn na obr. 4.3.

Samotná operace sejmutí obrázku se spustí funkcí `atmel_isi_start_capture()`. Funkce zavolá metodu `start_capture()` struktury `image_sensor`. Tuto metodu implementuje driver pro konkrétní snímač a měl by mu dát pokyn ke spuštění snímání. Dále je aktivován DMA kanál voláním funkce `atmel_isi_start_dma()`. Nakonec je zápisem do ovládacího registru ISI povoleno snímání.

V tomto okamžiku je nutné vyčkat, dokud není snímání dokončeno. Toto čekání realizuje funkce `atmel_isi_wait_for_frame()`. Funkce čeká, dokud není probuzena obsluhou přerušení nebo dokud nevyprší timeout. Na tom, který z těchto dvou případů nastane dřív, závisí návratová hodnota této funkce. Modul ISI po dokončení snímání jednoho obrázku generuje přerušení SOF (start of frame), kterým signalizuje, že se chystá přenést další obrázek. V tomto okamžiku obsluha přerušení snímání zastaví a probudí čekací funkci.



Obrázek 4.3: Znázornění postupu při snímání obrazu

Proces snímání se řídí zápisem do registru *ISI Control 1 Register* (`ISI_CR1`), konkrétně bitem `ISI_DIS` (Image sensor disable) (viz [3]). Nulováním tohoto bitu je spuštěna funkce ISI a zahájen přenos, zatímco nastavením bitu je dokončeno sejmutí aktuálního snímku a pak se modul vypne.

4.5.6 Operace `read()`

Zpětné volání `read()` je použito v momentu, kdy uživatel vyčítá data ze souboru zařízení. Je tedy úkolem této funkce očekávaná data poskytnout.

V záhlaví funkce máme mj. vstupní parametr `count`, který nás zpravuje o velikosti bufferu v uživatelském prostoru, do něhož máme zapsat data. Protože tento parametr jen velmi těžko dosáhne stovek kilobajtů, které obrazová data dohromady čítají, je funkce volána několikrát a data se vyčítají postupně. Musíme si tedy zaznamenávat množství přenesených dat, abychom věděli, kde v bufferu máme při dalším přenosu začít vyčítat. K tomu jsme ve struktuře `atmel_isi_fh` definovali člen `read_off`.

Při vstupu do funkce `read()` mohou nastat tři stavy:

- Funkce je volána poprvé od otevření souboru nebo předchozího kompletního vyčtení dat. Musíme zajistit sejmutí nových dat, nulovat `read_off` a začít vyčítání.
- Funkce je volána v průběhu vyčítání, aby získala další dávku dat. Už nic nesnímáme, pouze kopírujeme data a inkrementujeme `read_off`.
- Funkce je volána po vyčtení všech dat, `read_off` je roven množství dat v bufferu. Neděláme nic, pouze vracíme nulu jako znamení konce souboru dat.

4.5.7 Operace `ioctl()` a konfigurace snímače

Volání `ioctl()` se používá pro speciální operace, které se vymykají běžnému čtení nebo zápisu dat. U ovladače pro videosnímač se tímto způsobem provádí konfigurace a zjišťují se možnosti zařízení.

V kapitole o Video4Linux2 jsem už popsal, jak tato vrstva usnadňuje použití `ioctl()`. Zatímco v běžných ovladačích funkce `ioctl()` obsahuje obrovskou konstrukci *switch–case*, která řeší různé příkazy, vrstva V4L2 tento problém řeší sama a programátor musí pouze definovat funkce o předem daných prototypech, které jednotlivé příkazy *ioctl* realizují.

V4L2 definuje mnoho *ioctl* příkazů a všechny jsou reprezentovány konstantami ve tvaru `VIDIOC_xxx`. Jejich podrobný popis a definice příslušných funkčních prototypů lze nalézt ve specifikaci V4L2 [8]. Jsou mezi nimi zahrnuty ale příkazy pro všechny třídy zařízení, které V4L2 podporuje. My budeme v našem driveru řešit jen velmi malou část.

Příkaz	Popis
`QUERYCAP`	dotaz na schopnosti zařízení
`ENUMFMT`	dotaz na podporované formáty
`G_FMT`	dotaz na aktuální formát
`S_FMT`	příkaz k nastavení formátu
`TRY_FMT`	dotaz na možnost nastavení daného formátu
`CROPCAP`	dotaz na možnosti ořezu obrazu
`G_CROP`	dotaz na aktuální nastavení ořezu
`S_CROP`	příkaz k nastavení ořezu
`QUERYCTRL`	dotaz na možnosti nastavení parametrů
`G_CTRL`	dotaz na hodnotu parametru
`S_CTRL`	příkaz k nastavení parametru

Tabulka 4.1: Podporované *ioctl* příkazy

Příkazy, které mají vazbu na použitý senzor, jsou vyřizovány prostřednictvím zpětných volání ve struktuře `image_sensor`.

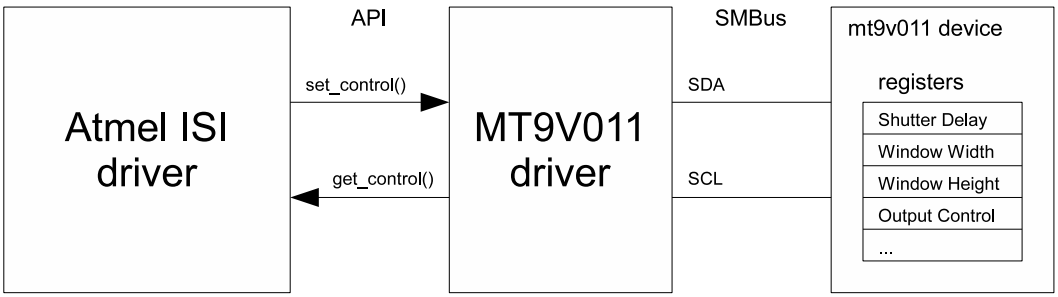
Formát dat: náš snímač MT9V011 podporuje jenom jeden formát, proto funkce pro nastavení asi moc nevyužijeme, ale driver pro ISI musí být univerzální a předpokládat možnost připojení snímače s více podporovanými formáty.

Parametry: u MT9V011 podporujeme dobu expozice a zesílení jednotlivých barevných kanálů na Bayerově masce. Podporované parametry ale závisí na typu senzoru, proto je na jeho ovladači, aby sdělil jejich seznam.

4.6 Ovladač snímače

Ovladač pro snímač je specifický pro konkrétní výrobek Micron MT9V011. Nenabízí žádné rozhraní pro komunikaci s uživatelskou aplikací, protože komunikuje pouze s ovladačem ISI.

Úkolem ovladače je přijímat pokyny od ovladače ISI a předávat je snímači. Komunikace se snímačem probíhá po sériové lince protokolem I2C a jedná se pouze o **zápis nebo čtení ovládacích registrů snímače**. Význam driveru je znázorněn na obr. 4.4.



Obrázek 4.4: Význam ovladače `mt9v011`

4.6.1 Úpravy linuxového jádra

Stejně jako u ovladače ISI, i pro driver snímače je nutné provést několik drobných změn ve zdrojových kódech linuxového jádra. V souboru `board-eye02.c` definujeme pole

```
static struct i2c_board_info __initdata eye_i2c_devices[] = {
    ...
    {
        I2C_BOARD_INFO("mt9v011", 0x5d),
    },
};
```

V něm každý prvek reprezentuje jedno zařízení na sběrnici I2C a uvádí jeho název a sedmibitovou adresu. Senzor se na sběrnici hlásí adresou `0x5d` a ponese název `mt9v011`, stejně jako jeho ovladač (to je podmínka).

V inicializačním souboru desky potom nezapomeneme volat funkci

```
at91_add_device_i2c(eye_i2c_devices, ARRAY_SIZE(eye_i2c_devices));
```

Tento kód musí být uveden pro každou použitou aplikaci v jejím souboru `board-xxx.c`.

4.6.2 Protokol I2C

Komunikace mikroprocesoru se snímačem probíhá kromě rozhraní ISI také po sériové lince, kde je použitý protokol I2C. Tento protokol definuje komunikaci na sběrnici, kde je kromě jednoho masteru (mikroprocesor) přítomno také několik zařízení slave (ty jsme definovali v souboru `board-eye02.c`). Každé zařízení má svou unikátní sedmibitovou adresu. V kameře EYE-02 je kromě videosnímače na sběrnici I2C připojena také paměť EEPROM.

Protokol I2C definuje dvě základní operace s daty: *čtení* a *zápis*. Při každé z těchto operací musí být zadána sedmibitová adresa zařízení. Pro případ zařízení jako je náš snímač nebo EEPROM je ještě potřeba zadat adresu uvnitř těchto zařízení, kterou specifikujeme, s jakými daty budeme pracovat. U videosnímače se jedná o adresu registru, který budeme číst nebo zapisovat.

Pro usnadnění komunikace je mikroprocesor vybaven modulem TWI (Two-Wire Interface). Bohužel s ním programátoři nemají dobré zkušenosti, proto se doporučuje nepoužívat ho a provozovat I2C komunikaci přímo posíláním bitů na sériovou linku.

V Linuxu je pro tento účel k dispozici ovladač *i2c-gpio*. Ten provozuje komunikaci protokolem *SMBus*, který je jakousi podtřídou I2C a definuje komunikaci právě pro zařízení jako EEPROM nebo náš videosenzor, která fungují jako úschovna dat a při čtení nebo zápisu je u nich nutné definovat vnitřní adresu, na které se nacházejí data, s nimiž hodláme pracovat.

4.6.3 Struktura kódu

V `init` a `exit` funkci modulu se provádějí pouze dvě věci – registrace a odstranění driveru pro zařízení I2C:

```
i2c_add_driver(&mt9v011_i2c_driver);
i2c_del_driver(&mt9v011_i2c_driver);
```

Struktura `mt9v011_i2c_driver` reprezentuje ovladač zařízení I2C a je definována následovně:

```
static struct i2c_driver mt9v011_i2c_driver = {
    .driver = {
        .name = "mt9v011",
    },
    .probe      = mt9v011_probe,
    .remove     = mt9v011_remove,
    .id_table   = mt9v011_id,
};
```

Název driveru musí být shodný s názvem zařízení, které jsme definovali v souboru `board-eye02.c`. Potom při jeho registraci jádro zavolá funkci `mt9v011_probe()`, která provádí inicializaci.

Při inicializaci jsou nastaveny potřebné parametry pro provozování I2C komunikace a dále se zavádí struktura `image_sensor`, která slouží jako rozhraní mezi ovladačem snímače a ovladačem ISI. Funkčním ukazatelům v této struktuře jsou přiřazeny funkce, které ovladač snímače definuje a které bude ovladač ISI při svojí práci volat.

Na závěr je provedena registrace struktury `image_sensor`:

```
struct image_sensor *ims;
...
ims->set_crop = mt9v011_set_crop;
...
atmel_isi_register_camera(ims);
```

Přitom se volá registrační funkce, kterou exportuje ovladač ISI.

Ovladač snímače dále obsahuje funkce, které se dají rozdělit do dvou kategorií:

- **zápis a čtení registrů snímače** – obalují funkce ovladače *i2c_core*, který se v jádře stará o I2C komunikaci.
- **zpětná volání struktury** `image_sensor` – tyto funkce volá ovladač ISI, když chce nastavit nějaký obecný parametr senzoru a nezajímá ho, jak se toto nastavení provede na konkrétním typu senzoru. To už je starost tohoto ovladače senzoru. Příkladem je volání `set_crop` – nastavení ořezu obrázku. Ovladač ISI chce nastavit jistý ořez obrázku, tak zavolá tuto funkci. Ovladač snímače ví, jak se toto nastavení provede na aktuálním snímači, a tuto akci provede.

4.7 Použití ovladačů

Použití obou ovladačů jsem otestoval úloze sejmutí obrázku. Po překladu jsem měl k dispozici dva jaderné moduly:

```
atmel-isi.ko
mt9v011.ko
```

Tyto moduly jsem zavedl do jádra:

```
insmod atmel-isi.ko
insmod mt9v011.ko
```

Dokud jsou moduly mimo jádro, je nutné dodržet toto pořadí, protože modul `mt9v011` využívá symboly exportované modulem `atmel-isi` (funkci pro registraci snímače) a pokud by v momentě jeho zavádění nebyl `atmel-isi` v jádře přítomen, program `insmod` by ohlásil chybu, že modul obsahuje nedefinované symboly.

Pokud by moduly byly součástí jádra, mohla by se na jejich zavedení použít funkce `modprobe`, která by poznala, že modulu `mt9v011` chybí symboly definované v modulu `atmel-isi` a jako první by sama zavedla právě `atmel-isi`.

V jádře musí být dále přítomno API Video4Linux2, které vyžaduje modul `atmel-isi` a podpora pro I2C, kterou vyžaduje modul `mt9v011`. V tabulce zařízení `/proc/devices` bychom měli vidět položku `video4linux` s *major* číslem 81.

Abychom mohli k ovladači ISI přistupovat, musíme si vytvořit soubor `/dev/video0` příkazem

```
mknod /dev/video0 c 81 0
```

Teď už můžeme vesele snímat obraz. Využijeme k tomu operaci `read` na souboru zařízení. Tuto operaci provádějí např. utility `dd` nebo `cat`, ale můžeme si vytvořit i vlastní program.

```
cat /dev/video0 > rawdata
```

Po této operaci budeme mít v souboru `rawdata` po řádcích sekvenčně uložené jasové hodnoty všech pixelů ze snímače, ovšem v barevném rozložení podle Bayerovy masky. Při zpracování je nutné provést debayerizaci (z Bayerovy masky vytvořit 24-bitový RGB formát).

Ukázka obrázku pořízeného kamerou je na obr. 4.5.

Obrázek 4.5: Ukázka obrázku sejmutého kamerou EYE-02

Kapitola 5

Čidlo PIR

Kamera EYE-02 je vybavena pohybovým senzorem PIR (passive infrared). Tento senzor generuje analogový signál a je připojen na první kanál analogově-číslicového převodníku v mikroprocesoru.

Ovladač pro tento senzor vyčítá data z analogově-digitálního převodníku a nechává na uživatelské aplikaci, aby signál zpracovala a detekovala z něho pohyb.

5.1 Inicializační funkce

Ovladač jsem pojal velice klasicky a jeho funkce je jednoduchá.

V inicializační funkci modulu se registruje jádru znakové zařízení a vyžádá si přidělení *major* čísla.

```
err = alloc_chrdev_region(&pir_dev, PIR_MINOR, PIR_NR_DEV,
                          PIR_DEVNAME);
cdev_init(&pir_cdev, &pir_fops);
err = cdev_add(&pir_cdev, pir_dev, PIR_NR_DEV);
```

Driver si registruje dvě znaková zařízení. Zařízení s *minor* číslem 0 na žádost `read` vrátí aktuální hodnotu analogového signálu přímo v binární podobě, zatímco zařízení č. 1 vrací tuto hodnotu převedenou na text.

Dále inicializační funkce aktivuje zdroj hodinového signálu pro periferii ADC:

```
struct clk *adc_clk = clk_get(NULL, "adc_clk");
clk_enable(adc_clk);
```

Blok ADC se stejně jako každá jiná periferie ovládá prostřednictvím řídicích registrů, proto k nim musíme mít přístup. Inicializační funkce si vyžádá přístup do paměťové oblasti, do které jsou registry mapované, a následně provede přemapování do virtuálního paměťového prostoru.

```
struct resource *adc_res;
void __iomem *adc_regs;
adc_res = request_mem_region(AT91SAM9260_BASE_ADC, ADC_REGS_LEN,
                             PIR_DEVNAME);
adc_regs = ioremap_nocache(AT91SAM9260_BASE_ADC, ADC_REGS_LEN);
```

Protože převod analogové hodnoty na digitální nějakou chvíli trvá, informuje nás blok ADC o jeho dokončení přerušením. Musíme si tedy registrovat obsluhu přerušení:

```
err = request_irq(AT91SAM9260_ID_ADC, pir_irqhandler, 0, PIR_DEVNAME,
                  NULL);
```

Nakonec provedeme inicializaci A/D převodníku. Povolíme příslušný analogový kanál a přerušení pro případ dokončení převodu.

```
mode = AT91_ADC_LOWRES | AT91_ADC_PRESCAL_(9) | AT91_ADC_SHTIM_(9) |
       AT91_ADC_STARTUP_(9) | AT91_ADC_SLEEP;
adc_writeb(CR, SWRST);
adc_writeb(CHER, CH(pir_adc_channel));
adc_writel(MR, mode);
adc_writeb(IER, EOC(pir_adc_channel));
```

5.2 Souborová operace `read()`

Náplní obsluhy operace `read()` je převod aktuální hodnoty na PIR senzoru a jeho přenos do uživatelského prostoru.

Převod probíhá velmi jednoduše – zapíšeme do řídicího registru převodníku, aby začal s převodem, a následně čekáme, dokud nás obsluha přerušení neinformuje o dokončení převodu.

```
static struct completion data_ready;
...
init_completion(&data_ready);
adc_writeb(CR, START);
wait_for_completion_timeout(&data_ready, msecs_to_jiffies(1000));
```

Po dokončení převodu přeneseme výslednou hodnotu do uživatelského prostoru. Ve funkci `open()` jsme si zjistili, který znakové zařízení bylo otevřeno (tedy jeho *minor* číslo) a podle toho přeneseme do uživatelského prostoru buď binární hodnotu nebo řetězec. Nezapomeneme na ošetření konce přenosu (to je důležité především u řetězce) – po vyčtení hodnoty si nastavíme příznak a při příštím volání `read` vrátíme nulu signál konce souboru dat.

5.3 Obsluha přerušení

Úlohou obsluhy přerušení je informovat čekající funkci `read()` o dokončení převodu, aby mohla převedená data odevzdat do uživatelského prostoru.

```
static irqreturn_t pir_irqhandler(int irq, void *dev_id)
{
    if (adc_readb(SR, EOC(pir_adc_channel))) {
        adc_data = adc_readl(CHR(pir_adc_channel));
        complete(&data_ready);
```

```
    }

    return IRQ_HANDLED;
}
```

V obsluze přerušení také vyčteme převedenou hodnotu z datového registru, čímž se zruší příznak přerušení a procesor bude připraven na další. Přenesená data uložíme do globální proměnné, kde si je vyzvedne funkce `read()`.

5.4 Další možnosti vývoje

Tento driver jsem nestihl dovést do pokročilejší podoby. Určitě by bylo užitečnější, kdyby nejen vyčítal data z A/D převodníku, ale také je zpracovával a detekoval v nich situace, kdy se před senzorem někdo pohybuje.

V takovém případě by měl driver obsahovat kruhový buffer, do kterého by se zaznamenávala určitá délka průběhu analogového signálu a tento úsek dat by se zpracovával. Operace `read()` by potom namísto digitální hodnoty signálu z čidla vracela logickou hodnotu podle toho, jestli byl před kamerou detekován pohyb nebo ne.

Kapitola 6

USB velkokapacitní zařízení

Jak už bylo zmíněno v úvodních kapitolách, plánovanou funkcí kamery je střežení vymezeného prostoru a v případě detekce pohybu sejmutí obrázků, jejich uložení do paměti a následné upozornění uživatele formou SMS nebo MMS. Aby si uživatel po příchodu ke kameře mohl sejmuté obrázky prohlédnout, bude kamera podporovat připojení k PC protokolem USB. Počítači by se kamera měla představit jako velkokapacitní paměťové zařízení (USB file storage device), podobně jako např. USB klíčenka.

Drivery pro zařízení s USB protokolem se dělí na dvě skupiny podle toho, na které straně kabelu budou aktivní:

(a) **Drivery na hostujícím systému** – tím je obvykle PC, k němuž připojujeme nějaké USB zařízení.

(b) **Drivery na zařízení** – fungují v embedded zařízení, které připojujeme k PC. Říká se jim také *USB gadget drivers*.

Z tohoto rozdělení je zřejmé, že pro kameru bude potřeba driver z druhé skupiny, tedy driver na zařízení.

V této části práce za mě veškerou práci už odvedli druzí – gadget driver pro velkokapacitní zařízení je v jádře již přítomen. Lze ho nalézt v adresáři `drivers/usb/gadget` spolu s několika dalšími drivery, které podporují např. emulaci sériového portu nebo ethernetového rozhraní na USB portu. Driver pro velkokapacitní zařízení nese název `file_storage`.

Tento driver je pro kameru překládán jako modul, přičemž při zavádění se mu jako parametr předá souborový systém, který má driver zpřístupnit. V kameře je to třetí oddíl Micro SD karty, který je formátován jako VFAT a má velikost cca 480 MB. Modul zavedeme do jádra příkazem

```
modprobe g_file_storage stall=n file=/dev/mmcblk0p3
```

Kapitola 7

Závěr

Úkolem mé diplomové práce bylo prozkoumat možnosti použití operačního systému Linux v kameře JABLOCOM EYE-02.

Úloha instalace operačního systému do kamery byla již vyřešena v rámci týmového projektu, já jsem se věnoval tvorbě ovladačů pro hardwarové součásti kamery. Psal jsem ovladače pro rozhraní ISI mikroprocesoru Atmel AT91SAM9260, připojený CMOS videosenzor Micron MT9V011 a detektor pohybu PIR. Požadavkem firmy JABLOCOM bylo také vytvořit ovladač pro USB device port, který by kameru prezentoval jako velkokapacitní paměťové zařízení, avšak takový ovladač je již součástí jádra, proto jsem tuto úlohu neřešil, neboť by to byla zbytečná práce.

Vytvořil jsem ovladač pro rozhraní ISI, které slouží pro připojení CMOS videosenzorů. Můj ovladač nepokrývá všechny možnosti tohoto rozhraní. Některé funkce jsou omezeny na formát obrazových dat, který bohužel použitý videosenzor neposkytuje. Nepsal jsem tedy kód, který bych neměl možnost odladit. Dále jsem se při práci zaměřil spíše na požadavek sejmutí jednoho snímku na povel a neřešil jsem možnosti streamování.

Spolu s ovladačem pro ISI jsem vytvořil také ovladač pro použitý videosnímač. Snímač se totiž neovládá pouze prostřednictvím rozhraní ISI, to slouží jen pro přenos dat. Je také potřeba provádět konfiguraci snímače a to lze např. prostřednictvím sběrnice I2C, jako u použitého senzoru firmy Micron. Tento způsob ale není unifikovaný, může se u různých typů senzorů lišit. Proto by tento úkol měl provádět zvláštní driver. Jednak je to čistší, protože driver pro ISI by měl obsluhovat skutečně jenom ISI a jinou práci nechat jiným driverům, a dále je potřeba odlišit různé typy senzorů a jejich specifika. Můj driver pro ISI tedy definuje API a může díky tomu používat každý senzor, jehož driver s tímto API spolupracuje.

Dále jsem vytvořil ovladač pro detektor pohybu PIR. Tento ovladač jsem nestihl dokončit, takže jenom přenáší signál, který tento detektor generuje. Bylo by vhodné rozšířit ho o zpracování signálu tak, aby uživateli poskytl jasnou informaci o přítomnosti pohybujícího se objektu před kamerou.

Přínos této práce vidím v získání dovedností při tvorbě linuxových ovladačů, v bližším seznámení se s tímto operačním systémem a jeho využitím na procesorech ARM.

Literatura

[1] J. Corbet, A. Rubini, G. Kroah-Hartman: *Linux Device Drivers, Third Edition*, O'Reilly Media (2005)

[2] P. J. Salzman, M. Burian, O. Pomerantz: *The Linux Kernel Module Programming Guide*, (2001)

[3] *AT91SAM9260 Preliminary, Revision 6221G*, Atmel Corporation (2008)

[4] *Katalogový list CMOS snímače MT9V011, Rev B*, Micron Technology, Inc. (2004)

[5] N. Matthew, R. Stones: *Beginning Linux Programming, 4th Edition*, Wiley Publishing (2008)

[6] Kurzy na serveru `http://free-electrons.com`, Free Electrons 2004–2008

[7] R. Krátký: Seriál *Jaderné noviny – Video4Linux2 API* na serveru AbcLinuxu.cz (`http://www.abclinuxu.cz`), publikován v letech 2006–2007

[8] M. H. Schimek, B. Dirks, H. Verkuil, M. Rubli: *Video for Linux Two API Specification, Revision 0.24*, (1999–2008)

[9] Dokumentace zdrojového kódu linuxového jádra (adresář `Documentation`)

Dodatek A

Obsah přiloženého CD

K tištěné verzi práce je přiloženo CD s následujícím obsahem:

- elektronická verze práce ve formátu PDF
- zdrojový kód driveru `atmel-isi`
- zdrojový kód driveru `mt9v011`
- zdrojový kód driveru `pir`
- patche pro doplnění podpory pro ovladače do zdrojových kódů linuxového jádra verze 2.6.29.3
- katalogový list mikroprocesoru Atmel AT91SAM9260
- katalogový list CMOS videosnímače Micron MT9V011