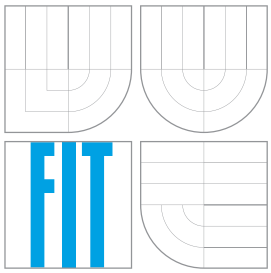
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

NÁVRH OVLADAČŮ PRO VESTAVĚNÉ SYSTÉMY V OS LINUX

LINUX-BASED DRIVERS FOR EMBEDDED SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAROSLAV KOPÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ROLAND DOBAI, Ph.D.

BRNO 2016

Zadání bakalářské práce/16882/2015/xkopac01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Kopáček Jaroslav**
Obor: Informační technologie
Téma: **Návrh ovladačů pro vestavěné systémy v OS Linux**
Linux-Based Drivers for Embedded Systems
Kategorie: Operační systémy

Pokyny:

1. Seznamte se s problematikou návrhu ovladačů pro vestavěné systémy v OS Linux.
2. Navrhněte a implementujte vzorové ovladače pro vybrané úlohy.
3. Ověřte funkčnost implementovaných ovladačů.
4. Zhodnoťte dosažené výsledky a diskutujte jejich vlastnosti a možnosti dalšího vývoje.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodu 1 zadání, demonstrace rozpracovanosti bodu 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dobai Roland, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Problematika návrhu a tvorby ovladačů je obšírná a proto se v této práci zaměříme na návrh ovladačů pro zařízení s programovatelným hradlovým polem. Oproti procesorům typu aplikačně-specifického integrovaného obvodu, kde je funkcionalita pevně daná, je pro každou novou konfiguraci programovatelného hradlového pole nutné pro správnou funkčnost vytvořit nový ovladač. Tato práce se zabývá analýzou požadavků a možných variant řešení návrhu a implementace ovladačů pro vestavěné systémy založené na operačním systému Linux a možnosti automatizace vývoje. Součástí práce je též navrhnutí a implementace generátoru, jež bude schopný generovat ovladače pro takové systémy. Tvorba ovladače je modulární, aby bylo možné generovat ovladač, jež obsahuje požadovanou funkcionalitu. Navrhnutý generátor byl otestován na úloze řízení svitu různých světlo emitujících diod, které jsou využity na diagnostické účely vestavěného systému.

Abstract

Issues of design and writing device driviers is wide-ranging and therefore in this thesis we focus on the design of drivers for devices with field-programmable gate array (*FPGA*). Compared to the application-specific integrated circuit procesors, where functionality is immutable, it is necessary for each new FPGA configutation to write a new driver for the required behavior. This thesis deals with the analysis of requirements and posible solutions of designing and implementation of device drivers for embedded systems based on OS Linux and the possibility of development automatization. This thesis includes the design and implementation a driver generator which can generate Linux-based drivers for embedded systems. The driver generator is modular so the final driver can contain only the required functionality and no unnecessary functionality. Designed driver generator has been tested on the task of controlling light-emitting diodes which are used for diagnostics of the embedded system.

Klíčová slova

operační systém, jádro, strom zařízení, modul, ovladač, vestavěné systémy, Xilinx, Zynq, Zedboard, FPGA

Keywords

OS, kernel, device-tree, module, driver, embedded systems, Xilinx, Zynq, Zedboard, FPGA

Citace

KOPÁČEK, Jaroslav. *Návrh ovladačů pro vestavěné systémy v OS Linux*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dobai Roland.

Návrh ovladačů pro vestavěné systémy v OS Linux

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Rolanda Dobaie. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jaroslav Kopáček
17. května 2016

Poděkování

Chtěl bych poděkovat panu Rolandu Dobaiovi za pomoc při tvorbě této práce. Jsem rád, že jako vedoucí mne při tvorbě této práce podporoval a poskytoval věcné připomínky.

Obsah

Slovník

- **API** - Application programming interface - Rozhraní pro programování aplikací
- **ASIC** - Application-specific integrated circuit - Aplikačně specifický integrovaný obvod
- **AXI** - Advanced eXtensible interface - Rozhraní pro komunikaci s periferiemi
- **DDR** - Double Data Rate - typ operační paměti
- **DEVFS** - virtuální souborový systém
- **DT** - device tree - Strom zařízení
- **DTC** - device tree compiler - Překladač zdrojového kódu stromu zařízení
- **FPGA** - Field-programmable gate array - Programovatelné hradlové pole
- **GPIO** - General Purpose Input/Output - vstupy/výstupy obecného určení
- **GPL** - GNU General Public License - všeobecná veřejná licence GNU
- **IP core** - Intellectual Property core - Blok tvořící určitou funkcionalitu v FPGA konfiguraci
- **JTAG** - Joint Test Action Group - architektura pro testování hardware
- **kernel** - jádro operačního systému
- **kernel space** - jaderný paměťový prostor, v němž běží veškeré jaderné procesy
- **LED** - Light-emitting diode - světelo vyzařující dioda
- **Linux** - operační systém typu UNIX
- **modul** - část jaderného kódu, jež není zkompilována jako součást jádra ale lze jej dynamicky načítat a odebírat
- **module stacking** - vrstvení modulů - koncept používaný při vývoji modulů
- **open source** - software s veřejně dostupným zdrojovým kódem, obvykle šířeným pod svobodnou licencí
- **OS** - Operating system - operační systém
- **PCI** - Peripheral Component Interconnect - počítačová sběrnice pro připojení periferií k základní desce
- **PL** - Programmable logic - programovatelná logika hradlového pole
- **PS** - Processing system - procesorová část použitého hradlového pole
- **PROCFS** - virtuální souborový systém

- **RSoC framework** - Reconfigurable SoC framework - návrhová kostra a knihovna potřebných nástrojů pro vývoj na SoC s FPGA
- **SoC** - System on Chip - systém, jež má části různého typu umístěny na křemíkové plošce
- **SSD** - Solid State Drive - pevný disk bez pohyblivých částí
- **SYSFS** - virtuální souborový systém
- **UART** - Universal asynchronous receiver/transmitter - rozhraní pro asynchronní komunikaci
- **user space** - uživatelský paměťový prostor, v němž běží veškeré uživatelské procesy

Kapitola 1

Úvod

Vestavěné systémy mnohdy představují specializovaná výpočetní zařízení zaměřená na konkrétní úlohu, které jsou přizpůsobeny jak po výkonnostní stránce, tak i po softwarové. Hardware lze optimalizovat dle různých požadavků, např. ceny, příkonu, velikosti či výkonu. Zařízení tak mohou obsahovat i specializované obvody nebo komponenty. Typicky si lze pod vestavěnými systémy představit různé mikrokontrolery a jiná zařízení sloužící jako řídící systémy jiných zařízení od alarmu, přes CNC (*computerized numerical control*) soustruhy až po elektroniku obsaženou v televizi.

Dnešní vestavěné systémy ale nemusí nutně sloužit pouze jako řídící jednotky, ale vzhledem k technologickému pokroku vedoucímu k dobrému poměru vysokého výkonu, nízké ceny a přijatelného příkonu je možné vytvářet i systémy. To vede k postupně se snižujícímu rozdílu mezi vestavěnými systémy a výpočetními systémy obecného určení (nejen osobní počítače ale i servery) [6].

Pokročilé vestavěné systémy je možné navrhnout na bázi architektury Soc (*System-on-Chip*), kdy jsou části systému umístěny na jedné komponentě. Vestavěný systém může obsahovat jako hlavní jednotku mikrokontroler (8 nebo 16 bitový) nebo procesor (32 či 64 bitový). Dále, paměť s kapacitou pohybující se od pár kilobyte do stovek megabyte. Typickými prvky tedy jsou výpočetní jednotka, operační paměť, řadič přerušení, časovač a dále řadiče pro komunikaci s periferiemi, např. řadič pro vstupně-výstupní rozhraní obecného určení (GPIO) nebo USB řadič. V některých případech je zde možné nalézt i další řadiče, např. řadič přímého přístupu do paměti (*DMA controller*) či různé převodníky. Součástí vestavěného systému může být i programovatelné hradlové pole (FPGA), jež nám umožní optimalizovat systém pro řešení daného problému, aniž bychom při změně úlohy museli vyměnit celý systém za jiný [3].

Software řídící činnost vestavěného systému se liší dle hardwarových prostředků a složitosti systému. Jednoduché mikrokontrolerové systémy bývají řízeny softwarem uloženým v paměti ROM (*firmware*). U komplexnějších systémů, obsahující procesor, je činnost řízena zejména operačním systémem, jež je tvořen minimálně jádrem (*kernel*) operačního systému [6].

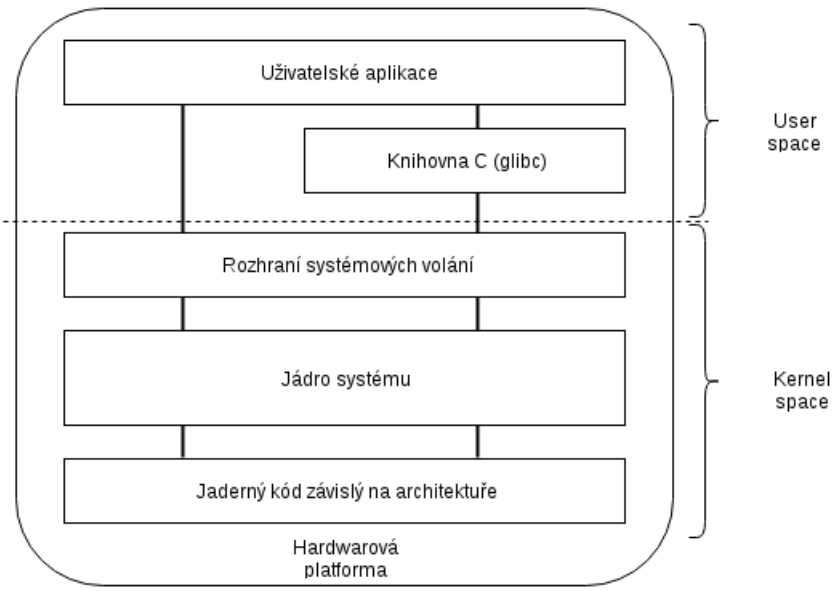
Vestavěné systémy je ovšem možné používat i bez operačního systému (tzv. *barematal*) [10]. V tomto případě se pracuje přímo se samotným hardware, nicméně práce s ním nemusí být úplně snadná, případně efektivní. Například pokud máme vícejádrový procesor, je nutné jej synchronizovat, aby systém pracoval správně. Mohou zde vznikat i další zpoždění, jež nepříznivě ovlivňují běh a efektivitu systému. Proto, pokud pracujeme s výkonnějším zařízením, je zcela namístě použít operační systém, jež nám zefektivní a usnadní práci s daným zařízením.

Kapitola 2

Linux

Linuxové jádro (dále jen jádro) je jádro operačního systému typu UNIX, jež je *open source* a lze jej modifikovat a používat dle vlastních potřeb. Jádro samotné podporuje mnoho procesorových architektur, například x86, PowerPC, SPARC či ARM. Tento fakt umožňuje použití v široké škále zařízení, od vestavěných systémů přes mobilní zařízení a stolní počítače až po servery a superpočítače. Zároveň s tím je podporováno mnoho zařízení, takže jej lze použít na starších i novějších zařízeních. Nicméně, ne všechny zařízení jsou podporována, nebo jsou podporovány pouze proprietárními ovladači, jež z nějakého důvodu nechceme nebo nemůžeme využít nebo máme specifické zařízení, jež bychom rádi používali. V těchto případech vyvstává potřeba vytvořit podporu jádra pro takové zařízení.

Linuxové jádro je monolitického typu, což je znázorněno ve schématu systému Linux na obrázku 2.1. To znamená, že veškerý jaderný kód běží ve stejném paměťovém prostoru (*kernel space*). Dále nám umožňuje dynamicky za běhu načítat a odebírat moduly, důsledkem čehož lze docílit minimalistického jádra a další zařízení, případně volitelné vlastnosti můžeme přidávat až v případě potřeby za běhu. To nám dává možnost jádro přizpůsobit našim potřebám a této vlastnosti lze využít zejména u vestavěných systémů, kde často máme omezené paměťové prostředky [4].

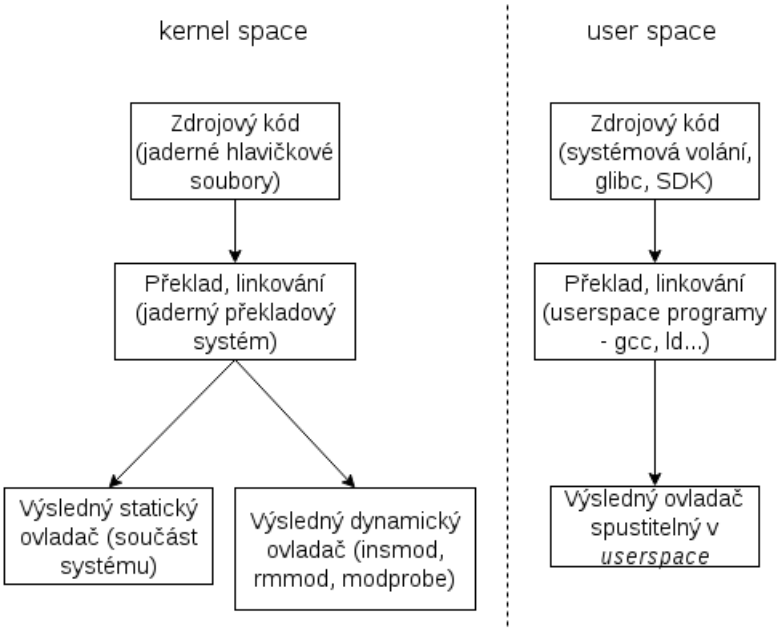


Obrázek 2.1: Schéma systému Linux

2.1 Ovladače

Ovladač představuje softwarové rozhraní mezi hardware a operačním systémem. Umožňuje tak operačnímu systému i uživatelským aplikacím používání daného zařízení, aniž by bylo nutné znát podrobnou technickou specifikaci. Ovladač je software závislý na daném typu zařízení a je i specifický vzhledem k použitému operačnímu systému [2].

Je možné vytvořit dva různé typy ovladačů, jejichž rozdíl je znázorněn na obrázku 2.2. Ovladač zařízení tvořený pro jaderný paměťový prostor může být v jednom ze tří stavů a to (1) zkompilovaný a plně integrovaný do jádra, (2) ve formě modulu nebo (3) zcela nepoužíván (pouze ve formě zdrojového kódu). Je také možné vytvořit pseudo-ovladač, jež je spouštěn v userspace. Tyto ovladače využívají paměťově mapovaného přístupu (funkce `mmap` a `munmap`) [5].



Obrázek 2.2: Rozdíl mezi user space a kernel space ovladači

Ovladač je možné vytvořit několika způsoby. Musíme si tedy ujasnit, zda vytvoříme ovladač zcela od začátku, nebo zda budeme využívat již existujícího ovladače (*module stacking*). Vrstvení modulů (*module stacking*) je vhodné používat, neboť umožňuje tvorbu přehlednějších a snadno čitelných ovladačů. Typické použití si lze představit tak, že vytvoříme nízkoúrovňovou vrstvu ovladače, jež pracuje se samotným hardware a vytváří tak abstraktní rozhraní nad prací s ním. Na vyšších úrovních abstrakce se pak využívá rozhraní vytvořeným nižšími vrstvami. Je tedy doporučeno používat skládání modulů, neb nám to usnadní mnoho práce.

Pro vytvoření ovladače zařízení je nejdříve nutné alokovat hlavní číslo (*major*) a vedlejší číslo (*minor*) číslo, což lze udělat automatizovaně nebo manuálně. K tomuto lze použít například funkci pro alokaci čísla `alloc_chardev_region`, jež nám přidělí systém. Pokud chceme zařízení zařadit do třídy, je nutné tuto třídu vytvořit. K tomu lze použít například funkci `create_class`. Následně je nutné ovladač zaregistrovat. Výše uvedené se provádí při inicializaci ovladače a při odstranění ovladače je nutné alokované prostředky uvolnit, což lze provést funkcemi `device_destroy`, `class_destroy` a `unregister_chrdev_region`.

Pokud využíváme vrstvení modulů, tak výše zmíněná nutnost registrace ovladače odpadá. Namísto toho se volá makro, jež zaregistruje náš ovladač do nadřazeného ovladače [5].

2.2 Typy zařízení

Zařízení se v Linuxu/UNIXu dělí do tří základních, jimiž jsou:

- *Bloková zařízení* - jsou zařízení, která jsou dostupná jako uzly souborového systému v adresáři `/dev`. Takovými zařízeními jsou například pevné a optické disky.
- *Znaková zařízení* - jsou zařízení, jež mohou být přístupné jako tok bitů. Znaková zařízení mají na starosti implementaci tohoto chování. Do této kategorie spadá většina zařízení a to jak platformních (LED, přepínače, tlačítka...), tak i neplatformních (zařízení připojená přes sériový či paralelní port).
- *Síťová rozhraní* - jsou většinou hardwarová, nicméně mohou být i softwarová. Síťová rozhraní jsou řízena síťovým subsystémem jádra a jsou odpovědné za odesílání a přijímání datových paketů. Typicky se jedná o síťové karty.

Znaková zařízení jsou zařízení, ke kterým lze přistupovat jako k proudu bajtů. Za toto chování je odpovědný ovladač takového zařízení, jež musí tuto činost implementovat. Obvykle se jedná o operace otevření (*open*), uzavření (*close*), čtení (*read*) a zápisu (*write*). Bloková zařízení, v systémech Unixového typu, umožňují, oproti znakovým zařízením, přenášet jeden či více celých bloků. Linuxové jádro místo toho umožňuje čtení a zápis jako u znakových zařízení. Rozdíl je tak pouze ve způsobu, jakým jsou data v jádru spravována. Obvykle se používají pro úložná zařízení jako jsou pevné disky, SSD, optická uložiště či flash uložiště. Síťová zařízení jsou podobná blokovým zařízením. Umožňují výměnu dat mezi dalšími zařízeními.

Zařízení může být jak fyzické, jako LED (*Light-emitting diode*) nebo síťová karta, tak i virtuální jako například virtuální síťový most (*bridge*) pro komunikaci mezi virtuálním strojem a síťovým rozhraním připojeným k internetu či lokální síti. V případě znakových virtuálních zařízení můžeme uvést například `/dev/null` nebo `/dev/zero`.

2.3 Strom zařízení

Strom zařízení (*device tree* - DT) je koncept pro popis hardware. Jedná se o stromovou datovou strukturu s uzly, jež popisují fyzické zařízení v systému. Pro práci se DT v ovladačích slouží zdrojové soubory mající předponu `of` a je nutné při konfiguraci jádra parametr `CONFIG_OF`.

Strom zařízení může být ve třech formách:

- Zdrojový kód - *.dts(i)
- Objektový kód - *.dtb
- Souborový systém v adresáři `/proc` - vhodné pro ladění a reverzní inženýrství.

Zdrojový kód je překládán do objektového kódu pomocí speciálního překladače (*Device tree compiler* - DTC). Tento překladač je součástí jaderných zdrojových kódů.

2.4 Platformní ovladače

Platformní ovladače se používají pro taková zařízení, která nelze ze systému odebrat. Je to dáno tím, že zjišťování jejich přítomnosti není prováděno na úrovni hardware ale na úrovni software. Takový hardware musí být znám a měl by být registrován během zavádění systému co nejdříve, neboť jde většinou o kritický hardware systému. Platformní zařízení se vážou na ovladač shodou jmen. K tomu slouží v DT parametr `.compatible`, který je poté vyhledán ovladačem. K definování jmen slouží struktura `of_device_id`, makro `MODULE_DEVICE_TABLE` a samotná struktrura `of_device_id` musí být uvedena ve struktuře `platform_driver`.

2.5 Neplatformní ovladače

Oproti platformním ovladačům se normální ovladače používají pro zařízení, jež lze ze systému odebrat a to jak ve vypnutém stavu tak i zařízení s funkcí hot-plug. Detekce zařízení je na hardwarové úrovni. Typickým příkladem tak je například sběrnice PCI, jež porovnává PCI Device ID každého zařízení s tabulkou podporovaných PCI ID poskytované ovladači.

2.6 Adresář /dev

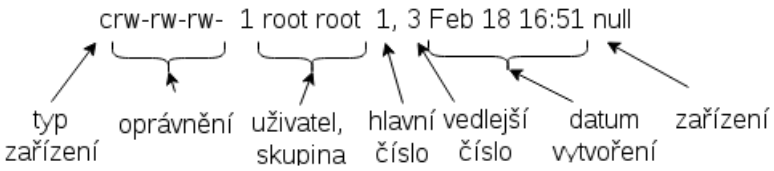
V Linuxu/UNIXu je vše reprezentováno soubory. Není tomu jinak ani u zařízení. Tyto soubory jsou organizovány ve speciálním adresáři `/dev`, jež je vytvořen jako přípojný bod pro virtuální systém DEVFS. Jednotlivá zařízení zde mají každý svůj soubor. Tyto soubory mohou býti organizovány samostatně nebo též ve třídách. Třída je určená ke sdružování zařízení s podobnou funkčností a vede tak ke snažší orientaci v adresáři, neboť tak vytváří hierarchickou souborovou strukturu.

Vlastnosti každého souboru zařízení lze zjistit použitím příkazu `ls -l`, ze kterého lze zjistit typ zařízení, oprávnění, vlastníka, hlavní a vedlejší čísla, datum poslední změny a název souboru.

```
crw-rw-rw-  1 root root      1,   3 Feb 18 16:51 null
crw-rw-rw-  1 root root      1,   8 Feb 18 16:51 random
brw-rw----  1 root disk      8,   0 Feb 18 16:51 sda
brw-rw----  1 root disk      8,   1 Feb 18 16:51 sda1
crw--w----  1 root tty       4,   0 Feb 18 16:51 tty0
crw--w----  1 root tty       4,   1 Feb 18 16:51 tty1
crw-rw-rw-  1 root root      1,   5 Feb 18 16:51 zero
```

Obrázek 2.3: Ukázka výstupu `ls -l /dev`

Popis výstupu se nachází na obrázku 2.4. První znak řádku nám udává, o jaký typ zařízení se jedná. Pro znaková zařízení je použito `c` (*char*), pro bloková zařízení je použito `b` (*block*). Dále, hlavní číslo slouží k rozpoznání ovladače daného zařízení. Vedlejší číslo slouží pro identifikaci zařízení, na které je odkazováno. Z výpisu uvedeného na obrázku 2.3 lze vypozorovat, že například pro null, random a zero používají stejný ovladač. Liší se pouze vedlejším číslem.



Obrázek 2.4: Popis výstupu `ls -l /dev`

2.7 Ovládání z userspace

Za účelem výměny dat mezi jaderným prostorem a uživatelským prostorem poskytuje Linux několik virtuálních souborových systémů. Obvykle každý soubor obsahuje pouze jednu hodnotu, nicméně je možné v jednom souboru mít uloženou i sadu hodnot. Z uživatelského prostoru se využívá běžných funkcí `read` a `write`. V jaderném prostoru se využívá funkcí `copy_to_user` a `copy_from_user`.

Těchto virtuálních souborových systémů existuje několik a mají i podobnou funkčnost, ale mohou mít rozdílné hlavní zaměření. Nás ovšem budou zajímat dva konkrétní, PROCFS a SYSFS.

2.7.1 Adresář /proc

Virtuální souborový systém PROCFS bývá obvykle připojen do adresáře `/proc`. PROCFS byl původně navržen pro exportování různých informací o procesech jako například stavy procesů či otevřené souborové deskriptory. Bývá ale používán pro získání dalších informací o běžícím systému jako informace o CPU, o dostupné paměti nebo verzi jádra, informace o síťovém připojení a mnoho dalšího.

Prostřednictvím tohoto virtuálního souborového systému by měli býti poskytovány pouze informace o procesech. Poskytování jiných informací prostřednictvím PROCFS je zastaralé [1]. Pro tyto účely slouží SYSFS, jež je popsán v následující podkapitole.

2.7.2 Adresář /sys

SYSFS byl navržen pro reprezentaci celého modelu zařízení (*device model*) tak, jak jej vidí Linuxové jádro. Obsahuje tak informace o zařízení, sběrnicích, jejich propojeních, ovladačích. Oproti PROCFS je SYSFS i velmi strukturovaný a obsahuje mnoho odkazů mezi jednotlivými adresáři [1].

Základní strukruta obsahuje následující adresáře:

- `/sys/block/` - obsahuje všechna známá bloková zařízení jako například hda/, ram/, sda/,
- `/sys/bus/` - obsahuje všechny registrované sběrnice. Každý podadresář obahuje další dva podadresáře:
  - `device/` - pro všechna zařízení připojené ke sběrnici,
  - `driver/` - pro všechny ovladače přiřazené ke sběrnici,
- `/sys/class/` - pro každý typ zařízení je zde podadresář,
- `/sys/device/` - všechna známá zařízení organizována podle sběrnice, k níž jsou připojeny,

- `/sys/firmware/` - soubory řídící firmware některých zařízení,
- `/sys/fs/` - soubory pro řídící souborový systém,
- `/sys/kernel/` - adresáře (přípojné body) pro ostatní souborové systémy,
- `/sys/module/` - každý načtený modul je reprezentován adresářem,
- `/sys/power/` - soubory spravující napájení některých zařízení.

Pro řízení stavu zařízení tak lze využít soubory vytvořené ovladačem právě v tomto adresáři. Pokud chceme například řídit svit LED diody připojené k GPIO pomocí výchozího ovladače od firmy Xilinx, nalezneme zde patřičný adresář, ve kterém lze najít soubory *brightness* či *trigger*, jejichž prostřednictvím můžeme ovládat svit dané diody. Toto je ovšem závislé na implementaci a každý ovladač, byť se stará o stejné zařízení či periferii, může implementovat rozdílné rozhraní pro ovládání z *user space*.

2.8 Stávající řešení

Problematiku tvorby ovladačů v *HW/SW Co-design* dnes řeší například aplikace DDGen (*Device Driver Generator*)[1] vytvořená společností Vayavya Labs, která zvládá automatizovanou tvorbu ovladačů pro různá zařízení a to pro různé platformy včetně OS Linux a Windows®.

Další variantou, jež se zaměřuje na OS Linux a zařízení Zynq™, je *RSoC framework* vytvořený studentem magisterského studia na FIT VUT v Brně Janem Viktorinem. Tento *framework* vytváří vrstvu mezi PS a PL a unifikuje tak přístup k vývoji aplikací urychlovaných pomocí FPGA [6].

Nevýhodou stávajících řešení je, že žádné z nich nemá otevřený kód a nejsou ani volně dostupné.

2.8.1 Cíle práce

Cílem této práce je navrhnout a implementovat generátor, jež bude schopen produkovat ovladače dle parametrů uvedených v definičním souboru. Ovladače budou generované pro znaková zařízení a tato práce tak hodlá nastínit, jakým způsobem lze řešit problém *HW/SW Co-design* v případech, kdy návrhář vytvoří vlastní konfiguraci FPGA a potřebuje k němu vytvořit ovladač, aby bylo možné danou konfiguraci využívat, avšak nezná jaderné API pro tvorbu ovladačů.

2.8.2 Porovnání se stávajícími řešeními

Nevýhodou dosavadních řešení je, že žádný produkt není volně dostupný. Jedním z cílů této práce je tedy vytvořit generátor ovladačů, jež bude volně dostupný, byť za cenu menší komplexnosti řešení. Nástroj je navrhován tak, aby bylo možné jej později, například formou diplomové práce, rozšířit o generování ovladačů pro bloková a síťová zařízení.

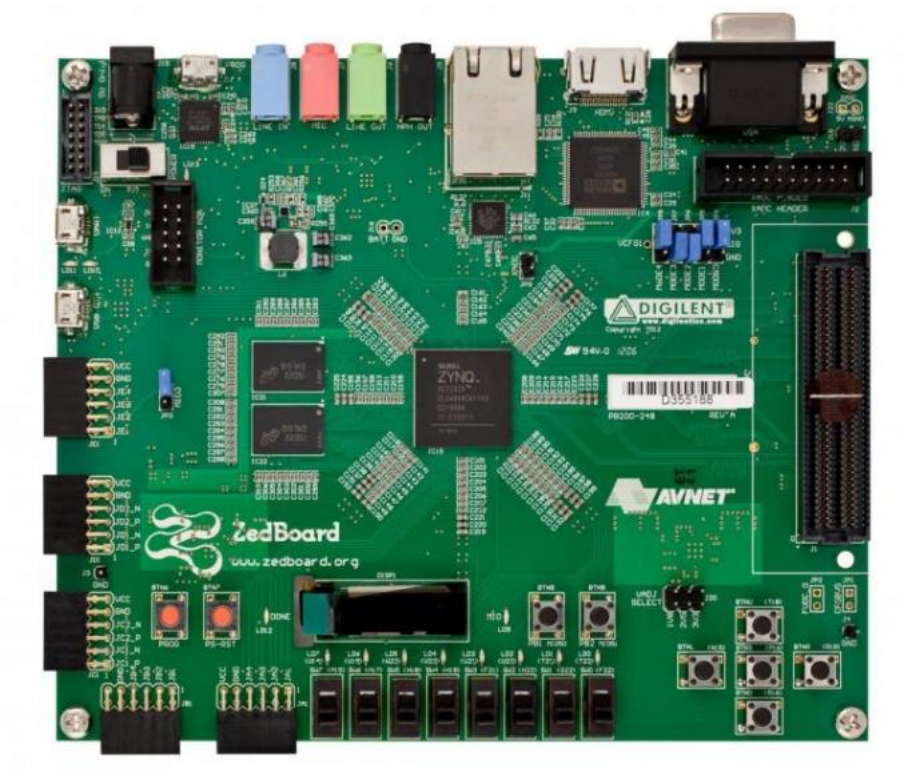
[1] http://vayavyalabs.com/technology/ddgen/

Kapitola 3

Návrh automatického generátoru ovladačů

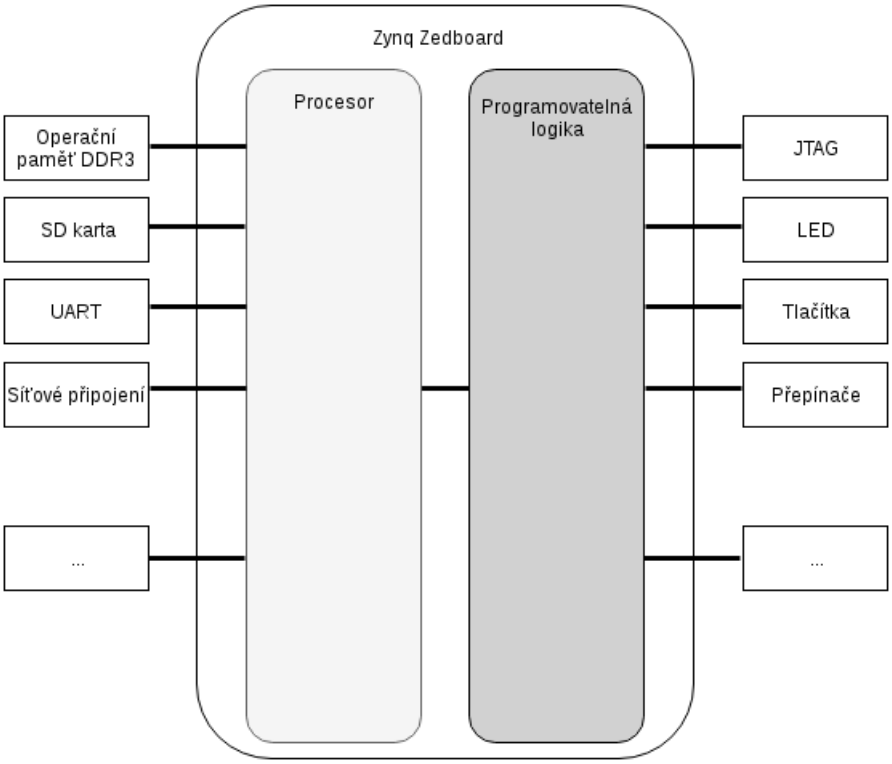
3.1 Referenční zařízení

Jako referenční zařízení bylo zvoleno SoC řešení Xilinx ZedBoard Zynq™-7000 Development Board, jež je znázorněno na obrázku 3.1, neboť se jedná o jedno z nejmodernějších zařízení využívající koncept Soc a je dostupné zaměstnancům pracoviště, kde byla tato práce vypracována. Na této desce byl použit výchozí operační systém *Linux-4.0.0*. Zynq je založen na procesoru ARM® Cortex™-A9 a dále obsahuje programovatelné hradlové pole (FPGA), jež budeme v ovladačích využívat.



Obrázek 3.1: Referenční zařízení

Deska obsahuje mnoho periferií, jako například DDR3 operační paměť, uživatelské LED a přepínače, vstupy a výstupy [9]. Zjednodušené blokové schéma lze nalézt na obrázku 3.2. K některým částem systému má přístup přímo procesor (*Processing system* - PS), jiné jsou přístupné přes programovatelnou logiku (*Programmable logic* - PL).



Obrázek 3.2: Zjednodušené blokové schéma Zynq - ukázka zapojení některých částí systému

3.2 Typy zařízení

Tato práce se zabývá znakovými ovladači, která jsou, oproti blokovým či síťovým ovladačům, postačující pro základní práci se systémem implementovaným v FPGA. Lze tak implementovat přístup k jednoduchým periferiím, ale třeba i datové přenosy pro datová zařízení, které ale nebudou natolik efektivní jako při použití blokových ovladačů.

3.3 Strom zařízení

Generátor ovladačů je tvořen tak, aby byl schopný generovat platformní ovladače. Tyto ovladače využívají k rozpoznání zařízení hodnotu `compatible`. Je tedy nutné, aby zařízení bylo uvedeno v DT. Jako možné rozšíření by bylo možné imlementovat neplatformní ovladače, jež nevyužívají hodnotu `compatible`.

3.4 Možnosti rozšíření

Rozšíření základního generátoru, jež generuje platformní ovladače v zjednodušeném režimu (používá se pouze struktura pro znakové zařízení a veškeré operace je nutné provádět čtením

a zápisem přímo nad registrem) je možnost rozšíření o neplatformní ovladače, případně o specializovaný režim (používající specializovaných struktur).

V případě, že bychom chtěli generátor rozšířit o možnost tvorby ovladačů pro bloková a síťová zařízení, je nutné vzít v úvahu, jaký typ a případně způsob implementace ovladače umí generátor produkovat. Zde tedy přichází v úvahu možnosti dle typu ovladače:

- platformní a
- neplatformní;

a dle způsobu implementace:

- zjednodušený - používá se struktura pro popis zařízení a operace čtení a zápisu probíhají přímo s daným registrem,
- specializovaný - používají se specializované struktury jako například `struct gpio_led_platform_data` pro platformní ovladač LED.

Rozšíření tak závisí na kombinaci dílčího způsobu generování, pokud tedy nechceme upravit všechny možné režimy. Základní generátor, kdy se využívá platformního typu ovladače ve zjednodušeném režimu, tak lze rozšířit poměrně snadno. Z pohledu samotného ovladače se vymění struktura `struct cdev` za `struct gendisk` a nahradí se volání funkcí pro znaková zařízení za funkce pro bloková zařízení, přičemž samotná struktura ovladače pak zůstává stejná [2].

Například ovladač pro bloková zařízení je vhodné použít při práci s úložnými zařízeními. Použití blokového ovladače oproti znakovému má výhodu, že umožňuje přístup prostřednictvím vyrovnávací paměti (*buffer*) a operace jsou tak rychlejší. Například zápis probíhá až v momentě, kdy je plná vyrovnávací paměť. Do té doby se systém nemusí zdržovat přístupem k pomalému zařízení [2].

Kapitola 4

Implementace a ověření generátoru ovladačů

Pro implementaci generátoru ovladačů byl jako implementační jazyk zvolen Python ve verzi 3, pro snadné prototypování a rychlou a efektivní implementaci výsledného řešení. Generátor je veřejně dostupný na webovém GIT repozitáři pod veřejnou licencí GPL ve verzi 3 [1]. Výstupem tohoto generátoru je ovladač v jazyce C, což vyplývá z toho, že jádro je napsáné v jazyce C. Vygenerovaný zdrojový kód ovladače je nutné poté zkompilovat do formy jaderného modulu.

4.1 Architektura generátoru

Generátor je tvořen několika třídami, které jsou znázorněny ve schématu na obrázku 4.1, jež implementují specifickou funkčnost. Výjimku tvoří specializované třídy (`Probe`, `Remove` a `SysFS`), jež jsou obsáhlé a bylo lepší je vyčlenit do samostatných tříd.

4.1.1 Zpracování definičního souboru

Při spuštění generátoru se ihned začíná zpracovávat definiční soubor. K tomuto zpracování slouží třída `Parser`, jež obsahuje několik proměnných, do kterých jsou jednotlivé části definičního souboru ukládány. Využívá se zde nejen polí, ale i pokročilých datových struktur jako jsou množiny, seznamy či slovníky dle ukládaných dat.

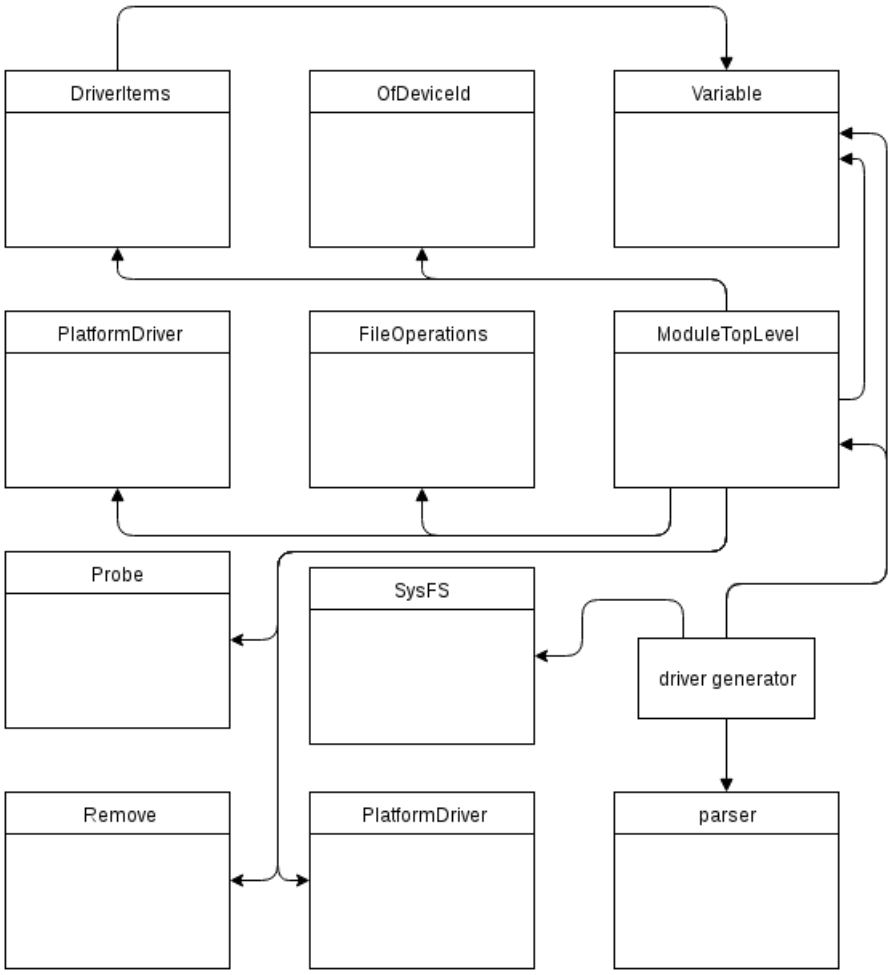
4.1.2 Systémové struktury

Jaderný kód obsahuje mnoho struktur a není tomu jinak i v případě rozhraní pro ovladače zařízení. Potřebné struktury jsou v generátoru implementovány a jsou reprezentovány jednotlivými třídami. Jedná se o struktury `of_device_id`, `platform_driver` a `file_operations`. Ty jsou, jak lze vidět na obrázku 4.1, reprezentovány třídami `OfDeviceId`, `PlatformDriver` resp. `FileOperations`. Tyto třídy jsou následně využity ve třídě `ModuleTopLevel`.

4.1.3 Data s globálním rozsahem

Data, jež mají být umístěna na globální úroveň ovladače jsou zpracována a ukládána ve třídě `ModuleTopLevel`, která tak zpracovává a uchovává direktivy preprocesoru (`include`

[1] https://bitbucket.org/jaroslav_kopacek/linux-driver-generator/



Obrázek 4.1: Schéma generátoru

a `define`), globální proměnné, struktury a funkce. Zde jsou využity i systémové struktury, které obsahují data z definičního souboru. Dále je zde uložena i struktura `driver_items`, která je reprezentována třídou `DriverItems`. Tato struktura zapouzdřuje některé proměnné potřebné pro ovladač.

4.1.4 Registrace ovladače

Funkce zavádějící ovladač do systému (`probe`) by měla být, vzhledem k vazbě na strukturu `platform_driver` umístěna ve třídě `PlatformDriver`, kde je také uvedena jako jedna z hodnot struktury. Nicméně z důvodu rozsahu samotné funkce byla její tvorba a generování vyčleněno do samostatné třídy `Probe`.

Tato třída se stará o zaregistrování ovladače do systému včetně podporovaných funkcionalit jako je například vytvoření souborů ve virtuálních souborových systémech `SYSFS`, případně `PROCFS`, pokud byla jejich funkcionalita specifikována v definičním souboru.

4.1.5 Odregistrace ovladače

Obdobně jako funkce pro registraci ovladače byla vyčleněna i opačná funkce, tedy funkce pro odregistrování ovladače ze systému (`remove`). Tato funkce však byla vyčleněna spíše z důvodu symetričnosti, aby nebyla jedna funkce uvedena ve třídě samotné struktury `platform_driver` a druhá jinde. To vede k větší přehlednosti jak při tvorbě kódu, tak i z významu obou funkcí, neboť se jedná o nejdůležitější funkce samotného ovladače.

4.1.6 Podpora SYSFS

Aby bylo možné s ovladačem komunikovat z uživatelského prostoru, je součástí generátoru i porpoda virtuálního souborového systému `SYSCFS`. Tato podpora je soustředěna do samostatného modulu, jež je tvořen třídou `SysFS`. Lze tedy prostřednictvím tohoto rozhraní jádra komunikovat.

4.2 Definiční soubor

Vstupem generujícího skriptu je definiční soubor, jež obsahuje informace nezbytné pro vygenerování zdrojového kódu ovladače. V tomto souboru jsou uvedeny základní informace o modulu jako je jméno autora, licence, popis a další, ale také informace popisující hardware, pro který má být ovladač vytvořen.

Definiční soubor je rozdělen do dvou hlavních bloků, kde každý blok má určitý význam. Dále je využíváno několika vnořených bloků, specifikující funkcionalitu určité části ovladače. Význam jednotlivých bloků bude vysvětlen v následujících podkapitolách.

4.2.1 Obecné informace o modulu

Obecné informace se specifikují v bloku `MODULE_DESC`, kde jsou vyžadovány položky jako jsou jméno autora nebo licence. Licence je zde vyžadována zejména kvůli jadernému makru `MODULE_LICENSE`, jež ovlivňuje dostupnost exportovaných symbolů z jaderných knihoven a modulů. V případě využití proprietární licence tak nelze využívat exportované symboly, jež jsou například pod licencí *GPL*.

Další prvky jako je popis jsou prvky volitelné, avšak doporučené a jsou využity například při tvorbě dokumentačních komentářů souboru.

4.2.2 Informace ovlivňující tvorbu výkonného kódu

Informace, jež jsou využity pro tvorbu výkonného kódu, jsou specifikovány v bloku `MODULE_DEF`. Tento blok obsahuje jak dílčí položky, tak i celé podbloky, v nichž lze například specifikovat funkcionalitu pro operace v prostředí virtuálního souborového systému `SYSCFS` či třeba při registraci ovladače (operace *probe*).

Vyžadovanými položkami jsou zejména jméno modulu, jméno ovladače a typ ovladače. Další jsou závislé na zvolené konfiguraci jako například řetězec pro vyhledání zařízení ve stromu zařízení (hodnota `compatible`) v případě zvolení generování platformního ovladače.

Hlavičkové soubory

Definiční soubor umožňuje do ovladače přidat hlavičkové soubory, které uživatel využívá v uživatelem definovaných částech kódu. K tomu slouží blok `HEADER`, ve kterém je možné tyto

hlavičkové soubory specifikovat. Blok podporuje přidání systémových hlavičkových souborů, ale také uživatelských hlavičkových souborů. Pro přidání systémových hlavičkových souborů slouží položka `header`, kdežto pro přidání uživatelem definovaných hlavičkových souborů složí položka `ownHeader`. Toto rozdělení je nutné proto, aby bylo možné na daný soubor aplikovat správný zápis vkládací (*include*) direktivy.

Symboly a makra

Další blok, jež je součástí bloku `MODULE_DEF`, je blok `DEFINE`. Tento blok je používán pro generování direktiv pro definování symbolů a maker (*define*). V tomto bloku je tak možné si definovat vlastní symboly a makra, která poté můžeme používat v uživatelských částech kódu.

Funkconalita pro dílčí operace

Je také možné specifikovat funkcionalitu pro dílčí operace, jako je například zaregistrování (operace *probe*) či odregistrování (operace *remove*) ovladače. Funkcionalitu lze specifikovat v bloku `FEATURE`. V tomto bloku je tak nutné uvést místo spuštění kódu (`exec`), proměnné a samotný kód. Hodnota `exec` je definována v rámci definičního souboru a říká generátoru, kam se má uvedený kód a proměnné umístit. Samotný kód musí být v jazyce C a je limitován vlastnostmi jádra (nelze použít `glibc` běžící v *user space*).

Blok `FEATURE` umožňuje specifikovat proměnné a struktury, které uživatel následně používá v kódu. Pro definici slouží bloky `VARIABLES` a `STRUCTURE`. Pokud definujeme strukturu, je nutné, aby byla ukončena pomocí `STRUCTURE_END`, neboť je možné do tohoto bloku zanořovat další bloky `VARIABLES` a `STRUCTURE`. Pokud bychom ale chtěli za strukturou definovat další proměnnou nebo strukturu, generátor by neměl jak poznat, že se jedná o dalš strukturu, případně proměnnou a nikoli o další položku aktuální struktury.

Pro zápis samotného kódu slouží blok `CODE`. Tento kód je přímo vkládán bez jakýchkoli změn na místo specifikované pomocí `exec`. V tomto kódu je možné používat definované symboly a makra uvedených v bloku `define` a samozřejmě i další části kódu, jež je obsažen v přiložených hlavičkových souborech.

V případě použití funkcí či maker dostupných v hlavičkových souborech systému Linux nebo i vlastních hlavičkových souborech je nutné tyto soubory definovat v definičním souboru. Stejně tak je možné definovat vlastní symboly či makra.

Podpora SYSFS

Aby bylo možné zařízení ovládat, vkládat data nebo je naopak získávat, je nutné využít některého z rozhraní pro komunikaci mezi jádrem a uživatelským prostorem. K tomu slouží blok `SYSFS`, který vytvoří atribut, jehož výsledkem je vytvoření souboru ve virtuálním souborovém systému `SYSFS`. U souboru je nutné nastavit oprávnění. Tato oprávnění jsou do jaderného makra kopírovány bez úprav, je tedy nutné použít oprávnění definované v jáderném kódu. Jako příklad lze uvést oprávnění `S_IWUSR`. Oprávnění je možné, stejně jako v jaderném kódu kombinovat.

Atribut má definovanou operaci `show`, která slouží pro čtení ze souboru. Funkcionalita, která se má vykonat při čtení ze souboru, je pak definována v bloku `SHOW`. Tím, že generátor do těla funkce nijak nezasahuje, je součástí těla i definice používaných lokálních proměnných.

Druhou operací, jíž atribut disponuje je `store`. Ta slouží pro zápis a tedy i předání dat z uživatelského prostoru do jádra. Vlastnostmi je jinak zcela totožná s operací `show`.

Výše uvedené operace lze využít nejen pro předání dat mezi jaderným prostorem a uživatelským prostorem nebo naopak, ale lze jej použít také pro vyvolání určité operace či procesu. Není tedy nutné používat prostředky pro výměnu dat, ale lze například vyvolat nějakou operaci či spustit například diagnostiku.

Operace `show` i `store` mají hlavičku funkce vždy totožnou s hlavičkou uvedenou v dokumentaci a zdrojových kódech jádra. Ta se pochopitelně liší pouze v identifikátoru funkce, který generátor vytváří sám. Lze se tak spolehnout na to, že argumenty funkcí mají shodné pojmenování a lze je tak snadno použít ve vkládaném kódu.

Podpora PROCFS

V aktuální implementaci je vytvořena pouze jednoduchá podpora, sloužící spíše pro testovací účely. Pokud chceme využít čtení ze souboru v `PROCFS`, je možné použít blok `FEATURE` a jako `exec` parametr zvolit `proc_read`. V případě čtení je nutné provést totéž jako u čtení, ovšem jako parametr `exec` je nutné zvolit `proc_write`.

Přesto je ale silně doporučeno pro komunikaci mezi ovladačem a uživatelským prostorem používat virtuální souborový systém `SYSFS`, kde je podpora kompletní.

Podpora definice funkcí

Další důležitou částí, jež lze v definičním souboru specifikovat, je blok `FUNCTION`, který umožňuje generování funkcí definovaných pouze uživatelem a to jak hlavičky funkce, tak i tělo funkce. Uživatel tak může definovat vlastní funkce, jež jsou zcela v režii uživatele.

Takto definovanou funkci je samozřejmě možné dále volat z uživatelem definovaných částí kódu.

4.2.3 Podpora pro rozšířitelnost

V definičním souboru se nacházejí některé položky, jež nejsou dosud implementovány, ale počítá se s nimi v případě rozšíření. Příkladem takové položky může býti `driver_type`, která umožní zvolit tvorbu platformního i neplatformního ovladače, avšak dosud není podporována, neb je možné generovat pouze platformní ovladače.

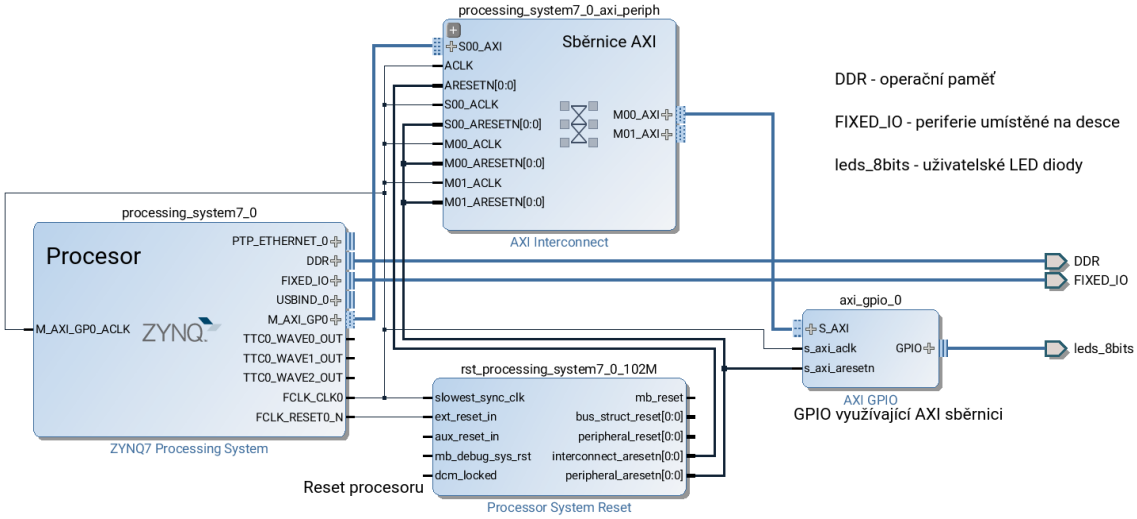
Další možností rozšíření může být například parametrizace bloku `FUNCTION`, tak aby bylo možné definovat komentář k funkci a samotný kód. V případě potřeby rozsáhlejší automatizace se nabízí vyčlenění proměnných, obdobně jako v bloku `FEATURE`. Případně i oddělení názvu, typu návratové hodnoty a parametrů.

Rozšíření je též možné u podpory `PROCFS`, jenž nyní pracuje v testovacím režimu, avšak prostředí je připraveno na plnohodnotnou podporu.

4.3 Příprava prostředí

Předtím, než se pustíme do tvorby samotného ovladače pomocí generátoru, je nutné si připravit samotné prostředí, které je poté použito pro výsledný ovladač.

Je nutné nejdříve provést návrh samotné konfigurace FPGA, podobně jako je znázorněno na obrázku 4.2, a vytvořit konfigurační bitstream, kterým se FPGA konfiguruje. Následně je nutné si vygenerovat novou konfiguraci pro spuštění systému, tedy zejména nový strom zařízení, zavaděč systému a další nezbytné soubory [8].



Obrázek 4.2: Ukázka blokového návrhu v Xilinx Vivado

4.3.1 Blokový návrh

Na obrázku 4.2 lze vidět blokový návrh, jež je tvořen IP jádry (*IP cores*), které v tomto zapojení umožňují ovládat LED diody dostupné pouze prostřednictvím programovatelné logiky. Tato konfigurace slouží jako zařízení, pro které je tvořen testovací ovladač generovaný pomocí tvořeného generátoru.

Procesor

Aby bylo možné komunikovat se zařízením dostupným pouze prostředníctvím programovatelné logiky 3.2, je nutné použít také procesor, jež zde má označení `processing_system7_0` jež tvoří další vazby potřebné pro svůj běh ale i komunikaci s programovatelnou logikou. Součástí připojených komponent je i připojení k operační paměti (`DDR`) a všem důležitým komponentám (`FIXED_IO`).

Reset procesoru

Jednou z těchto nezbytných vazeb je i možnost resetování procesoru, jež můžeme vidět pod názvem `rst_processing_system7_0_102M`. Je to jeden z nezbytných modulů, které automaticky doplní návrhové prostředí Xilinx Vivado při automatickém vytvoření propojení.

Sběrnice AXI

Dalším takovým modulem je modul sběrnice AXI (*Advanced eXtensible interface*), která slouží k propojení periferií. Tato sběrnice existuje ve více variantách, které jsou vhodné pro různé periferie a různé přenosy [7]. V příkladu byla použita verze AXI4-Lite, která je určena pro jednoduchou, nízkopropustnou paměťově mapovanou komunikaci.

Uživatelský blok

Uživatelský blok, pro který navrhujeme ovladač, umožňuje připojení diod GPIO blok využívající sběrnici AXI (`axi_gpio_0`), který slouží jako vstupně-výstupní rozhraní obecného

určení. K tomuto bloku jsou připojeny samotné diody (`leds_8bits`).

Tento blok je možné nahradit uživatelským blokem, který bude poskytovat požadovanou funkčnost. Může to být například blok implementující zpracování obrazu nebo blok implementující složitou matematickou operaci. Můžeme si ale vytvořit i vlastní blok a ten následně použít. Lze tak například schovat procesor a jeho rozhraní ukrýt do samostatného bloku a pak jej používat namísto několika samostatných bloků, což může u složitějších konfigurací pomoci ke zvýšení přehlednosti výsledného schématu.

4.3.2 Zdrojové soubory jádra

Další část tvorby ovladače zařízení je kompilace zdrojového textu ovladače do binární podoby. K tomu je nutné stáhnout si zdrojové kódy jádra použitého operačního systému ze stránky výrobce FPGA nebo třeba přímo ze stránek Linuxového jádra. Po stažení je nutné zdrojové kódy přeložit, což obnáší konfiguraci a následné spuštění jaderného sestavovacího skriptu [2]. Takto připravené jádro bude následně využito pro překlad modulu, jež vygeneruje výsledný modul ve formátu `*.ko`. Takto přeložený modul je následně možné vložit na paměťovou kartu a spustit systém.

4.3.3 Automatický překlad

Generátor též umožňuje generovat `Makefile` pro automatický překlad. Je tedy vhodné mít nainstalovaný i program `make`. Překladový systém jádra též podporuje možnost meziplatformního překladu (*cross-compile*). V případě, že je překlad prováděn na jiné než cílové platformně, potřebuje sestavovací skript znát cestu k překladači pro cílovou platformu. Je tedy nutné, aby systém znal cestu k těmto programům nebo je nutné uvést cestu při spuštění parametr `--cross_compile` s cestou k potřebné sadě nástrojů (*toolchain*).

4.4 Různé způsoby využití generátoru

Pokud máme připravenou konfiguraci, můžeme se pustit do tvorby definičního souboru. V něm je nutné uvést všechny potřebné položky a jakoukoli funkcionalitu, jež požadujeme, zde specifikovat. Po vytvoření pak můžeme navrhnutý generátor spustit v jednom z podporovaných režimů.

4.4.1 Generování ovladače připraveným pro překlad

V prvním režimu je nutné spustit skript s parametry, jež jsou potřebné jak pro generování ovladače tak i samotného Makefile. Je tak nutné použít parametry `--input`, `-m`, `--kernel` a optimálně i `--cross_compile`. Výsledkem tak bude vygenerovaný zdrojový kód ovladače a Makefile, jež při zadání správných hodnot bude možné přeložit pomocí příkazu `make`. Pokud chceme výsledek uložit do určitého adresáře, je nutné specifikovat i parametr `--output`. Tento režim je vhodné použít v situaci, kdy máme připravené prostředí a veškerou funkcionalitu je možné vytvořit pouze pomocí generátoru.

Výsledný příkaz tak může vypadat například takto: `python -O driver_generator.py --input=/cesta/k/definicnimu_souboru --output=/cesta/pro/ulozeni/ -m --kernel=/cesta/k/adresari/jadra/ --cross_compile=/cesta/k/sade/nastroju`

4.4.2 Generování ovladače předpřipraveným pro překlad

Tento režim je prakticky stejný jako předchozí režim. Liší se v tom, že se neuvádí parametr `--kernel` a před překladem je tedy nutné do vygenerovaného `Makefile` doplnit cestu k adresáři s jádrem do připravené proměnné ručně. To je vhodné, pokud budeme chtít překládat modul pro více verzí či konfigurací jádra. Samotný soubor je jinak plně připraven a po doplnění je ihned možné provést překlad stejně jako v předchozím režimu.

Výsledný příkaz tak může vypadat například takto: `python -O driver_generator.py --input=/cesta/k/definicnimu_souboru --output=/cesta/pro/ulozeni/ -m`

4.4.3 Generování samotného ovladače

Pokud však nechceme využít vygenerovaného `Makefile` například z důvodu tvorby komplexního ovladače využívajícího skládání modulů, lze generátor použít ke generování jednotlivých vrstev, jež následně jsou přeloženy v rámci vlastního `Makefile` nebo jiným software. Výstupní adresář je opět možné specifikovat pomocí přepínače `--output`.

Výsledný příkaz tak může vypadat například takto: `python -O driver_generator.py --input=/cesta/k/definicnimu_souboru --output=/cesta/pro/ulozeni/`.

4.5 Použití a správa jaderného modulu

Překlad pomocí jaderného překladového skriptu nám vygeneruje několik souborů. Jedním z nich je i soubor s příponou `.ko`. Tento soubor je výsledný modul, který poté můžeme například nakopírovat na paměťovou kartu. Po nakopírování modulu na paměťovou kartu včetně všech potřebých souborů pro spuštění systému můžeme systém spustit.

Po naběhnutí systému připojíme paměťovou kartu do souborového systému příkazem `mount /dev/sd_card_device /mnt`, kde `sd_card_device` je zařízení čtečky paměťové karty. Následně můžeme zavést náš modul do systému například pomocí následujícího příkazu `insmod /mnt/driver.ko`, kde `driver.ko` je jméno našeho ovladače. Ovladač je tak zaregistrován do systému a ihned může začít pracovat. Pokud chceme ovladač ze systému odregistrovat, použijeme příkaz `rmmod driver.ko`.

V případě, že modul přesuneme do adresáře `/lib/module/‘uname -r‘/` a vygenerujeme závislosti modulu příkazem `depmod -a`, můžeme pak používat příkaz `modprobe driver`, resp. `modprobe -r driver`, jež spolu s modulem přidá, resp. odebere modul včetně jeho závislostí.

Pokud se modul nepodaří zaregistrovat, je to pravděpodobně způsobeno chybnou verzí stažených zdrojových souborů nebo chybnou konfigurací jádra před jeho překladem. Taková chyba bývá často způsobená:

- špatnou verzí systému,
- neodpovídající příponou verze systému.

To lze řešit buď:

- použitím správné verze,
- úpravou konfiguračního souboru pro překlad jádra,
- použití konfigurace, jež nevyžaduje přísnou kontrolu závislostí, tedy i například verzi systému [2].

[2] Lze použít pouze pro verze, jež mají shodné rozhraní.

V případě chyby, jež nelze vyřešit žádným z uvedených souborů, je nutné zjistit další informace, abychom mohli daný problém řešit. Zdrojem takových informací může být například samotný výstup z `insmod`, případně z `modprobe`. Dále je možné se pokusit získat informace například ze systémového logu `dmesg`.

4.6 Ladění modulu

Pro ladění je možné využít běžné techniky pro ladění jádra a jaderných ovladačů. Lze tedy využít například kontrolních výpisů (doporučeno používat `printk` s úrovní `KERN_DEBUG`), jež budou součástí funkčního kódu v definičním souboru.

Další variantou je použít jádro přeložené s podporou ladění a využít další techniky jako je použití ladícího software (*debugger*) [4].

4.7 Použití pro další vestavěné systémy

Ačkoli je systém navrhnut s ohledem na použité referenční zařízení, není vázán na určitou platformu a lze jej tak použít i pro jiné vestavěné systémy. Je to dáno tím, že ovladač generuje zejména samotnou kostru ovladače včetně všech potřebných úkonů. Samotnou funkcionalitu je ale nutné dodat v podobě definičního souboru.

4.8 Výhody, nevýhody a omezení

Jak je uvedenov sekci 4.7, tak je nutné dodat funkcionalitu v definičním souboru psanou v jazyce C. Z toho plyne výhoda i nevýhoda zároveň a to, že je nutné znát alespoň pár příkazů pro čtení a zápis do registrů (`readl` a `writel`) a samozřejmě i další funkce, které chceme použít. Tuto nevýhodu ale kompenzuje ta výhoda, že programátor není omezen na určitou platformu či jen úzce vymezený typ zařízení. Generátor tak slouží jako rozhraní, jehož prostřednictvím lze vytvořit jak jednoduchý ovladač (jako je řízení LED diod), tak i velmi složité FPGA konfigurace.

Jistá omezení se však, vzhledem k rozsáhlosti rozhraní jádra OS Linux, mohou objevit a takový kód je nutné do vygenerovaného zdrojového souboru doplnit ručně. Příkladem může být užití některých systémových maker.

Další výhodou je, že se tak programátor může soustředit na implementaci funkcionality a nikoli samotného ovladače, což uživateli ušetří čas. Tím, že je podporován také blok `FUNCTION`, je možné rozložit funkcionalitu do více funkcí a vytvářet tak přehledný kód, nikoli jen rozsáhlé a nepřehledné části kódu v místech, kde to generátor uživateli umožňuje.

4.9 Testovací ovladač

Pro testování generátoru byl vytvořen ovladač, jež provádí v jednotlivých částech definovanou činnost, ať se již jedná o registraci či odregistraci modulu.

Zařízení tak při zaregistrování do systému postupně rozsvěcuje jednotlivé LED diody, čímž potvrzuje zaregistrování a správnou funkčnost ovladače. Při odregistrování ovladače ze systému pak zařízení několikrát zabliká všemi diodami.

Při testování interakce s uživatelským prostorem prostřednictvím virtuálního souborového systému `SYSFS` pak bylo testováno, zda se záznam skutečně vytvoří a pokud ano, zda

je funkční. Pro testování funkcionality tak byl vytvořen soubor s oprávněním zápisu, jež po zápisu jakékoli hodnoty rozsvítí na referenčním zařízení všechny uživatelské LED diody.

4.9.1 Definiční soubor

Jedním z testovacích případů byl například definován v definičním souboru takto:

```
MODULE_DESC:
  author: "Jaroslav Kopacek"
  license: "GPL v2"
  description: "LED blinkig module"
MODULE_DEF:
    module_name: "my_led"
    driver_name: "LED_switcher"
    driver_type: "platform"
    compatible: <"xlnx,xps-gpio-1.00.a">
    HEADER:
      header: "linux/delay.h"
    FEATURE:
        exec: "init"
        description: "Switching active leds"
        VARIABLES:
            int:dir
            int:i
            int:value
        CODE:
            i = 16;
            dir = 0;
            value = 1;
            while (i) {
                if (value == 128 || value == 1) {
                    if (dir)
                        dir = 0;
                    else
                        dir = 1;
                }
                if (dir)
                    value = value << 1;
                else
                    value = value >> 1;
                writel (value, instance1->dev_virtaddr);
                msleep_interruptible (1000);
                i--;
            }
            return 0;
    SYSFS:
      name:"test"
      mode: "S_IWUSR"
      STORE:
```

```
        writel (0xFF, instance1->dev_virtaddr);
        return 0;
```

Můžeme zde najít blok popisující samotný modul jako je jméno autora, licence a popis modulu. Dále definici samotné funkcionality. Zde lze nalézt jméno samotného modulu i ovladače, která následně slouží pro identifikaci například ve virtuálním souborovým systému `SYSFS`. Dále je zde specifikován řetězec `compatible` pro nalezení hardware. Ten se musí shodovat s řetězcem uvedeným v DT. Poté následuje specifikace využitých hlavičkových souborů a samotné funkcionality.

Uvedený kód provádí rozsvícení jedné diody a po jedné vteřině přepíná na další, což provede šestnáctkrát a poté zůstane poslední rozsvícená dioda svítit. V parametru `exec` je uvedena značka `init`, která značí, že se daný kód má vykonat při zavádění ovladače. Je však možné použít i značku pro jiná umístění jako například `exit` pro umístění ve funkci pro odregistrování ovladače (`remove`) ze systému. Tyto značky jsou vytvořené v rámci definičního souboru, kde jsou všechny možnosti popsány a nejsou tedy definovány v rámci jádra systému.

V kódu, jež vkládá uživatel můžeme mimo jiné vidět i proměnnou `instance1`, kterou generuje samotný generátor. Tato proměnná reprezentuje strukturu `driver_items` zapouzdřující některé proměnné jako například strukturu `struct cdev`, jež reprezentuje znaková zařízení v jádru systému, nebo počáteční adresu a velikost přidělené paměti, která byla získána při registraci ovladače. Podrobnější informace lze nalézt v definičním souboru, kde je tato struktura popsána.

V bloku `SYSFS` je uvedeno jméno souboru, který se vytvoří ve virtuálním souborovým systému `SYSFS` po zaregistrování ovladače do systému. Tento soubor má nastavené oprávnění pouze pro zápis (`S_IWUSR`), kde toto oprávnění je definováno v rámci jádra systému a je tak přímo vkládáno do vygenerovaného kódu. Po zápisu jakékoli hodnoty provede rozsvícení všech diod současně. Zápis do takového souboru můžeme provést například pomocí tohoto příkazu
`echo 1>/sys/module/my_led/drivers/platform LED_switcher/41200000.gpio/test`.

4.9.2 Vygenerovaný ovladač

Vygenerovaný ovladač pak je vygenerován do souboru `my_led.c`, jež je dán názvem modulu. V následujících úryvcích výsledného kódu můžeme spatřit části definované v definičním souboru.

Část popisující modul generátor využívá mimo jiné i pro tvorbu komentářů o souboru. Následující úryvek zobrazuje vygenerovanou hlavičku souboru:

```
/**
 * @file my_led.c
 *
 * LED blinkig module
 *
 * @author Jaroslav Kopacek
 * @created 2016-05-12
 *
 *
 *
 * Generated by Linux driver generator developed as bachalor's thesis
```

```
 * project.
 */

        ⋮
```

Z uvedeného souboru můžeme vidět, že bylo použito jméno autora a popis modulu z bloku `MODULE_INFO`. Dále generátor použil název modulu z bloku `MODULE_DEF`. Generátor navíc přidává datum, kdy byl ovladač vygenerován a poznámka, že byl použit.

Následný blok ukazuje vygenerované hlavičkové soubory. Lze zde nalézt soubory, které přidává generátor na základě tvořené kostry, ale také hlavičkové soubory, které byly zadány v definičním souboru.

```
            ⋮

#include <linux/cdev.h>
#include <linux/platform_device.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <asm/io.h>
#include <linux/ioport.h>
#include <linux/fs.h>
#include <linux/of_address.h>
#include <linux/kernel.h>
#include <linux/module.h>

            ⋮
```

Generátor generuje také definující direktivy. Některé používá sám, například název modulu či ovladače. Jiné jsou definovány v definičním souboru.

```
            ⋮

#define MODULE_NAME "my_led"
#define DRIVER_NAME "LED_switcher"

            ⋮
```

Zde lze vidět definované názvy, jež jsou následně využívány v generovaných funkcích, zejména tedy ve funkci pro registraci ovladače (`probe`).

V dalších částech kódu můžeme nalézt například funkci pro zápis a struktury, jež vytvářejí soubory ve virtuálním souborovém systému SYSFS:

```
            ⋮


static ssize_t sys_test_store (struct device *dev, struct
device_attribute *attr, const char *buf, size_t count) {
writel (0xFF, instance1->dev_virtaddr);
return 0;
}

DEVICE_ATTR (test, S_IWUSR, NULL, sys_test_store);

static const struct attribute *reg_attrs[] = {
```

```
&dev_attr_test.attr,
NULL,
};
```

⋮

Tyto struktury jsou následně zaregistrovány ve funkci pro registraci ovladače.

Následný obsah souboru obsahuje běžné struktury a funkce, jež jsou v ovladačích využívány a které též mohou obsahovat definovanou funkcionalitu namísto výchozí funkcionality.

⋮

```
int driver_probe(struct platform_device * pdev){
```

⋮

```
// Add device to kernel
if (cdev_add (&instance1->cdev, instance1->dev_num, 1) < 0){
 dev_err (&pdev->dev, "Unable add chrdev");
 goto fail;
}

// USER CODE
i = 16;
dir = 0;
value = 1;
while (i) {
if (value == 128 || value == 1) {
if (dir)
dir = 0;
else
dir = 1;
}
if (dir)
value = value << 1;
else
value = value >> 1;
writel (value, instance1->dev_virtaddr);
msleep_interruptible (1000);
i--;
}
return 0;

fail:
driver_remove (pdev);
return -ENODEV;
}
```

⋮

```
MODULE_LICENSE ("GPL v2");
MODULE_AUTHOR ("Jaroslav Kopacek");
```

⋮

V posledním úryvku lze nalézt definovanou funkcionalitu uvnitř funkce registrující ovladač. Tato funkcionalita je vždy vykonána až po řádném zaregistrování ovladače do systému. Na konci lze též i systémová makra pro licenci a jméno autora, které jsou již umístěna mimo funkci a obsahují parametry z bloku `MODULE_INFO`.

4.9.3 Výsledky testování

Testovací ovladač je možné přeložit pomocí vygenerovaného `Makefile`. Vytvořený ovladač lze po nakopírování na paměťovou kartu po spuštění systému spustit. Výsledný ovladač je poté zaregistrován do systému příkazem `insmod` a na referenčním zařízení ovladač funguje bezvadně. Poté lze ovladač ze systému úspěšně odregistrovat pomocí příkazu `rmmod`.

Kapitola 5

Závěr

Vestavěné systémy mnohdy představují specializovaná výpočetní zařízení zaměřená na konkrétní úlohu, které jsou přizpůsobeny jak po výkonostní stránce, tak i po softwarové. Pokročilé vestavěné systémy je možné navrhnout na bázi architektury Soc, kdy jsou části systému umístěny na jedné komponentě. Součástí vestavěného systému může být i FPGA, jež nám umožní optimalizovat systém pro řešení daného problému, aniž bychom při změně úlohy museli vyměnit celý systém za jiný.

Cílem této práce bylo analyzovat požadavky a možná řešení návrhu a implementace ovladačů pro vestavěné systémy založené na operačním systému Linux a možnosti automatizace vývoje. Dalším cílem bylo vytvořit generátor ovladačů založených na operačním systému Linux. Tvorba generátoru obnášela také analýzu již existujících řešení. Samotný generátor je modulární a otevřeným zdrojovým kódem. Lze jej, stejně jako jeho dílčí části použit i v jiných projektech či jej pro jiné projekty přizpůsobit. Možným využitím tak může být rozšíření o neplatformní ovladače či o bloková, případně síťová zařízení.

Výsledný ovladač může být problém zaregistrovat do systému, avšak s velkou pravděpodobností se nebude jednat o chybu vygenerovaného ovladače, ale o chybu způsobenou samotným překladem. Jak již bylo nastíněno, při vhodné konfiguraci je možné použít ovladač pro jiné verze, avšak je nutné, aby byla použitá část rozhraní nezměněná.

Testování probíhalo vytvořením definičního souboru ovladače, jež řídil svit různých LED na základě aktuální události.

Generátor má modulární strukturu a lze jej tak snadno rozšiřovat. Jednou z možností, jak generátor rozšířit, je podpora pro bloková a síťová zařízení, kde lze aplikovat i mnohá rozšíření, jako například nárazový režim (*burst mode*).

Jinou možností je rozžíření o generování neplatformního typu ovladačů. Poté by bylo možné generovat i ovladače například pro dynamicky připojitelná zařízení.

Jistou cestou také může být optimalizace a rozšíření automatizace generátoru. Bylo by tak možné provádět kontrolu za běhu a nikoli až při překladu pomocí překladače. Případně by bylo možné i uživateli napovídat například to, jaký datový typ dané funkce akceptuje nebo identifikátory dle shody právě psaného textu. To by bylo využitelné například při vytvoření grafického rozhraní například pomocí frameworku `PyGtk` nebo `PyQt`.

Literatura

[1] Daniel P. Bovet, M. C.: *Understanding the Linux Kernel*. O'Reilly Media, Inc., třetí vydání, únor 2005, ISBN 0-596-00565-2.

[2] Jonathan Corbet, A. R.; Kroah-Hartman, G.: *Linux Device Drivers*. O'Reilly Media, Inc., třetí vydání, únor 2005.

[3] Kamal, R.: *Embedded Systems: Architecture, Programming and Design*. McGraw-Hill Education, březen 2009, ISBN 0070151253, 978-0070151253.

[4] Love, R.: *Linux Kernel Development*. Addison-Wesley, třetí vydání, 2009, ISBN 0-67-232946-8, 978-0-07015125-3.

[5] Venkateswaran, S.: *Essential Linux Device Drivers*. Prentice Hall, březen 2008, ISBN 0-13-239655-6, 978-0-13-239655-4.

[6] Viktorin, J.: *HW/SW Co-design for the Xilinx Zynq Platform*. diplomová práce, FIT VUT v Brně, Brno, 2013.

[7] Xilinx: *AXI Reference Guide, UG761, v 13.1*. březen 2011.

[8] Xilinx: *Zedboard hardware user's guide, v 1.1*. srpen 2012.

[9] Xilinx: *Zynq-7000 All Programmable SoC - Technical Reference Manual, UG585, v 1.10*. únor 2015.

[10] Xilinx, J. M.: *Simple AMP: Bare-Metal System Running on Both Cortex-A9 Processors, XAPP1079, v 1.0.1*. leden 2014.

Přílohy

Seznam příloh

Příloha A

Obsah CD

CD obsahuje:

- zdrojové soubory generátoru
- popisující definiční soubor
- ukázkový definiční soubor