TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Ovladače a řídící software pro rekonfigurovatelný embedded systém pod OS Linux

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 3902T005 Automatické řízení a inženýrská informatika

Autor: **Bc. Jiří Čech**
Vedoucí práce: **Ing. Martin Rozkovec, Ph.D.**
ITE FM TUL
Konzultant: **Ing. Roman Nádhera**
Applic s.r.o.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2014/2015

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří Čech**
Osobní číslo:	**M13000207**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Automatické řízení a inženýrská informatika**
Název tématu:	**Ovladače a řídící software pro rekonfigurovatelný embedded systém pod OS Linux**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s AP SoC Zynq, dodanými IP jádry a nízkoúrovňovým firmware, který ovládá stávající HW
2. Seznamte se s strukturou jádra OS Linux a způsoby tvorby a užívání modulů ovladačů periferií
3. Pro dodaná IP jádra - Accelerator, Sensor Capture, Camera Caputure, aj. - naprogramujte ovladače.
4. Vytvořte aplikaci (daemon), která bude integrovat služby ovladačů a bude umožňovat ovládání systému přes TCP/IP

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 40 až 50 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Crocket, H.L., Elliot, R.A., Enderwitz, M.A., Stewart, R.W. - The Zynq Book, Strathclyde Academic Media, 2014, ISBN: 9780992978709**

[2] **Matthwe, N., Stones, R.: Linux, Programujeme profesionálně, Computer Press, 2001, ISBN:80-7226-532-6**

Vedoucí diplomové práce:	**Ing. Martin Rozkovec, Ph.D.** Ústav informačních technologií a elektroniky
Konzultant diplomové práce:	**Ing. Roman Nádhera**
Datum zadání diplomové práce:	**12. září 2014**
Termín odevzdání diplomové práce:	**15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 - L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
vedoucí ústavu

V Liberci dne 12. září 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15. 5. 2015

Podpis:

Abstrakt

Cílem této práce je přetvořit stávající nízkoúrovňové řízení komplexního kamerového systému na modulární řízení pod OS Linux. Vytvořím ovladače ve formě modulů pro klíčové komponenty kamerového systému a zprovozním jejich ovládání přes protokol TCP/IP. Při tvorbě se pokusím zachovat stávající strukturu řízení a kompatibilitu s vyvíjenou aplikací na zpracování dat z kamerového systému.

Abstract

The goal of my work is to rebuild an existing firmware based control of complex camera system into modular-driver based system using OS Linux. I will create driver modules for key components of the system and utilize them in an application build around TCP/IP protocol. The application will be used to control the system. It is required to preserve compatibility with already developed PC application that is used for remote control of the camera system.

Klíčová slova

Zedboard, PetaLinux, ovladače, kamerový systém

Key words

Zedboard, PetaLinux, drivers, camera system

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

PS	Processing System, Procesorový systém
PL	Programable logic, Programovatelná logika
IP	Intellectual Property, duševní vlastnictví
SC	Sensor Capture, komponenta pro zisk dat ze senzoru
CC	Camera Capture, komponenta pro zisk dat z kamery
ACC	Accelerator, akcelerátor přenosu dat
MLV	Multi Layer Video, komponenta pro zobrazení dat
PWM	Pulse Width Modulator, pulzně šířkový modulátor
XPS	Xilinx Platform Studio, studio pro návrh desky
SDK	Xilinx Software Development Kit, studio pro programování desky
BSP	Board Support Package, vytvořené zapojení desky
OS	operační systém
SD	Secure Digital, typ paměťové karty
I^2C	Inter-Integrated Circuit, sériová sběrnice (polo duplexní)
SPI	Serial Peripheral Interface, sériová sběrnice (plně duplexní)
HDMI	High-Definition Multi-media Interface, přenos digitálního obrazu
VGA	Video Graphics Array, starší standard přenosu digitálního obrazu
ISE	Integrated Synthesis Environment, balíček programovacích nástrojů
USB	Universal Serial Bus, univerzální sériová sběrnice
RAM	Random-Access Memory, paměť s přímím přístupem
TCP/IP	Transmission Control Protocol/Internet Protocol, síťoví komunikační protokol
APSoC	All Programmable System-on-Chip, plně programovatelný systém
UART	Universal Synchronous / Asynchronous Receiver and Transmitter, synchronní/asynchronní sériové rozhraní
JTAG	Joint Test Action Group, IEEE 1149.1 programovací/testovací rozhraní
QSPI	Quad SPI, SPI se čtyřmi datovými vodiči
HDL	Hardware Description Language, programovací jazyk
RGBA32	Red Gren Blue Alpha 32bit, barevný model
QEMU	Quick EMUlator, simulátor desky
FPGA	Field-Programmable Gate Array, programovatelné hradlové pole
ASIC	Application Specific Integrated Circuit, zákaznický integrovaný obvod
FAT32	File Allocation Table, souborový systém
EXT4	Fourth Extended filesystem, souborový systém
IOCTL	Input/Output Control, systémové volání
SCCB	Serial Camera Control Bus, sběrnice podobná I^2C
CLK	Clock, hodiny
MOSI	Master Output Slave Input, výstupní vodič Master
MISO	Master Input Slave Output, vstupní vodič Master
SS	Slave Select, výběr Slave zařízení

Úvod

Vývojová deska Zedboard využívá programovatelné hradlové pole a procesor. Díky tomu je možné vytvářet vlastní zařízení v hradlové logice a ovládat je přes procesor. Typy zařízení mohou být různé, od jednoduchých multiplexorů a automatů, přes sčítačky a jiné akcelerátory, až po procesory a neuronové sítě. To vše umožňuje převedení zbytečně zdlouhavých prací procesoru na hardwarovou úroveň.

Pro využití komplexních a stabilních systémů již nestačí klasické firmwarové ovládání a přechází se pod některý operační systém. Ten zajišťuje funkci a správu jednotlivých zařízení, která by byla jinak těžko řešitelná. V této práci se zaměřím na využití operačního systému Linux. Konkrétně distribuce PetaLinux, pro kterou má firma Xilinx, inc. podporu svých produktů.

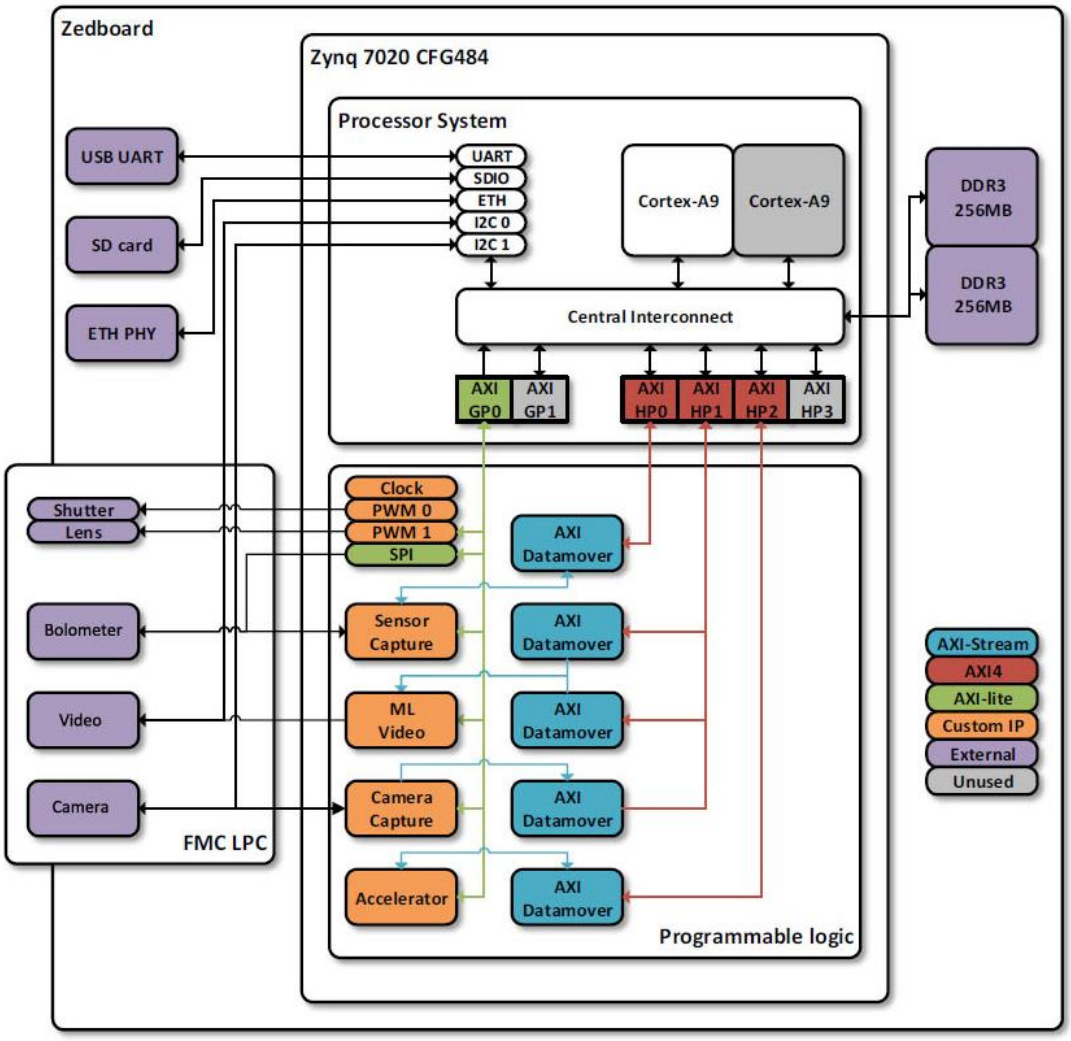
Do operačního systému integruji funkcionalitu firmware pro prototyp kamerového systému. Tento prototyp je součástí projektu firmy APPLIC s.r.o., na kterém se podílí i Technická univerzita v Liberci. V současné době jsou již funkční nové verze kamerového systému, které se intenzivně testují. V budoucnu se plánuje rozšíření stávajícího systému o další zařízení a implementaci vyhodnocovacích algoritmů.

1 Hardwarové a softwarové prostředky

V této práci se jedná o řízení prototypu kamerového systému vyvíjeného firmou Applic [1], který využívá vývojové desky ZedBoard. Používám zde programový balíček „Xilinx ISE Design suite“, pro vytvoření návrhu zapojení desky a tvorbu řídícího programu. Všechny zdrojové soubory a vytvořené moduly je potřeba přeložit z jazyka C++ pomocí cross-compilátoru pro ARM. To je vhodné provádět na počítači s operačním systémem Linux, ale pod systémem Windows je to možné také.

1.1 Prototyp kamerového systému

Jedná se o systém pracující se dvěma obrazovými čipy, z nichž jeden je pro viditelné a druhý pro infračervené spektrální pásmo. Vše je propojeno a ovládáno pomocí komunitní desky ZedBoard, která je založena na platformě Xilinx Zynq®-7000 All Programmable System on Chip (APSoC) od firmy Avnet. Na Obr. 1 je schéma zapojení celého systému.



Obr. 1 Schéma kamerového systému

Fialově jsou vyznačeny externí komponenty, které je možné ze systému ovládat a komunikovat přes ně s okolím. Jedná se o komunikační rozhraní gigabitového ethernetu a převodníku UART na USB. Dále jsou dostupné dvě paměti DDR3 o celkové kapacitě 512 MB a slot na SD kartu. Pro zobrazení obrazu je k dispozici HDMI řadič a VGA výstup, ty jsou sdružené v komponentě „Video“. Pro zisk obrazu v infračerveném pásmu slouží bolometrický čip „Bolometer“ a pro viditelné pásmo je k dispozici VGA kamera „Camera“ od firmy OmniVision [2]. Ovládání objektivu „Len“ a záklopky „Shutter“ je prováděno dvěma krokovými motůrky řízenými PWM signálem.

Oranžově jsou vyznačená vytvořená uživatelská „Intellectual Property“ (IP) jádra, která plní nebo zpracovávají data v paměti nezávisle na procesoru, avšak jsou z něj plně ovladatelná. IP jádra jsou obecná zařízení složená z logických bloků a realizovatelná v programovatelných polích FPGA nebo obvodech ASIC. Podle implementace se dělí do tří kategorií. Mohou mít buď pevně dané rozložení všech jejich součástí „hard core“, nebo jen jejich část „firm core“. Zde je použit třetí typ „soft core“, který má pouze popsanou svou strukturu pomocí některého HDL jazyka a jeho fyzické zapojení je pak řešeno generátorem při tvorbě zapojení pro cílovou desku.

Mezi uživatelská „Custom“ IP jádra patří: „Sensor Capture“ (SC) obsluhující VGA kameru, „Camera Capture“ (CC) obsluhující bolometrický čip, „Multi Layer Video“ (MLV) obsluhující komponenty „Video“, dvě jádra „PWM“ pro obsluhu krokových motůrků a „Accelerator“ (ACC) umožňující operace nad obrazem jako jsou konvoluce či obarvování. Doplňujícím IP jádrem je „Clock“, které generuje hodiny rozdílných frekvencí pro jednotlivé komponenty.

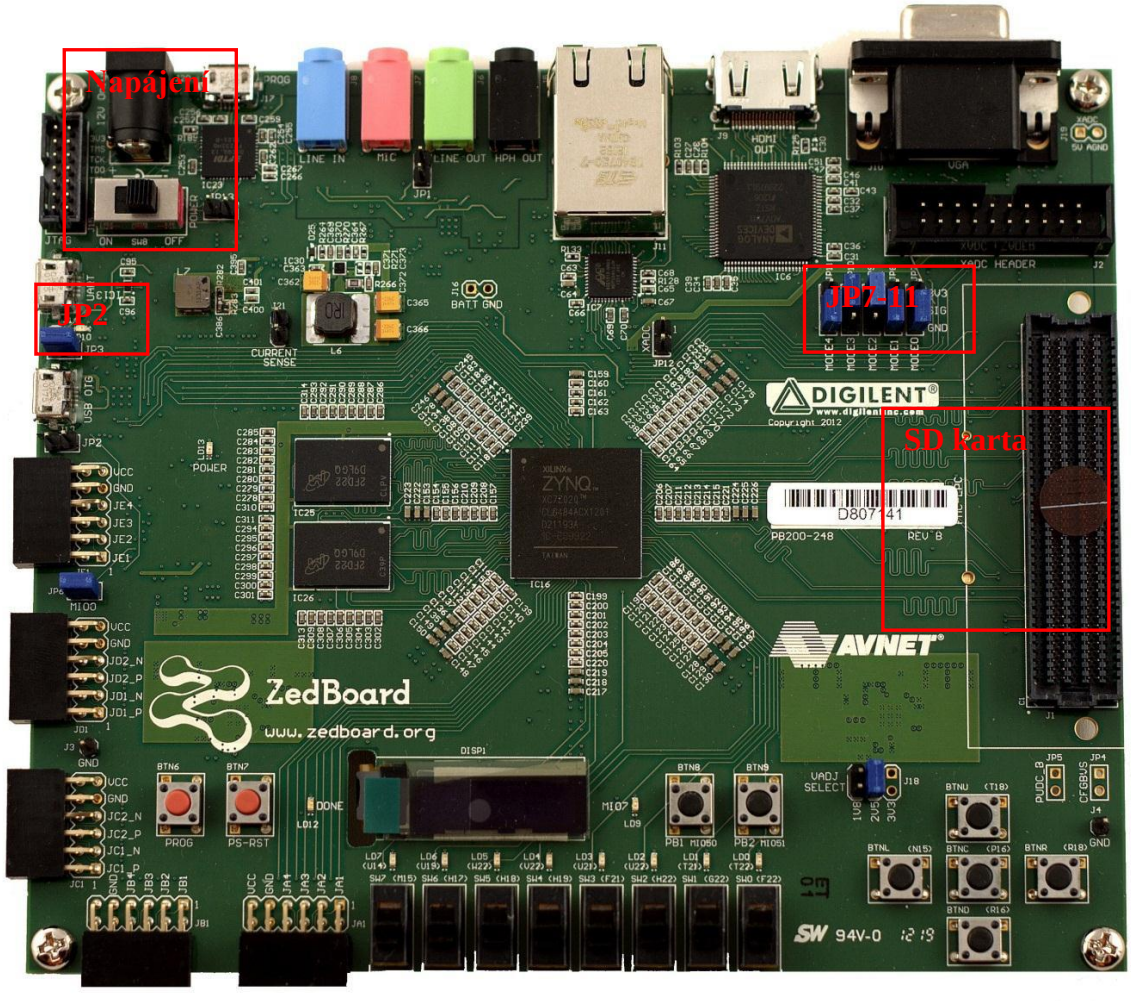
Čip „Zynq 7020“ obstarává propojení dvou-jádrového procesoru „ARMv7 Cortex-A9“ s frekvencí 666 MHz, FPGA pole a připojených systémových komponent. Skládá se ze dvou vzájemně propojených částí „Processing system“ (PS) a „Programmable logic“ (PL).

Část PS obsahuje systémové komponenty desky propojené pomocí „Central Interconnect“. Patří mezi ně krom dostupné RAM paměti i dvě I^2C rozhraní, dvě SPI rozhraní, Ethernet rozhraní, USB rozhraní, slot na SD kartu a jiné.

Část PL obsahuje FPGA modul, do kterého lze vkládat jednotlivá uživatelská IP jádra. V základu je návrh tvořen jednou AMBA sběrnicí typu AXI-Lite označenou jako „AXI

GP0“ na kterou se IP jádra připojují. Sběrnic jde vytvořit i více a lépe tak propojit IP jádra mezi sebou ale pouze přes „AXI GP0“ mají k dispozici přímí přístup do RAM paměti. Paleta dostupných IP jader je široká a za mnohé se platí nemalé peníze. V návrhu je z této palety použito IP jádro „SPI“, které má odlišné vlastnosti od „SPI“ v PS.

Deska může být programována přes: JTAG rozhraní, převodníkem JTAG na USB, nebo automaticky při zapnutí souborem *boot.bin*. Ten je možné uložit do QSPI Flash paměti nebo na SD kartu. Propojky JP7-11 určují, jaký režim programování desky bude použit. Já použiji programování přes SD kartu a režim USB Host. To znamená nastavit propojky do následujících poloh. Piny JP9, JP10 přivedeny na plus (3,3 V), piny JP7, JP8, JP11 přivedeny na zem a rozpojit pin JP2. Rozmístění pinů je vyznačené na Obr. 2.



Obr. 2 Vývojová deska Zedboard [3]

1.2 Vytvořená uživatelská IP jádra

Pro zisk dat z bolometrického čipu slouží IP jádro SC. Maximální dosažitelný počet přenesených snímků je 70 fps o rozlišení 640×480 pixelů. Hodnota pixelu je 14bit slovo

a odpovídá naměřenému napětí na jednom mikrobolometru, z nichž je celý čip složen. Získaná data jsou zašuměná a to tak, že nejnižší dva bity jsou nepoužitelné. Mikrobolometr je citlivý na intenzitu záření o dané vlnové délce a výstupní napětí je mezi saturacemi lineární. Protože není možné zajistit, aby všechny mikrobolometry byly stejné, je nutné provádět korekci ofsetu a zesílení pro každý získaný pixel.

Pro zisk dat z VGA kamery slouží IP jádro CC. Maximální dosažitelný počet přenesených snímků je 30 fps o rozlišení 1280×1024 pixelů. Použité je ale rozlišení 640×480 pixelů. Formát dat obrazu je RGBA32, což znamená čtyři 8bit složky: červené, zelené, modré barvy a hodnoty průhlednosti. Po zapnutí je nutné kameru nejdříve nastavit na požadovaný režim přes I^2C rozhraní pomocí dlouhé série příkazů.

Pro zobrazení obrazu na HDMI a VGA výstup slouží IP jádro MLV. Umí zobrazit až dva různé obrazy, měnit jejich polohu a překrytí. Používám režimy rozlišení obrazu 640×480 pixelů až 1280×480 pixelů s obnovovací frekvencí 60 Hz. Fyzické adresy obrazů ve formátu RGBA32 stačí uložit do dvou zdrojových registrů a IP jádro se postará o jeho zobrazení. HDMI řadiči je nutné nastavit hodnoty řídících registrů přes I^2C rozhraní.

IP jádro ACC umožňuje kopírování dat mezi dvěma poli v systémové paměti a provést nad nimi nějakou operaci. Přenos dat je realizován pomocí sady modulů napojených na vnitřní sběrnici akcelerátoru, z nichž každý provede nad daty určitou operaci. Zatím je implementován modul černobílého obarvení dat, modul vícebarevné palety na obarvení dat a modul prostého kopírování dat.

IP jádra „PWM“ generují PWM signál pro krokové motůrky záklopky a objektivu. Označení IP jádra ovládající motůrek záklopky je „Shutter“ (SHU) a pro objektiv „len“ (LEN). Mají nastavitelnou střídou signálu po 256 krocích a měnitelnou polaritou. Jejich frekvence je nastavitelná 32bit hodnotou předděliče. Protože záklopka má pouze dvě polohy „otevřeno“ a „zavřeno“ má smysl měnit střídu jen mezi dvěma hodnotami.

IP jádro „Clock“ generuje ze vstupní hodinové frekvence 100 MHz pět výstupních frekvencí: 100 MHz, 62,5 MHz pro SC, 24 MHz pro CC, 25 a 50 MHz pro MLV.

1.3 Xilinx ISE Design Suite

Jedná se o rozsáhlé vývojové prostředí pro produkty firmy Xilinx inc. [4]. Jde o několik programů, z nichž každý je určen pro tvorbu jiné části celkového zapojení a naprogramování desky. Já používám studia z balíčku „Embedded Development Kit“, a to

konkrétně „Xilinx Platform Studio“ (XPS) a „Software Development Kit“ (SDK). Studio XPS slouží k tvorbě propojení mezi jednotlivými IP jádry a ke konfiguraci jednotlivých komponent.

Studio SDK je určeno pro vývoj a ladění programů. Obsahuje nejen knihovny pro vývoj firmwarových aplikací, ale také pro vývoj Linuxových „standalone“ programů. Obsahuje i nástroj pro generování bootovacích souborů „Create Zynq Boot Image“. Tím lze vytvořit soubor „boot.bin“, který spustí zvolený program po startu desky.

V současné době je vývojový balíček ISE nahrazován modernějším studiem Vivado. To má tu výhodu, že obsahuje všechny nástroje v jednom. Jeho hromadné využití je omezováno licenční politikou firmy. Ke koupené desce ZedBoard získáte doživotní licenci balíčku ISE, ale jen roční licenci Vivado studia.

1.4 Vývoj v Linuxu

Já používám pro vývoj ovladačů operační systém CentOs 7 [5] ve virtuálním prostředí pomocí Oracle VM Virtual box [6]. Doporučovaný je i operační systém Ubuntu [7]. Pro jiné distribuce není takový výběr manuálů a mohou nastat problémy s kompatibilitou u některých použitých programů. Zdrojový kód ovladače píšu v textovém editoru „gedit“.

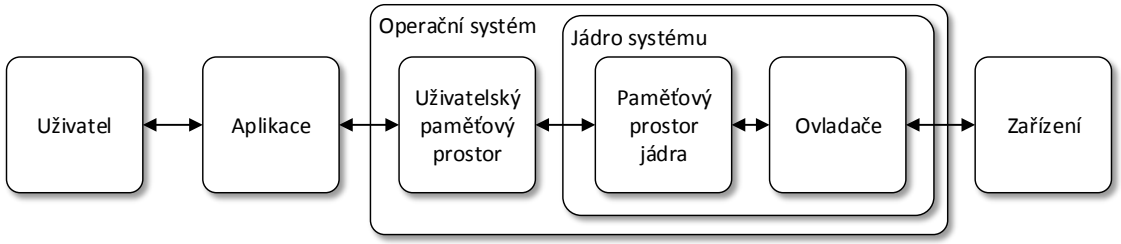
Pro zprovoznění překladu kódu pro ARM procesor je nutné nainstalovat kompilátor „xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin“ [3], který přeloží kódy vytvořené v jazyce C++ pro danou architekturu procesorů. Po nainstalování je vhodné si vytvořit či upravit soubor „.bash_profile“ a přiřadit význam symbolu „CROSS_COMPILE“ podle použitého kompilátoru. Poté stačí v konzoli využívat funkce „source“ místo opětovného psaní zdlouhavých výrazů.

Doporučená posloupnost příkazů pro nainstalování a nastavení cross-compilátoru:

```
yum install make gcc kernel-devel perl
chmod ugo+x xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin
./xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin
gedit .bash_profile
export CROSS_COMPILE=arm-xilinx-linux-gnueabi-
source .bash_profile
```

2 Operační systém Linux

Operační systém je program, který umožňuje uživateli ovládat počítač. Jeho součástí je jádro a pomocné softwarové nástroje. Jádro umožňuje spouštění více aplikací a přerozděluje jim systémové prostředky. Softwarové nástroje implementují firmware jednotlivých komponent a umožňují tak komunikaci s nimi či zpřístupňují některé jejich funkce. Jádro má oddělenou svoji paměť od paměti přístupné uživatelským aplikacím. Dělíme je podle úrovně funkcionality řešené v paměti jádra na: „mikrojádra“ malá jádra s pouze nejnutnější funkcionalitou, „hybridní jádra“ implementující i funkce náročné na rychlost, „monolitická jádra“ řešící vše (správa paměti, ovladače, protokoly, …) v paměti jádra. Hierarchie přístupu uživatele k zařízení v OS s monolitickým jádrem je na Obr. 3.

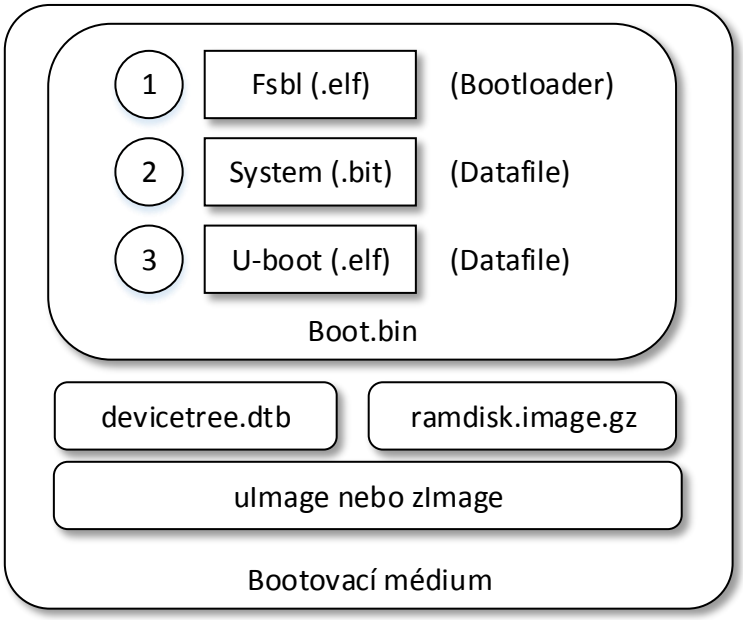


Obr. 3 Schéma Operačního systému s monolitickým jádrem

Operační systém Linux má čistě monolitické jádro a nové ovladače jdou do něj vkládat za běhu ve formě modulů. V minulosti byly operační systémy dominantou desktopových počítačů. Dnes se ale čím dál víc rozmáhají i v embedded zařízeních jakou jsou například telefony, síťové prvky, robotické systémy. Pro řízení výrobních procesů je důležité znát přesnou dobu zpoždění odezvy na vstupech a výstupech. Pro takovéto aplikace se používají buď programovatelné logické automaty (PLC) nebo operační systémy reálného času (RTOS), které mají s určitou přesností všechna zpoždění definovaná. Klasické operační systémy mají tuto dobu proměnnou v závislosti na aktuálním vytížení.

2.1 Embedded OS Linux

Embedded, či vestavěný OS Linux se využívá jako řídící software pro systémy, které nekladou důraz na přesně definovanou dobu zpoždění odezvy vstupů a výstupů. Jedná se většinou o jednoúčelová zařízení s potřebou větší kompaktnosti a komunikace s ostatními počítači. Ve většině aplikací postačí pro řízení firmware, nicméně jeho vývoj a údržba pro rozsáhlejší systémy se snadno stane velmi náročnou záležitostí. Systém bootuje z paměťového média, jehož struktura je zobrazena na Obr. 4.



Obr. 4 Struktura bootovacího média

Operační systém implementuje do svého jádra ovladače pro jednotlivé systémové komponenty. Ohlídá si jejich funkčnost či nefunkčnost a zpřístupní je uživateli. Embedded systémy k tomuto účelu používají soubor *devicetree.dtb*, který definuje základní parametry všech dostupných zařízení. Ten je při startu systému načten a k příslušným zařízením se spustí příslušný ovladač.

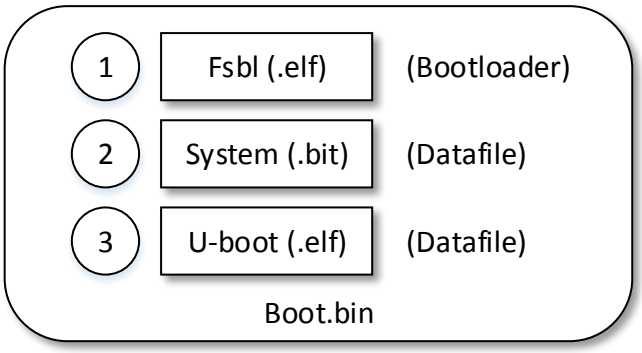
Pro připravení desky a spuštění operačního systému slouží soubor *boot.bin*, který rozbalí jádro Linuxu. To je uloženo v souboru *zImage* nebo *uImage* v závislosti na druhu použité komprese. Pro nastavení systému po spuštění a inicializaci programů a služeb slouží *ramdisk.image.gz.* Ten se po startu systému nahraje do RAM paměti, kde vytvoří strom složek „rootfs“ a poskytne operačnímu systému jednotlivé soubory.

K vytvoření všech potřebných souborů se využívají nástroje Petalinux tool, Buildroot a jiné. Je možné modifikovat zdrojové soubory a vytvořit vše nebo část ručně. K tomu je zapotřebí stáhnout zdrojové soubory jednotlivých vyvíjených projektů, které často najdeme na webu „github.com“.

2.1.1 Tvorba souboru *Boot.bin*

Tento soubor slouží pro připravení desky k zavedení operačního systému do paměti desky. Jeho hlavním úkolem je naprogramovat FPGA modul desky a připravit paměť pro rozbalení obrazu jádra operačního systému, popřípadě i načtení ramdisku. Pro jeho vytvoření slouží program „Xilinx boot build“ dostupný v menu „Tool“ v SDK studiu. Ten vytvoří adresový souboru s příponou „*bif*“, ve kterém je určeno v jakém pořadí které

soubory spustit a jaký mají typ. Tyto adresy souborů jsou požity při tvorbě souboru *boot.bin* jehož struktura je na Obr. 5. Pro jeho vytvoření jsou potřebné soubory: *fsbl.elf* jako „bootloader“, *system.bit* jako „datafile“ a *u-boot.elf* jako „datafile“.



Obr. 5 Struktura souboru *Boot.bin*

Soubor *Fsbl.elf* se vytvoří pomocí stejnojmenné šablony jako aplikace v SDK studiu. FSBL je zkratka „first stage boot loader“ a označuje program obsahující počáteční nastavení komponent desky. Je tudíž vázaný na její zapojení a při jakékoliv změně je třeba aplikaci znovu přeložit.

Soubor *System.bit* je automaticky vygenerován při překladu hardwarového zapojení desky a obsahuje nastavení FPGA modulu. Lze ho najít na několika místech v kořenové složce projektu. Například ve složce „SDK/hw-platform“, kde „hw-platform“ je vytvořený „Board Support Package“ (BSP) pro dané zapojení desky. BSP slouží jako zdrojové soubory pro jednotlivé aplikace a definuje, jaká zařízení jsou k dispozici.

Soubor *U-boot.elf* je takzvaný zavaděč operačního systému. Jeho verze je párována s verzí jádra operačního systému, takže není zaručena kompatibilita se starší verzí. Má na starost rozbalení obrazu ramdisku a jádra Linuxu do paměti systému. Pokud není ramdisk dostupný, předpokládá se existence kořenových souborů systému na některé z dostupných paměťových medií. Pokud není potřeba měnit místo kořenových souborů či formát zdrojových dat jádra, je dostupná předkompilovaná verze [8].

2.1.2 Tvorba souboru *u-boot.elf*

Tvorba tohoto souboru pro ZedBoard je jednoduchá. Potřebujeme k tomu zdrojové soubory dostupné na webu „github.com/Xilinx“ [9] ve složce „u-boot-xlnx“. Pro nastavení kompilace jsou důležité konfigurační soubory ve složce „u-boot-xlnx/include/configs“, ze kterých využijeme konfiguračního souboru *zynq_zc70x.h*. Ten

definuje důležité parametry, použité v konfiguračním souboru *zynq_common.h,* pro desku ZedBoard.

Ze všech možných parametrů nás zajímá „CONFIG_EXTRA_ENV_SETTINGS“, který přepisuje bootovací parametry desky. Jedná se o definici názvu či cesty ke zdrojovým souborům *uImage, uramdisk.image.gz a devicetree.dtb* a jejich cílovou adresu v paměti. Zároveň definuje různé bootovací metody z dostupných paměťových úložišť desky. V tomto případě využíváme „sdboot“. Jednotlivé bootovací metody se liší ve formátu příkazu „fatload“ respektive v jeho prvním parametru, kterým je umístění souboru. Například u „sdboot“ metody je hodnota parametru „mmc 0“, což označuje první oddíl na SD kartě.

Pokud vynecháme použití souboru *ramdisk.image.gz* a patřičně upravíme závěrečný příkaz k bootování „bootm“, nedojde k nahrání kořenových souborů systému do paměti a je možné využít jejich jiné umístění, například na jiném oddílu SD karty. Celou bootovací metodu a její parametry můžeme napsat také ručně v bootovací konzoli desky.

Série použitých příkazů pro tvorbu *u-boot.elf* vypadá následovně:
```
make zynq_zc70x_config
make
```

Upravený příkaz „bootm“ je:
```
bootm ${kernel_load_address} - {devicetree_load_address};
```

2.1.3 Tvorba jádra Linuxu

V řadě aplikací stačí využít již zkompilované jádro. Pro vývoj ovladačů je ale zapotřebí mít zdrojové soubory jádra, pro které ovladače píšeme. Ty jsou k dispozici na webu „github.com“ daného projektu. Nejrozšířenější je ArchLinux [10], který je možné potkat na mnoha platformách. Další podobné distribuce jsou Digilent Embedded Linux [11] a PetaLinux. Vesměs se liší přiloženými ovladači pro podporu systému od dané firmy. Pro produkty od firmy Xilinx [12] je vyvíjen právě PetaLinux. Jeho zdrojové soubory jsou k dispozici na webu „github.com/Xilinx“ [9] ve složce „linux-xlnx“.

Pokud máme stažené zdrojové soubory a správně nastavený kompilátor, stačí nám vytvořit konfigurační soubor *.config* a spustit kompilaci. V distribuci jsou předpřipravená nastavení pro nejrůznější systémy, ze kterých využijeme soubor *xilinx_zynq_defconfig* jako přednastavení pro desku ZedBoard. Dodatečná nastavení můžeme provést buď přímou editací souboru nebo parametrem „menuconfig“.

Pro naše potřeby potřebujeme jádro rozšířit o podporu I^2C a SPI rozhraní. Potřebné ovladače se liší v závislosti na použitých IP jádrech. Pro komponenty I^2C a SPI v PS jsou ovladače *I2c_cadence.c* a *Spi_cadence.c* a pro komponenty I^2C či SPI z PL jsou ovladače *I2c_xilinx.c* a *Spi_xilinx.c*. Po každé editaci souboru *.config* je nutné znova zkompilovat celé jádro. U příkazu ke spuštění kompilace musíme přidat parametr „UIMAGE_LOADADDR“ pro nastavení korektní bootovací adresy.

Série použitých příkazů pro kompilaci jádra vypadá následovně:

```
make ARCH=arm xilinx_zynq_defconfig
make ARCH=arm menuconfig
make ARCH=arm UIMAGE_LOADADDR=0x8000 uImage
```

Celý překlad zabere běžnému počítači středního výkonu asi půl hodiny času. Výsledný soubor *uImage* najdeme ve složce „linux-xlnx/arch/arm/boot“. Vytvoří se i soubor *zImage*, který slouží k tomu samému účelu. Jejich použití záleží jen na konfiguraci zavaděče *u-boot* a liší se v použité kompresy.

2.1.4 Tvorba souboru *Devicetree.dtb*

Zdrojovým souborem pro *devicetree.dtb* je textový soubor *devicetree.dt*s. Pro konverzi mezi nimi slouží program „dtc“, který lze najít ve složce „linux-xlnx/scripts/dtc/“. Soubor *devicetree.dts* obsahuje základní informace o použitých komponentách. Jeho plné znění je v příloze A. Umožňuje inicializaci ovladačů používaných danými komponentami při startu sytému.

Každý *devicetree.dts* je přiřazen k určitému typu desky parametry „compatible“ a „model“. Je to proto, že jsou zde definovány velikosti pamětí a procesory desky, které se vztahují k danému typu. Neméně důležitým parametrem je i „chosen“, který definuje parametry při spouštění jádra Linuxu. Jedná se o „bootargs“, které nastavují parametry rozhraní UART na desce a adresu umístění kořenových složek. Další parametr „linux,stdout-path“ přesměrovává textový výpis ze systému na již zmíněný UART.

Správné vytvoření tohoto souboru vyžaduje rozsáhlé znalosti o struktuře zapojení desky a potřebách jednotlivých ovladačů. Pokud bude ovladači chybět nějaký potřebný údaj o zařízení, inicializace selže a zařízení nebude možné používat přes daný ovladač.

Základním údajem určeným návrhem zapojení desky ve studiu XPS jsou použitá IP jádra, jejich bázové adresy, rozsah paměťového prostoru a čísla přerušení. Vnitřní komponenty v PS dostupné přes „processing_system7“ mají adresu vyšší než 0xE0000000 a parametry

k nim dohledáme v dokumentaci desky [13]. Bázové adresy pro jednotlivá IP jádra jsou vypsány v Tab. 1. Prefix „0x“ u bázové adresy označuje, že je číslo psané v hexadecimálním kódu.

Tab. 1 Bázové adresy IP jader

Jméno IP jádra	Bázová adresa	Velikost
Processing_system7 (DDR)	0x00000000	512 MB
Sensor_capture (SC)	0x62220000	64 KB
Camera_capture (CC)	0x66000000	64 KB
Multi_layer_video (MLV)	0x6AC00000	64 KB
Accelerator (ACC)	0x77600000	64 KB
Shutter_pwm (SHU)	0x77800000	64 KB
Lens_pwm (LEN)	0x77820000	64 KB
Uart	0xE0001000	4 KB
Usb	0xE0002000	4 KB
I2C0	0xE0004000	4 KB
I2C1	0xE0005000	4 KB
SPI	0x42000000	64 KB
Gpio	0xE000A000	4 KB
Enet	0xE000B000	4 KB
Sdio	0xE0100000	4 KB
Ttc	0xF8001000	4 KB

Předpřipravená IP jádra z PS a jiné systémové komponenty mají pevně daný identifikátor přerušení, viz Tab. 2. Přidaným IP jádrům v PL je možné přiřadit číslo přerušení z rezervovaných rozsahů 61-68 a 84-91. Maximální počet přerušení generovaný z PL je 16. Přiřazená čísla přerušení k našim komponentám jsou v Tab. 3. O funkčnost přerušení se stará systémová komponenta „Generic Interrupt controller“.

Tab. 2 Čísla přerušení PS komponent

Název IP jádra	Číslo přerušení
Uart	59
Usb	53
I2C0	57
I2C1	80
GPIO	52
Enet	54
Sdio	56
Ttc	42-44

Tab. 3 Čísla přerušení PL komponent

Název IP jádra	Název přerušení	Číslo přerušení
Multi_layer_video	Hsync_interrupt	91
Multi_layer_video	Vsync_interrupt	90
Sensor_capture	Line_interrupt	89
Sensor_capture	Frame_interrupt	88
Sensor_capture	Transfer_interrupt	87
Sensor_capture	Prog_line_interrupt	86
Sensor_capture	Prog_transfer_interrupt	85
Camera_capture	Frame_interrupt	84
Camera_capture	Prog_line_interrupt	68
Camera_capture	Prog_transfer_interrupt	67
Accelerator	Transfer_interrupt	66
SPI	Ip2intc_Irpt	65

Struktura souboru *devicetree.dts* je taková, že každé zařízení má ve složených závorkách definovány svoje parametry. Pokud má komponenta nějaká přidružená zařízení, jsou definována v prostoru mezi jeho závorkami. Je dodržována hierarchie „sběrnice-zařízení“ nebo „sběrnice-rozhraní-zařízení“. To, jaký ovladač patří k danému zařízení, určuje parametr „compatible“. Bázová adresa zařízení a velikost je obsažena v parametru „reg“.

Hodnoty přerušení jsou v parametru „interrupts“, který je tvořen třemi hodnotami „zdroj přerušení“, „číslo přerušení“, „typ přerušení“. Zdroj přerušení z PL má hodnotu 0. Číslo přerušení je zde nutné uvádět přepočtené na vzdálenost od zdroje, což pro PL znamená odečíst od každého čísla hodnotu 32. Typ přerušení na náběžnou hranu má hodnotu 4. Komponenta odpovědná za obsluhu přerušení je určena parametrem „interrupt-parent“. Ostatní parametry jsou závislé na konkrétním ovladači.

Bohužel ani z dostupných zdrojových kódů nejsou všechny potřebné parametry dobře čitelné. K dispozici je ukázková dokumentace ve zdrojových souborech jádra Linuxu „linux-xlnx/Documentation/devicetree/“,ve které najdeme ukázky záznamu různých typů komponent. Ale ani tam se člověk nedozví všechny potřebné informace.

Další možností je použití záznamů z jiných *devicetree.dts* souborů, které najdeme ve složce „linux-xlnx/arch/arm/boot/dts/“. Při jejich zkoumání narazíme na několik různých nuancí, například s používání přepisu názvu komponent za symbolický název a jeho používání jako parametr. Používaný způsob pro tvorbu *devicetree.dts* je vygenerováním zdrojových souborů pomocí nějakého vhodného nástroje. Při stažení a doinstalování

zdrojových souborů „device-tree-xlnx“ lze k vygenerování využít studia SDK či Vivado nebo využít nástroje „Petalinux tool“.

Zmíněné nástroje pracují s obecnými předepsanými strukturami vytvořenými pro každou dostupnou desku. Do výsledného *devicetree.dtb* souboru pak přidávají specifikace podle zvoleného nastavení desky. Pokud jsou v nastavení obsaženy vlastní IP jádra, může metoda generování selhat.

2.1.5 Využití nástroje Petalinux tool

V nástroji „Petalinux tool“ lze vytvořit projekt z vygenerovaného zapojení desky pomocí programu XPS a SDK. Ten umí vygenerovat všechny potřebné soubory pro spuštění na desce a dokonce obsahuje simulátor desky „QEMU“, na kterém lze operační systém také spustit. Zásadním problémem všech automatických nástrojů je to, že neumí správně pracovat s přidanými uživatelskými IP jádry v PL. Je to hlavně proto, že pro většinu z nich nejsou dostupné ovladače.

Nové verze generátorů uživatelských IP jader už získávají podporu generování ovladače pro dané jádro. Ty jsou dodávány jako šablony (předvyplněné kostry), které nabízejí pouze základní funkcionalitu – zápis a čtení z registrů. Jejich přínos pro usnadnění vývoje ovladače je malý.

Nástroj „Petalinux tool“ je k dispozici na webu firmy Xilinx [12]. Pro integraci do systému CentOs 7 je zapotřebí doinstalovat potřebné knihovny „Development Tools“. Poté stačí nastavit povolení k instalaci instalačnímu souboru a nainstalovat.

Příkaz k instalaci knihoven a všech potřebných návazností:

```
sudo yum groupinstall "Development Tools"
```

Dále uvedu postup pro vytvoření projektu a vygenerování spouštěcích souborů pomocí tohoto nástroje. Přesuneme se do složky, do které jsme nástroj nainstalovali a importujeme nastavení v souboru *settings.sh* do příkazové řádky pomocí příkazu „source“. Zpřístupní příkazy „petalinux-create“, „petalinux-config“, „petalinux-build“, „petalinux-package“.

Projekt vytvoříme pomocí příkazu „petalinux-create“ s parametry „--type project“, „--template zynq“ a „--name *jméno_projektu*“. Tím získáme stejnojmennou složku projektu, ve které budeme nadále pracovat. Dalším krokem je nastavení hardwarové konfigurace desky. Slouží k tomu příkaz „petalinux-config“ s parametrem „--get-hw-description

cestaH -p *cestaP*“, kde „*cestaP*“ je cesta k projektu petalinuxu a „*cestaH*“ je cesta k hardwarovému popisu (BSP) v SDK projektu.

Pomocí příkazu „petalinux-config“ je možné nastavovat konfigurační soubory pro jádro systému a ostatní komponenty. Pro nás jsou důležité vygenerované zdrojové soubory pro *devicetree.dtb*, ze kterých vyčteme potřebné parametry k jednotlivým komponentám. Tyto zdrojové soubory jsou velmi obsáhlé, protože obsahují parametry každé dostupné komponenty na desce ZedBoard. Komponenty, které nejsou použity v návrhu, jsou pomocí parametru „status“ deaktivovány.

Vygenerovány jsou i parametry k IP jádrům obsaženým v PL, ale některé nemají přiřazený ovladač. Mohou mít špatné číslo parametru „interrupt-parent“, které je potřeba zkontrolovat, aby bylo stejné jako u ostatních komponent.

Pomocí příkazu „petalinux-build“ se spustí kompilace všech nakonfigurovaných souborů. Pokud jsou všechna nastavení pořádku, vygenerují se všechny potřebné soubory pro spuštění systému. Na středně výkonném počítači to zabere půl hodiny. Vygenerované soubory najdeme ve složce „jméno_projektu/images/linux“, jejíž obsah stačí nahrát na SD kartu. Testovat systém můžeme na desce nebo na virtuální desce příkazem „petalinux-boot --qemu“.

Jelikož vytvořený projekt obsahuje desítky tisíc souborů, pro jeho distribuci slouží zdrojový balíček „BSP“. Ten vytvoříme příkazem „petalinux-package –bsp –p cesta_projektu“. Z vytvořeného BSP souboru lze opět vytvořit plnohodnotný projekt v „Petalinux tool“ příkazem „petalinux-create –t project –s cesta_k_bsp“.

2.1.6 Příprava SD karty

Bootovací karta musí obsahovat výše zmíněné soubory, a to tak, aby byly pohromadě. Jména souborů musí odpovídat názvům použitých v nastavení při jejich tvorbě. Je možné změnit jejich názvy i umístění, ale vyžaduje to úpravu inicializačních souborů. Pro rekapitulaci se jedná o soubory: *boot.bin, uImage, devicetree.dtb, ramdisk.image.gz.*

Pokud nechceme používat ramdisk, není problém (při patřičné úpravě konfiguračních souborů) využívat speciálního datového oddílu vytvořeného na SD kartě. Minimální velikost karty je pak 4GB, doporučená 8GB. Pomocí programu schopného práce s oddíly na datovém úložišti, například „GParted“, vytvoříme dva oddíly „boot“ a „rootfs“. První oddíl „boot“ s formátem FAT32 začínající na 4MB a s rozumnou velikostí 50MB až 1GB.

Druhý oddíl „rootfs“ s formátem EXT4 (je možné použít i nižší verze) vyplňující zbytek karty. Zde je omezení pro minimální velikost, protože oddíl „rootfs“ je nutné naplnit daty pro daný typ Linuxu.

Pro verzi Linaro 12 jsou zdrojové soubory například *linaro-precise-ubuntu-desktop-20120923-436.tar.gz* dostupné v archivu na webu „linaro.org“ [14]. Pro kopírování souborů je metod více, já používám následující postup. Rozbalení souboru příkazem „tar“ s parametry „zxf“ a kopírování příkazem „rsync“, to vše s administrátorským oprávněním.

```
sudo tar zxf linaro-precise-ubuntu-desktop-20120923-
436.tar.gz
sudo rsync -a --progress ./ /media/uzivatel/rootfs/
```

Proces trvá řádově desítky minut. Výhodou tohoto typu řešení kořenových souborů Linuxu je to, že po restartu systému neztratíte konfigurační data. Ale tato výhoda bývá při tvorbě a ladění spíše na obtíž, protože je mnohdy potřeba některé provedené změny vracet do původních stavů.

2.2 Zprovoznění OS Linux

Pro spuštění operačního systému Linux na desce ZedBoard je potřeba mít všechny již zmíněné soubory na SD kartě nebo v QSPI paměti desky. K tomu patřičně nastavené propojky na desce. V mém případě se vše týká spouštění z SD karty. Pokud je správně nastaven „boot env“, začne nabíhat jádro Linuxu. Postupně se čte soubor *devicetree.dtb* a spouští se jednotlivé ovladače.

Spouštění probíhá od inicializace paměti a procesorů po jednotlivé komponenty. Při inicializaci sběrnice se obvykle žádná hláška nezobrazí. Ale zobrazují se údaje o připojených zařízeních. Na závěr se spustí jednotlivé předdefinované služby a přihlašovací služba. Používá se standardní přihlašovací jméno „root“ a heslo „root“.

2.3 Ovladače a moduly

Ovladače umožňují aplikacím přístup k rozličným zařízením. Zdrojové soubory k ovladači se skládají z mnoha souborů, z nich každý definuje různou úroveň přístupu k zařízení. Od přístupu na nejnižší úrovni (nastavení jednotlivých registrů), přes definici jednotlivých funkcí až po počáteční inicializaci zařízení a obsluhu potřeb zařízení. Takový způsob rozdělení do mnoha souborů přináší velkou variabilitu vývojářům. Přístup

ke všem možným nastavení zařízení se může stát užitečným v pozdějších vývojích ovladače. Vývoj ovladače poskytující velké množství variant přístupu k zařízení je časově velmi náročný. Je zapotřebí nahlížet na ovladač z několika možných úhlů. Velmi důležité je chování ovladače z hlediska ochrany dat před uživateli, či ošetření maximálních fyzických hodnot zařízení, které nemusí korespondovat s maximálními hodnotami použitých proměnných.

Ovladač se zavádí přímo do jádra systému a může mít přístup ke všem datovým strukturám, funkcím a proměnným napříč jádrem. To může vést ke kolizi názvu proměnných a zhroucení sytému. Každý ovladač/programátor by měl využívat svůj prefix před každým názvem, který tak unifikuje jednotlivé názvy. Dále by se mělo využívat příznaku „static“ pro skrytí proměnných a funkcí v jádře.

Aby ovladač mohl využívat funkce již definovaných přístupů, musí implementovat předepsané struktury. Ty lze poté speciálními registračními funkcemi přidat do již běžících ovladačů „vyšší“ úrovně. Ukázkou toho může být přidání zařízení na sběrnici, kde se vyplní předepsaná struktura zařízení příslušnými parametry a poté se zařízení ohlásí obsluze sběrnice jako dostupné. Všechny struktury a funkce ovladače jsou definovány v hlavičkových souborech, které se implementují do ovladače funkcí „include“.

To, jaký hlavičkový soubor je potřeba, se odvíjí od funkcí, které chceme v ovladači využít. Většina jich je ve zdrojových souborech Linuxu ve složce „include/linux“ nebo „arch/arm/include/asm“. Z těch základních to je soubor *kernel.h*, který je nutný pro práci v jádře systému. Dále pak *version.h* pro zjištění verze jádra a informací o systému. Pro definování běžných datových typů tu je *types.h*. Pro možnost obsluhy ovladače přes soubor je *fs.h*. Pro umožnění funkce ovladače jako modulu slouží *module.h.*

2.3.1 Rozdíl mezi modulem a ovladačem

Ovladač je již zabudován v jádře systému a spouští se zároveň se systémem. Některé z nich pomáhají systému nabootovat. Aby nebyl potřeba po naběhnutí systému zásah uživatele, mohou se po nabootování systému automaticky vkládat i moduly.

Ukázkou ovladače je třeba ovladač implementující hlavičkový soubor *platform_device.h,* který definuje obecnou strukturu zařízení „platform_device“, která je plněna daty ze souboru *devicetree.dtb*. Takový ovladač obsahuje definovanou strukturu funkcí

„platform_driver“, ze které jsou nejdůležitější funkce „probe“ a „remove“. Tyto funkce jsou volány, při připojování a odpojování odpovídajícího zařízení. Při inicializaci ovladače se musí zaregistrovat do seznamu ovladačů zařízení funkcí „platform_driver_register“, tím si aktivuje svoje funkce z „platform_driver“. Pro identifikaci příslušného ovladače a zařízení slouží makro „MODUL_DEVICE_TABLE“. Pomocí něho si ovladač určí textový parametr „compatible“, který musí dané zařízení obsahovat.

Modul je před použitím zapotřebí přidat do jádra systému příkazem „insmod“ nebo „modprobe“. Příkaz „modprobe“ je inteligentnější než příkaz „insmod“ a automaticky vyhledává přidružené ovladače či dostupná zařízení. Nespornou výhodu má modul při vývoji. Jelikož je překládán samostatně, není tedy nutné po vytvoření nové verze překládat celé Linuxové jádro a měnit zdrojové bootovací soubory.

Každý ovladač zařízení může být modulem, ovšem mohou existovat moduly, které nemohou být ovladači. Modul má funkce „init_module“ a „cleanup_module“, ve kterých provádí svou inicializaci. To může být pro řadu ovladačů zbytečné.

2.3.2 Základní kostra modulu/ovladače

Pro inicializaci modulu jsou nutné hlavičkové soubory *module.h* a *kernel.h*. Ty potřebují implementaci funkcí „init_module“ a „cleanup_module“, pro inicializaci modulu. Pokud se nám názvy inicializačních funkcí nelíbí, pomocí maker „module_init(funkce)“ a „module_exit(funkce)“ je přesměrujeme. Jsou definované v souboru *init.h*.

Další užitečná makra slouží pro identifikaci modulu. Jedná se o „MODULE_AUTOR (*autor*)“ pro jméno autora, „MODULE_DESCRIPTION (*popis*)“ pro popis k modulu, „MODULE_LICENCE (*licence*)“ pro nastavení licence. Některé systémy podporují použití jen modulů s „GPL“ licencí. Tyto informace jdou v systému zobrazit příkazem „modinfo“.

Následující funkce a přístupy se napříč verzemi jádra Linuxu mění často. Přesněji řečeno proběhla změna mezi verzí 2.6 a 2.7. Což vzhledem k rozšířenosti starší verze vnáší do vyhledané literatury zmatky. Mnoho ovladačů má podporu obou verzí. Ta ovšem dělá zdrojový kód značně nečitelným díky zdvojené deklaraci většiny použitých funkcí.

Pro základní interakci mezi uživatelem a modulem slouží vytvořený soubor v kořenové složce „/dev“. K tomu je zapotřebí implementovat soubor „fs.h“. Díky němu jsme schopni

vytvořit soubor a reagovat na jeho otevření, čtení, zápis, zavření a předávat si skrz něj data z uživatelského paměťového prostoru do paměťového prostoru jádra.

K tomu slouží unifikovaná posloupnost funkcí, která bezpečně přidá soubor do složky „/dev“, který je propojen s ovladačem pomocí čísla „Major_num“. Počet čísel v systému je omezen na 256 a plánuje se jeho navýšení. Číslo „Major_num“ slouží pro identifikaci typu ovladače. V rámci jednoho typu ovladače existují čísla „Minor_num“, která dále identifikují jednotlivý ovladač.

Posloupnost funkcí začíná s „alloc_chardev_region“ pro bezpečnou alokaci paměti pro pointer struktury ovladače. Následuje „class_create“ pro vytvoření typu ovladače. Poslední a hlavní funkcí je „device_create“, která pod danými „Major_num“ a „Minor_num“ vytvoří soubor s daným názvem ve složce „/dev“. Pokud jsou na místech čísel nuly, jsou jim hodnoty přiřazeny z volného rozsahu.

Součástí inicializace je i struktura „file_operations“, která obsahuje funkce obsluhující události „read“, „write“, „open“, „release“, „unlocked_ioctl“ a jiné. Funkce „read“ je volána při pokusu čtení souboru. Funkce „write“ při pokusu zápisu do souboru. Funkce „open“ při otevření souboru a funkce „realease“ při jeho zavření. Ačkoliv se mohou zdát funkce „read“ a „write“ vhodné pro předávání dat z a do modulu, na předávání jednotlivých hodnot či nastavování parametrů modulu je využívána funkce „unlocked_ioctl“. Ve starší verzi byla pouze „ioctl“, ale v novější verzi se rozšířila na „unlocked_ioctl“ a „compat_ioctl“. Důvodem je vylepšení správy paměti při volání těchto funkcí.

Pro usnadnění inicializace ovladače lze využít funkce „misc_register“ ze souboru „miscdevice.h“. Ta automaticky vytvoří obslužný soubor v „/dev“ a nastaví „Major_num“ ovladače do neutrální třídy „Misc“. Potřeba je jen nastavit parametry struktury „miscdevice“, kde je jméno souboru jako „name“, „Minor_num“ ovladače jako „minor“ a „file_operations“ struktura jako „fops“. Číslo „Minor_num“ se dá nastavit na neobsazenou hodnotu makrem „MISC_DYNAMIC_MINOR“.

2.3.3 Využití IOCTL

Makra použitá pro usnadnění práce s funkcí „ioctl“ jsou v souboru *ioctl.h*. V modulu obsažená obslužná funkce „ioctl“ obsahuje parametry volaného souboru, čísla „ioctl_num“ a parametru „ioctl_param“. Kde „ioctl_num“ slouží k rozpoznání volané

akce a „ioctl_param“ předává příslušná data. Pro každé „ioctl_num“ se definuje určitá akce, kterou modul vykoná při jeho přijmutí.

Volané akce mohou být různého charakteru (přijímat či vracet data) a „ioctl_param“ může mít libovolný typ. Aby při větším množství akcí (čísel „ioctl_num“) v modulu nenastal zmatek, využívá se předdefinovaných maker „_IOR“, „_IOW“, „_IOWR“. Ty rozdělují „ioctl_num“ do čísel: směr „dir“, typ „type“, pořadového čísla „nr“ a velikosti parametru „size“. Hodnota „dir“ je určena použitým makrem. Hodnota „type“ je většinou „Major_num“ daného ovladače. Hodnota „nr“ je základní rozlišovací hodnotou dvou maker. Makra s příslušnými hodnotami si definujeme například do zvláštního hlavičkového souboru. Ten bude společný modulu i aplikaci, která jej využívá.

2.3.4 Mapování paměťových prostorů

Dostupný paměťový prostor je reprezentován 32 bit slovem a dělíme ho na fyzický a virtuální. IP jádra pracují s fyzickou adresou, která koresponduje s adresami jednotlivých zařízení. Pokud potřebujeme využít adresový prostor, můžeme si vybrat ze spojitého či nespojitého prostoru. Spojitý virtuální adresový prostor nemusí být zároveň fyzicky spojitý.

Pro alokaci spojitého fyzického paměťového prostoru slouží funkce „kmalloc“, a pro to samé ve virtuálním prostoru funkce „vmalloc“. Pro získání virtuální adresy odkazující na určitý spojitý fyzický prostor je funkce „mmap“. Pro přidělení fyzické adresy virtuální adrese je funkce „kmap“. Pro uvolnění virtuální paměti je funkce „vfree“ a fyzické paměti „kfree“. Pro uvolnění paměti vázané na konkrétní fyzickou adresu je funkce „iounmap“.

Pro překlad mezi virtuální a fyzickou adresou je funkce „virt_to_phys“ nebo obráceně „phys_to_virt“. Pro předávání informací z paměťového prostoru jádra do uživatelského prostoru aplikacím jsou funkce „put_user“ nebo „copy_to_user“. Pro obrácený směr zase funkce „get_user“ nebo „copy_from_user“. Ty umožňují přenos buď jednoho znaku, nebo pole. Všechny zmíněné funkce jsou obsaženy v souborech *vmalloc.h, io.h, uaccess.h, slab.h.*

2.3.5 Využití I^2C

Pro komunikaci po I^2C sběrnici jsou všechny potřebné funkce definovány v souboru i2c.h. Ten definuje standardy pro zprávu „i2c_msg“, pro ovladač „i2c_driver“, pro zařízení „i2c_client“ a pro celou sběrnici „i2c_adapter“.

Při správném záznamu v *devicetree.dtb* se určený ovladač postará o zpřístupnění sběrnice z operačního systému. To znamená, že obsahuje strukturu „i2c_driver“ pro obsluhu příslušné sběrnice. Dále vytvoří instanci struktury „i2c_adapter“ a vytvoří jí příslušný soubor ve složce „/dev“. V záznamech může být uvedeno nějaké zařízení připojené na sběrnici, ke kterému existuje ovladač. Pak se vytvoří struktura „i2c_client“, která je přidružena příslušnému „i2c_adapter“.

Komunikace napříč různými zařízeními není přesně definovaná. Existují dvě formy přenosu dat: originální „i2c“ a odvozený „smbus“. Obě umožňují komunikaci po dvou vodičích. Mají však rozdílné napěťové úrovně a dosažitelné rychlosti. Ovladač i2c_cadence podporuje pouze originální i2c s možnou úpravou posílané zprávy, tj. definice relevance start, stop a potvrzovacích bitů.

Běžně se setkáme s takovou logikou posílání zpráv, že každý odeslaný bajt či adresa příjemci je následován jednobitovou odpovědí odesílateli. Pokud odesílatel neobdrží odpověď, ukončuje vysílání. Produkty firmy OmniVision Technologies, Inc. mají tuto logiku upravenou (protokol SCCB) tak, že ignorují posílanou odpověď [2].

2.3.6 Využití SPI

Pro komunikaci po SPI sběrnici jsou definované standardy v souboru *spi.h*. Komunikace je po 4 drátech [hodiny (CLK), k zařízení (MOSI), od zařízení (MISO), výběr zařízení (SS)] a je plně duplexní. Ovladač schopný komunikace implementuje strukturu „spi_driver“. Zde definuje „spi_master“ zařízení, které řídí komunikaci po sběrnici označenou číslem „bus_num“. K němu se připojuje „spi_device“, kterých může být i více, a adresují se přes SS. Vzájemně si vyměňují „spi_message“, která má jedno pole dat pro čtení a jedno pro zápis.

Komunikace začíná změnou napětí na pinu SS, kterým se vybírá cílové zařízení. Poté CLK udává validaci hodnot na MOSI a MISO pinech. Pomocí nich se zároveň přenesou data z i do zařízení. Po dokončení přenosu jednoho slova s definovanou délkou 8, 16 nebo 32 bitů se opět změní hladina napětí na SS. Hodnoty mohou být validní na hranu nebo hladinu signálu CLK.

2.3.7 Přerušení

Přerušení nastává na událost zařízení a vykoná se nějaká část kódu nezávisle na hlavní smyčce programu. Funkce k tomu dostupné jsou v souboru *interrupt.h*. Vytvoříme si

obslužnou funkci „sample_irq“ s parametry „irq“ a „dev_id“, kde „irq“ je číslo přerušení a „dev_id“ je číslo zařízení. Tuto obslužnou funkci musíme registrovat do tabulky přerušení v systému funkcí „request_irq“. Uvolnit obsazené přerušení můžeme funkcí „free_irq“.

Výpis všech aktuálních přerušení získáme příkazem „cat /proc/interrupts“. Každé přerušení má své číslo, popis, obslužnou funkci a typ vyvolání události (na hranu, hladinu). Součástí výpisu je i počet, kolikrát byly jakým jádrem procesoru obslouženy.

2.3.8 Propojení více modulů

Už jsem zmínil, že jednotlivé funkce či proměnné jsou napříč jádrem viditelné. Lze rozlišit celkem tři druhy viditelnosti: „static“, „external“, „exported“. „Static“ znamená viditelnost v daném dokumentu, „external“ umožňuje jinému kódu v jádře a „exported“, který zviditelní a zpřístupní funkce či proměnné každému modulu.

Funkce pro práci s „exported“ funkcemi či proměnnými nalezneme v souboru *kallsyms.h*. Ten zviditelní funkci nebo proměnnou pomocí makra „EXPORT_SYMBOL“ nebo „EXPORT_SYMBOL_GPL“ pouze pro moduly s GPL licencí. Všechny dostupné funkce či proměnné lze vypsat příkazem „cat /proc/kallsyms“. Pozor, výpis je extrémně dlouhý. Vypisují se adresy, názvy a příznaky typu a viditelnosti funkcí či proměnných. To, zda budou funkce či proměnné napříč jádrem viditelné, lze zakázat při překladu jádra.

3 Tvorba modulů

Pro překlad zdrojového souboru modulu pro danou verzi jádra musíme vytvořit soubor *Makefile*. V něm definujeme cestu ke zdrojovým souborům jádra, a který soubor se má přeložit jako modul. Výsledkem je to, že místo zdlouhavého příkazu k přeložení napíšeme v konzoly jen „make“. Ten může mít příznaky „all“ nebo „clean“. Jejich využití je vhodné při kompilaci více zdrojových dat naráz.

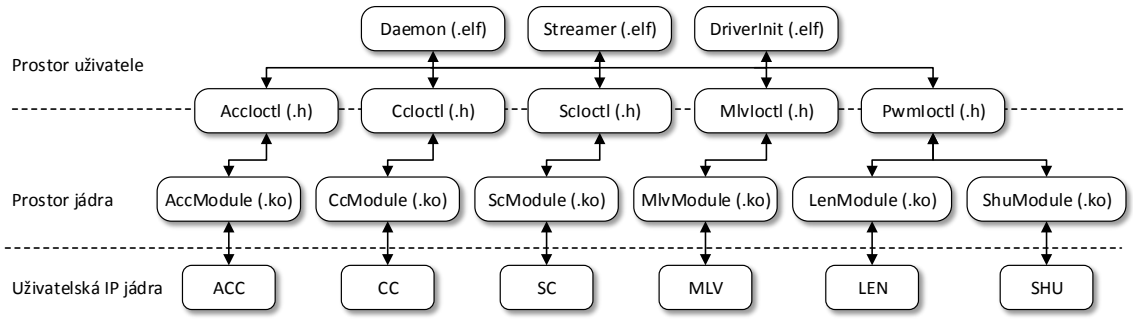
Ukázka obsahu souboru *Makefile*:

```
KERN_SRC =/home/uzivatel/linux-xlnx/
obj-m += soubor_modulu.o
all:
     make -C $(KERN_SRC) ARCH=arm M=`pwd` modules
```

Výsledkem kompilace je *soubor_modulu.ko*, který lze za běhu systému vložit do jádra. Výsledný soubor obsahuje číselný klíč jako zámek pro použitelnost modulu pouze na verzi jádra, pro kterou byl kompilován.

3.1 Vytvořené moduly

Moduly, které jsem vytvořil, vychází ze zdrojových souborů již vytvořeného firmwaru ke kameře. Protože využité knihovny pro správu paměti a periferií nejsou v jádře Linuxu, musel jsem je nahradit obdobnými. Pro komunikaci aplikace v uživatelském paměťovém prostoru s modulem v paměťovém prostoru jádra využívám funkci IOCTL. Ta pomocí IOCTL maker definovaných ke každému modulu předá informace z nebo do IP jádra, které modul obsluhuje, viz Obr. 6.



Obr. 6 Hierarchie vytvořených aplikací a modulů

Moduly obsahují potřebné funkce „init“ a „exit“ pro přidání a odebrání modulu z jádra Linuxu. Implementují strukturu „file_operations“ pro vytvoření obslužného souboru ve složce „/dev“ a mají ochranu před jeho vícenásobným otevřením.

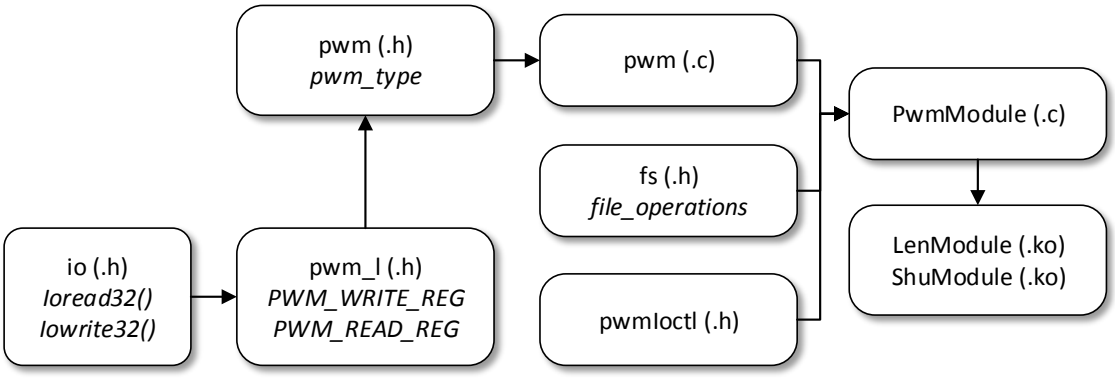
Všechny moduly jsou typu „Misc“ a mají dynamicky volené „Minor_num“. Každý modul má svůj hlavičkový soubor pro IOCTL makra a obsluhuje jedno IP jádro. Pro umožnění nastavení řídících registrů IP jader jsem využil funkci „ioremap“, která alokuje příslušný fyzický prostor dané velikosti a vrátí virtuální adresu na tento prostor. Pomocí této adresy a funkcí „iowrite32“ či „ioread32“ z *asm/io.h* lze s tímto prostorem komunikovat. Získaná adresa se uloží příslušné struktuře jako parametr „DeviceBaseAddress“. Při odebrání modulu se všechny alokované prostory uvolní funkcí „iounmap“.

Pro práci s daty obrazu vytvářím dostatečně velké a fyzicky spojité adresové prostory funkcí „kmalloc“. Při ukončování modulu je uvolním funkcí „kfree“. Získaná adresa na tyto prostory je virtuální. Příslušné prostory jsou použity jako cílová či zdrojová paměťová pole pro jednotlivá IP jádra. Protože IP jádra s nastavenou adresou paměťového pole pracují samostatně, potřebují jeho zápis ve fyzické podobě, tudíž adresy při nastavení překládám pomocí „virt_to_phys“.

Vytvořené moduly jsou: modul PWM *PwmModule.c* pro obsluhu IP jádra PWM, modul CC *CcModule.c* pro obsluhu kamery a IP jádra CC, modul SC *ScModule.c* pro obsluhu bolometru a IP jádra SC, modul MLV *MlvModule.c* pro obsluhu HDMI a IP jádra MLV a modul ACC *AccModule.c* pro obsluhu IP jádra ACC.

3.1.1 Modul PWM

Modul obsluhuje IP jádro PWM, které je použité ve dvou podobách LEN a SHU. Pro vytvoření cílových souborů *LenModule.ko* a *ShuModule.ko* je potřeba celá řada definičních souborů vyobrazená na Obr. 7.



Obr. 7 Souborová struktura modulu PWM

Moduly po přidání do systému vytváří soubory *ShuDevice* a *LenDevice*, přes které je možné s moduly komunikovat pomocí maker definovaných v souboru *pwmIoctl.h,* viz Tab. 4. Vnitřně se moduly liší pouze použitou bázovou adresou IP jádra

Pro nastavení daného IP jádra používám upravených funkcí v souboru *pwm.c*. Pro správu IP jádra implementuji strukturu „pwm_type“ ze souboru *pwm.h*. V níž jsou uloženy všechny aktuálně nastavené hodnoty. Pomocné hodnoty jsou definovány v souboru *pwm_l.h*.

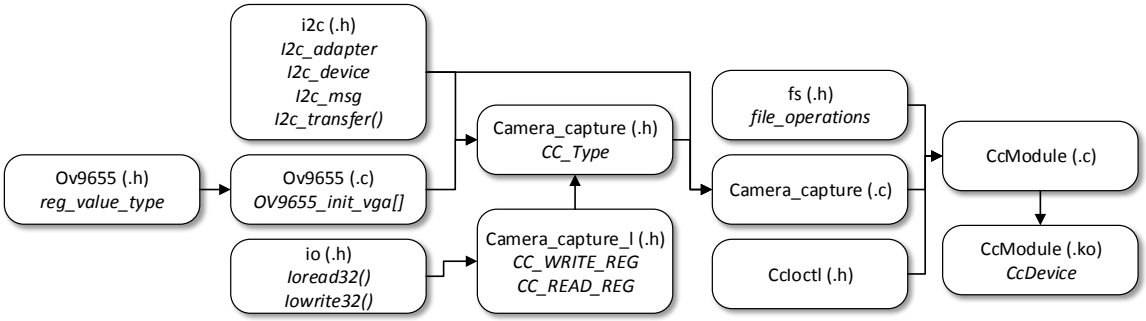
Pomocí modulu jdou nastavovat parametry IP jádra PWM: povolení výstupu, nastavení předděliče frekvence PWM, nastavení polarity PWM a nastavení plnění PWM. Pro funkci otevření a zavření záklopky ovládané modulem SHU je implementovaná funkce přepínání mezi dvěma předdefinovanými hodnotami výstupu.

Tab. 4 IOCTL makra PWM modulu

Název makra	Typ parametru	Popis funkce
PWM_ENABLE	Číslo	0 – zakáže výstup 1 – povolí výstup Jinak vrací nastavenou hodnotu
PWM_VALUE_SET	Číslo	Nastaví hodnotu plnění PWM
PWM_VALUE_GET	Číslo	Vrátí nastavenou hodnotu plnění PWM
PWM_PRESCALER_SET	Číslo	Nastaví hodnotu předděliče PWM
PWM_PRESCALER_GET	Číslo	Vrátí hodnotu předděliče PWM
PWM_POLARITY	Číslo	0 – plnění výstupu zleva 1 – plnění výstupu zprava Jinak vrací nastavenou hodnotu
PWM_SHUTTER	Číslo	0 – nastavení pro zavření záklopky 1 – nastavení pro otevření záklopky Jinak vrací nastavenou hodnotu

3.1.2 Modul CC

Modul obsluhuje IP jádro CC a vytváří na I^2C sběrnici zařízení VGA kamery „OV9655“, které nastaví do požadovaného režimu. Pro vytvoření cílového souboru *CcModule.ko* je potřeba celá řada definičních souborů vyobrazená na Obr. 8.



Obr. 8 Souborová struktura modulu CC

Modul po přidání do systému vytváří soubor *CcDevice*. Skrz něj je možné s modulem komunikovat, pomocí IOCTL maker definovaných v souboru *CcIoctl.h*, viz Tab. 5. Pro nastavení IP jádra CC implementuji upravený soubor *camera_capture.c*, který definuje jednotlivé obslužné funkce pracující se strukturou „CC_type“. Tato struktura je definována v souboru *camera_capture.h* a pomocné hodnoty pro správné nastavení IP jádra jsou definovány v souboru *camera_capture_l.h*.

Při inicializaci modulu je potřeba nastavit kameru připojenou na I^2C sběrnici. K tomu si žádám o sběrnici „I2c_adapter“ a na něj přidám zařízení „i2c_device“ s nastavenou adresou kamery. Soubory *ov9655.c* a *ov9655.h* definují strukturu „reg_value_type“, která obsahuje osmibitovou adresu registru a jeho osmibitovou hodnotu. Z ní je vytvořené velké inicializační pole „OV9655_init_vga“, které se odešle zařízení přes I^2C. Odesílání probíhá pomocí struktury zprávy „i2c_msg“ s upraveným parametrem příznaku „flags“ na zanedbání bitu odpovědi „I2C_M_IGNORE_NAK“ (kvůli kompatibilitě se SCCB protokolem). Po odeslání celého pole se zkontroluje obsah řídícího registru kamery, zda odpovídá předpokládané hodnotě.

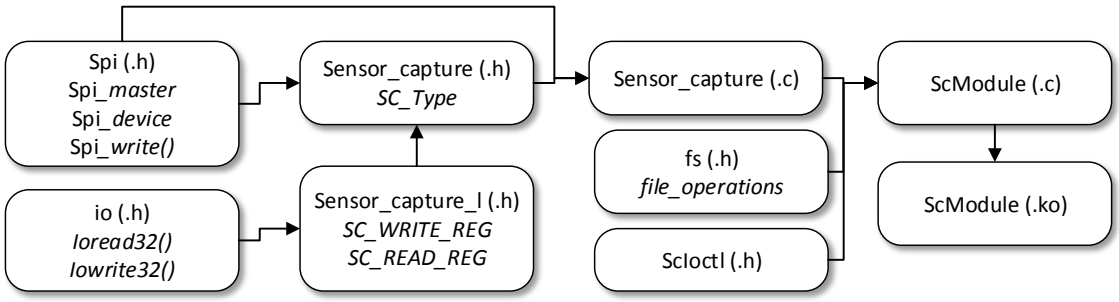
Modul vytváří dvě paměťová pole „camBuffer0“ a „camBuffer1“ pro ukládání obrazu. Mezi nimi se přepíná při přerušení „Frame_interrupt“ a zároveň se nastaví příznak „NextFrame“. Díky tomu jsou vždy v jednom z nich k dispozici neměnná data pro případné kopírování snímku. Adresy paměťových polí jsou rovněž k dispozici pro možné nastavení jiných modulů. Modul obsahuje příznak „cameraStream“ pro identifikaci povolení streamování obrazu.

Tab. 5 IOCTL makra CC modulu

Název makra	Typ parametru	Popis funkce
CC_NEXT_FRAME	Číslo	Vrací příznak snímku 0 – starý 1 – nový
CC_IMAGE	Pole	Vrací data snímku
CC_IMAGE_ADRESS1	Číslo	Vrací adresu camBuffer0
CC_IMAGE_ADRESS2	Číslo	Vrací adresu camBuffer1
CC_ENABLE	Číslo	0 – zakáže výstup 1 – povolí výstup Jinak vrací nastavenou hodnotu
CC_STREAM_ENABLE	Číslo	0 – zakáže stream 1 – povolí stream Jinak vrací nastavenou hodnotu

3.1.3 Modul SC

Modul obsluhuje IP jádro SC a vytváří na SPI sběrnici zařízení bolometru, se kterým komunikuje. Pro vytvoření cílového souboru *ScModule.ko* je potřeba celá řada definičních souborů vyobrazená na Obr. 9.



Obr. 9 Souborová struktura modulu SC

Modul po přidání do systému vytváří soubor *ScDevice*, přes který je možné s modulem komunikovat pomocí maker definovaných v souboru *ScIoctl.h,* viz Tab. 6. Pro nastavení IP jádra SC implementuji upravený soubor *sensor_capture.c*, který definuje jednotlivé obslužné funkce pracující se strukturou „SC_type“. Ta je definována v souboru *sensor_capture.h* a pomocné hodnoty jsou definované v souboru *sensor_capture_l.h.*

Při inicializaci je potřeba nastavit bolometrický čip připojený na SPI sběrnici. K tomu si žádám o „spi_master“ a na něj přidám zařízení „spi_device“ s příslušným nastavením. Pro zápis požadovaných parametrů do čipu používám funkci „spi_write“, čtení z čipu bohužel není možné. Všechny nastavené hodnoty parametrů se zároveň ukládají i do struktury „SC_type“, aby byly známy jich aktuální hodnoty.

Modul vytváří dvě paměťová pole „bolBuffer0“ a „bolBuffer1“ pro ukládání obrazu. Mezi nimi se přepíná při přerušení „Frame_interrupt“ a zároveň se nastaví příznak „NextFrame“. Díky tomu jsou vždy v jednom z nich k dispozici neměnná data pro kopírování snímku. Při každém novém snímku se uloží jeho parametry minimální, maximální a průměrné hodnoty do struktury „SC_type“. Adresy paměťových polí jsou k dispozici pro možné nastavení jiných modulů. Modul obsahuje příznak „StreamBolEna“ pro identifikaci povolení streamování obrazu.

Dále jsou tvořeny další dvě paměťová pole „corBuffer“ a „comBuffer“. Pole „corBuffer“ je určeno pro uložení korekční matice, která má stejné rozměry jako obraz. A pole „comBuffer“ slouží pro uložení průběžných výsledků při výpočtu korekční matice. Korekce jednotlivých pixelů obrazu je nutná, protože čip nemá žádnou vnitřní kalibraci.

Kalibraci provádím manuálně, zakrytím čipu předmětem o konstantní teplotě v celé své ploše. K tomu slouží například ovladatelná krytka „Shutter“. Prvním krokem výpočtu kalibrační matice je výpočet posunutí „compute_offset“. Při něm se určí jednotná nulová hodnota všech pixelů obrazu z čipu. Pro správnou korekci obrazu s velkými rozdíly hodnot by se měl provádět i výpočet zesílení „compute_gain“. Ten se provede při zakrytí čipu předmětem o jiné konstantní teplotě v celé své ploše, než byla při provedení výpočtu posunutí.

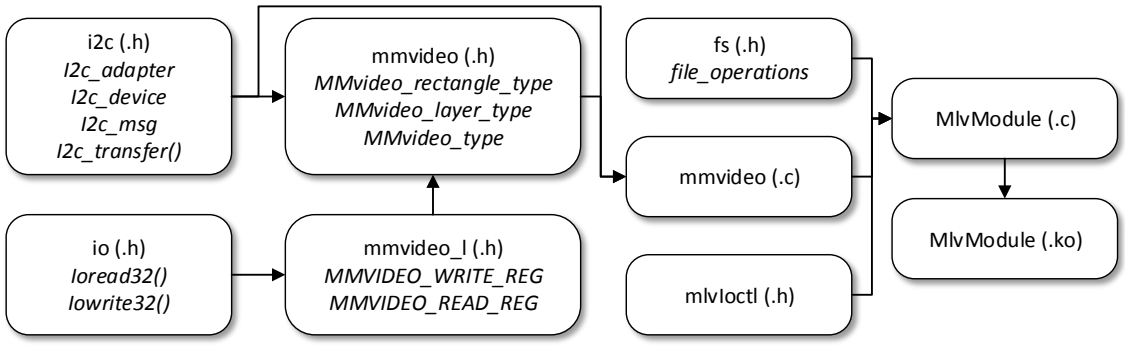
K výpočtům korekce jdou volit dva parametry „ComputationAverage“ a „ZeroOffset“. Parametr „ComputationAverage“ říká, z kolika snímků se má korekce počítat (proběhne mezi nimi průměrování) a parametr „ZeroOffset“ se odčítá od výsledků celé vypočtené korekční matice.

Tab. 6 IOCTL makra SC modulu

Název makra	Typ parametru	Popis funkce
SC_NEXT_FRAME	Číslo	Vrací příznak snímku 0 – starý 1 – nový
SC_IMAGE	Pole	Vrací data snímku
SC_IMAGE_ADRESS1	Číslo	Vrací adresu bolBuffer0
SC_IMAGE_ADRESS2	Číslo	Vrací adresu bolBuffer1
SC_ENABLE	Číslo	0 – zakáže výstup 1 – povolí výstup Jinak vrací nastavenou hodnotu
SC_CORRECTION_ENABLE	Číslo	0 – zakáže korekci 1 – povolí korekci Jinak vrací nastavenou hodnotu
SC_STREAM_ENABLE	Číslo	0 – zakáže stream 1 – povolí stream Jinak vrací nastavenou hodnotu
SC_COMPUTE_AVERAGE	Číslo	Nastaví počet snímků k výpočtu
SC_ZERRO_OFFSET	Číslo	Nastaví posun pro výpočet
SC_COMPUTE_OFFSET	Číslo	Spočítá ofsety pixelů
SC_COMPUTE_GAIN	Číslo	Spočítá zesílení pixelů
SC_SET_CORRECTION	Pole	Nastaví korekční matici
SC_GET_CORRECTION	Pole	Vrátí korekční matici
Parametry čipu	Číslo	Nastaví/vrátí hodnoty parametrů čipu

3.1.4 Modul MLV

Modul obsluhuje IP jádro MLV a vytváří na I^2C sběrnici zařízení HDMI, které nastaví do požadovaného režimu. Pro vytvoření cílového souboru Mlv*Module.ko* je potřeba celá řada definičních souborů vyobrazená na Obr. 10.



Obr. 10 Souborová struktura modulu MLV

Modul po přidání do systému vytváří soubor *MlvDevice*, přes který jde s modulem komunikovat pomocí maker definovaných v souboru *MlvIoctl.h*, viz Tab. 7. Pro nastavení

IP jádra MLV implementuji upravený soubor *mmvideo.c*, který definuje jednotlivé obslužné funkce pracující se strukturou „MMvideo_type“. Ta je definována v souboru *mmvideo.h* a pomocné hodnoty jsou definovány v souboru *mmvideo_l.h*.

Modul MLV obsahuje dvě instance struktury „MMvideo_layer_type“ pro ukládání zdrojových dat obrazu k zobrazení. Kromě adres na paměťová pole se zdrojovými daty obrazu obsahují i informace o jeho průhlednosti a pozici. Pozice obrazu je určena dvěma body uloženými ve struktuře „MMvideo_rectangle“.

Pro nastavení HDMI výstupu modul přidává na příslušnou sběrnici „i2c_adapter“ zařízení „i2c_device“ s nastavenou adresou řadiče HDMI. Tomu je pak posláno konfigurační nastavení výstupu. Modul si vytváří dvě paměťová pole „videoBuffer0“ a „videoBuffer1“ obsahující počáteční zobrazovaná data obrazu.

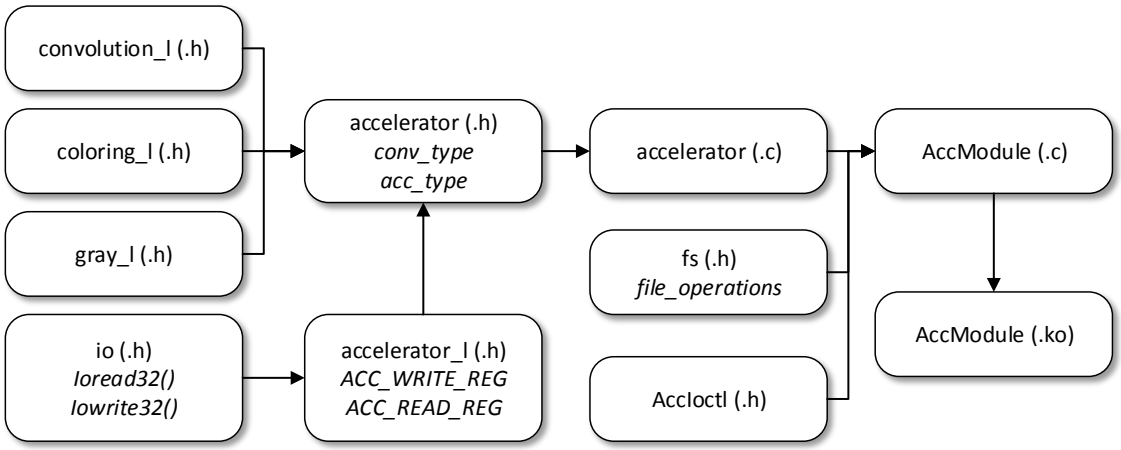
Adresy na zobrazovaná pole jdou z modulu vyčítat nebo je lze přepsat adresou na jiné pole. Modul umožňuje streamování obou dat těchto polí a k tomu účelu obsahuje identifikaci o povolení streamovaní a funkce na vyčítání obrazu.

Tab. 7 IOCTL makra MLV modulu

Název makra	Typ parametru	Popis funkce
MLV_SET_LAYER_0	Číslo	Nastaví adresu obrazu 0
MLV_SET_LAYER_1	Číslo	Nastaví adresu obrazu 1
MLV_SET_OVERRIDE_LAYER_0	Číslo	Nastaví průhlednost obrazu 0
MLV_SET_OVERRIDE_LAYER_1	Číslo	Nastaví průhlednost obrazu 1
MLV_SET_POSITION_LAYER_0	MMvideo_rectangle	Nastaví pozici obrazu 0
MLV_SET_POSITION_LAYER_1	MMvideo_rectangle	Nastaví pozici obrazu 1
MLV_GET_LAYER_0	Číslo	Vrátí adresu videoBuffer0
MLV_GET_LAYER_1	Číslo	Vrátí adresu videoBuffer1
MLV_SET_LAYER_0_DEFAULT	Číslo	Nastaví adresu videoBuffer0
MLV_SET_LAYER_1_DEFAULT	Číslo	Nastaví adresu videoBuffer1
MLV_GET_IMAGE_0	Pole	Vrátí obraz 0
MLV_GET_IMAGE_1	Pole	Vrátí obraz 1
MLV_STREAM_0_ENABLE	Číslo	Povolí stream obrazu 0
MLV_STREAM_1_ENABLE	Číslo	Povolí stream obrazu 1

3.1.5 Modul ACC

Modul obsluhuje IP jádro ACC a cyklicky kopíruje data mezi dvěma nastavenými poli s využitím zvolených modulů jádra. Pro vytvoření cílového souboru Acc*Module.ko* je potřeba celá řada definičních souborů vyobrazená na Obr. 11.



Obr. 11 Souborová struktura modulu ACC

Modul po přidání do systému vytváří soubor *AccDevice*, přes který je možné s modulem komunikovat pomocí maker definovaných v souboru *AccIoctl.h,* viz Tab. 8. Pro nastavení IP jádra MLV implementuji upravený soubor *accelerator.c*, který definuje jednotlivé obslužné funkce pracující se strukturou „acc_type“. Ta je definovaná v souboru *accelerator.h* a pomocné hodnoty jsou definovány v souboru *accelerator_l.h*. Pomocné hodnoty pro různé části IP jádra ACC jsou definovány v souborech *coloring_l.h, gray_l.h* a *convolution.h.* Pro konvoluční část je vytvořená struktura „conv_type“, která obsahuje konvoluční jádro. To je čtvercová matice o velikosti pět s přidaným normovacím koeficientem.

Modul si vytváří dvě paměťová pole pro mezi výpočty „comBuffer0“ a „comBuffer1“. Ústřední částí modulu je stavový automat, který podle aktuálního nastavení ve struktuře „acc_type“ rozhoduje o dalším provedeném kopírování dat. Automat má tři stavy ve kterých vždy zkopíruje data mezi dvěma poli za použití jedné části IP jádra ACC. První stav je kopírování podle volby rozmazání obrazu pro odstranění šumu „gaus_blur“. Druhým stavem je kopírování podle zvoleného konvolučního jádra „custom_kernel“, pokud je tato možnost zakázána, je tento stav přeskočen. Třetím stavem je obarvení obrazu podle zvolených parametrů „coloring_dynamic“.

Po každém kopírování je obslouženo přerušení „Transfer_interrupt“, při kterém se provede další krok stavového automatu. Tento cyklus lze spustit IOCTL makrem „ACC_RUN“ nebo zastavit makrem „ACC_STOP. Před spuštěním kopírování je potřeba nastavit zdrojovou a cílovou adresu paměťových polí s obrazovými daty.

Tab. 8 IOCTL makra ACC modulu

Název makra	Typ parametru	Popis funkce
ACC_SET_ADRESS_DEST	Číslo	Cílová adresa
ACC_SET_ADRESS_SOURCE	Číslo	Zdrojová adresa
ACC_RUN	Číslo	Zahájení kopírování
ACC_STOP	Číslo	Zastavení kopírování
ACC_GAUSS_BLUR_ENABLE	Číslo	0 – zakáže rozostření 1 – povolí rozostření Jinak vrací hodnotu
ACC_COLORING_DYNAMIC_ENABLE	Číslo	0 – statické obarvování 1 – dynamické obarvování Jinak vrací hodnotu
ACC_COLORING_STATIC_MIN	Číslo	Minimální hodnota obarvování
ACC_COLORING_STATIC_MAX	Číslo	Maximální hodnota obarvování
ACC_GRAY_DYNAMIC_ENABLE	Číslo	0 – statické odstíny šedi 1 – dynamické odstíny šedi Jinak vrací hodnotu
ACC_GRAY_STATIC_MIN	Číslo	Minimální hodnota šedi
ACC_GRAY_STATIC_MAX	Číslo	Maximální hodnota šedi
ACC_INTENSIVE_BLUR	Číslo	0 – zakáže větší rozostření 1 – povolí větší rozostření Jinak vrací hodnotu
ACC_CUSTOM_KERNEL	Číslo	0 – zakáže konvoluci 1 – povolí konvoluci Jinak vrací hodnotu
ACC_SET_KERNEL_TRANSFER	Conv_type	Nastaví konvoluční jádro
ACC_GET_KERNEL_TRANSFER	Conv_type	Vrátí konvoluční jádro

3.2 Testování modulů

Syntaktické chyby či varování jsou nalezeny překladačem při překladu. Chyby inicializace jsou odhaleny při vkládání modulu do jádra a končí většinou kritickou chybou. Před dalším pokusem je nutný restart systému.

Pro zjištění vadné části kódu je vhodné proložit ho výpisy zajímavých hodnot či jen „oddělení“ jednotlivých úseků. K tomu slouží funkce „printk“. Ta obsahuje i různé

příznaky typu zprávy, ale to je v našem případě zbytečné. Pokud jsou použité funkce schopné vracet informaci, zda proběhly úspěšně, je možné tyto chyby odchytit a včas korektně modul ukončit. Takto ošetřené bývají již kompletní odzkoušené ovladače a pro nahlášení takto zjištěné chyby využívají makra „dev_err“.

Pro ladění jednotlivé funkce je vhodné provést její implementaci do funkce „read“ nebo „open“. Tu pak jednoduše spustíme příkazem „cat /dev/jmeno_modulu“. Funkčnost uvidíme záhy. Zejména pokud se budeme snažit zapisovat data na špatné adresy v paměti.

3.2.1 Detekce periferií

Po nabootování systému by měly být ve složce „/dev“ k dispozici obslužné soubory pro jednotlivá zařízení. Konkrétně dva pro obsluhu I^2C s názvy *i2c-0* a *i2c-1*, jedno SPI zařízení a po implementaci všech modulů příkazem „insmod“ i soubory *CcDevice*, *ScDevice*, *MlvDevice*, *AccDevice*, *LenDevice*, *ShuDevice*. Obsah SD karty je v souboru *mmcblk0p1*. Ten můžeme zpřístupnit skrz složku „/mnt“ příkazem „mount /dev/mmcblk0p1 /mnt“. Poté můžeme bezpečně odebrat SD kartu příkazem „umount /mnt“.

3.3 Komunikace s moduly

Každý modul má zvláštní soubor, ve kterém má definované podporované IOCTL funkce. Každá funkce má pomocí makra generované své unikátní číslo „ioctl_makro“ a obsahuje zmínku o typu přenášeného parametru „ioctl_param“. Celkem rozeznávám tři typy hodnot parametru: jednobitové číslo, velké číslo a adresu na pole hodnot.

Jednobitové číslo je reprezentované datovým typem „int“ a jeho validní hodnoty jsou nula a jedna. Při jiné hodnotě parametru je příkaz chápán jako dotaz a vrací aktuálně nastavenou hodnotu. Ta je uložena v příslušné struktuře obsluhující IP jádro. Příkazy „set_param“ a „get_param“ se postarají, aby byla hodnota náležitě aktualizována v registrech IP jádra a aktuální hodnota uložena do příslušné struktury. Následuje ukázka obslužného kódu.

```
case ioctl_makro:
     switch(ioctl_param) {
     case 0: set_param(&struktura, 0); break;
     case 1: set_param(&struktura, 1); break;
     default: return get_param(&struktura); }
break;
```

Dalším typem parametru je velké číslo, většinou šestnáctibitové. To je reprezentováno datovým typem „int“. Většinou je u funkcí pracující s tímto typem parametru možná maximální nastavitelná hodnota „PARAM_MAX“. Pokud je hodnota parametru větší než toto maximum, je příkaz chápán jako dotaz a vrací aktuálně nastavenou hodnotu. Pokud je číslo využito v celém svém rozsahu je nutné mít dvě IOCTL funkce. Jednu pro uložení hodnoty a druhou pro její čtení. Funkce pro čtení pak ignoruje předávanou hodnotu parametru. Následuje ukázka obslužného kódu s kontrolou maximální hodnoty.

```
case ioctl_makro:
     if(ioctl_param<=PARAM_MAX)
          set_param(&struktura,ioctl_param);
     else return get_param(&struktura);
break;
```

Posledním typem parametru je adresa na pole hodnot. Hodnoty pole jsou datového typu „char“, tj. jednobajtové číslo. Tento typ parametru se používá pro přenos objemnějších dat jako je obraz, korekční matice či konvoluční jádro. Protože nás zajímá přenos všech hodnot pole a nikoli jen jeho adresy, je potřeba jej „ručně“ zkopírovat z uživatelského paměťového prostoru do paměťového prostoru jádra či naopak. Pro kopírování celých polí využívám funkci „copy_from_user“. Ta přenese data o velikosti „velikost_budice“ z uživatelského prostoru na adrese „ioctl_param“ do paměti jádra adresované „adresa_budice“. Pro obrácený směr je funkce „copy_to_user“. Následuje ukázka obslužného kódu, velikost paměťového pole je přesně daná pro každou funkci.

```
case ioctl_makro:
     copy_from_user(adresa_budice, (char *) ioctl_param,
          velikost_budice);
break;
```

4 Tvorba řídícího software

Aplikaci píšu ve vývojovém studiu SDK. Při tvorbě aplikace se musí nastavit v „Target Software“ parametr „OS Platform“ na „linux“. Pro komunikaci s ovladačem poslouží soubor *sys/ioctl.c*, pro otevírání souborů slouží *fcntl.h* a pro TCP/IP komunikaci přes ethernet jsou *sys/socket.h, arpa/inet.h, netinet/in.h.*

Kamerový systém komunikuje po třech portech. Na jednom běží nastavovací rozhraní a zbylé dva jsou pro streamování obrazu. Protože se mi nepodařilo v rámci jedné aplikace úspěšně vytvářet nová vlákna, vytvořil jsem pro každý port zvláštní aplikaci. Pro nastavování parametrů kamery mám „komunikační server“ a pro video streamy „streamovací server“. Pro nastavení požadované funkce všech modulů v kamerovém systému jsem vytvořil dávkovou inicializační aplikaci.

Pro zachování kompatibility s vyvíjeným řídícím software pro prototyp kamerového systému jsem implementoval v něm použité komunikační rozhraní. To definuje přenos pole dat a dekódování jednotlivých příkazů a jejich odpovědi. Rámec zprávy je definován v souboru *transfer_buffer.h*. Přenosové limity a možné příkazy jsou definovány v souboru *cam_server.h.*

Do své aplikace jsem implementoval možné příkazy použité v nejnovější verzi kamerového systému. Díky tomu mohu komunikaci testovat s nejnovější verzí ovládacího softwaru, který poskytuje lepší zpracování dat. Přidané cílové komponenty a jejich možné příkazy zde nebudu uvádět, protože vrací pouze základní nulové hodnoty.

4.1 Nastavení kamerového systému

Požadované nastavení modulů zajistí aplikace *driverInit.elf*. Ta si vyčte z modulu CC jedeno jeho výstupní pole a nastaví ho jako jedeno ze vstupních polí pro modul MLV. Poté nastaví modul ACC. Ten bude kopírovat data z jednoho výstupního pole modulu SC do druhého vstupního pole modulu MLV. Pak nastaví výchozí hodnoty barvícího režimu a spustí kopírování. Poté se program ukončí.

4.2 Komunikace po TCP/IP

Vytvoření serveru a připojení klienta je umožněno použitím struktury „sockaddr_in“ a obslužné funkce připojení. Průběh inicializace serveru začíná zažádáním si o číslo soketu funkcí „socket“. Ten se propojí funkcí „bind“ s vyplněnou strukturou serveru, ta obsahuje povolené adresy a port. Poslouchání na daném portu se zapne funkcí „listen“. Poté se ve

smyčce volá funkce „accept“. Ta čeká na připojení klienta. Až se klient připojí, přiřadí se mu obslužná funkce.

Posílání zpráv je možné přes funkce „send“ a „recv“. Posílaná nebo přijímaná zpráva se plní do zvláštního pole „send_buffer“ či „command_buffer“. Ty mají přesně určenou velikost. Protože zprávy neobsahují ukončovací znaky, byla by problematická detekce různě velkých zpráv. Pro odesílání dat větších než velikost pole „send_buffer“ slouží funkce „mySend“. Ta odesílá celé pole po kouskách a skončí až po odeslání posledního bajtu pole.

4.2.1 Komunikační server

Komunikace s připojeným klientem začíná jeho první odeslanou zprávou. Ta se odešle na vyhodnocení funkci „camsrv_process_request“. Po jejím zpracování se cyklicky čte další zpráva.

Získanými daty ze zprávy se plní struktura „camsrv_command_type“. Základní typy datové zprávy začínají znaky „GET“ nebo „SET“. Pokud je zpráva obsahuje, vyhodnocuje se cílová komponenta. Tou může být bolometr „BOL“, kamera „CAM“, PWM modul „LEN“, paměťová karta „SDC“ nebo všeobecné nastavení „ACC“. Každá z cílových komponent má svoje parametry s unikátním třímístným kódem. U zpráv typu „SET“ náleží zbylý prostor zprávy pro číselný parametr.

4.2.2 Streamovací server

Každý klient, který se připojí na streamovací server, obdrží zprávu o velikosti posílaného objemu dat, tj. velikost obrázku navýšený o jedničku. Poté, pokud je povoleno streamování dat, jsou odesílány jednotlivé snímky klientovi. Při povoleném streamování se na konec snímku přidá bajt o hodnotě jedna. To signalizuje návaznost posílání dalšího snímku. Pokud je streamování zakázáno, je na konci aktuálního posílaného snímku doplněna nulová hodnota.

Při zakázání probíhá v sekundových intervalech kontrola obnovy povolení. Pokud je streamování obnoveno, znovu se odešle zpráva o velikosti posílaných dat. Poté cyklicky probíhá odesílání dat, dokud se klient neodpojí nebo nebude streamování opět zastaveno. Snímky se vyčítají buď z modulu SC, nebo CC. Je možné streamovat i obrazy z modulu MLV.

4.3 Zisk dat z modulů

Zisk dat z modulů provádím pomocí IOCTL maker, které jsou definované v souborech *CcIoctl.h, ScIoctl.h, MlvIoctl.h, AccIoctl.h, PwmIoctl.h*. Dále si pak definuji v souboru *myIoctl.h* obslužné funkce pro jednotlivý modul. Jedná se o funkce pro uložení čísla do modulu „set_int“, pro čtení čísla z modulu „get_int“, pro odeslání pole do modulu „set_file“ a pro čtení pole z modulu „get_file“. Funkce typu „set“ vrací při úspěšném nastavení hodnotu nula.

Protože obslužný soubor modulu může být naráz otevřen pouze jednou, je součástí mých funkcí čekací smyčka pro jeho otevření. K otevření souboru používám funkci „open“ ze souboru *ioctl.h*. Po otevření souboru se pomocí funkce „ioctl“ předává souboru číslo vygenerované IOCTL makrem a příslušný parametr. Tato funkce je dostupná v souboru sys/*ioctl.h.* Po přijmutí návratové hodnoty se soubor zavře funkcí „close“.

Pro předávání číselných hodnot datového typu „int“ je funkce „set_int“ jednoduchou implementací funkce „ioctl“. Opačná funkce „get_int“ posílá v parametru maximální hodnotu 32bit čísla. To odpovídající logika vyhodnocení parametru funkce v modulu pochopí jako žádost o aktuální nastavenou hodnotu.

Funkce pro předávání celých polí předává parametrem „ioctl_param“ adresu na dané pole. Pokud je pole předáváno do modulu funkcí „set_file“, předpokládá se již jeho existence. Tudíž předání jeho adresy není problém, musí se dbát na dostatečnou velikost pole. Nesmí se pochopitelně prohazovat korekční a konvoluční matice a podobně. Pro opačnou funkci „get_file“ je požadován parametr očekávané velikosti dat z modulu. Funkce si vytvoří dostatečně velké pole pro příjem dat z modulu. Poté se pracuje s adresou tohoto pole.

4.4 Spuštění na pozadí

Aby aplikace běžela na pozadí, stačí připsat při jejím spouštění parametr „&“. To nám umožní nadále pracovat v systému. Aplikace ale i nadále reaguje na všechny klasické podněty z kontrolního terminálu a komunikuje s ním.

Aby aplikace byla opravdu službou a dala se označit jako „daemon“, je potřeba tuto komunikaci ukončit a přesměrovat výstup „log“ aplikace do souboru. Abychom toho dosáhly, musíme zjistit přiřazené číslo procesu spuštěné aplikace (PID). K tomu je funkce „fork“ ze souboru *unistd.h.* Ta zduplikuje volající proces a přiřadí ho jako podproces

volanému procesu. Tím získáme dva stejné procesy, z nichž jeden je podprocesem toho druhého. Když nyní ukončíme první spuštěný proces, ten druhý zdvojený bude pracovat na pozadí.

Pro oddělení procesu od terminálu, ze kterého byl vyvolán, použijeme funkci „setsid“. Změníme nastavení práv nově vytvořených souborů funkcí „umask(0)“ a změníme pracovní složku funkcí „chdir(„/“)“. Zavřeme standardní popisovače „STDIN“, „STDOUT“ a „STDERR“ funkcí „close(popisovac_FILENO). Tím jsme vytvořily samostatně běžící aplikaci.

Dále můžeme použít funkcí ze souboru „syslog.h“ pro vytvoření souboru pro logování zpráv pomocí funkcí „openlog“ a „closelog“. Pro zápis do tohoto souboru pak slouží funkce „syslog“.

4.5 Podporované příkazy

Kamerový systém se skládá z mnoha různých komponent. V příloze B uvádím příkazy obslužných funkcí komponent dostupných v prototypové verzi. Z nich je zatím vynechána implementace práce s SD kartou, protože ke své obsluze nepotřebuje žádný z vytvořených modulů. Přístup na ní a práce se soubory jsou standardní funkce operačního systému a v práci již bylo využito některých jejich základních implementací.

4.6 Testování aplikace

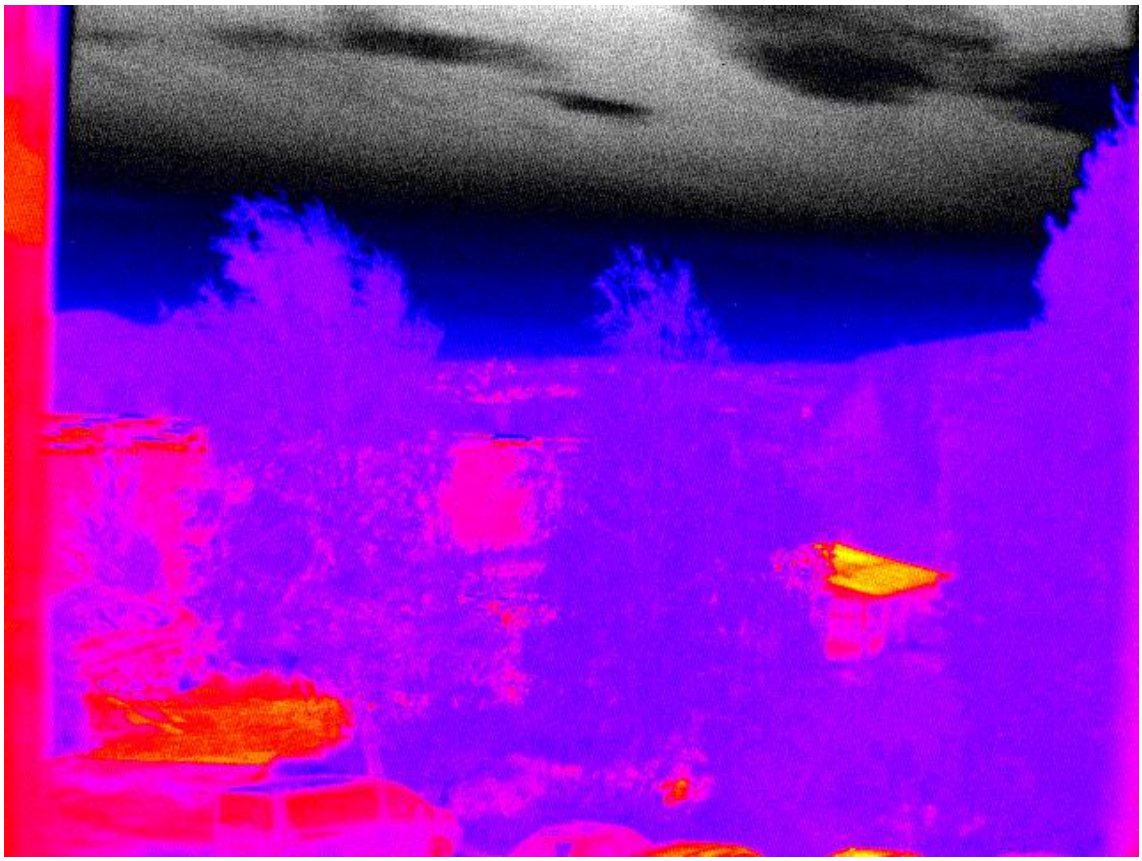
Kamerový systém používá gigabitové připojení a reálně jsem ho využil asi na 40 %. Tento údaj je závislý na klientském počítači (jeho výkonu a vytížení). Pro diagnostiku vytížení systému na desce jsem použil příkaz „top“. Z něj je patrné vytížení procesoru jednotlivou aplikací. Při jednom aktivním streamu je vytížen jeden procesor až na 50 % a výsledná rychlost obnovy obrázku v aplikaci je 60 fps. To je, při rychlosti firmwarového řešení 50 fps, zlepšení o 20 %. Pokud jsou aktivní oba streamy je dosažená rychlost obnovy obrázku každého z nich 30 fps. To je, při rychlosti firmwarového řešení 25 fps, také zlepšení o 20%. Vytíženy jsou tentokrát oba procesory, každý na 30 %.

Funkci barvícího a konvolučního modulu IP jádra ACC, lze vidět na připojeném monitoru k desce nebo pomocí uživatelské aplikace. Výstupy pro jednotlivá nastavení modulu ACC s provedenou korekcí obrazu a pohledem z okna jsou na následujících obrázcích.

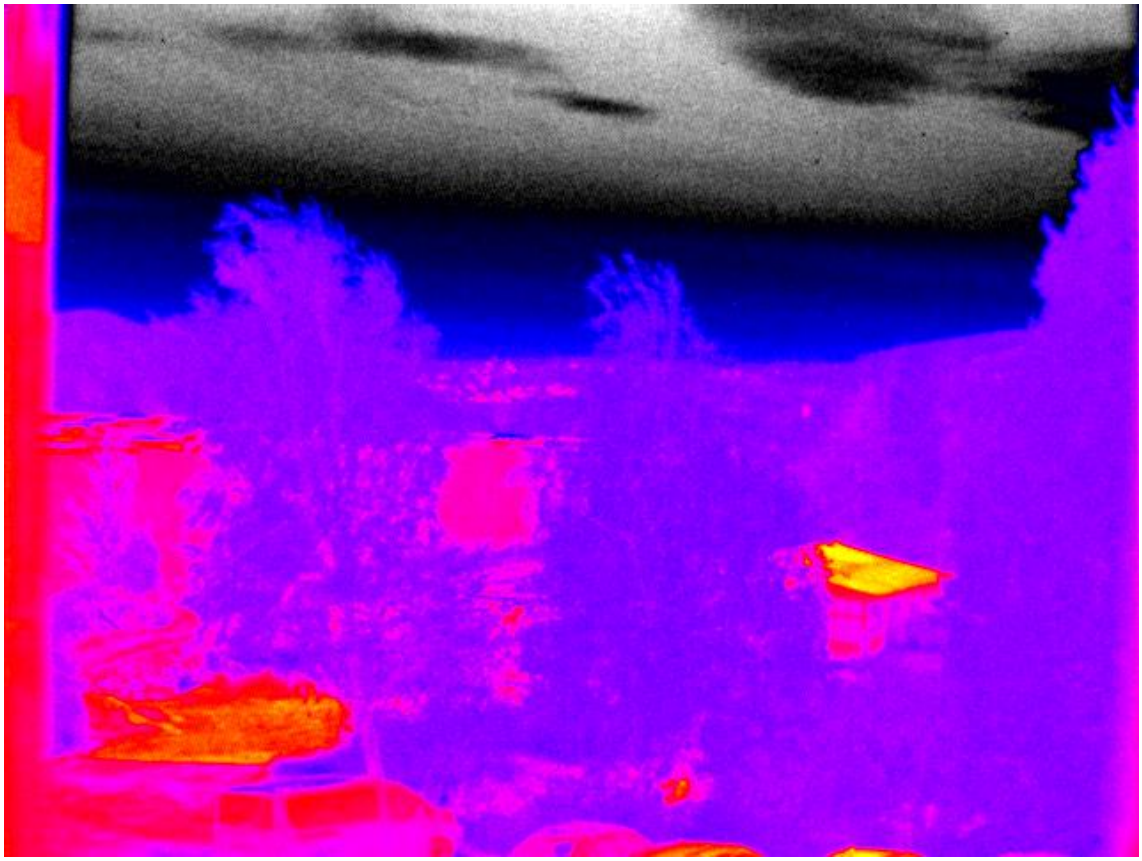
Obraz z barevné VGA kamery je na Obr. 12 a skýtá pohled z okna na chatku, továrnu a horizont za slunného dne. Ostatní obrázky skýtají stejný pohled, ale v infračerveném

spektru. Jsou tvořena daty získanými z bolometru a zobrazena stávající uživatelskou aplikací. Na Obr. 13 jsou data po korekci a je krásně vidět rozpálená střecha chatky nebo střešní okna továrny. V obraze je stále přítomen šum a proto se aplikují rozmazávací funkce. Na Obr. 14 je aplikované rozmazání obrazu a na Obr. 15 jeho intenzivnější verze. Jak obraz vypadá s aktivním obarvováním, je ukázáno na Obr. 16. Obecně jsou teplejší objekty označeny žlutou barvou a chladnější červenou. Aplikace obarvuje nechladnější místa bílou barvou.

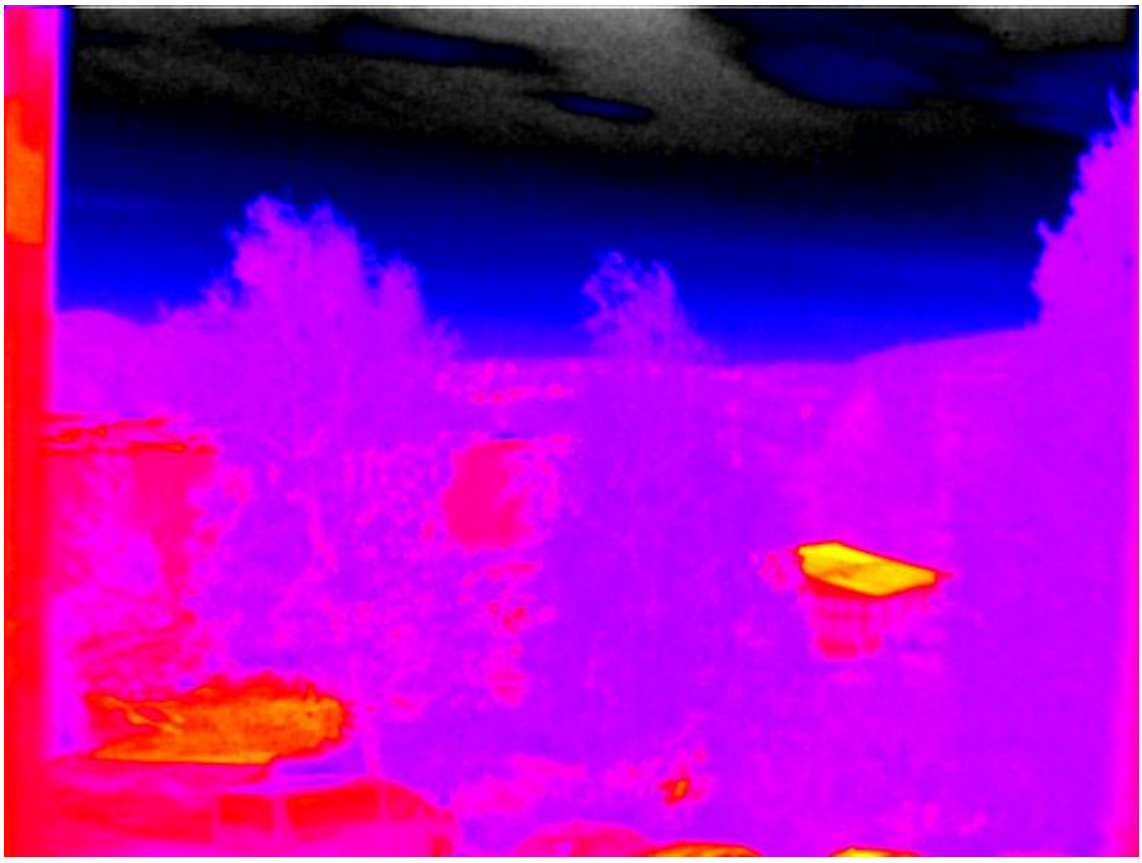
Obr. 12 Obraz z VGA kamery



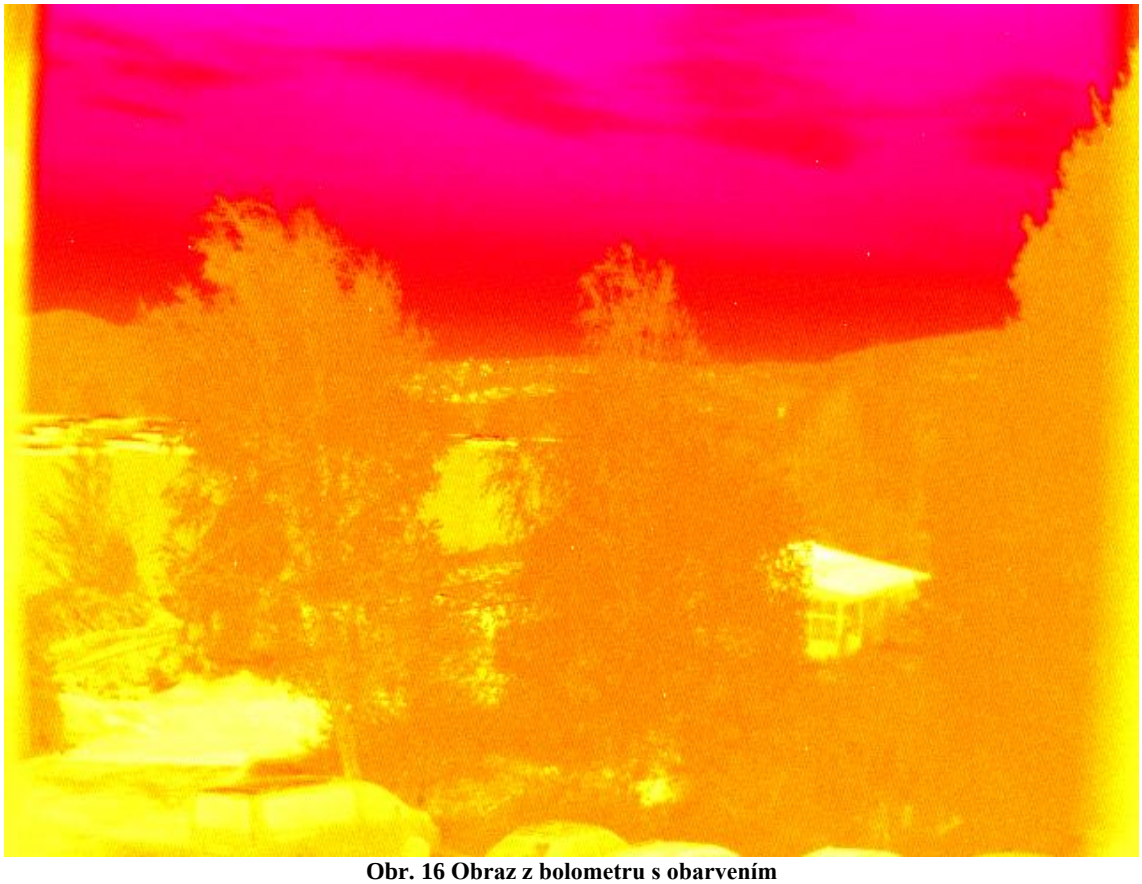
Obr. 13 Obraz z bolometru bez rozmazání



Obr. 14 Obraz z bolomoteru s rozmazáním



Obr. 15 Obraz z bolomoteru s větším rozmazáním



Obr. 16 Obraz z bolometru s obarvením

5 Závěr

Realizoval jsem ovládání kamerového systému pomocí OS Linux v distribuci PetaLinux, která má podporu produktů od firmy Xilinx. Veškeré potřebné soubory jsou na SD kartě, ze které celý systém bootuje. Kořenové složky operačního systému jsou v předem vytvořeném obrazu *ramdisk.image.gz*, který se po spuštění nahraje do RAM paměti systému. Provedené změny nastavení systému a soubory uložené jinam než na SD kartu jsou tudíž po restartu ztraceny.

Vytvořil jsem ovladače ve formě modulů pro dodaná IP jádra a zprovoznil komunikaci mezi použitými komponentami. Jedná se o komunikaci po sběrnicích I^2C, SPI, mezi moduly samotnými a mezi moduly a aplikací. Vše jsem nejdříve otestoval v rámci jednoho velkého modulu, jehož zdrojový kód je v souboru *nulD.c*. Pro komunikaci po I^2C jsem zkoušel jak IP jádra v PS tak jejich obdobu v PL. Pouze u systémového I^2C z PS se mi podařilo zprovoznit režim komunikace přes SCCB protokol. Taktéž jsem zkoušel IP jádra SPI z PS a PL, přičemž pouze ty z PL jsem zprovoznil v 16bit režimu, který je potřeba pro komunikaci s bolometrickým čipem.

Každý modul, který pracuje s obrazovými daty, má vstupní či výstupní pole, které je reprezentované adresou a má velikost 640×480×4 bajtů. Mezi těmito poli systém realizuje obrazové funkce zisku, zobrazení, přesunu, konvoluce a obarvení. Z koncových komponent SC, CC a MLV lze nastavit streamování obrazu. Komponenta ACC je použita pro zpracování obrazu ze SC do MLV a komponenta PWM ovládá záklopku pro tvorbu korekce dat z bolometru. Ostatní komponenty nemají přiřazenou žádnou funkci.

Napsal jsem aplikace pro komunikaci po TCP/IP na ovládání a zisk dat z kamerového systému. Původně bylo v plánu vytvořit jednu více vláknovou aplikaci, ale nezdařila se mi implementace funkcí „pthread“. V základu se jedná o tři servery a jednu aplikaci pro počáteční nastavení modulů. Každý server poslouchá na jednom portu Ethernetového rozhraní desky a slouží buď pro vzdálené nastavení systému, nebo pro streamování dat obrazu. Výsledkem jsou dva „Streamovací servery“ pro stream dat z bolometru a VGA kamery a jeden „Komunikační server“, který je kompatibilní s nejnovější uživatelskou aplikací.

Dominantou celého systému jsou Streamovací servery, které jsou optimalizované pro přenos obrazových dat do uživatelské aplikace. Při porovnání rychlosti sběru dat pracuje nový systém o 20% rychleji než původní a k tomu využívá jen třetinu svého

procesorového výkonu. Většina tohoto výkonu je využita pro kopírování dat mezi paměťovými prostory uživatele a jádra systému.

Zprovoznit jsem zkoušel i jiné zmíněné distribuce, z nichž zajímavější je Linaro 12. Využívám pro něj jiného zapojení BSP a jeho „rootfs“ mám na zvláštním oddílu SD karty. Pro využití grafického výstupu je nutné vytvořit zařízení „framebuffer“, v našem případě by se jednalo o vytvoření jiného ovladače IP jádra MLV. Díky tomu je možné pracovat s deskou jako klasickým desktopovým počítačem. Bohužel je poté většina výkonu spotřebována na grafickou obsluhu.

Výhoda modularity systému se projeví v možnosti testování systému po částech, což vede k zlepšení zjišťování chyb a k jejich rychlejší nápravě. Dále umožňuje nové kombinace propojení jednotlivých části zpracování obrazu a tím rozšiřuje možné zdroje dat. Je možné například zdvojit IP jádro ACC a vytvořit tak paralelní nebo sériové struktury pro zpracování a korekci dat.

Vytvořený funkční systém budu dál vyvíjet již na nejnovější verzi kamerového systému, která má více zařízení k obsluze. Funkcionalita systému bude uzpůsobena pro zisk všech možných dat ze systému a jeho variabilnímu nastavení, což většina komerčních řešení neumožňuje. Experimentovat budu i s použitím některého průmyslového komunikačního protokolu.

Literatura

[1] I. Xilinx, „zedboard book source," Mentor Graphics, [Online]. Available: https://code.google.com/p/zedboard-book-source/downloads/detail?name=xilinx-2011.09-50-arm-xilinx-linux-gnueabi.bin&can=2&q=. [Přístup získán 2015].

[2] I. Xilinx, „Xilinx repository," GitHub Inc., [Online]. Available: https://github.com/xilinx. [Přístup získán 2015].

[3] I. OmniVision Technologies, „OmniVision Serial Camera Control Bus (SCCB) Functional Specification," OmniVision Technologies, Inc., 2007. [Online]. Available: http://www.ovt.com/download_document.php?type=document&DID=63. [Přístup získán 2015].

[4] I. B. Moshe Gavrielov, „ISE Design Suite," Xilinx Inc., [Online]. Available: http://www.xilinx.com. [Přístup získán 2015].

[5] Avnet, „www.zedboard.org," 2015. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.

[6] Xilinx, Inc, „Zynq-7000 All Programmable SoC Technical Reference Manual," [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. [Přístup získán 2015].

[7] Pre-built Zynq Linux Image, „Zynq 2014.4 Release," Xilinx, Inc., [Online]. Available: http://www.wiki.xilinx.com/Zynq+2014.4+Release. [Přístup získán 2015].

[8] The VirtualBox team, Oracle, [Online]. Available: https://www.virtualbox.org. [Přístup získán 2015].

[9] Leann Ogasawara, Jason Warner, Canonical Ltd, [Online]. Available: http://www.ubuntu.cz. [Přístup získán 2015].

[10] Xilinx, Inc., „PetaLinux Tools," [Online]. Available: http://www.xilinx.com/tools/petalinux-sdk.htm. [Přístup získán 2015].

[11] Mark Brown, Deepak Saxena, Vicky Janicki, „Linaro Releases," Linaro, [Online]. Available: http://releases.linaro.org/12.09/ubuntu/precise-images/ubuntu-desktop. [Přístup získán 2015].

[12] Ing. Vladimír Hampl, „Hyperspektrální kamera," Applic s.r.o, [Online]. Available: http://www.applic.cz. [Přístup získán 2015].

[13] Clint Cole, Gene Apperson, „Digilent Embedded Linux," Digilent Inc. , [Online]. Available: https://digilentinc.com/Products/Detail.cfm?NavPath=2,66,1143&Prod=EMBEDDED-LINUX. [Přístup získán 2015].

[14] Karanbir Singh, Johnny Hughes, Tru Huynh, „CentOS 7 Updates," [Online]. Available: https://www.centos.org. [Přístup získán 2015].

[15] Kevin Mihelich, Jason Plum, Mike Brown, David Higham, [Online]. Available: http://archlinuxarm.org. [Přístup získán 2015].

Příloha A: Device Tree

```
/dts-v1/;/ {
#address-cells = <0x1>;
#size-cells = <0x1>;
compatible = "xlnx,zynq-zed", "xlnx,zynq-7000";
model = "Zynq Zed Development Board";
chosen {
      bootargs = "console=ttyPS0,115200 root=/dev/ram rw earlyprintk";
      linux,stdout-path = "/amba/serial@e0001000";};
memory {    device_type = "memory";
            reg = <0x0 0x20000000>;};
cpus {#address-cells = <0x1>;
      #size-cells = <0x0>;
      cpu@0{compatible = "arm,cortex-a9";
            device_type = "cpu";
            reg = <0x0>;
            clocks = <0x1 0x3>;
            clock-latency = <0x3e8>;
            cpu0-supply = <0x2>;
            operating-points = <0xa2c2b 0xf4240 0x51616 0xf4240>;};
      cpu@1{compatible = "arm,cortex-a9";
            device_type = "cpu";
            reg = <0x1>;
            clocks = <0x1 0x3>;};};
pmu { compatible = "arm,cortex-a9-pmu";
      interrupts = <0x0 0x5 0x4 0x0 0x6 0x4>;
      interrupt-parent = <0x3>;
      reg = <0xf8891000 0x1000 0xf8893000 0x1000>;};
fixedregulator@0 {
      compatible = "regulator-fixed";
      regulator-name = "VCCPINT";
      regulator-min-microvolt = <0xf4240>;
      regulator-max-microvolt = <0xf4240>;
      regulator-boot-on;
      regulator-always-on;
      linux,phandle = <0x2>;
      phandle = <0x2>;};
amba {compatible = "simple-bus";
      #address-cells = <0x1>;
      #size-cells = <0x1>;
      interrupt-parent = <0x3>;
      ranges;
      adc@f8007100 {
            compatible = "xlnx,zynq-xadc-1.00.a";
            reg = <0xf8007100 0x20>;
            interrupts = <0x0 0x7 0x4>;
            interrupt-parent = <0x3>;
            clocks = <0x1 0xc>;};
      gpio@e000a000 {
            compatible = "xlnx,zynq-gpio-1.0";
            #gpio-cells = <0x2>;
            clocks = <0x1 0x2a>;
            gpio-controller;
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x14 0x4>;
            reg = <0xe000a000 0x1000>;};
      i2c@e0004000 {
            compatible = "cdns,i2c-r1p10";
            status = "okay";
            clocks = <0x1 0x26>;
```

```
		interrupt-parent = <0x3>;
		interrupts = <0x0 0x19 0x4>;
		reg = <0xe0004000 0x1000>;
		#address-cells = <0x1>;
		#size-cells = <0x0>;
		clock-frequency = <0x61a80>;
		i2c-reset = <0x4 0xd 0x0>;};
	i2c@e0005000 {
		compatible = "cdns,i2c-r1p10";
		status = "okay";
		clocks = <0x1 0x27>;
		interrupt-parent = <0x3>;
		interrupts = <0x0 0x30 0x4>;
		reg = <0xe0005000 0x1000>;
		#address-cells = <0x1>;
		#size-cells = <0x0>;
		clock-frequency = <0x61a80>;};
	interrupt-controller@f8f01000 {
		compatible = "arm,cortex-a9-gic";
		#interrupt-cells = <0x3>;
		interrupt-controller;
		reg = <0xf8f01000 0x1000 0xf8f00100 0x100>;
		linux,phandle = <0x3>;
		phandle = <0x3>;};
	cache-controller@f8f02000 {
		compatible = "arm,pl310-cache";
		reg = <0xf8f02000 0x1000>;
		arm,data-latency = <0x3 0x2 0x2>;
		arm,tag-latency = <0x2 0x2 0x2>;
		cache-unified;
		cache-level = <0x2>;};
	memory-controller@f8006000 {
		compatible = "xlnx,zynq-ddrc-1.0";
		reg = <0xf8006000 0x1000>;
		xlnx,has-ecc = <0x0>;};
	ocmc@f800c000 {
		compatible = "xlnx,zynq-ocmc-1.0";
		interrupt-parent = <0x3>;
		interrupts = <0x0 0x3 0x4>;
		reg = <0xf800c000 0x1000>;};
	serial@e0001000 {
		compatible = "xlnx,xuartps", "cdns,uart-r1p8";
		status = "okay";
		clocks = <0x1 0x18 0x1 0x29>;
		clock-names = "uart_clk", "pclk";
		reg = <0xe0001000 0x1000>;
		interrupts = <0x0 0x32 0x4>;
		current-speed = <0x1c200>;
		device_type = "serial";
		port-number = <0x0>;};
	spi@42000000 {
		compatible = "xlnx,xps-spi-2.00.a";
		clock-names = "ref_clk", "pclk";
		clocks = <0x1 0x19 0x1 0x22>;
		interrupt-parent = <0x3>;
		interrupts = <0x0 0x21 0x4>;
		reg = <0x42000000 0x10000>;
		xlnx,fifo-exist = <0x1>;
		xlnx,instance = "sensor_spi";
		xlnx,num-ss-bits = <0x1>;
		xlnx,num-transfer-bits = <0x10>;
```

```
            xlnx,sck-ratio = <0x20>;
            #address-cells = <0x1>;
            #size-cells = <0x0>;
            num-cs = <0x1>;
            is-decoded-cs = <0x0>;};
      ethernet@e000b000 {
            compatible = "xlnx,ps7-ethernet-1.00.a";
            reg = <0xe000b000 0x1000>;
            status = "okay";
            interrupts = <0x0 0x16 0x4>;
            clocks = <0x1 0xd 0x1 0x1e>;
            clock-names = "ref_clk", "aper_clk";
            local-mac-address = [00 0a 35 00 00 00];
            xlnx,has-mdio = <0x1>;
            #address-cells = <0x1>;
            #size-cells = <0x0>;
            phy-mode = "rgmii-id";
            phy-handle = <0x4>;
            phy@0{reg = <0x0>;
                  linux,phandle = <0x4>;
                  phandle = <0x4>;};};
      sdhci@e0100000 {
            compatible = "arasan,sdhci-8.9a";
            status = "okay";
            clock-names = "clk_xin", "clk_ahb";
            clocks = <0x1 0x15 0x1 0x20>;
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x18 0x4>;
            reg = <0xe0100000 0x1000>;
            clock-frequency = <0x2faf080>;};
      slcr@f8000000 {
            #address-cells = <0x1>;
            #size-cells = <0x1>;
            compatible = "xlnx,zynq-slcr", "syscon";
            reg = <0xf8000000 0x1000>;
            ranges;
            clkc@100 {
                  #clock-cells = <0x1>;
                  compatible = "xlnx,ps7-clkc";
                  ps-clk-frequency = <0x1fca055>;
                  fclk-enable = <0xf>;
                  clock-output-names = "armpll", "ddrpll", "iopll",
"cpu_6or4x", "cpu_3or2x", "cpu_2x", "cpu_1x", "ddr2x", "ddr3x", "dci",
"lqspi", "smc", "pcap", "gem0", "gem1", "fclk0", "fclk1", "fclk2",
"fclk3", "can0", "can1", "sdio0", "sdio1", "uart0", "uart1", "spi0",
"spi1", "dma", "usb0_aper", "usb1_aper", "gem0_aper", "gem1_aper",
"sdio0_aper", "sdio1_aper", "spi0_aper", "spi1_aper", "can0_aper",
"can1_aper", "i2c0_aper", "i2c1_aper", "uart0_aper", "uart1_aper",
"gpio_aper", "lqspi_aper", "smc_aper", "swdt", "dbg_trc", "dbg_apb";
                  reg = <0x100 0x100>;
                  linux,phandle = <0x1>;
                  phandle = <0x1>;};};
      dmac@f8003000 {
            compatible = "arm,pl330", "arm,primecell";
            reg = <0xf8003000 0x1000>;
            interrupt-parent = <0x3>;
            interrupt-names = "abort", "dma0", "dma1", "dma2", "dma3",
"dma4", "dma5", "dma6", "dma7";
            interrupts = <0x0 0xd 0x4 0x0 0xe 0x4 0x0 0xf 0x4 0x0 0x10
0x4 0x0 0x11 0x4 0x0 0x28 0x4 0x0 0x29 0x4 0x0 0x2a 0x4 0x0 0x2b 0x4>;
            #dma-cells = <0x1>;
```

```
            #dma-channels = <0x8>;
            #dma-requests = <0x4>;
            clocks = <0x1 0x1b>;
            clock-names = "apb_pclk";};
      devcfg@f8007000 {
            clock-names = "ref_clk", "fclk0", "fclk1", "fclk2",
"fclk3";
            clocks = <0x1 0xc 0x1 0xf 0x1 0x10 0x1 0x11 0x1 0x12>;
            compatible = "xlnx,zynq-devcfg-1.0";
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x8 0x4>;
            reg = <0xf8007000 0x100>;};
      timer@f8f00200 {
            compatible = "arm,cortex-a9-global-timer";
            reg = <0xf8f00200 0x20>;
            interrupts = <0x1 0xb 0x301>;
            interrupt-parent = <0x3>;
            clocks = <0x1 0x4>;};
      timer@f8001000 {
            interrupt-parent = <0x3>;
            interrupts = <0x0 0xa 0x4 0x0 0xb 0x4 0x0 0xc 0x4>;
            compatible = "cdns,ttc";
            clocks = <0x1 0x6>;
            reg = <0xf8001000 0x1000>;};
      timer@f8002000 {
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x25 0x4 0x0 0x26 0x4 0x0 0x27 0x4>;
            compatible = "cdns,ttc";
            clocks = <0x1 0x6>;
            reg = <0xf8002000 0x1000>;};
      timer@f8f00600 {
            interrupt-parent = <0x3>;
            interrupts = <0x1 0xd 0x301>;
            compatible = "arm,cortex-a9-twd-timer";
            reg = <0xf8f00600 0x20>;
            clocks = <0x1 0x4>;};
      watchdog@f8005000 {
            clocks = <0x1 0x2d>;
            compatible = "xlnx,zynq-wdt-r1p2";
            device_type = "watchdog";
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x9 0x1>;
            reg = <0xf8005000 0x1000>;
            reset = <0x0>;
            timeout-sec = <0xa>;};
      usb@e0002000 {
            clocks = <0x1 0x1c>;
            compatible = "xlnx,ps7-usb-1.00.a", "xlnx,zynq-usb-1.00.a";
            status = "okay";
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x15 0x4>;
            reg = <0xe0002000 0x1000>;
            dr_mode = "host";
            phy_type = "ulpi";};};}
```

Příloha B: Tabulka podporovaných příkazů

Do ovladače „SET“	Z ovladače „GET“	Komponenta	Příkaz
Ano	Ano	Bolometr „BOL“	Povolení komponenty „ENA“
Ano	Ano		Povolení korekce „CEN“
Ano	Ano		Obrázek „IMG“
Ano	Ano		Data korekce „CDT“
Ano	Ano		Parametry čipu
Ano	Ano		Povolení streamu „STR“
Ano	Ne		Výpočet korekce ofsetu „COC“
Ano	Ne		Výpočet korekce zesílení „CGC“
Ano	Ano		Počet průměrů „AVG“
Ano	Ano		Posun ofsetu „ZOF“
Ne	Ne		Záznam videa „AVI“
Ano	Ano	Kamera „CAM“	Povolení komponenty „ENA“
Ano	Ano		Obrázek „IMG“
Ano	Ano		Povolení streamu „STR“
Ano	Ano	Ovládání „ACC“	Rozmazání „GBE“
Ano	Ano		Dynamické obarvení „CDE“
Ano	Ano		Minimální hodnota „CSL“
Ano	Ano		Maximální hodnota „CSH“
Ano	Ano		Šedotónové obarvení „GDE“
Ano	Ano		Minimální hodnota „GSL“
Ano	Ano		Maximální hodnota „GSH“
Ano	Ano		Větší rozmazání „INB“
Ano	Ano		Vlastní korelace „CUK“
Ano	Ano		Jádro korelace „KET“
Ne	Ne	SDkarta „SDC“	Ukládání dat a nastavení
Ne	Ne	Ostatní periferie	Pro novou verzi

Příloha C: Obsah přiloženého CD

- Text diplomové práce
- Obsah bootovací SD karty
    - Bootovací soubory
    - Přeložené moduly
    - Spustitelné aplikace
- Zdrojové soubory k souboru *Boot.bin*
- Zdrojoví kód modulů
- Zdrojoví kód aplikací a hardwarový popis systému
- Pořízené fotografie