Fakulta elektrotechnická
Telekomunikační a multimediální systémy

Diplomová práce

HW implementace obrazové analýzy kamerových dat z parkovacích stání

Autor práce: Daniel Hořejší

Vedoucí práce: Ing. Vladimír Pavlíček Ph.D.

Plzeň 2017

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Daniel HOŘEJŠÍ**
Osobní číslo:	**E15N0062P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Telekomunikační a multimediální systémy**
Název tématu:	**HW implementace obrazové analýzy kamerových dat z parkovacích stání**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Proveďte analýzu potřebného HW a SW vývojového kitu OMAP L138 pro zpracování statických obrazových dat z průmyslové kamery, příp. diskutujte možnost použití kamery s průmyslovým PC.
2. Realizujte snímání statického obrazu z kamery (příp. z více kamer) do vnitřní paměti vývojového kitu (průmyslového PC) a obraz ekvalizujte.
3. Dle možností aplikujte některý z navržených algoritmů zpracování a detekce volných parkovacích míst.
4. V případě úspěšného získání informace o volných místech tuto informaci zapište do definované datové struktury.
5. Dosažené výsledky zhodnoťte, diskutujte klady a zápory navrženého řešení a popište možné budoucí kroky.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah kvalifikační práce: **40 - 60 stran**

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

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Vladimír Pavlíček, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce: **14. října 2016**

Termín odevzdání diplomové práce: **19. května 2017**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 14. října 2016

Abstrakt

Práce je součástí projektu, jehož cílem je vyhodnocovat z kamerových snímků parkoviště školy obsazenost jednotlivých parkovacích míst. Tímto způsobem je možné získávat informace o obsazenosti míst bez finančně náročného zásahu do infrastruktury parkoviště. Stavy jednotlivých parkovacích míst by poté měly být odesílány na školní síť, odkud by měly být přístupné pomocí mobilní aplikace.

V předchozích pracích na projektu již byl vytvořen vhodný algoritmus na rozpoznávání vozidel na parkovacích místech. Výpočty algoritmu jsou však při použití standardního PC časově příliš náročné. Vzhledem k charakteru algoritmů na zpracování obrazu by bylo vhodné výpočty realizovat na speciálním typu procesoru - digitálním signálovém procesoru. Pro jeho optimální využití je ale vhodné jej doplnit pomocným procesorem, který bude provádět funkce aplikace netýkající se samotných výpočtů. Komplikací pak může být vhodná volba a realizace meziprocesorové komunikace.

Cílem práce je zjistit možnosti procesoru OMAP-L138 pro uvedený způsob využití. OMAPL-138 se skládá z procesorů ARM-9 a DSP procesoru C674x. Pro ověření možností architektury by měla být vytvořena aplikace, v níž ARM procesor bude získávat obrazová data z kamery, která poté pomocí meziprocesorové komunikace předá k výpočtům na DSP procesoru. DSP procesor by měl provést základní ekvalizaci obrazu a výsledky předat zpět ARM procesoru. Ten by měl poté snímek zobrazit na displeji vývojové desky a nahrát jej na server.

Abstract

This master thesis is part of a project which aims to scan a parking lot around the University with a camera and then analyze these pictures to determine if each single parking spot is free or not. This approach allows us to get this information without a need to modify the infrastructure of the parking lot which could be quite expensive. The information about each parking spot should be then sent over to the school network from where it should be accessible via a phone app.

Suitable algorithm for free spot detection has been done yet in previous works on this project. However, the execution of this algorithm is too slow when it is being run on a standard PC. It should be better to realize these computations on a more suitable hardware platform. Due to the nature of data processing in this algorithm, digital signal processors should be the best choice fot that. It is better though to select a platform which contains also a general purpose processor. That should be used for general tasks of the algorithm in order that DSP could focus on parts for which it is the most suitable. Main problem of this solution can be communication between both of these processors.

The aim of this thesis is to explore suitability of the architecture of the processor OMAPL-138 for this kind of algorithm. OMAP-L138 is a multicore processor which contains general purpose processor ARM-9 and DSP processor C674x. The ARM processor should be used for getting image data out of a camera and these data should be then handed over to the DSP processor for computations. DSP processor should then perform some basic image equalizations and then results of these computations were handed over to the ARM processor, should then display the image and also load it up to the server.

Prohlášení

Prohlašuji, že předkládanou diplomovou práci jsem vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu literatury, který je součástí této diplomové práce. Dále prohlašuji, že veškerý software použitý k řešení této diplomové práce je legální.

V Plzni dne

.....

Daniel Hořejší

.....

Poděkování

Tato diplomová práce vznikla za podpory projektu SGS.

Obsah

1 ÚVOD 1

2 HARDWARE OMAPL138 2

2.1 ARM926EJ-S 2

2.2 C674X 2

2.3 PERIFERIE 4

2.4 VÝVOJOVÁ DESKA OMAPL138 EXPERIMENTER KIT [9] 4

3 VÝVOJOVÉ NÁSTROJE PRO OMAPL138 6

3.1 DVSDK 6

3.2 VIRTUAL BOX 7

3.3 U-BOOT 7

3.4 DSP/BIOS 8

3.4.1 Hierarchie procesů 9

3.4.2 Statistické nástroje 10

3.4.3 Dynamické alokování paměti 11

3.5 CODE COMPOSER STUDIO 12

4 DSP/BIOS LINK 13

4.1 ARCHITEKTURA DSPLINK 13

4.1.1 GPP 14

4.1.2 DSP 15

4.2 PŘEKLAD A SESTAVENÍ MODULU DSPLINK 15

4.2.1 Překlad pro DSP v CCS 16

4.3 INSTALACE MODULU DSPLINK DO OS 17

4.4 ZMĚNY KONFIGURACE 19

4.4.1 GPP 19

4.4.2 DSP 20

4.4.3 Linux 20

4.5 KOMPONENTY DSPLINK 22

4.5.1 Stručný popis komponent 22

4.5.2 PROC 22

4.5.3 POOL 23

4.5.4 MSGQ 24

4.6 BĚH APLIKACÍ [39] 26

4.6.1 Inicializace 26

4.6.2 Vykonávání 27

4.6.3 Ukončení 27

4.7 LAZENÍ DSPLINK APLIKACÍ 29

4.7.1 DSP [38] 29

5 POSTUP REALIZACE APLIKACE 31

6 VÝVOJ - SD, NFS 32

7 SNÍMÁNÍ PŘEPÍNAČE 34

8 OVLÁDÁNÍ DISPLEJE 36

9 KAMERA 39

10 JPEG KODEK 41

11 NEGATIV SNÍMKU 45

12 DSPLINK ROZDĚLENÍ PAMĚTI 49

13 MOŽNOSTI ANALÝZY PARKOVACÍCH STÁNÍ NA OMAPL138 [20] 53

14 ZÁVĚR 55

SEZNAM POUŽITÝCH ZDROJŮ 56

PŘÍLOHY 61

1 Úvod

Práce je součástí projektu, jehož cílem je z kamerových snímků parkoviště školy vyhodnocovat obsazenost jednotlivých parkovacích míst. Tímto způsobem je možné získávat informace o obsazenosti míst bez finančně náročného zásahu do infrastruktury parkoviště. Stavy jednotlivých parkovacích míst by poté měly být odesílány na školní síť, odkud by měly být přístupné pomocí mobilní aplikace.

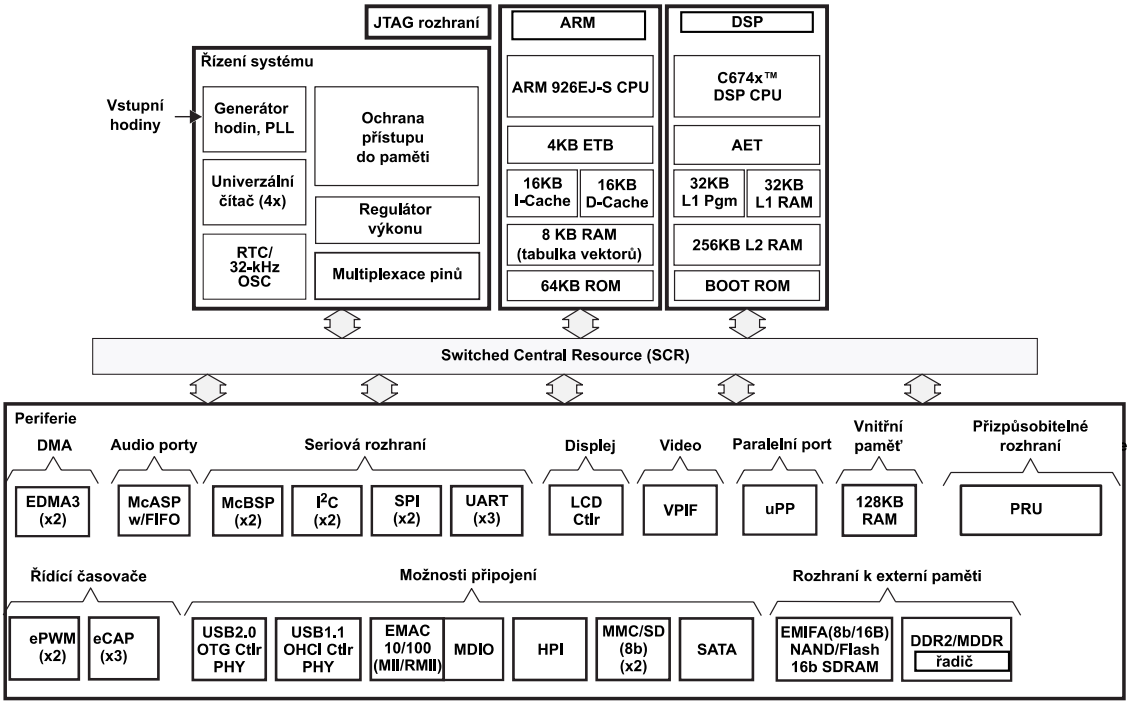
V předchozích pracích na projektu byl vytvořen vhodný algoritmus na rozpoznávání vozidel na parkovacích místech. Výpočty algoritmu jsou však při použití standardního PC časově příliš náročné. Vzhledem k charakteru algoritmů na zpracování obrazu by bylo vhodné výpočty realizovat na speciálním typu procesoru - digitálním signálovém procesoru. Pro jeho optimální využití je ale vhodné jej doplnit pomocným procesorem, který bude provádět funkce aplikace netýkající se samotných výpočtů. Komplikací pak může být vhodná volba a realizace meziprocesorové komunikace.

Cílem práce je zjistit možnosti meziprocesorové komunikace dvojjádrového procesoru OMAP-L138, skládajícího se z procesorů ARM-9 a DSP procesoru C674x. Pro zkoumání byla vytvořena aplikace, v níž ARM procesor získává obrazová data z kamery, která následně pomocí meziprocesorové komunikace předá k výpočtům na DSP procesoru. DSP procesor provede základní ekvalizaci obrazu a výsledky předá zpět ARM procesoru. Ten poté data nahraje na server.

V práci se podařilo zprovoznit základní obsluhu kamery, odesílání dat na server a informativní zobrazení výsledného obrázku na displeji vývojové desky. Tyto funkce byly navázány na meziprocesorovou komunikaci s DSP procesorem, na kterém byla prováděna jednoduchá obrazová operace - negativ snímku.

2 Hardware OMAPL138

OMAPL138 je dvojjádrový procesor s nízkým příkonem skládající se z jader ARM926EJ-S a DSP procesoru C674x. Základní architektura procesoru je na Obr. 1.[1]



Obr. 1 Architektura procesoru OMAPL138 (Upraveno z [1])

2.1 ARM926EJ-S

ARM926EJ-S je 32 bitový RISC procesor schopný vykonávat 32 či 16 bitové instrukce a zpracovávat 32, 16 či 8 bitová data. Pro souvislý běh programu využívá zřetězenou strukturu (pipeline). ARM9 jádro má vlastní 16 kB programovou a 16 kB datovou cache. Obě tyto paměti jsou 4-cestně mapované. [1]

2.2 C674x

Jádro C674x se skládá z 8 výpočetních bloků, dvou souborů registrů každého s vlastním přístupem do datové cache. Každý soubor registrů se skládá z 32 32 bitových registrů, celkově tedy z 64 registrů. Každá z 8 výpočetních jednotek zvládne zpracovat jednu instrukci za jeden hodinový cyklus. [1]

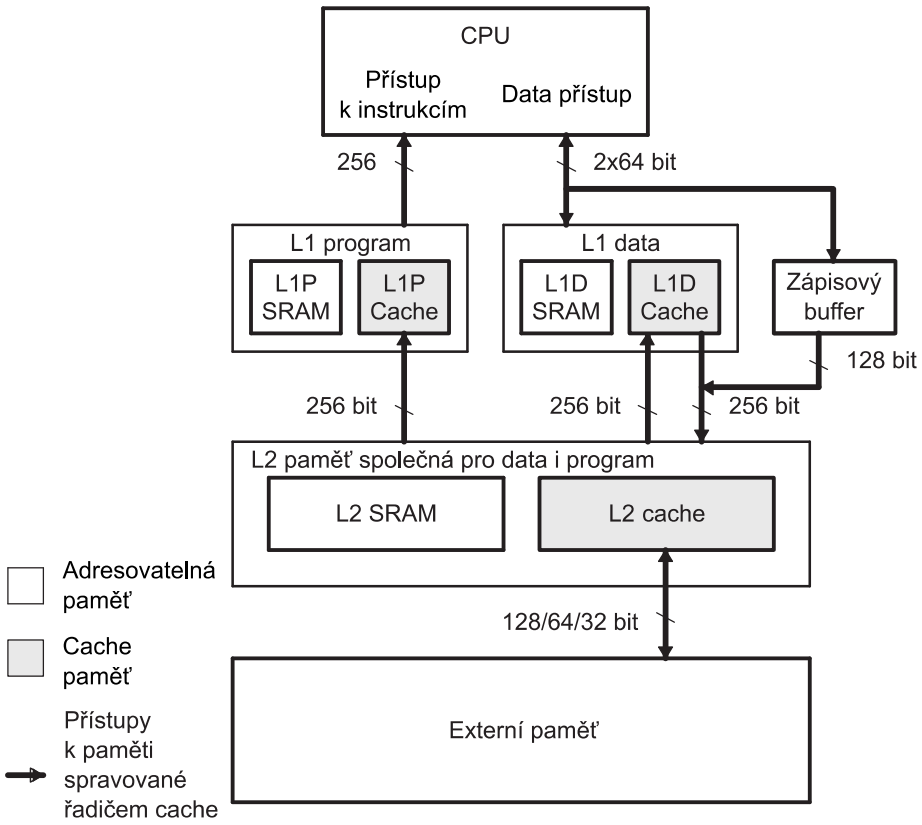
Všechna násobení jsou vykonávána násobícími jednotkami .M. Jednotka .M obsahuje i násobičku pro výpočty s desetinnou čárkou (datový typ float). Jednotky .S a

.L umožňují vykonávat obecné matematické a logické operace a jednotka .D slouží primárně k nahrávání dat z paměti do registrů a k ukládání dat z registrů do paměti. [1]

Jádro C674x má dvouúrovňovou paměťovou architekturu založenou na cache. První úroveň paměti je rozdělena na programovou a datovou cache. Programová cache první úrovně je přímo mapovaná (direct mapped) o velikosti 32 kB. Datová cache první úrovně má stejnou velikost, ale je tzv. dvoucestná (2-way-set-associative). Cache druhé úrovně (L2) je společná pro program i data a je čtyřcestná (4-way-set-associative) o velikosti 256 kB. [1], [7]

Paměti L1 i L2 je dále možné konfigurovat buď celé jako cache, nebo celé jako adresovatelné vnitřní RAM (IRAM), anebo různé poměry obou variant (např. 32kB IRAM/224kB cache). [1], [7]

Paměť L2 DSP procesoru je přímo přístupná i z ARM procesoru. DSP do vnitřních pamětí ARM procesoru (RAM, ROM, vektor přerušení) přístup naopak nemá. [1], [7]



Obr. 2 Paměťová architektura procesoru C674x (upraveno z [7])

2.3 Periferie

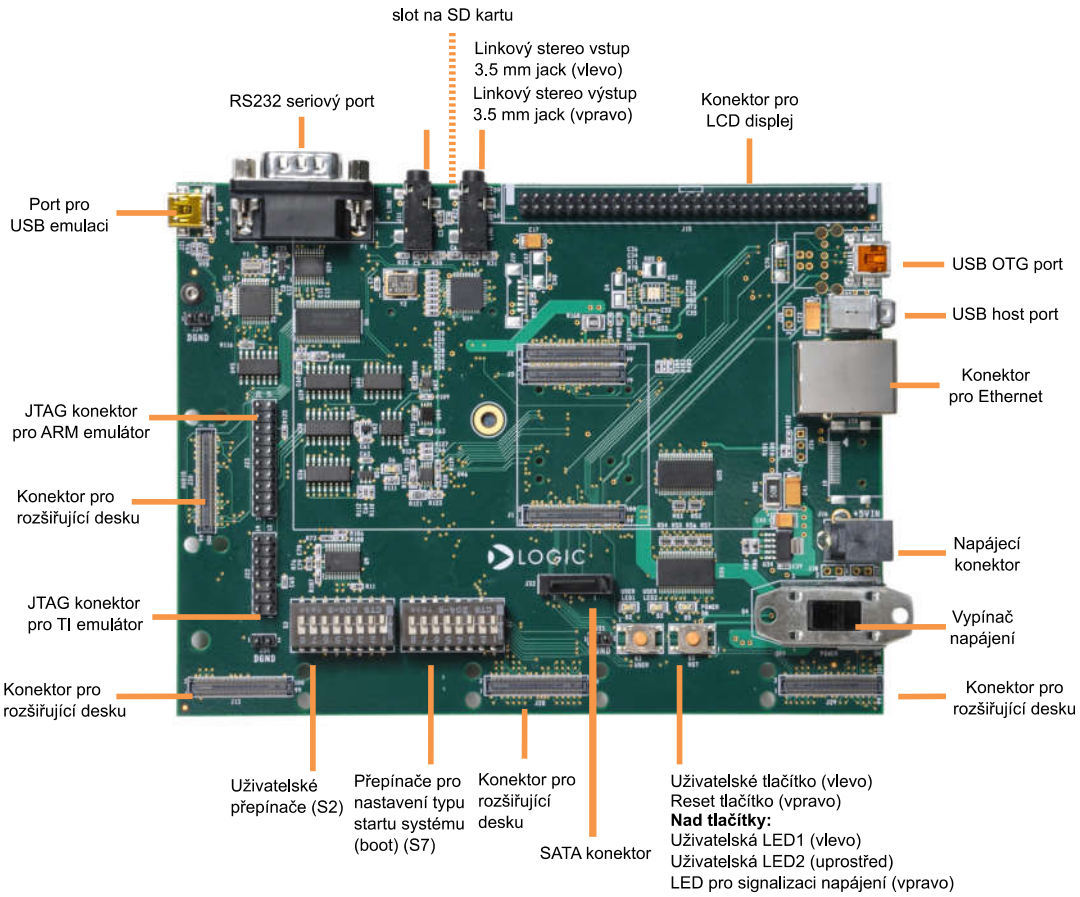
Oba procesory mohou přistupovat ke všem periferiím, jak je patrné z Obr. 1. Vzhledem k velkému množství periferií a úspoře vývodů čipu slouží různé piny více funkcím. Výběr jejich aktuálního nastavení je pak řešen multiplexory. Všechna přerušení mohou být obsluhována jak z ARM procesoru, tak z DSP. Některá přerušení jsou pak dedikována pro meziprocesorovou komunikaci mezi ARM a DSP procesorem. [1], [7]

Procesor umožňuje několik variant zavádění a spouštění programu (boot). Volba konkrétní varianty zavedení programu je nastavována přepínači na vývojové desce, nebo ve specifickém softwaru (U-Boot). [1][9]

2.4 Vývojová deska OMAPL138 experimenter kit [9]

Vývojový kit OMAP-L138 je platformou, která umožňuje ověření možností procesoru OMAP-L138. Kit obsahuje následující:

- procesor OMAP-L138 s externí pamětí mDDR o velikosti 128 MB
- 4.3" LCD dotykový displej
- RJ45 Ethernet
- USB OTG 2.0 rozhraní
- USB 1.1 rozhraní
- RS232 seriový port, rychlost 115.2 kbps
- slot k připojení MMC/SD karty
- konektory pro JTAG



Obr. 3 Vývojová deska OMAPL138 eXperimenter Kit (upraveno z [9])

3 Vývojové nástroje pro OMAPL138

3.1 DVSDK

DVSDK (Digital Video Software Development Kit) je balíček softwaru firmy Texas Instruments (TI), umožňující rychlý vývoj multimediálních aplikací v prostředí OS Linux. Je adaptován pouze pro vybrané procesory TI. Pro procesory obsahující také DSP jádro pak poskytuje i SW umožňující využití DSP ve vhodných částech aplikací. [4]

DVSDK je volně ke stažení ze stránek TI. Vývoj i údržba balíčku však již byly ukončeny. Existuje i novější verze obdobného balíčku softwaru, MCSDK (Multicore Software Development Kit), která je také určena pro procesor OMAPL138. Ta je však vyvíjena pro novější vývojovou desku procesoru OMAPL138 LCDK. Pro starší varianty vývojových desek – OMAPL138 EVM a OMAPL138 experimenter kit – je pak doporučeno pouze DVSDK. Je možné, že komponenty MCSDK jsou funkční i na těchto starších vývojových platformách, ale kompletní funkce balíčku pro ně není testována. [4][6][5]

Součástí balíčku DVSDK (4.03.00.06) je:

- software pro zavádění OS (bootloader) U-boot
- jádro OS Linux (2.6.37) včetně příslušejících modulů OS
- výchozí souborový systém pro Linux
- soubor knihoven umožňujících ovládání HW periferií dané vývojové desky (Board Support Libraries)
- RTOS DSP/BIOS (5.41.10.36)
- software pro meziprocesorovou komunikaci DSPLink (1.65.00.02)
- kodeky pro DSP
- balíček softwaru pro překlad a sestavení programů (kompilátor, linker), jak pro ARM procesor, tak pro DSP procesor

Většina součástí DVSDK je distribuována v podobě zdrojového kódu a je tedy potřeba je přeložit. Balíček DVSDK obsahuje i skripty spouštěné v příkazové řádce

vývojového PC (host PC), které usnadní nastavení a překlad jednotlivých částí DVSDK, včetně konfigurace a překladu jádra Linux. [45]

DVSDK je na vývojovém PC podporována pouze pro operační systém (OS) Linux, konkrétně pro distribuci Ubuntu 10.04. Ta je dodávána s vývojovým kitem, případně je i volně ke stažení na oficiálních stránkách Ubuntu distribuce. [5]

3.2 Virtual Box

Jelikož některé části vývoje programů je nutné provádět v OS Linux a jiné v OS Windows (lazení kódu s použitím JTAG), je možné použít software Virtual Box, který slouží k virtualizaci různých OS. Umožní tedy spouštění jiného OS uvnitř již běžícího OS v PC. Virtual Box je vyvíjený společností Oracle, ale je distribuován s licencí open-source a je tedy i volně ke stažení. Snadné předávání dat mezi nativním OS a virtualizovaným OS umožňuje funkce sdílených složek. Při ní je v rámci nativního OS nastavena složka, jejíž obsah je poté sdílen mezi oběma OS.

3.3 U-Boot

U-Boot je software sloužící k zavádění a spouštění operačního systému (bootloader). Jeho velikost je typicky pouze několik desítek až set kB, aby bylo možné jej provozovat i v omezených paměťových možnostech zabudovaných (embedded) procesorů. Umožňuje více způsobů zavádění operačního systému, včetně stažení jádra OS ze serveru pomocí TFTP či k zavádění OS z paměťové karty.

Některé z možných zdrojů k zavádění OS jsou:

- SD karta
- NOR flash
- NAND flash
- USB
- TFTP (jádro OS staženo z FTP serveru)

Zmíněné varianty zavádění OS umožňuje i OMAPL138.

Okamžitě po zapnutí napájení je možné běh U-Boot přerušit pomocí konzolového vstupu. To umožní nastavení typu a parametrů zavádění OS. Nastavení je prováděno specifickými příkazy příkazové řádky. U-boot umožňuje práci s proměnnými prostředí (environment variables). Do nich je možné uložit jednotlivé části nastavení a

zpřehlednit tak zapsanou konfiguraci. Tu je možné zapsat do permanentní paměti a tím uchovat nastavení pro jednotlivé typy zavádění OS i přes vypnutí zařízení. [44]

Pro řídící příkazy U-boot a způsob použití byl zvolen manuál firmy Logic PD, která vývojovou desku sestavuje [44]. Konkrétní význam příkazů použitých pro řešení práce bude součástí praktické části.

3.4 DSP/BIOS

DSP/BIOS je proprietární (TI) jádro operačního systému, určené pro DSP procesory TI. Je to preemptivní operační systém reálného času (RTOS). Cílem RTOS je dosažení rychlého provádění funkcí OS, minimalizace velikosti OS, dosažení co nejpřesnějšího určení doby trvání jednotlivých operací OS a zajištění stability těchto dob. U preemptivního OS je výpočetní čas jednotlivým procesům přidělován pouze na základě jejich nastavených priorit (rozdíl např. vůči desktop variantám OS Linux či Windows). DSP/BIOS je objektově orientovaný[1], proto změna priority jednoho z běžících procesů nevyžaduje změnu nastavení ostatních procesů. [16]

V současných systémech je používán OS SYS/BIOS, který je následníkem jádra DSP/BIOS. Základním rozdílem mezi nimi je, že DSP/BIOS je OS pouze pro DSP procesory, kdežto SYS/BIOS rozšiřuje možnosti použití OS i na standardní mikrokontroléry TI. Oba OS mají podobnou filozofii i mnoho funkcí systému. Na webových stránkách TI jsou dostupná instruktážní videa sloužící k seznámení s jeho principy (zahrnuta v kapitolách TI-RTOS). Jejich součástí jsou také laboratorní praktika, při nichž je jedna aplikace řešena různými variantami v rámci BIOS. [12]

Součástí DSP/BIOS jsou například funkce na řízení cache DSP procesoru, software pro analýzu běhu programu (grafy zatížení CPU, časově efektivní varianty funkce printf, informace o době trvání jednotlivých procesů), funkce sloužící k synchronizaci mezi jednotlivými procesy a také plánovač (scheduler), který řídí, která z úloh má být aktuálně vykonávána. [16]

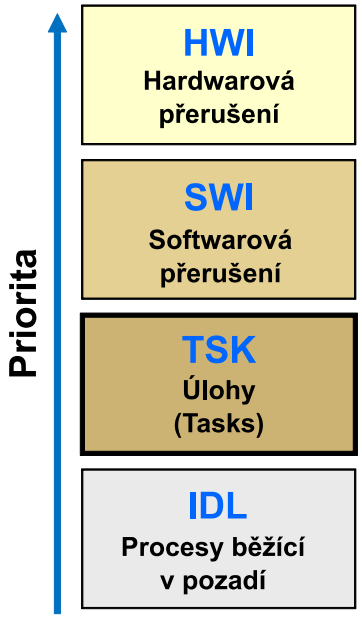
[1] objekty jsou na sobě navzájem nezávislé, takže změna jednoho neovlivní ostatní – pro RTOS důležité, ovlivňuje determinismus [16]

3.4.1 Hierarchie procesů

Jednotlivé procesy vždy žádají plánovač o provedení. Ten je poté umístí do fronty procesů, kterou řadí podle priorit jednotlivých procesů. V jeden čas běží vždy pouze jeden proces, ten s nejvyšší prioritou z aktuálně aktivních procesů. Změna vykonávaného procesu je provedena buď při požadavku nového procesu o spuštění, pokud má tento vyšší prioritu než všechny čekající procesy, anebo po dokončení prováděného procesu. [16]

DSP/BIOS rozlišuje několik typů úloh s rozdílnými prioritami:

- HWI – nejvyšší priorita v systému; vázané na HW přerušení
- SWI – typicky volané z HWI; slouží k post-zpracování dat (získaných například pomocí HWI), tedy k operacím, které není nutné provést okamžitě při nastání přerušení; jejich cílem je umožnit běh procesů, které mají vyšší prioritu, než jakou má zpracování nových dat, ale které by nemohly být provedeny v HWI kvůli jeho vysoké prioritě v rámci systému
- Task – nižší priorita než SWI, ale vyšší než Idle
- Idle – nejnižší priorita v systému; může obsahovat více funkcí, které bude vykonávat; pokud v systému není žádný proces čekající na provedení, pak je cyklicky prováděn tento typ procesu



Obr. 4 Priority procesů v DSP/BIOS (upraveno z [16])

V rámci jednotlivých typů procesů je dále možné rozlišit priority mezi procesy stejného typu.

Objekty HWI a SWI jsou navrženy tak, aby vždy doběhly až do svého ukončení a až poté uvolnily místo procesům s nižšími prioritami. Pokud tedy není jejich běh přerušen procesem s vyšší prioritou, pak nepředávají výpočetní čas jinému procesu. [16]

Úlohy (Task) jsou naopak navrženy tak, aby umožňovaly pozastavit svou činnost, pokud čekají na výpočetní data. Tím mohou předávat výpočetní čas jiným úlohám. Jednotlivé úlohy navíc mohou dynamicky měnit prioritu, umožňující řešení problémů prioritních inverzí. Aby bylo umožněno přerušování běhu a dynamické změny priorit má každá úloha svůj vlastní zásobník (stack), do kterého před pozastavením ukládá dosavadní pozici ve vykonávaném programu. [16]

Jednotlivé typy procesů je možné vytvářet buď staticky pomocí grafického rozhraní (GUI), pomocí konfiguračního skriptu, nebo dynamicky pomocí příkazů v kódu. [16]

3.4.2 Statistické nástroje

Součástí OS jsou i nástroje umožňující získávání statistik jednotlivých procesů. Výsledky statistik je pak možné zobrazit v rámci vývojového prostředí Code Composer Studio (viz dále). Některé typy těchto statistik je možné pozorovat v reálném čase, jiné naopak pouze při zastavení procesoru. Statistiky sledované v reálném čase odesílají informace o běhu programu v rámci Idle procesu, aby neblokovaly vykonávání výpočetních úloh. Vzhledem k preemptivnímu charakteru systému tak může nastat, že úlohy s vyššími prioritami plně vytíží procesor a Idle proces není vůbec vykonáván. V takovém případě nelze informace o statistikách jednotlivých procesů zobrazit. [16]

Nástroje statistiky jsou např. následující:

- LOG_printf – obdoba standardní printf funkce, ale její trvání je jen 40 cyklů (normální printf řádově např. 1000 cyklů)
- CPU load – graf zatížení procesoru, zjišťován jako poměr času vykonávání Idle procesu vůči měřící časové periodě
- ROV – RTOS Object Viewer - zjišťuje stav jednotlivých procesů systému - které jsou aktuálně aktivní, jaký je stav

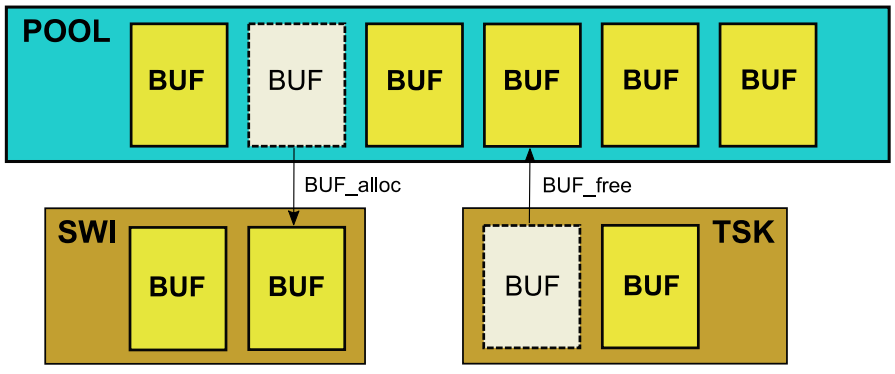
3.4.3 Dynamické alokování paměti

DSP/BIOS umožňuje vytvořit více typů haldy (heap). Na dynamické alokování paměti pak není používána standardní funkce jazyka C malloc(), ale DSP/BIOS funkce MEM_alloc (případné volání funkce malloc() je vnitřně řešeno pomocí MEM_alloc). Ta umožňuje nastavit, z jakého typu haldy bude paměť alokována. [16]

DSP/BIOS rozlišuje několik typů haldy pro dynamické alokování paměti. Standardní, kdy alokované bloky mohou být "libovolné" velikosti, dále variantu, která umožňuje alokovat pouze buffery definované velikosti, anebo variantu s více buffery předem definovaných velikostí. [16]

Varianta standardní, tedy obdoba funkce malloc, má výhodu v tom, že předem nemusí být známá velikost bloků, které bude potřeba alokovat. Nevýhodou však je, že není deterministická. Počáteční alokace tedy budou rychlé, kdežto při pozdějších bude muset potřeba déle prohledávat seznam bloků paměti, aby byl nalezen dostatečně velký volný blok. Tento typ navíc při frekventivním alokování a uvolňování paměti zpravidla způsobí fragmentaci paměti. [16]

Varianta alokování pevných velikostí bufferů je naopak plně deterministická. Principem je vytvoření banky předalokovaných bloků paměti, všech stejné velikosti, o jejichž přidělení si při běhu aplikace jednotlivé procesy žádají. Pro správu volných bloků paměti není potřeba použít např. spojový seznam, ale postačí pouhé pole informující o obsazení jednotlivých bloků. Toto řešení také brání fragmentaci paměti – uvolňovaný blok bude mít vždy stejnou velikost jako ostatní bloky a snadno se tak udržuje přehled o celkovém množství dostupných bloků paměti. [16]



Obr. 5 Princip alokování paměti ze staticky vytvořeného souboru bufferů (upraveno z [16])

3.5 Code Composer Studio

Code Composer Studio (CCS) je vývojové prostředí (IDE, Integrated Development Environment) firmy Texas Instruments. Umožňuje editaci, překlad a lazení programů pro TI procesory. [16]

Výstupem překladu programu pro DSP je spustitelný (executable) .out soubor. Ten je pak možné např. pomocí JTAG rozhraní nahrát do paměti procesoru, spustit jeho vykonávání a řídit jeho běh. V rámci vývojového prostředí je pak při běhu programu možné nahlížet do obsahu jednotlivých registrů procesoru, do obsahu buněk paměti, na aktuální hodnotu proměnných, nebo, v případě DSP/BIOS, na výsledky statistik procesů. [16]

4 DSP/BIOS Link

DSP/BIOS Link (DSPLink) je soubor softwarových komponent sloužících k meziprocesorové komunikaci v zabudovaných (embedded) aplikacích, v nichž GPP řídí DSP procesor firmy Texas Instruments a komunikuje s ním. Cílem modulu DSPLink je vytvořit snadno přenositelný systém, u kterého má být možné na různých procesorech a operačních systémech používat stejný soubor uživatelem volaných DSPLink funkcí. Podmínkou je, že na DSP musí běžet OS DSP/BIOS. Na GPP pak není potřeba specifický OS - může to být např. Linux, či Windows. [35]

DSPLink obsahuje varianty jak pro komunikaci mezi dvěma procesory, které jsou na jednom čipu (SoC, System on Chip), tak i mezi dvěma procesory předávajícími si informace pomocí DMA. Nepodporuje však komunikaci mezi více GPP procesory (tu umožňují až nové varianty meziprocesorových komunikací, které ale vycházejí z podobných principů).

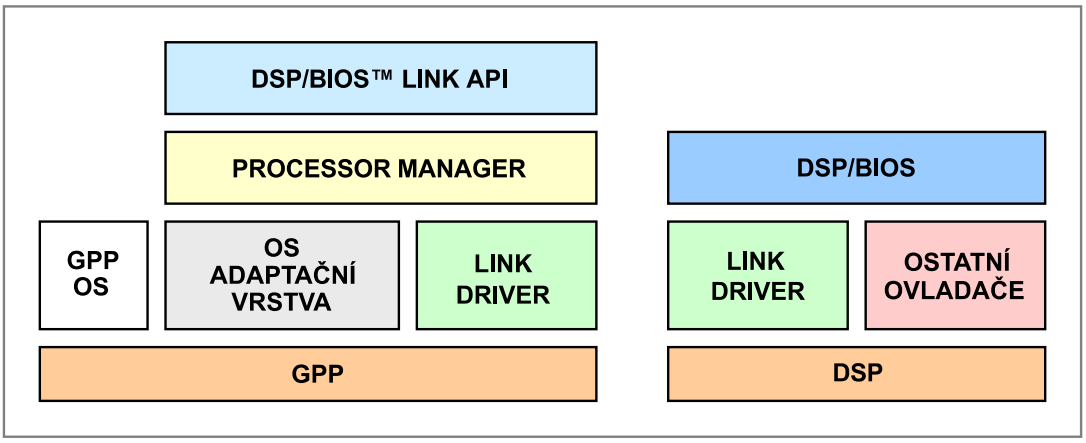
Funkce DSPLink obsahují: [36]

- základní řízení DSP procesoru
- sdílený synchronizovaný soubor bufferů (pool) mezi procesory
- notifikaci o událostech v uživatelských aplikacích (user event)
- nastavitelný vzájemně exkluzivní přístup (mutually exclusive access, mutex) ke sdíleným datovým strukturám
- předávání meziprocesorových zpráv (messaging)
- data streaming na bázi spojového seznamu
- data streaming na bázi kruhového bufferu

Kromě samotného řízení DSP a komunikace s ním pak DSPLink umožňuje i lazení (debug) meziprocesorové komunikace.[35]

4.1 Architektura DSPLink

Architektura DSPLink je rozdělena do vrstev. Tím je usnadněna přenositelnost DSPLink na jiné procesory nebo OS.[35]



Obr. 6 Systém vrstev architektury DSPLink (upraveno z [35])

4.1.1 GPP

OS adaptační vrstva slouží ke snadné přenositelnosti celého DSPLink mezi různými operačními systémy. Funkce OS, které jsou schopné vykonat operaci požadovanou modulem DSPLink, jsou strukturami této vrstvy pouze zapouzdřeny (encapsulate) do jednotných názvů. Ostatní komponenty DSPLink pak vždy používají k volání OS funkcí právě funkce OS adaptační vrstvy místo přímých volání OS specifických funkcí. Díky tomu je realizace vnitřních funkcí DSPLink nezávislá na typu OS, na kterém je DSPLink provozován. [35][39]

Vrstva LINK DRIVER realizuje nízkoúrovňové operace sloužící k přímému řízení běhu DSP a k přenosům dat mezi procesory. Pro vrstvu PROCESOR MANAGER vytváří abstrakci konkrétního hardware, na němž je DSPLink provozován. Ve vrstvě LINK DRIVER nedochází k ověřování platnosti vstupních parametrů jejích funkcí. Správnost vstupních parametrů tedy musí být ověřena ve vyšších vrstvách DSPLink. Díky tomu může LINK DRIVER pouze rychle vykonávat jednotlivé požadavky. Součástí této vrstvy jsou i struktury na překlad adres mezi fyzickými a virtuálními adresami. [35][38]

Vrstva PROCESSOR MANAGER spravuje hlavní objekty s nimiž komunikuje uživatel. Pro části svých funkcí vyžadujících realizaci v HW volá funkce vrstvy LINK DRIVER, pro části realizované operačním systémem pak OSAL konkrétního GPP OS a jak je patrné z obrázku, realizuje jejich vhodné propojení. [35] [37]

DSP/BIOS LINK API je rozhraní pro všechny klienty na GPP straně. Funkce této vrstvy jsou přímo volány v uživatelských aplikacích. Je to pouze tenká vrstva, která zpravidla provádí pouze ověření správnosti vstupních parametrů a která také umožňuje překlenout rozhraní mezi prostorem jádra OS a uživatelským prostorem v případě OS

jako je např. Linux. Její součástí je tedy např. i překlad adres mezi prostorem jádra a uživatelským prostorem. Dále pak předává řízení vrstvě PROCESSOR MANAGER. [35]

4.1.2 DSP

Na DSP straně komunikace musí běžet DSP/BIOS. Díky podpoře pouze specifického OS pak není potřeba na DSP adaptační vrstva OS. LINK DRIVER je na DSP jedním z modulů DSP/BIOSu. Účelem této vrstvy je opět nízkoúrovňová část komunikace. DSP/BIOS LINK API je rozhraním pro všechny klienty na DSP straně.

Některé z komponent DSPLink (MSGQ, POOL) jsou realizovány nativně již v prostředí DSP/BIOS. Ty pak DSPLink pouze zapouzdří do vlastních variant volání. API ostatních komponent, jsou (např. MPCS, MPLIST, NOTIFY a RingIO), je realizována samotným DSPLink. [35]

4.2 Překlad a sestavení modulu DSPLink

DSPLink je distribuován ve formě zdrojového kódu. Před jeho použitím je tedy nutné nastavit jeho požadovanou konfiguraci a následně jej přeložit a sestavit. To je realizováno pomocí GNU utility make. Při překladu lze postupovat buď podle návodu v dokumentu [35], nebo podle hlavního dokumentu DVSDK [45], při němž je překlad řešen jedním ze skriptů DVSDK.

Jak již bylo uvedeno v kapitole o DVSDK, je použití DVSDK podmíněno provozováním OS Linux na vývojovém PC. Pro vývoj aplikací podle dokumentu [35] jsou pak na vývojovém PC podporovány dva typy OS - Windows a Linux. Pokud však na GPP poběží OS Linux, pak k překladu jeho zdrojových souborů musí být použit na vývojovém PC OS Linux. Pro překlad DSP strany je možné použít na vývojovém PC jak OS Linux, tak Windows.

Postup podle [35] vyžaduje vlastní soubor nástrojů pro překlad a sestavení kódu (toolchain). Tyto nástroje je potřeba před překladem nastavit v konfiguraci překladu modulu. Dále jsou do interaktivního skriptu zadávány např. typy obou procesorů, typy OS obou procesorů a také které komponenty mají být do modulu zahrnuty. Na základě této konfigurace jsou pak při překladu automaticky vybrány jednotlivé podsložky DSPLink, které obsahují potřebné detaily nastavených procesorů či OS.

Součástí každé ze vzorových aplikací DSPLink jsou 3 soubory, jejichž obsah je potřeba pro vlastní aplikace upravit. Jedná se o soubory MAKEFILE, COMPONENTS a SOURCES.

- MAKEFILE - standardně v případě DSPLink aplikací neměněn
- COMPONENT - ovlivňuje překlad a sestavení dané aplikace; je v něm nastavováno jméno výstupního spustitelného souboru, přepínače pro kompilátor i linker (např. na nastavení úrovně optimalizací), či knihovny k překladu.
- SOURCES - uvádí seznam jednotlivých souborů, které tvoří danou aplikaci; pokud je potřeba do aplikace přidat nějaký zdrojový .c soubor, je potřeba jej připsat do seznamu souborů v SOURCES. Pokud je potřeba přidat do aplikace hlavičkový .h soubor, který má být používán pouze danou aplikací, stačí jej pouze vložit do složky aplikace. Pokud se však na daný .h soubor mají odkazovat i jiné aplikace, je potřeba jej přidat do proměnné souboru COMPONENT, která obsahuje seznam exportovaných hlavičkových souborů.

Při překladu DSPLink a jeho výchozích aplikací jsou mimo jiných generovány ve složce *$DSPLINK\gpp\BIN\Linux\Davinci\[RELEASE|DEBUG]* (podobně i v případě Windows vývojového PC) textové soubory:

- COMPONENT_defines.txt - obsahuje definice pro preprocesor, které byly potřeba k překladu dané komponenty
- COMPONENT_includes.txt - obsahuje seznam knihoven, které byly potřeba k překladu dané komponenty.

Místo COMPONENT je však vždy název jednotlivých vzorových aplikací či komponent DSPLink.

4.2.1 Překlad pro DSP v CCS

DSP stranu aplikací je možné překládat i pomocí CCS. Tato možnost však není z dokumentace přímo zřejmá, byla zjištěna až z fóra TI. V případě provozování této varianty v OS Windows je potřeba do Windows nainstalovat DSPLink, dostupný z [17]. Pro samotný překlad aplikací je pak potřeba překladu zadat informace o tom, které DSPLink specifické knihovny a z jakých složek OS do překladu zahrnout (include). Při instalaci DSPLink jsou generovány výše zmíněné soubory *_includes.txt a *_defines.txt jednotlivých komponent a vzorových aplikací. Proto je nejjednodušší je zkopírovat, nejlépe z té vzorové aplikace, která obsahuje všechny DSPLink komponenty, které mají být ve vlastní aplikaci použity. Kopii těchto textových souborů je poté potřeba umístit

do složky vlastní aplikace a následně v nastaveních překladu (Build settings) projektu v CCS nastavit tyto konfigurační textové soubory jako Compiler Command files. Poté by již mělo být možné aplikaci vyvíjet v rámci CCS. [35]

4.3 Instalace modulu DSPLink do OS

Po přeložení modulu DSPLink a aplikací, které jej využívají, je potřeba jej zkopírovat do souborového systému vývojového kitu (tedy například na SD kartu či do složky pro NFS), odkud jej spouštěný OS může vkládat do jádra. Zkopírování do souborového systému je v případě použití skriptů DVSDK zařízeno automaticky. V případě použití postupu podle [35] je pak popsáno v daném dokumentu.

Modul DSPLink je v OS Linux možné vkládat do běžícího jádra jako jeden z jeho dynamicky nahrávaných modulů. Jelikož je v OS Linux používáno ověřování platnosti jednotlivých modulů pro danou verzi přeloženého jádra, je při jakékoli změně jádra vedoucí k jeho novému překladu potřeba znovu přeložit i modul DSPLink i jeho jednotlivé aplikace.[45]

Při používání skriptů DVSDK je vložení DSPLink modulu do jádra provedeno jedním ze skriptů spouštěných při zavádění OS. Aplikace využívající DSPLink je tedy potom možné spouštět ihned po naběhnutí OS. [45]

V případě potřeby je možné modul vkládat do jádra manuálně a to buď pomocí skriptu v *$DSPLINK\etc\host\scripts\Linux\loaddsplink.sh*, nebo příkazem (provedeným v terminálu vývojové desky):

modprobe dsplinkk.ko (záměrně dvě k)

a odebrat příkazem

rmmod dsplinkk.ko

```
.
.
.
DSPLINK Module (1.65.01.05_eng) created on Date: Apr 19 2017
Time: 08:31:44
.
.
```

Obr. : Hláška oznamující správné nahrání modulu do jádra. Ta by se měla objevit jak v případě manuálního nahrávání modulu, tak v případě nahrávání jedním ze skriptů při startu OS. (vlastní záznam)

```
root@arago:~# lsmod
Module                  Size  Used by
dm365mmap               1716  0
dsplinkk              119140  0
cmemk                  20694  0
ipv6                  230498  12
minix                  25958  0
root@arago:~# rmmod dsplinkk
root@arago:~# lsmod
Module                  Size  Used by
dm365mmap               1716  0
cmemk                  20694  0
ipv6                  230498  12
minix                  25958  0
root@arago:~# modprobe dsplinkk
DSPLINK Module (1.65.01.05_eng) created on Date: Apr 19 2017
Time: 08:31:44
root@arago:~# lsmod
Module                  Size  Used by
dsplinkk              119140  0
dm365mmap               1716  0
cmemk                  20694  0
ipv6                  230498  12
minix                  25958  0
```

Obr : Ukázka vložení modulu do jádra a jeho vyjmutí. Příkazem lsmod jsou zobrazovány aktuálně běžící dynamické moduly jádra. (vlastní

4.4 Změny konfigurace [2]

Některé obecné parametry meziprocesorové komunikace je potřeba upravovat s ohledem na požadavky vyvíjených aplikací. Soubor často upravovaných parametrů je proto na GPP straně shrnut ve speciálním konfiguračním .c zdrojovém souboru. Na DSP straně je možné tyto změny provádět ve standardním tci/tcf konfiguračním souboru TI DSP programů.[35]

Parametry konfigurované těmito soubory jsou např. konfigurace paměti využívané pro DSPLink buffery. Výchozí konfigurace paměti pro DSPLink je nastavena následovně:

- velikost paměti pro sdílené buffery mezi procesory: 1 MB
- velikost paměti pro DSP kód/data : 1 MB

Pokud tato konfigurace není pro aplikace dostatečná, je potřeba konfiguraci změnit. Vzhledem k velikosti zpracovávaných obrázků v praktické části, které převyšovaly tuto hranici, bylo nutné konfiguraci upravit. [35]

4.4.1 GPP

Konfigurační soubor obsahuje informace o tom, jaké má být použito rozložení paměťové mapy použité k meziprocesorové komunikaci, pozici DSP reset vektoru v paměťové mapě a jeho velikost, endianitu paměti, čísla přerušení spjatých s jednotlivými typy meziprocesorové komunikace, či nastavení toho, která data jednotlivých DSPLink komponent mají být při jejich lazení (debug) vypisována.

Výchozí konfigurace pro GPP je ve složce

$DSPLINK\config\all\CFG_<Platform>.c

Od verze 1.40 DSPLink umožňuje na GPP straně několik způsobů změny výchozí konfigurace (dříve jen statická varianta) [35]:

- statickou variantu
- dynamickou variantu s využitím kopie konfiguračního souboru
- dynamickou variantu s nastavením požadovaných parametrů přímo v kódu aplikace

[2] Kapitola se netýká konfigurace DSPLink modulu, ale změn konfigurace pro jednotlivé aplikace

Statická varianta vyžaduje, aby byly potřebné změny konfigurace provedeny v konfiguračním souboru ještě před překladem modulu DSPLink. Změny konfigurace pak vyžadují i nový překlad aplikací, které DSPLink využívají.

Postup pro dynamickou variantu s využitím kopie konfiguračního souboru je následující:

- je vytvořena kopie konfiguračního souboru, která je poté umístěna do složky vytvářené aplikace
- následně jsou provedeny změny požadovaných parametrů v kopii konfiguračního souboru
- poté je změněn název řídící struktury ve které byly změny provedeny, např. z OMAPL138GEM_SHMEM_Config
  na OMAPL138GEM_SHMEM_AppConfig
- kopie konfiguračního souboru je přidána do seznamu souborů ke kompilaci (soubor SOURCES)
- externí deklarací je v kódu aplikace získán přístup ke změněné řídící struktuře
- v aplikaci je předán ukazatel na novou konfigurační strukturu jako argument funkce PROC_setup

Při poslední variantě postačí v kódu aplikace pomocí externí deklarace získat přístup k hlavní konfigurační struktuře. V té pak mohou být standardními postupy jazyka C změněny hodnoty požadovaných prvků struktury. Pro tyto změny je nutné znát tvar konfiguračních struktur, který je možné zjistit z [36]. Ukazatel na konfigurační strukturu je pak opět předán do funkce PROC_setup.

4.4.2 DSP

Změnu konfigurace pro DSP stranu je možné provést v konfiguračním TCF souboru DSP aplikace. Tento soubor je překládán s každou DSP aplikací a umožňuje tak změny konfigurace na straně DSP bez nutnosti opětovného překladu samotného DSPLink modulu. Výchozí hodnoty konfigurace jsou dány výchozím TCI konfiguračním souborem, který je specifický pro daný procesor a který je do TCF souboru zahrnut (include). [35]

4.4.3 Linux

V případě, že je v konfiguracích změněno rozložení paměťové mapy používané DSPLink aplikacemi, je nutné změny respektovat i v parametrech zavaděče Linux

(U-Boot). Konkrétně je potřeba pomocí mem parametru (součást U-Boot parametrů) nastavit, aby část paměťové mapy, používaná modulem DSPLink nebyla operačním systémem používána.

4.5 Komponenty DSPLink

4.5.1 Stručný popis komponent

- PROC - slouží k základnímu řízení DSP procesoru a k nahrávání kódu na DSP procesor
- POOL - alokátor bufferů ve sdílené paměti
- NOTIFY - umožňuje předávat upozornění na událost (notification of event) nastalou na vzdáleném procesoru; upozornění je realizováno meziprocesorovým přerušením a je možné s ním předávat 32 bitovou hodnotu (např. i ukazatel)
- MSGQ - slouží k předávání meziprocesorových zpráv ve formě datových struktur
- MPCS - zajistí vzájemně exkluzivní přístup (mutually exclusive access) ke sdíleným částem dat
- MPLIST - umožňuje předávat data ve formě obousměrného spojového seznamu; tím mimo jiné umožňuje zpracovávat datové "zprávy" v jiném pořadí, než v jakém byly odeslány
- CHANNEL - data streaming, kdy si dva komunikující procesy vyměňují vstupní a výstupní buffery (při zero-copy realizováno pouhou vzájemnou výměnou ukazatelů na zprávu ve sdílené paměti); odesílané buffery mohou být pro plynulost skládány do fronty; předání posledního zbývajícího naplněného bufferu fronty je blokováno, dokud není příjemcem předán i odpovídající prázdný buffer k naplnění
- RINGIO - předávání zpráv v podobě kruhového bufferu ve sdílené paměti

Ve vytvářené aplikaci byla zvolena kombinace komponent PROC, POOL a MSGQ. V aplikaci totiž bylo potřeba předávat několik meziprocesorových informací, přehledně organizovaných v datové struktuře. Dále jsou proto podrobněji rozebírány pouze tyto komponenty DSPLink.

4.5.2 PROC

PROC je zkratkou pro 'procesor'. PROC musí být součástí každé DSPLink aplikace.

Umožňuje:

- inicializaci (setup) DSPLink

- připojení se (attach) k DSP
- inicializaci hardware prostředků umožňujících komunikaci s DSP
- nahrání kódu na DSP
- spouštění běhu DSP od adresy určené ve spustitelném souboru DSP aplikace
- čtení a zápis do DSP paměti
- zastavení běhu DSP

Kód nahrávaný do DSP je nahráván ze souborového systému GPP procesoru. Pro čtení souboru jsou použity funkce OSAL vrstvy, které v daném OS umožňují čtení souborů (např. funkce fread()). Aby byl GPP procesor schopen nahrávat kód na DSP, obsahuje GPP parser DSP spustitelných souborů. Při parsování spustitelného souboru je z něj získána také vstupní adresa (entry point) programu, který je na DSP nahráván. Ta je poté uložena do jedné z řídících struktur DSPLink ve sdílené paměti. [35][39][41]

Nahrávání kódu ze souborového systému GPP potenciálně umožňuje, že pokud bude GPP připojen na nějaký server, pak je možné na tento server vložit nový kód pro DSP a spustit jej.

V případě SoC typu DaVinci či OMAP je řídícím procesorem (master) ARM procesor. Řízení v podobě volání PROC API je tedy prováděno pouze tento procesor. [36]

4.5.3 POOL

DSPLink komponent POOL poskytuje funkce ke správě bufferů ve sdílené paměti mezi procesory. Vlastnosti POOL jsou následující [36]:

- umožňuje konfigurovat sdílenou část paměti před zahájením komponenty
- alokování a uvolňování bufferů z oblasti sdílené paměti
- zajišťuje překlad adres mezi různými adresními prostory (např. mezi GPP do DSP)

Jelikož jsou buffery sdílené mezi GPP a DSP, musí být jejich velikosti zarovnané na velikost DSP cache. Zarovnání velikostí je možné provádět pomocí funkce DSPLINK_ALIGN.

Aplikacím poskytuje možnost dynamického alokování bufferů. Na správu meziprocesorové komunikace se sdílenou pamětí je použita odnož POOL komponenty, SMAPOOL (shared memory allocator pool, volným překladem alokátor bloků paměti ze souboru předdefinovaných bufferů ve sdílené paměti). SMA obsahuje seznam předem definovaných bufferů sdílených mezi GPP a DSP a informace o tom, jestli jsou

tyto buffery volné. Jak již bylo uvedeno v kapitole DSP/BIOS, tento způsob podstatně zvyšuje determinismus trvání alokování paměti při běhu programu a také zamezuje vzniku fragmentace paměti. Tím, že jsou buffery z obou procesorů alokovány od stejné entity, je udržován přehled o aktuálně volném paměťovém prostoru. [36][35][43]

Pro SMAPOOL je potřeba nakonfigurovat jeho řídící strukturu SMAPOOL_Attrs a tu pak předat do volání funkce konfigurující POOL - POOL_open. Elementy řídící struktury jsou [36]:

- numBufPools - počet buffer poolů
- bufSizes - vektor velikostí jednotlivých bufferů (velikosti musí být zarovnané na velikost DSP cache)
- numBuffers - vektor počtu bufferů jednotlivých velikostí
- exactMatchReq - určuje, jestli POOL umožní předat aplikaci buffer o nejbližší vyšší velikosti, pokud aplikace požádá o buffer velikosti neodpovídající žádnému z přednastavených

Při předávání bufferů aplikacím musí být pro ně proveden překlad adres. Veškerý překlad adres je nutné provádět na GPP. DSP překlady adres nepoužívá, protože k fyzickým adresám paměti přistupuje přímo, bez abstrakce virtuálního paměťového prostoru. DSP také nepotřebuje překlad adres mezi prostorem jádra OS a uživatelským prostorem, protože DSP/BIOS takto rozdělen není. [36]

4.5.4 MSGQ

Zkratka pro 'message queue' - fronta zpráv. Slouží k předávání zpráv mezi procesory na bázi fronty zpráv. Jednotlivé zprávy jsou předávány v podobě datových struktur. Fronty zpráv jsou jednosměrné, vyzvedávat je tedy může pouze příjemce zprávy (reader), odesilatel (writer) zprávy vždy pouze vkládá. Frontu musí vytvářet její příjemce a odesilatel ji pak musí před vkládáním zpráv lokalizovat. Fronta je rušena opět příjemcem. [36]

Každá fronta zpráv má vlastní jméno, které musí být unikátní v rámci celého systému. Díky unikátnímu jménu, definovanému shodně na obou komunikujících procesorech, je pak možné získat na vzdáleném procesoru přes řídící struktury MSGQ pozici fronty ve sdílené paměti, i když bude fronta vytvořena dynamicky a tedy na předem neurčitelném místě v paměti. [36][40]

MSGQ umožňuje předávat zprávy libovolné délky. Jako první pole každé zprávy vždy musí být dokumentací definovaná hlavička zprávy, která má konstantní velikost.

Její velikost musí být také zahrnuta do celkové velikosti bufferu, o který žádá MSGQ při alokování bufferu pro zprávu. Obsah hlavičky plní DSPLink řídícími informacemi potřebnými pro přenos zprávy. [36]

Jednotlivé funkce MSGQ pracují pouze s hlavičkou zprávy. Naopak, uživatel plní data zprávy do oblasti za úvodní hlavičkou bloku zprávy. Tím, že je pro zprávu alokován blok odpovídající součtu velikosti hlavičky i dat zprávy, je zajištěno, že i když DSPLink pracuje pouze s hlavičkou, tak do datové části zprávy nebude zapisovat. [36][40]

Jedním z návrhových požadavků na MSGQ je, že odeslání zprávy musí být deterministické a neblokující. Kvůli tomu je odesílací funkce realizována tak, že okamžitě po zadání odeslání zprávy dokončuje svůj běh. Samotný přenos zprávy potom nemusí proběhnout okamžitě, tedy, dokončení funkce neznamená doručenou zprávu. V případech, kdy je však potvrzení o příjmu žádoucí, je možné při odesílání specifikovat i původce odesílané MSGQ, na kterou pak může příjemce odpovědět.

4.6 Běh aplikací [36]

Jednotlivé komponenty DSPLink jsou na sobě různě závislé, kdy inicializace některé z nich může vyžadovat provedení jiné. Jednotlivé funkce komponent jsou již v rámci dokumentace rozděleny na inicializační, vykonávací (execute), a ukončovací (delete) fáze. V úvodu programu se provedou inicializační fáze komponent, kdy jsou pro jednotlivé z nich alokovány potřebné zdroje. Poté následuje vykonávací fáze, během které může aplikace až do ukončovací podmínky běžet ve smyčce. Může tedy například cyklicky přijímat buffer dat, zpracovávat jej, vracet jej druhému procesoru a čekat na nová data. Po vykonávací části je provedena ukončovací fáze, během níž se jednotlivé komponenty, v požadovaném pořadí, postupně ukončí tak, aby byly uvolněny všechny pro ně alokované prostředky a bylo umožněno opětovné zahájení komunikace s DSP při novém spuštění aplikace.

Pro stručnost jsou dále uvedeny jen příkazy těch komponent, které jsou v aplikaci použity.

4.6.1 Inicializace

PROC

- PROC_setup - základní inicializace komponenty
- PROC_attach - inicializace hardware potřebného ke komunikaci s DSP; provede reset DSP a zapne jeho napájení
- PROC_load - nahrání kódu na DSP

POOL

- POOL_open - nakonfiguruje soubor bufferů, z kterého bude aplikacím umožněno dynamicky alokovat buffery, či zprávy

MSGQ

- MSGQ_transportOpen - zahájí přenos zpráv na vzdálenou frontu
- MSGQ_open - zahájí lokální frontu zpráv, do které budou přijímány všechny zprávy určené pro tuto frontu
- MSGQ_setErrorHandler - otevře frontu zpráv, která bude přijímat asynchronní zprávy o chybách komponenty MSGQ (např. i vypršení času zadávaného jako maximální doba čekání na zprávu)
- MSGQ_locate - pomocí unikátního jména vyhledá frontu příjemce zprávy

4.6.2 Vykonávání

PROC

- PROC_start - spustí běh programu, který byl v předešlých krocích nahrán na DSP; uvolní DSP z reset a spustí jeho běh od adresy dříve získané při PROC_load
- PROC_read - čte z paměti DSP
- PROC_write - zapíše do paměti pro DSP
- PROC_stop - zastaví běh DSP

POOL

- POOL_alloc - alokuje buffer z POOL
- POOL_translateAddr - umožňuje překlad adres mezi různými adresovými prostory (adresy uživatelského prostoru OS, jádra OS, fyzické adresy, DSP adresy)
- POOL_free - uvolní dříve alokovaný buffer zpět do POOL

MSGQ

- MSGQ_alloc - alokuje buffer pro zprávu pomocí POOL
- MSGQ_put - pošle zprávu na určenou frontu; příkaz je neblokující a deterministický
- MSGQ_get - vyzvedne zprávu z přijímací fronty
- MSGQ_free - uvolní buffer zprávy

4.6.3 Ukončení

PROC

- PROC_detach - odpojí se od DSP; pokud byl klient prvním, kdo se k DSP připojil, tak zároveň ukončí běh DSP, provede reset DSP a vypne jej
- PROC_destroy - uvolní zdroje alokované pro PROC komponentu v inicializační fázi
- PROC musí být v aplikaci ukončován poslední.

POOL

- POOL_close - zavře POOL

MSGQ

- MSGQ_release - uvolní vzdálenou frontu zpráv
- MSGQ_transportClose - ukončí přenos zpráv na vzdálenou frontu
- MSGQ_close - ukončí frontu zpráv příjemce

4.7 Lazení DSPLink aplikací

DSPLink umožňuje ladit chování aplikací jak pro GPP stranu komunikace, tak pro DSP. Postupy obojího dokumentuje [35]. Dále je probrána pouze jedna z metod pro lazení aplikace na DSP procesoru, která byla během programu používána.

4.7.1 DSP [35]

Lazení DSP strany aplikace je možné provádět v rámci vývojového prostředí Code Composer Studio (CCS) po připojení se k procesoru pomocí rozhraní JTAG. CCS pak umožňuje například náhled do jednotlivých registrů DSP, do obsahu jednotlivých pamětí, krokování instrukcí DSP programu, či nahlížení do statistik jednotlivých částí programu.

Pro lazení touto metodou je však potřeba DSP uvést do definovaného stavu. Toho je možné dosáhnout umístěním nekonečné smyčky na pozici před spuštěním inicializace samotného DSPLink (tedy např. hned na začátek funkce main() DSP programu). Není však možné použít pouhou while(1) smyčku - při jejím použití by překladač během optimalizací odstranil veškerý po ní následující kód z důvodů jeho nedosažitelnosti. Místo toho je potřeba použít následující:

volatile Int i = 1;

while(i) ;

Proměnná i je typu volatile - kompilátor ji nemůže optimalizovat, musí uvažovat že její hodnota může být změněna mimo rozsah samotného optimalizovaného programu.

Lazení probíhá následovně:

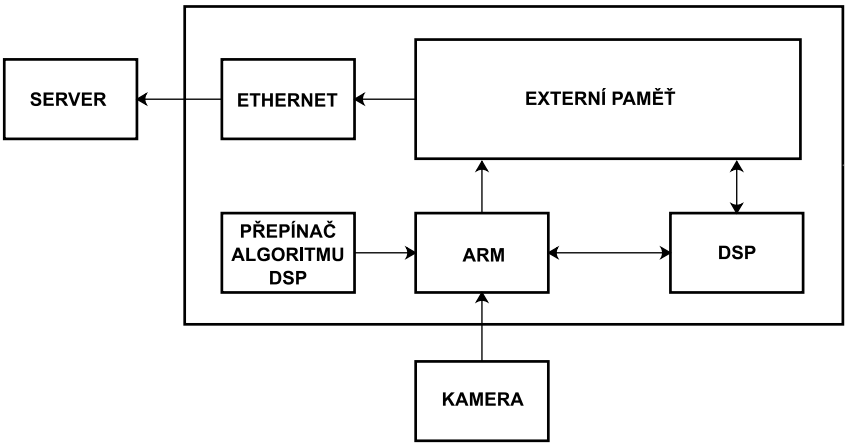
- spustí se aplikace ze strany GPP
- po rozběhnutí DSP pomocí volání funkce PROC_Start na GPP straně se DSP samo zastaví v uvedené nekonečné smyčce
- v CCS se z projektu odpovídajícího lazenému programu přepne do Debug perspektivy a pomocí JTAG se provede nahrání ladících (Debug) symbolů do vývojové desky. Je tedy nutné nastavit, aby na DSP nebyl nahráván i program, který je do DSP nahráván GPP procesorem
- po získání přístupu do programu DSP se provede přepsání hodnoty i lokální proměnné na 0 a spustí se pokračování programu, nebo umístění kurzoru za while smyčku a použití Set PC to Cursor

Poté je již umožněno standardní lazení DSP - krokování instrukcí programu, zavádění bodů zastavení, atd.

5 Postup realizace aplikace

Pro ověření principů a možností meziprocesorové komunikace by měla vzorová aplikace obsahovat následující funkce:

- snímání obrázku v nejvyšším možném rozlišení
- předání snímku do DSP
- volbu typu výpočetní operace, kterou DSP vykoná
- vykonání příslušné operace na DSP
- předání vypočtených dat do GPP procesoru
- zobrazení výsledného obrázku na displeji vývojové desky
- nahrání obrázku na server
- měření trvání důležitých operací



Obr. 7 Předběžné rozvržení vytvářené aplikace

Pro překlad DSP částí aplikace byl použit CCS, kvůli následné možnosti přehledně ladit kód aplikace. Nastavení CCS pro práci s procesory řady C6000, které je C674x součástí, bylo použito z v instruktážních videích TI-RTOS. [12] Pro nastavení CCS specifické

Překlad pro ARM procesor byl řešen pomocí skriptů DVSDK.

6 Vývoj - SD, NFS

Pro vývoj aplikací je kromě způsobu překladu programů potřeba vybrat vhodný způsob nahrávání programů na oba procesory. DSPLink umožňuje nahrávat kód na DSP ze souborového systému ARM procesoru a následně DSP spouštět. Díky tomu stačí najít způsob, kterým by při běhu ARM procesoru bylo možné do jeho souborového systému nahrávat soubory.

Díky tomu, že na vývojovém PC i na vývojovém kitu běží OS Linux, je možné použít tzv. NFS (Network Filesystem). Při něm je souborový systém pro vývojovou desku umístěn do jedné ze složek vývojového PC. Obsah této složky je poté sdílen přes rozhraní Ethernet s vývojovým kitem. Vývojové PC tak může snadno vkládat různé soubory do souborového systému ARM pouhým vložením těchto souborů do složky nastavené pro NFS.

Pro trvalý provoz aplikace je však NFS nevhodné - pro svou činnost vyžaduje spojení se serverem, na němž se souborový systém nachází. Při výpadku tohoto spojení je pak běh aplikace pozastaven až do doby opětovného navázání spojení. Pro nezávislý provoz byla tedy zvolena jiná varianta, při níž jsou jádro OS i souborový systém vývojového kitu umístěny na SD kartě. Před použitím SD karty pro tyto účely je potřeba ji vhodně naformátovat. Formátování sestává z vytvoření oddílů (partition) na kartě, z nichž jeden přísluší jádru OS a další pak souborovému systému. Formátování SD karty bylo provedeno podle návodu firmy Logic PD. [9]

Samotné nahrávání jádra i souborového systému na kartu i do složek NFS bylo řešeno pomocí DVSDK. To totiž obsahuje skripty, které automatizují:

- nastavení NFS a FTP serverů (FTP slouží k možnosti stažení jádra ze serveru ve vývojovém PC při zavádění OS, viz kapitola U-Boot)
- překlad a sestavení veškerých součástí DVSDK softwaru (jádra OS Linux, modulu DSPLink a dalších modulů)
- nahrání výchozího souborového systému i nově přeloženého jádra a jeho modulů do složek NFS či na SD kartu
- exportování parametrů pro zavádění OS pomocí U-Boot (jak pro SD kartu, tak pro NFS)

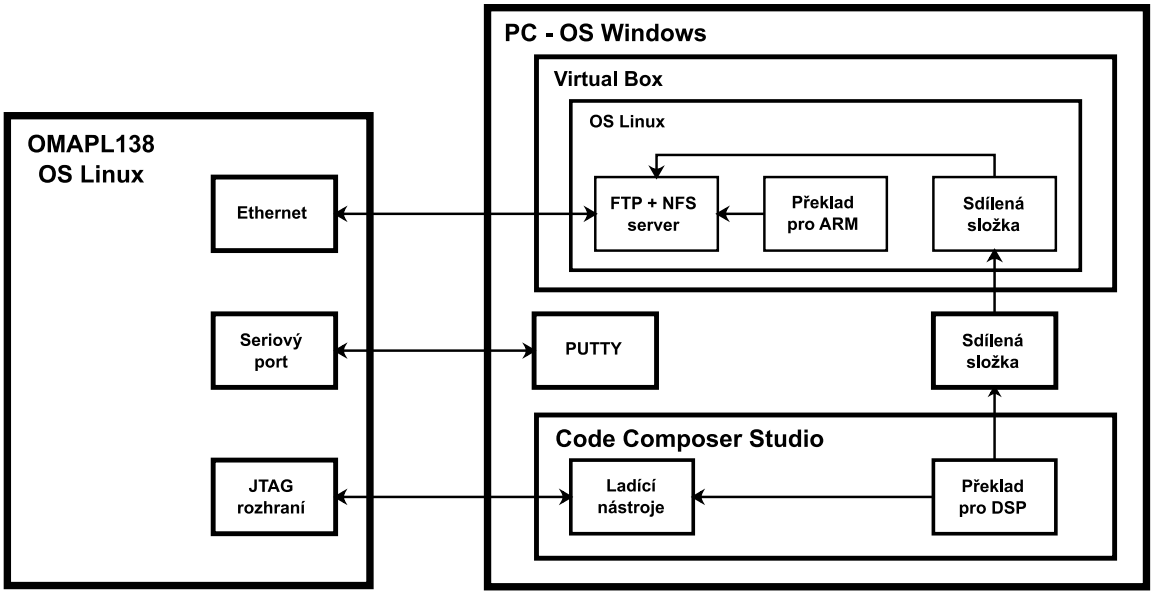
Jakákoliv změna v jádru OS Linux vyžaduje i znovupřeložení všech modulů jádra a mnohdy i aplikací, které tyto moduly využívají. Opakované změny, potřebné při vývoji, by pak byly bez konfiguračních skriptů DVSDK vzhledem k rozsáhlosti jeho software

komplikované. Pravděpodobně by pak při absenci skriptů DVSDK vedly k sepsání vlastních skriptů.

DSP strana aplikací byla překládána v software CCS ve Windows a to kvůli možnosti přehledného lazení DSP části aplikace přes rozhraní JTAG.

Při vývoji tedy bylo nutné provozovat na vývojovém PC OS Linux (pro sdílení souborového systému a provoz DVSDK) a zároveň OS Windows pro možnost lazení programů pomocí JTAG [3]. Proto byl OS Linux na vývojovém PC spouštěn v rámci software Virtual Box, uvedeného v teoretické části.

Schematický princip vývoje je na Obr. 8.



Obr. 8 Diagram použitého způsobu vývoje aplikací

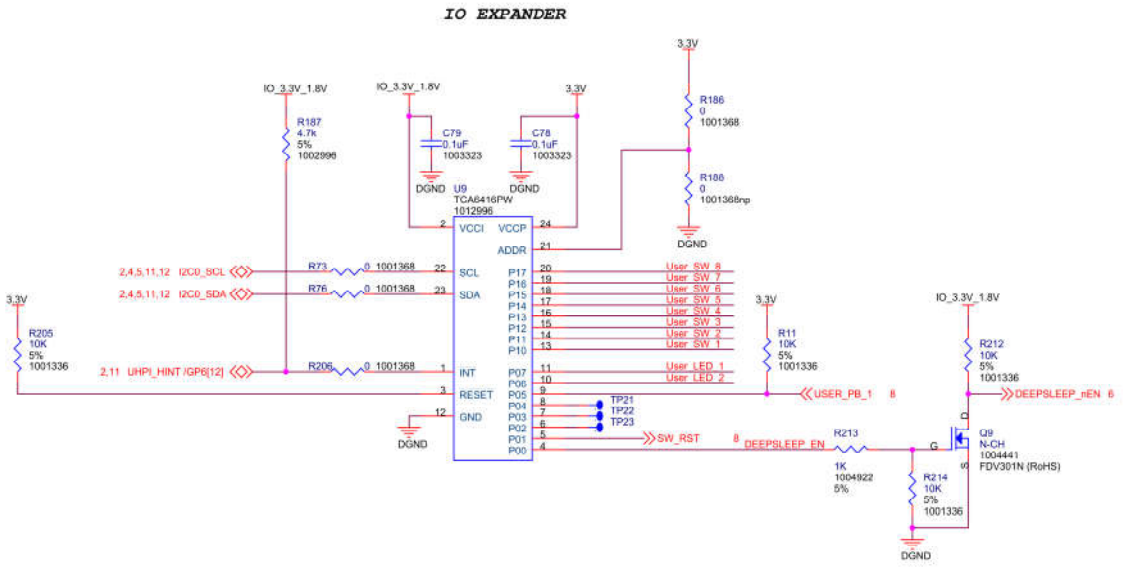
[3] Později bylo zjištěno, že od novější verze CCS je možný i provoz JTAG v rámci Linux [30][31], proto by bylo možné celý vývoj přenést do OS Linux.

7 Snímání přepínače

Změna výpočetního algoritmu na DSP za běhu aplikace je nastavována jedním z přepínačů vývojové desky. Z důvodu požadované plynulosti výpočtů na DSP je stav přepínače snímán ARM procesorem. Jelikož na ARM běží OS Linux, je možné snímat stav přepínače pomocí modulu, který přepínač v rámci OS obsluhuje.

Na vývojové desce je několik přepínačů a tlačítek - uživatelské tlačítko (User pushbutton), soubor 8 uživatelských přepínačů (User switches) a soubor 8 přepínačů pro nastavení typu zavádění OS (BOOT switches). Pro nastavení typu algoritmu byl využit jeden z uživatelských přepínačů. Soubor přepínačů nastavujících typ zavádění OS je samozřejmě nepřístupný a uživatelské tlačítko se po puštění vrací do původní polohy, není tedy na pohled zřejmé, jaký z módů je aktuálně nastavený.

Aby bylo možné uspořit piny čipu, jsou na desce jednotlivé uživatelské přepínače i uživatelské tlačítko snímány pomocí I^2C expanderu. Snímání stavů tlačítek je tedy možné pouze pomocí přístupu přes expander. Jak je patrné z hardwarové dokumentace ([13]), typ expanderu použitého na vývojové desce je TCA6416. Příslušný integrovaný obvod pak v rámci OS Linux obsluhuje modul pca953x. To je sice označení jiného typu expanderu, vzhledem k podobnosti řízení obou typů jsou však řídící funkce zastřešeny výše zmíněným modulem. Modul pca953x je tedy potřeba zahrnout do jádra OS při kompilaci jádra. [24]



Obr. 9 Část HW dokumentace vývojové desky zobrazující snímání přepínačů expanderem

Stav přepínače je v aplikacích možné snímat čtením vhodného souboru ve složce */sys/class/gpio* běžícího OS (GPIO = general purpose input output, v překladu univerzální vstupy/výstupy, dále jen GPIO). Aby však bylo možné do této složky přistupovat, je nutné její podporu nastavit při překladu jádra OS. [13]

Jádro OS při této variantě předává uživatelské aplikaci řízení příslušného pinu a informaci o jeho stavu. Tyto možnosti je do složky */sys/class/gpio* nutné tzv. exportovat z jádra. To je možné provést zápisem čísla popisujícího GPIO, který potřebujeme snímat, do souboru */sys/class/gpio/export*. Po tomto zápisu se ve složce */sys/class/gpio* objeví složka příslušného GPIO pinu. Ta pak obsahuje například soubor, v němž je zapsána aktuální hodnota pinu (0/1), či nastavení jeho směru (vstup/výstup). [13]

Číslo potřebné pro zadání exportování řízení je možné zjistit pomocí speciálního typu souborového systému debugfs. Ten primárně slouží k lazení chování jádra, kdy do něj vývojáři jádra vypisují např. podstatné informace k chování periferií. Podpora tohoto typu souborového systému musí být opět zahrnuta do jádra OS při jeho kompilaci. Za běhu OS je pak potřeba debugfs do jádra připojit vhodným příkazem. Následně je možné ve složce debugfs číst stavy jednotlivých GPIO a číselná označení těchto pinů. [13]

Některé z výchozích modulů jádra však mohou blokovat možnost řízení a snímání přepínačů ve složce /sys/class/gpio a to v případě když mají příslušné GPIO zabrány pro vlastní použití. Stavy přepínačů a jejich čísla se pak např. objevují ve výpisu debugfs, ale při zadání jejich čísla do souboru */sys/class/gpio/export* nedojde k vytvoření složky příslušného přepínače v */sys/class/gpio*.

Jelikož je stav přepínače zapisován do souboru, umístěného ve stálé složce OS, je možné stav přepínačů v rámci aplikace zjišťovat pouhým čtením tohoto souboru. Před prvním spuštěním aplikace po startu OS je však potřeba exportovat řízení a informace o stavu přepínače. Pro usnadnění těchto úkonů byly napsány skripty, které stačí před spuštěním aplikace po spuštění OS provést. Skripty i principy v této kapitole uvedené jsou pak dokumentovány v příloze práce.

8 Ovládání displeje

Výstup DSP algoritmu je zobrazován na displeji vývojové desky. Displej je v aplikaci používán v režimu framebuffer. V takovém případě jsou do bufferu jeho vstupních hodnot zapisovány parametry jednotlivých pixelů obrazu podle definovaného vstupního formátu displeje. Formáty podporované displejem, jeho rozlišení a obnovovací frekvenci je možné zjistit pomocí volání utility *fbset* [4], konkrétně společně s parametrem --*info*. [25]

```
root@arago:~# fbset --info

mode "480x272-52"
    # D: 7.895 MHz, H: 15.038 kHz, V: 52.214 Hz
    geometry 480 272 480 544 16
    timings 126667 2 2 3 3 41 10
    rgba 5/11,6/5,5/0,0/0
endmode

Frame buffer device information:
    Name        : DA8xx FB Drv
    Address     : 0xc7580000
    Size        : 524288
    Type        : PACKED PIXELS
    Visual      : TRUECOLOR
    XPanStep    : 0
    YPanStep    : 1
    YWrapStep   : 0
    LineLength  : 960
    Accelerator : No
```

Podstatné informace z výpisu jsou následující:

- rozlišení displeje - 480 x 272 pixelů
- frekvence snímků - 52 Hz

[4] Součást výchozího souborového systému používaného v DVSDK.

- barevný mód displeje - RGB565 (výpis vždy určuje počet bitů pro barevnou složku a od kterého bitu tato složka, v rámci 16 bitů pro pixel, začíná - tedy např. rgba 5/11 znamená, že R složka má 5 bitů a začíná na 11 bitu)
- délka dat pro jednu řádku displeje (LineLength) je 960 byte

Formát RGB565 pro každý pixel obrazu, skládající se ze 3 barevných složek (R - červená, B - modrá, G - zelená) používá nastavení zapsané do velikosti 2 byte - neboli do 16 bitů. Pro dynamický rozsah jednotlivých barev pak připadá následující počet bitů: R - 5 bitů, G - 6 bitů, B - 5 bitů.

Aby bylo možné výstupní snímky DSP algoritmu i dále zpracovávat, je potřeba zachovat dostatečný dynamický rozsah jednotlivých barev zpracovaných snímků. Proto je jako výstup z DSP použit formát RGB24, kdy jsou jednotlivé barevné složky reprezentovány vždy 8 bity. Každý pixel je pak zakódován pomocí 3 byte.

Kvůli rozdílnosti zobrazovacího a výstupního barevného formátu je nutné pro zobrazení formát převést. Vývojová deska neobsahuje HW, který by toto umožňoval, proto je nutné provádět konverzi v SW. Díky podobnosti obou barevných prostorů však stačí pro každý zobrazovaný pixel pouze bitovým maskováním omezit dynamický rozsah jednotlivých barevných složek a následně pomocí bitových posunů složit výslednou podobu obou zobrazovacích byte každého pixelu.

Zobrazovací TFT panel má také omezené rozlišení, maximálně 480 x 272 bodů. Výstup z DSP algoritmů je potřeba získávat v největším možném rozlišení, aby bylo možné z nich spolehlivě určovat stav jednotlivých parkovacích míst. Kvůli neodpovídajícímu rozlišení výstupů z DSP a zobrazovacího panelu je opět potřeba softwarově řešit konverzi. Jelikož je zobrazení výsledků na displeji vývojové desky pouze informativní, není nutné aplikovat složitější algoritmy zachovávající maximální věrnost obrázku. Byla tedy napsána zobrazovací funkce, při níž je nejdříve zjištěn dostatečný celočíselný koeficient, kterým je nutné dělit rozměry (šířka, výška) obrázku, aby byla velikost daného rozměru menší než maximální zobrazitelná. Z obou koeficientů je pak vybrán ten větší. Pro zobrazení dvou po sobě následujících pixelů je pak v původním obrázku vynechán počet pixelů daných koeficientem. Stejně tak je vynechán pro každou novou zobrazovanou řádku koeficientem daný počet řádek původního obrázku. Shodností koeficientu použitého pro redukci řádek i sloupců obrázku je pak zachován poměr stran obrázku.

Pro minimalizaci náročnosti výpočtů a tím i doby trvání zobrazování byl návrh smyčky volen tak, aby nejfrekventovanější výpočetní smyčka, konkrétně smyčka

zpracovávající data řádků, prováděla jen nezbytné operace. Koeficienty pro její operace jsou tak počítány vně této smyčky.

Základní řízení displeje je použito z aplikace *fbtest* [26]. Ta obsahuje pouze způsob jak zobrazit na displeji testovací čtverce. Byla tedy rozšířena o konverzi z RGB24 a o konverzi rozlišení snímků. Princip zobrazení je pak volen tak, že jsou nejdříve výpočty všech hodnoty obrazových bodů do vyrovnávacího bloku paměti a vypočtené hodnoty jsou pak zobrazovači předány až v závěru funkce jako celek.

Při používání vývojového kitu byla zjištěna závada řízení displeje. Po vypnutí a zapnutí napájení naběhne řízení displeje správně, ale pokud je pouze stisknuto reset tlačítko desky, naběhne přibližně v 50% případů pouze podsvícení displeje. V ovladačích se tento problém nepodařilo vyřešit, podařilo se však zjistit možnost jak tento problém "obejít" a to pomocí sekvence vypnutí a zapnutí displeje řízené softwarově. Sekvence byla následně zapsána do skriptu, který stačí provést, pokud je displej po spuštění OS pouze bílý. Skript je součástí přílohy práce.

9 Kamera

Kameru je možné k desce připojit více způsoby, např. pomocí USB, nebo pomocí speciálního VPIF portu. Při hledání kamery, u které by byla ověřená funkčnost s vývojovým kitem OMAPL138 experimenter, se však nepodařilo takovou kombinaci najít. USB kamery, zprovozněné s procesorem OMAPL138, byly používané pro typ vývojové desky OMAPL138 LCDK. To je novější platforma, jejíž podpůrný software má navíc (omezenou) podporu až do současnosti a pro kterou je adaptována i novější varianta jádra OS Linux. S používanou vývojovou deskou byl pak nalezen pouze jeden příklad funkčního propojení s kamerou. V tomto příkladě je však kamera připojována přes konektory rozšiřující desky dodávané pouze s vývojový kitem OMAPL138 EVM. Možnost dokoupení rozšiřující desky se nepodařilo dohledat. To je dáno hlavně tím, že platformy OMAPL138 EVM i experimenter jsou již dva roky ve statusu EOL (End Of Life) - je ukončena jejich produkce. Navíc kamera, s kterou byl příklad pro EVM kit demonstrován, již není vyráběna jako hotový modul, nýbrž pouze ve formě samotného čipu.[2][3][27][28]

Vzhledem k uvedeným skutečnostem byla zvolena USB kamera Logitech HD720p. Ovladače pro kamery Logitech jsou v OS Linux řešeny UVC moduly, které jsou součástí složek jádra OS. Tento modul je obsažen i v souborových složkách jádra OS v DVSDK [46]. Funkčnost UVC modulů ve spojení s použitou kamerou byla ověřena snímáním obrázků z kamery na PC. Ověření bylo provedeno jak v distribuci s novějším jádrem OS (Ubuntu 16.04), tak v distribuci Ubuntu 10.04, která používá dokonce starší jádro (2.6.32).[5] Kvůli podstatným rozdílům HW architektury PC a zabudovaných procesorů však funkčnost s PC neimplikuje funkčnost na vývojovém kitu.

Kamera Logitech HD720p podporuje dva typy výstupů - komprimovaný MJPEG a nekomprimovaný YUYV formát. Pro zachování kvality obrazu a možnosti přímého použití nasnímaných dat pro analýzu je vhodný nekomprimovaný formát. Naopak v

[5] Snímání fungovalo správně pouze pokud byl systém spouštěn jako nativní OS. Při spouštění OS ve virtualizačním software docházelo k chybám snímání, způsobeným zřejmě omezenou funkčností USB ovladačů v tomto režimu provozu. Pro Ubuntu 10.04 se podařilo test zprovoznit i při zavedení OS z tzv. LiveCD - specifické formy zavádění OS u Linux distribucí, umožňující spouštění OS pouze ze samotného CD, bez nutnosti instalace OS

případě HW komplikací s rychlostí komunikace by byl pro svou nižší přenosovou rychlost vhodnější komprimovaný formát obrazu.

Po nastavení UVC modulů do konfigurace jádra a jeho zkompilování byla kamera nejdříve připojena k USB 1.1 (kvůli přímé kompatibilitě konektorů kamery a USB 1.1 portu vývojové desky). To již v rámci své specifikace omezuje rychlost komunikace s kamerou na 1.1 MB/s.

V portu USB 1.1 se podařilo zprovoznit nekomprimovaný formát pouze do velikosti snímku 160 x 120. Naopak komprimovaný formát do velikosti 320 x 240 byl zprovozněn bez komplikací.

Následně byla kamera zapojena do USB portu 2.0. Kvůli nekompatibilitě konektorů byla použita krátká redukce z USB 2.0 B na USB 2.0 mini A. Podařilo se však zprovoznit pouze rozlišení 1280 x 960 a to navíc v komprimovaném módu. Navíc nemohl být při snímání použit DMA (kanál přímého přístupu do paměti), který by umožnil, aby procesor mohl během snímání vykonávat jinou činnost. Při snímání bylo také alokováno velké množství paměti, např. pro snímek 1280 x 960 bylo alokováno 45 MB. Jednotlivé problémy jsou společně s podstatnými výpisy terminálu rozebrány v příloze práce.

Objevil se ještě problém při cyklickém snímání. Kvůli principu aplikace uvccapture použité na snímání obrázků docházelo k opakovanému spouštění a zavíraní komunikace s kamerou, kdy nastávaly nedokončené přenosy.[6] To bylo ve vlastní aplikaci vyřešeno otevřením komunikace před vlastní smyčkou programu a zavřením komunikace až v závěru celé aplikace. Chyba je dokumentována v příloze práce.

Původní aplikace na snímání obrázků navíc ukládá obrázky do souborového systému OS. Ze souborového systému by pak musel být obrázek vlastním vytvářeným programem vyčten a předán do bufferu pro DSP. Adaptování snímání kamerou do kódu vlastního programu umožnilo, aby byl obrázek z kamery předáván přímo do bufferu pro DSP bez mezikroku do souborového systému.

[6] Toto by bylo možné vyřešit použitím jiné snímající aplikace. Aktuální byla zachována kvůli jejímu snadnému začlenění do vytvářeného programu.

10 JPEG kodek

Obrázky z kamery se podařilo zprovoznit v dostatečném rozlišení pouze v komprimovaném MJPEG formátu. MJPEG je formát, který komprimuje jednotlivé snímky videa do JPEG formátu. Narozdíl od kompresních formátů např. MPEG-2 tedy neumožňuje rozdílové snímky či snímkovou predikci. Na jeho dekodování však stačí standardní JPEG dekodér. Navíc je každý takový snímek dekódovatelný sámostatně - není nutné mít k jeho dekódování i další snímky, jako je to v případě rozdílových či predikčních snímků jiných formátů.

Formát JPEG je možné dekódovat jak pomocí ARM procesoru, tak pomocí DSP. Pro aplikaci byl zvolen kodek firmy TI, který je k dispozici zdarma. Kodek vytvořený TI je součástí tzv. XDAIS algoritmů. Ty byly vytvořeny právě TI a definují formáty struktur pro řízení operace kodeku. Pokud vývojáři externích kodeků dodrží způsob rozdělení algoritmu pro možnost použití formátu těchto struktur, je možné TI kodek v aplikaci kompletně nahradit externím pouze nahrazením informací v těchto strukturách. Tím je pak možné rychle porovnávat efektivitu různých kodeků v konkrétních aplikacích.

Použitý TI dekodér JPEG je optimalizován pro C6000 řadu procesorů. Umožňuje několik typů výstupů:

- RGB24
- RGB16 (RGB565)
- různé varianty YUV formátu

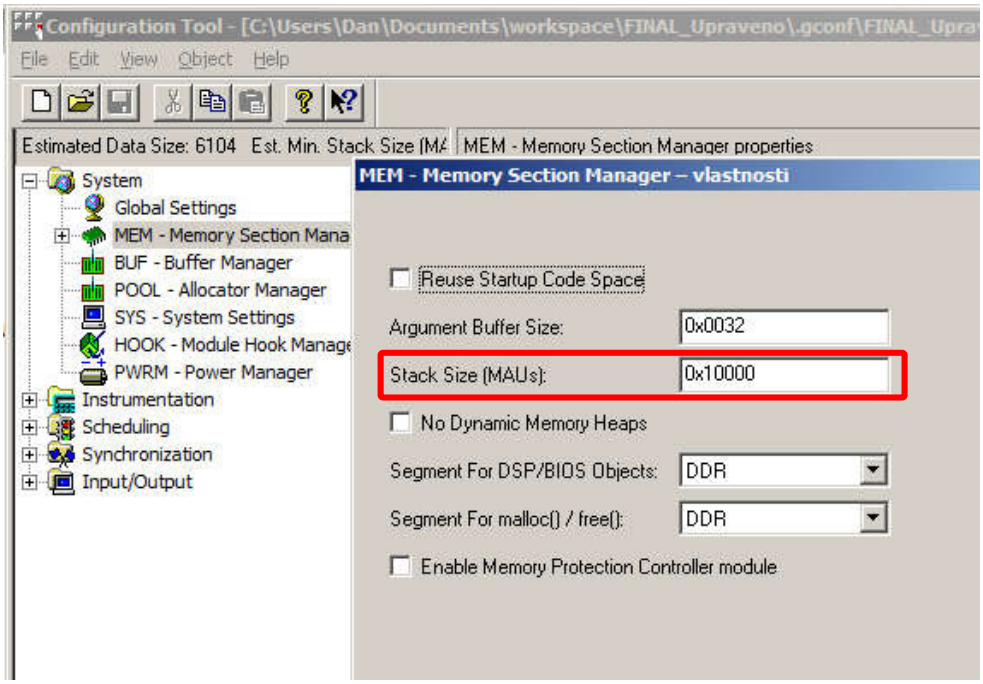
Součástí balíčku, v němž je dekodér distribuován je i vzorový kód jeho použití. Ten byl upraven pro snadné navázání na meziprocesorovou komunikaci. Konkrétně, v původním kódu jsou obrázek i konfigurační textový soubor pro dekódér čteny přes rozhraní JTAG ze souborového systému vývojového PC. Vzorový program byl tedy upraven tak, aby obrázek byl čten z bufferu ve sdílené paměti mezi procesory a aby potřebné konfigurace pro dekódování byly předávány pomocí meziprocesorové komunikace.

Součástí hlavičky snímků komprimovaných pomocí JPEG je i šířka a výška příslušného obrázku. Tyto parametry tedy není nutné pro dekódování snímků zadávat. Naopak šířka a výška dekodovaného obrázku jsou jedny z výstupních dat dekodéru. Aby bylo možné používat testovací obrázky předem neurčených rozměrů, je šířka a výška dekodovaného obrázku předávána do následujícího algoritmu řešícího negativ

obrázku a také do ARM kvůli možnostem zobrazení a pro určení konce obrázku ve výstupním bufferu. Aby bylo možné snadno navázat na DSP další algoritmy pracující s obrázkem, je výška a šířka obrázku zapsána do informační struktury ve sdílené paměti, na níž si algoritmy a jednotlivé části procesů na DSP předávají ukazatel. Tento způsob umožňuje předávání většího množství dat mezi algoritmy bez nutnosti jejich kopírování. Kvůli předání rozměrů do ARM procesoru jsou pak rozměry obrázku zapsány i do meziprocesorové zprávy.

```
/*
 * zapsani rozmeru dekodovaneho obrazku do informacni
 * struktury sdilene mezi jednotlivymi algoritmy na DSP
 * a castmi jednotlivych procesu
 *
 */
info->sirkaObr = (Uint16) status.imgdecStatus.outputWidth;
info->vyskaObr = (Uint16) status.imgdecStatus.outputHeight;
```

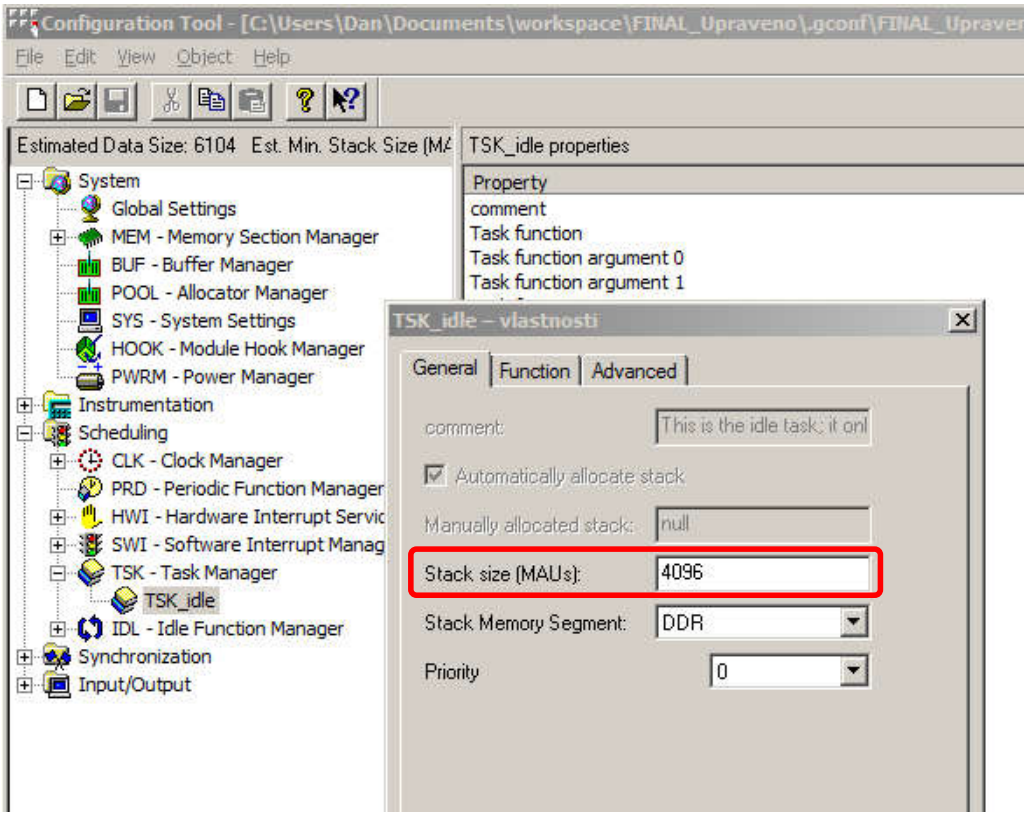
Kodek pro svou funkci potřebuje dostatečnou velikost zásobníku (stack). Velikost, která byla nastavena ve vzorové aplikaci je 0x2000, tedy 8192 byte (je zapsána v konfiguračním souboru vzorového programu **JPEGDecApp_OMAP3530.cmd**). Konfiguraci velikosti zásobníku je možné provést pomocí GUI konfigurátoru viz *Obr. 10*.



Obr. 10 Konfigurace velikosti hlavního zásobníku

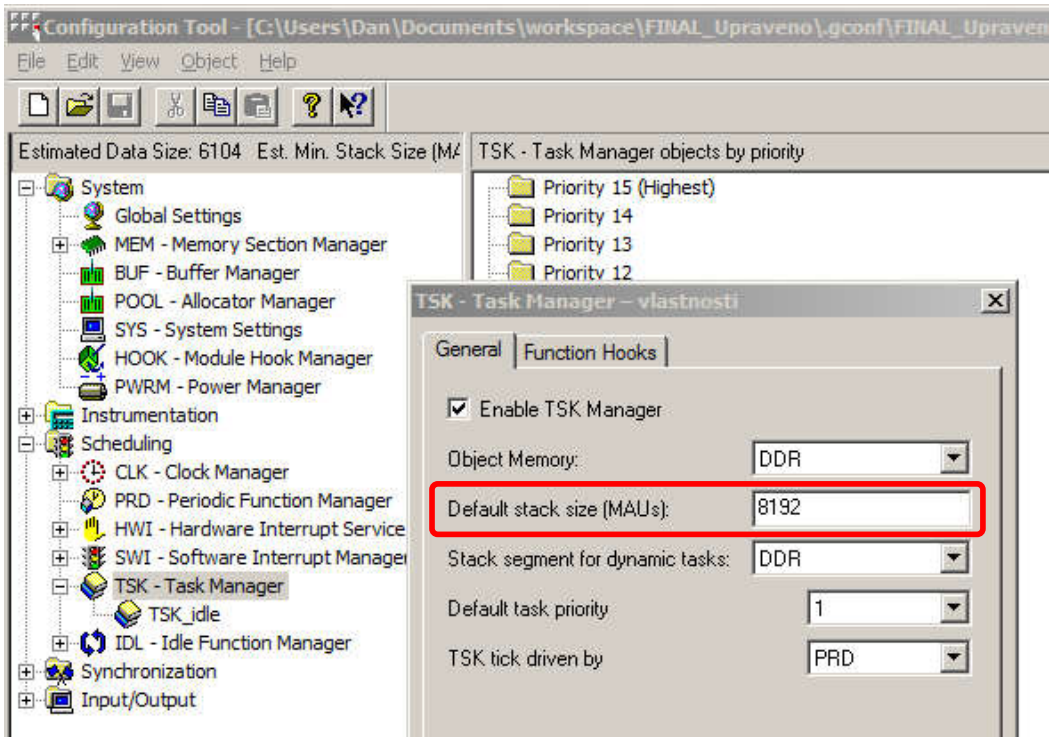
Tato hodnota se však týká pouze hlavního zásobníku, sdíleného například HWI a SWI typy procesů. Kodek je však spouštěn uvnitř dynamicky vytvářené úlohy (Tasks).

Každá úloha má svůj vlastní zásobník, jak již bylo zmíněno v kapitole o DSP/BIOS. Ty úlohy, které jsou vytvářené staticky (tedy ještě před startem programu), mají vlastní pozici v GUI konfigurátoru, kde je možné konfigurovat velikost jejich vlastního zásobníku, viz *Obr. 11*.



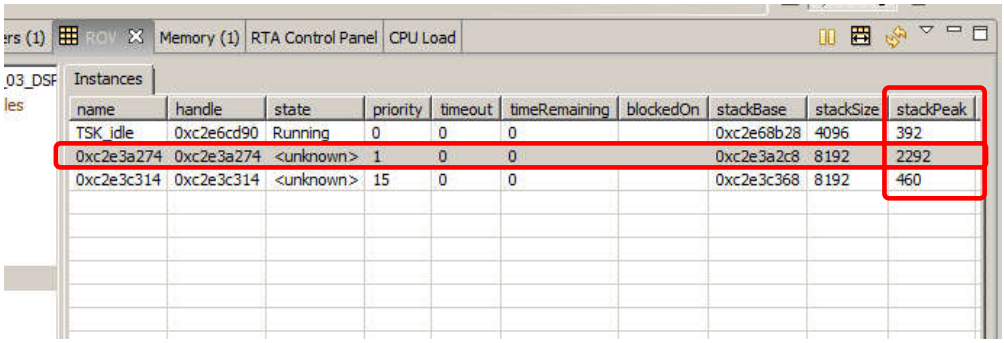
Obr. 11 Konfigurace zásobníku staticky vytvářené úlohy

Kodek je však v aplikaci navázán do původní formy DSPLink vzorové aplikace readwrite. V ní výpočetní úloha vytvářena dynamicky v kódu a jako taková tedy nemůže mít v GUI konfigurátoru svou pozici. V GUI je však možné konfigurovat výchozí velikost zásobníků vytvářených úloh, která platí i pro dynamicky vytvářené úlohy, viz *Obr. 12*. Ta je v základu nastavena na velikost 1024 byte. Tato hodnota je tedy pro funkci kodeku nedostatečná a dokud nebyla změněna, tak způsobovala nevyzpytatelné chování aplikace. Byla tedy zvětšena na 8192 byte.



Obr. 12 Konfigurace výchozí velikosti zásobníků úloh

To, že je nastavená hodnota dostačující, bylo ověřeno pomocí jednoho ze statistických nástrojů DSP/BIOS - konkrétně pomocí nástroje ROV, zmíněného v kapitole o DSP/BIOS. Jedním z parametrů, který tento nástroj vypisuje, je právě špičková hodnota ukazatele pozice v zásobníku (v Obr. 13 Záznam špičkové hodnoty ukazatele pozice zásobníku označena StackPeak). Záznam statistiky byl sejmut po proběhnutí analýzy několika snímků, při řízení běhu programu přes rozhraní JTAG. Alternativně je možné nastavit velikost zásobníku dynamicky vytvářené úlohy v kódu před jejím vytvořením. Zde popsaný způsob byl však ponechán, jelikož lépe nastiňuje komplexnost systému a také ukazuje výhody statistického nástroje ROV, který pomohl k odhalení příčiny chyby.



Obr. 13 Záznam špičkové hodnoty ukazatele pozice zásobníku

11 Negativ snímku

Jako jednoduchý algoritmus probíhající na DSP byl zvolen negativ snímku. Vytvoření negativu vyžaduje pouhé invertování úrovně jasu jednotlivých bodů obrazu. Při 8 bitovém kvantování úrovní jasu (tedy 256 kvantizačních hladin, při indexování od hodnoty 0 pak číslo 255) všech složek obrazu jej tedy lze popsat rovnicí:

$$negativ[i] = 255 - jas[i]$$

Výpočetní náročnost takové operace je minimální. Jelikož však DSP jádro obsahuje 8 vnitřních jednotek a sofistikovanou architekturu umožňující značnou paralelizaci operací procesoru potřebných k výpočtu, je potřeba rovnici zapsat tak, aby se možnosti jádra maximálně využily.

TI poskytuje k procesoru OMAPL138 i několik druhů knihoven s optimalizovanými verzemi funkcí, které jsou standardní součástí DSP algoritmů. Jednou z nich je knihovna IMGLIB [23], která obsahuje ručně optimalizované funkce typické pro zpracovávání obrazu. Její součástí jsou např. funkce na provedení histogramu, kosinových transformací, či mediánu bloku sousedících pixelů. Jednotlivé funkce mají navíc uvedeny i doby trvání pro definovaný počet řádek a sloupců vstupního obrazu, čímž umožňují plánovat využití procesoru, či porovnávání efektivity vlastních či externích algoritmů. Knihovna však neobsahuje přímo funkci pro negativ snímku. V závěru dokumentace knihovny je však uveden soubor menších optimalizovaních výpočetních smyček, z nichž je možné vytvářet další funkce. Jednou z nich je i smyčka na odečtení předem definované konstanty od bloku vstupních 8 bitových čísel (funkce IMG_subS_8), symbolicky:

$$rozdíl[i] = vstup[i] - konstanta$$

Při nastavení hodnoty konstanty na číslo 255 a přehození pořadí operandů v kódu funkce je tak možné získat funkci provádějící negativ, optimalizovanou specificky pro použitou architekturu. Rychlost provádění této funkce je v dokumentu určena 8 rozdílů/cyklus. Tato rychlost je však určena pouze s ohledem na rychlost výpočtů v procesoru a neuvažuje rychlost paměťových transakcí. Podle umístění hodnot v paměti se pak doba trvání funkce příslušně prodlouží.

Ještě je nutné poznamenat, že ukazatele na vstupní a výstupní data, zadávané do volání funkce na adresy, jsou definovány s klíčovým slovem jazyka C restrict. To oznamuje překladači, že vstupní i výstupní sekce dat se v žádném bodě celého pole v

němž bude výpočet probíhat nepřekrývají. Jen díky tomu je možné zvolit použitý druh optimalizací. Pokud však budou jako vstupní a výstupní sekce zadány překrývající se oblasti, pak není správnost výsledku zaručena a bude záviset na způsobu překryvu obou sekcí.

V aplikaci je výpočet negativu navázán přímo za dekódování JPEG obrázku. Součástí argumentů funkce realizující negativ jsou ukazatele na vstupní a výstupní buffer dat. Vstupním bufferem je tedy počáteční adresa bufferu, do nějž zaznamenal svůj výstup dekodér JPEG. Výstupním bufferem je pak jiná sekce ve sdílené paměti, která nesmí být v kontaktu se vstupním blokem. Argumentem funkce je i množství byte, pro něž má být výpočet proveden. Tato hodnota je určena velikostí a počtem barevných složek obrázku. Jelikož velikost obrázku je předem nedefinovaná (viz kapitola dekodéru JPEG), je spočtena před voláním funkce negativ. Výpočetní rovnice má následující charakter:

$$velikost\_obrazku = sirka \times vyska \times 3$$

kde číslo 3 reprezentuje tři barevné složky obrazu (R, G, B) a kde šířka a výška obrazu je získána z informační struktury, do níž je zapisována JPEG dekodérem.

Jelikož výstup negativu je v jiné oblasti paměti, než výstup dekodéru, musí být ARM procesoru předána informace z jaké oblasti má číst výsledná data, podle toho jestli je nebo není negativ prováděn. Kvůli tomu byla do meziprocesorové zprávy přidána proměnná, která slouží jako přepínač mezi výstupními buffery. Pokud není prováděn negativ, tato proměnná je nastavena na hodnotu 1, signalizující, že ARM procesor má výstupní data číst z prvního výstupního bufferu. Pokud negativ prováděn je, pak je po jeho dokončení před předáním meziprocesorové zprávy tato proměnná nastavena na hodnotu 2. Ta obdobně signalizuje, že výstupní data jsou v druhém výstupním bufferu. Oba výstupní buffery jsou alokovány na začátku aplikace ARM procesorem. Ten má počáteční adresy obou bufferů uloženy a tím je možné mezi nimi uvedeným způsobem přepínat. Systém přepínání demonstruje následující výňatek z řídícího kódu DSP:

```
            /* prepnuti vystupniho bufferu pro GPP */
            info->prepinacBuf = 2;
        } /* if(info->provedNegativ == 1) */

        /*
         * do meziprocesorove zpravy predan az posledni
         * stav promenne signalizujici buffer v nemz se
         * nachazi vysledek
         *
         */
        msg->prepinacBuf = info->prepinacBuf;

        /* Odeslani zpravy na GPP. */
            status  =  MSGQ_put  (info->gppFrontaZprav,  (MSGQ_Msg)
msg) ;
```

Při realizaci se objevily problémy u konce obrázku. Určité množství koncových pixelů, u větších obrázků menší procento obrázku a u menších obrázků větší procento, bylo vždy bílé (v případě pouze jedné iterace meziprocesorové smyčky), či obsahovalo pouze data z předchozího obrázku. To bylo způsobeno charakterem fungování pamětové architektury C674x procesoru. V aplikaci byla pro dodávání dat do procesoru využívána cache. Vzhledem k velikosti obrázků (při 1280 x 720 x 3 = 2,7 MB) není možné je zapsat celé do L2 cache (max. 256 kB) [7]. Obrázky tedy musejí být umístěny v externí paměti procesoru. S tou je však pomalá komunikace, která by vedla ke snížení efektivity algoritmů. Kopie bloků dat potřebných pro algoritmus jsou proto postupně z externí paměti umisťovány do rychlé cache paměti. To je realizováno již v hardwaru. Výsledky výpočtů procesoru jsou pak také udržovány pouze v cache a aby se omezilo množství komunikace s pomalou externí pamětí, jsou data z cache zapisována do externí paměti až když je nutné v cache uvolnit místo pro nová data.[7] Jelikož však mezi zpracováním posledních bodů obrazu a mezi předáním dat ARM procesoru nejsou DSP procesorem zpracovávána další data, tak nedojde k automatickému zapsání vypočtených dat do externí paměti. Kvůli tomu pak koncové části obrazu obsahují neplatná data. Proto je před předáním zprávy do ARM procesoru nutné volat funkci realizující tzv.

[7] Systém udržování synchronního obsahu v jednotlivých vrstvách paměti (v manuálech TI nazýváno cache coherence protocol) je v případě C674x procesoru velmi komplexní - záleží např. jestli je přistupováno do L2 cache nebo L2 SRAM, jestli se do komunikace zapojuje DMA, atd. Podle jednotlivých podmínek se pak chová hardware zařizující synchronizaci obsahu různě. Navíc tyto parametry velmi ovlivňují i rychlost komunikace s pamětí - je např. rozdíl mezi rychlostí komunikace s L2 cache vůči L2 SRAM. Podrobnosti jsou k nalezení v TI dokumentech [7][8].

cache writeback, která umožní softwarově zahájit zápis všech změněných dat z cache do externí paměti.

```
/*
 * Po dokonceni operace je potreba provest
 * programem volat prikaz zajistujici tzv. cache koherenci,
 * neboli zapsani obsahu cache do nizsich vrstev pameti.
 *
 * Pokud neni prikaz proveden, jsou z koncove oblasti
 * obrazku vyzvednuta GPP stara data, jelikoz cache
 * neni nucena nahradit data ktera v sobe obsahuje.
 *
 */
BCACHE_wbInv ((Ptr)(info->vystupBuf2), velikost_obrazku, 1) ;
```

12 DSPLink rozdělení paměti

Jak již bylo uvedeno v kapitole o DSPLink, pro komunikaci pomocí DSPLink je potřeba vhodně nakonfigurovat také paměťové rozložení. V paměti sdílené oběma procesory se nachází jak jádro a uživatelský prostor Linux (včetně bufferů aplikací), tak i prostor pro kód DSP, řídící struktury meziprocesorové komunikace a buffery použité k předávání dat mezi procesory.

Velikost sdílené RAM na vývojové desce je 128 MB. Pro jádro bylo vyhrazeno 32 MB prostoru (určeno experimentálně při provozu jádra). Pro kód DSP byly vyhrazeny 4 MB (velikost přeložené aplikace >1 MB + rezerva pro případné rozšíření aplikace). Pro předávání obrázků byly přiděleny 3 buffery - jeden vstupní a dva výstupní (mezi kterými aplikace přepíná). Maximální možná velikost obrázků z kamery je 1280 x 960 x 3 (3,7 MB), velikost bufferů však byla zvolena s ohledem na možnost vkládání testovacích obrázků 1920 x 1080 x 3 (6 220 800 byte).

Další buffery jsou však alokovány při snímání kamerou. Velikost alokovaných bufferů je určena velikostí obrázku. Jak ukazuje příloha práce zobrazující zatížení procesoru při snímání kamerou, v případě obrázku o velikosti 1280 x 720 jsou alokovány buffery 33 MB, v případě obrázků 1280 x 960 pak 55 MB. Pokud by se podařilo snímat správně i snímky 1920 x 1080, pak by alokovaná paměť byla 72 MB.

Bloky paměti pro kameru jsou alokovány z paměťového prostoru pro Linux, jejich velikost je tedy nutné do jeho paměťového prostoru zahrnout. V případě nedostatečného množství paměti pro kameru totiž způsobí chybu nedostatku paměti (OOM, Out Of Memory). Ta pak vede na zablokování jakékoli další možnosti ovládání procesoru a je nutné provést reset.

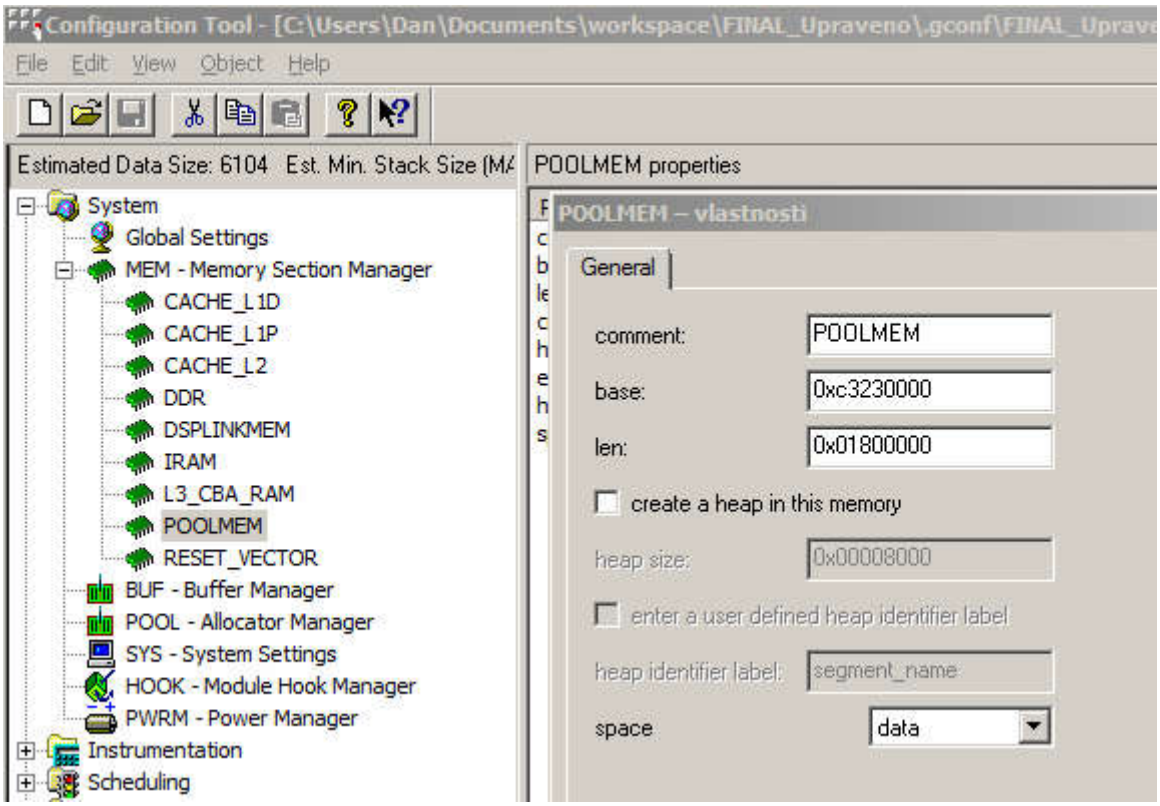
Sdílená paměť je v rámci fyzických adres 0xc000 0000 - 0xcFFF FFFF (viz Přílohy). Jelikož však vývojový kit má instalováno pouze 128 MB paměti, použitý rozsah je pouze 0xc000 0000 - 0xc7FF FFFF. Podle požadavků aplikace tedy bylo zvoleno rozložení paměti podle tabulky Tab. 1.

Linux (včetně prostoru pro kameru)	96 MB	0xC000 0000 - 0xC200 0000 (32 MB)
Reset vektor	80 B	0xC2E0 0000 - 0xC2E0 007F
Oblast pro DSP kód (blok DDR)	4 MB	0xc2E0 0080 - 0xC31F FFFF
Oblast pro sdílené řídící struktury (blok DSPLINKMEM)	1 MB	0xC320 0000 - 0xC32F FFFF
Oblast pro předávaná data (3 x 6,2 MB, blok POOLMEM)	18,5 MB	0xC323 0000 - 0xC53F FFFF
Linux (druhá část)	40 MB	0xC540 0000 - 0xC78F FFFF

Tab. 1 Rozložení paměťové mapy [8]

Jednotlivé části paměťové mapy pak byly nastaveny v konfiguracích pro ARM a DSP v rámci DSPLink a také pro Linux. Pro DSP stranu DSPLink jsou tyto parametry nastavitelné v GUI konfiguračním souboru, principem jako v *Obr. 14*.

[8] Důvodem rozdělení paměti pro Linux je původně plánovaný větší prostor pro komunikaci s DSP. Po zjištění množství paměti alokovaného snímáním kamery, které při nedostatku místa v RAM způsobovalo chybu OOM, byl tento prostor jednoduše



Obr. 14 Konfigurace bloků sdílené paměti v Code Composer Studio
parametr base nastavuje výchozí adresu bloku, parametr len délku bloku

Pro ARM stranu byla zvolena statická konfigurace [9], potřebné parametry tedy byly nastaveny v konfiguračním souboru uvedeném v kapitole 4.4 a po jejich nastavení byla znovu přeložena knihovna DSPLink i samotná aplikace a soubory obou pak byly nahrány do souborového systému vývojové desky. Nastavení jednotlivých částí konfiguračního souboru je následující:

```
#define  RSTENTRYID          0
#define  RESETCTRLADDR       0xC2E00000
#define  RESETCTRLSIZE       0x80

#define  CODEENTRYID         1
#define  CODEMEMORYADDR      (RESETCTRLADDR + RESETCTRLSIZE)
#define  CODEMEMORYSIZE      0x3fff80u

#define  SHAREDENTRYID0      2
#define  SHAREDMEMORYADDR0   (CODEMEMORYADDR + CODEMEMORYSIZE)
#define  SHAREDMEMORYSIZE0   0x5000
```

[9] Původně byla zamýšlena dynamická konfigurace pro své logickou výhodnost. Při změnách paměťové mapy však docházelo k nefunkčnosti překládaných programů. Proto byla zvolena jistější statická konfigurace. I při ní však docházelo k problémům. Důvody problémů byl pravděpodobně selhávající harddisk vývojového PC, který byl později zjištěn.

```
#define  SHAREDENTRYID1      3
#define  SHAREDMEMORYADDR1   (SHAREDMEMORYADDR0 + SHAREDMEMORYSIZE0)
#define  SHAREDMEMORYSIZE1   0x2B000

#define  POOLENTRYID         4
#define  POOLMEMORYADDR      (SHAREDMEMORYADDR1 + SHAREDMEMORYSIZE1)
#define  POOLMEMORYSIZE      0x1800000u
```

Parametry pro nastavení paměťového prostoru jsou zadávané do zavaděče OS U-Boot. Konkrétně parametry nastavující paměť jsou následující:

```
mem=32M@0xc0000000 mem=40M@0xc5400000
```

kde 32M značí velikost paměťového bloku a hexadecimální číslo za @ označuje paměťovou adresu od níž má být blok alokován.

13 Možnosti analýzy parkovacích stání na OMAPL138 [18]

V předcházejících pracích na projektu již byl napsán potřebný algoritmus na analýzu parkovacích stání. Problémem je však doba trvání jeho vykonávání. Proto by měl být program přenesen na vhodnější výpočetní hardware. PC variantu algoritmu zpracovávajícího snímky parkoviště by bylo možné převést do OpenCV. OpenCV je open-source knihovna zaměřená na zpracovávání obrazu. Knihovna je napsána v jazyce C++ a je možné ji volat i aplikacemi v psanými v jazyce C nebo Python. Je trvale vyvíjena a dochází tedy k přidávání nových funkcí či k opravám chyb. Je však cílena na architekturu PC a její funkce musejí být pro možnost použití na zabudovaných procesorech upravovány.

OpenCV je přeneseno např. na ARM architekturu procesorů. U jiných architektur je pak možné pouze přeložit zdrojové kódy OpenCV [10] a použít tento výsledek se zabudovaným procesorem, ale specifika těchto architektur způsobují při tomto procesu mnohé komplikace a tímto způsobem přenesená knihovna také nebude zdaleka optimální co se týče rychlosti.

TI nabízí pro TI DSP vlastní knihovnu IMGLIB, obsahující algoritmy na zpracovávání obrazu optimalizované konkrétně pro architekturu těchto procesorů. IMGLIB obsahuje různé OpenCV podobné funkce, rozsah IMGLIB či její aktualizovanost však nelze s OpenCV srovnávat. [23]

Možností jak funkce OpenCV na DSP optimálně přenést potom je nahradit pomocí intrinsických funkcí procesoru a pomocí bloků optimalizovaných knihoven TI podstatné části algoritmu napsaného v OpenCV. To by v případě rozsáhlých algoritmů mohlo vyžadovat velké množství času, s nejistým výsledkem. Pro DSP procesor řady Keystone II [32] jsou však již optimalizované verze OpenCV knihoven k dispozici [19]. Tento procesor obsahuje 8 DSP jader, umožňujících běžet na vyšších frekvencích než jádro na OMAPL138 (v procesoru Keystone >1GHz, oproti OMAPL138 - max 456 MHz) a zpracovávat tak data rychleji. Keystone má také rychlejší paměť (DDR3). Ta bude pro maximální rychlost provádění algoritmů detekce volných parkovacích míst potřebná. Původní program totiž obsahuje části, kde je nejdříve potřeba zpracovat jednou funkcí

[10] Open-source programy mají volně dostupný zdrojový kód

celý snímek a až na výsledky této funkce navázat další funkcí. [11] Tím pak není možné využít takového zpracování snímků po blocích, kdy by nad daným blokem byly provedeny všechny funkce algoritmu najednou. To by pak bylo možné provést pouze v rámci cache paměti a tedy s maximální rychlostí a až po jejich celém zpracování přesunout výsledek do externí paměti a zpracovat další blok (jak je tomu například u kompresních algoritmů). Vzhledem k časové náročnosti původní aplikace, vysokým časovým nárokům a zde uvedeným důvodům tak bylo rozhodnuto, že zpracovávání bude přeneseno až na tento novější a podstatně výkonnější typ procesoru.

[11] Tento problém je způsoben tím, že algoritmus rozpoznávání obsazených míst potřebuje pro svou funkci pracovat s celým obrazem auta. Jelikož jednotlivá auta mohou parkovat například částečně na dvou parkovacích místech, nelze rozdělit parkoviště na bloky obrazu jednotlivých parkovacích míst.

14 Závěr

Cílem práce bylo zjistit možnosti přenosu algoritmu na detekci obsazenosti parkovacích míst z architektury PC na procesor OMAP-L138.

Úvodní kapitoly práce se zabývají rozborem parametrů procesoru OMAP-L138 a vývojových nástrojů potřebných k jeho programování. Ty obsahují především popis způsobu meziprocesorové komunikace na tomto procesoru, přičemž se věnují jak popisu jejích vnitřních principů, tak i způsobu lazení programu a možností překladu programů při tomto způsobu práce s uvedeným procesorem. Zabývají se také hlubším rozborem softwarových komponent použitých v praktickém řešení.

Praktická část práce pak navazuje řešením částí aplikace potřebných k . Během této je vidět především rozsáhlost problematiky řešení, týkající se jak množství softwarových tak hardwarových specifik architektury.

Součástí práce je i průzkum možností přímého přenosu algoritmu na DSP procesor. Kvůli nedostupnosti optimalizovaných knihoven použitelných společně s PC, jako jsou OpenCV, by však přenos vyžadoval nahrazování jednotlivých vnitřních funkcí PC algoritmu optimalizovanými alternativami dostupnými pro použité DSP. Kvůli tomu bude další vývoj projektu směřován na architekturu procesoru KeyStone, který nejen podstatně výkonnější, ale jsou pro něj poskytovány i optimalizované verze OpenCV knihoven.

Ačkoli se praktické řešení podařilo zprovoznit, vzhledem k rozsáhlosti teoretických i praktických poznatků potřebných k řešení problematiky se nepodařilo včas dokončit dokumentaci praktického postupu.

Ve vytvořené aplikaci je prováděno měření trvání jednotlivých částí programu. Výsledky těchto měření budou společně s demonstrováním způsobu práce s aplikací prezentovány při obhajobě práce.

Seznam použitých zdrojů

[1] Texas Instruments. *OMAP-L138 C6000™ DSP+ ARM® Processor* [online]. [cit. 03.05.2017]. Dostupné z: http://www.ti.com/lit/ds/symlink/omap-l138.pdf

[2] LOGIC PD. *Zoom™ OMAP-L138 eXperimenter Kit* [online]. [cit. 07.05.2017]. Dostupné z: https://www.logicpd.com/dev_kits/zoom-omap-l138-experimenter-kit/

[3] LOGIC PD. *Zoom™ OMAP-L138 EVM* [online]. [cit. 07.05.2017]. Dostupné z: https://www.logicpd.com/dev_kits/zoom-omap-l138-evm/

[4] Texas Instruments. *Linux Software Development Kit (SDK) for OMAP-L138 Processors* [online]. [cit. 08.05.2017]. Dostupné z: http://www.ti.com/tool/linuxsdk-omapl138

[5] Texas Instruments. *DVSDK_4.03 04_03_00_06 Product Download Page* [online]. [cit. 08.05.2017]. Dostupné z: http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/dvsdk/DVSDK_4_00/latest/index_FDS.html

[6] Texas Instruments. *MCSDK 01_01_00_02* [online]. [cit. 09.05.2017]. Dostupné z:http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest1/index_FDS.html

[7] Texas Instruments. *TMS320C64x+ DSP Cache. User´s Guide* [online]. [cit. 20.05.2017] Dostupné z: http://www.ti.com/lit/ug/spru862b/spru862b.pdf

[8] Texas Instruments. *TMS320C64x+ DSP Megamodule. Reference Guide* [online]. [cit. 22.05.2017]. Dostupné z: http://www.ti.com/lit/ug/spru871k/spru871k.pdf

[9] LOGIC PD. *OMAP-L138 SOM-M1 (eXperimenter Kit) Downloads* [online]. [cit. 18.05.2017]. Dostupné z: https://support.logicpd.com/ProductDownloads/OMAP-L138SOM-M1(eXperimenterKit).aspx

▪ DENX Software Engineering. *U-Boot Design Principles* [online]. [cit. 12.05.2017]. Dostupné z: http://www.denx.de/wiki/U-Boot/DesignPrinciples

[10] Texas Instruments. *eXpressDSP Algorithm Standard – xDAIS Developer´s Kit and xDM* [online]. [cit. 15.05.2017]. Dostupné z: http://www.ti.com/tool/tmdxdaisxdm

[11] Texas Instruments. *DSP/BIOS Real-Time Operating System (RTOS)* [online]. [cit. 15.05.2017]. Dostupné z: http://www.ti.com/tool/dspbios

[12] Texas Instruments. *TI-RTOS Workshop Series 1 of 10 – Welcome* [online]. [cit. 18.05.2017]. Dostupné z: https://training.ti.com/ti-rtos-workshop-series-1-10-welcome

[13] eLinug.org. *GPIO* [online]. [cit. 05.02.2017]. Dostupné z: http://elinux.org/GPIO

[14] Ideas on Board. *Linux UVC driver and tools – FAQ* [online]. [cit. 08.05.2017]. Dostupné z: http://www.ideasonboard.org/uvc/faq/

[15] Texas Instruments Wiki. *C6000 Embedded Design Workshop.* [online]. [cit. 19.05.2017]. Dostupné z: http://processors.wiki.ti.com/index.php/C6000_Embedded_Design_Workshop

[16] Texas Instruments. *C6000 Embedded Design Workshop Using BIOS. Student Guide* [online]. [cit. 19.05.2017]. Dostupné z: http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/bios/BIOS_Workshop_Student_Guide_Rev5_93.pdf

[17] Texas Instruments. *DSP Link Overview* [online]. [cit. 25.05.2017]. Dostupné z: http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/DSPLink/

[18] COOMBS, Joseph and PRABHU, Rahul. OpenCV on TI's DSP+ARM® platforms: Mitigating the challenges of porting OpenCV to embedded platforms. *Texas Instruments* [online]. [cit. 24.05.2017]. Dostupné z: http://www.ti.com/lit/wp/spry175/spry175.pdf

[19] Texas Instruments Wiki. *C66x opencv* [online]. [cit. 23.05.2017]. Dostupné z: http://processors.wiki.ti.com/index.php/C66x_opencv

[20] Texas Instruments. *OpenCV and DSP Selection* [online]. [cit. 23.05.2017]. Dostupné z: https://e2e.ti.com/support/dsp/omap_applications_processors/f/42/t/416531

[21] Texas Instruments. *Building Open CV for ARM Cortex-A8* [online]. [cit. 22.05.2017]. Dostupné z: http://processors.wiki.ti.com/index.php/Building_OpenCV_for_ARM_Cortex-A8

[22] Texas Instruments. *c64p_imglib 2_02_00_00 Product Download Page* [online]. [cit. 19.05.2017]. Dostupné z: http://software-dl.ti.com/dsps/dsps_public_sw/c6000/web/c64p_imglib/latest/index_FDS.html

[23] Texas Instruments. *TMS320C64x+ DSP Image/Video Processing Library (v2.0.1). Programmer´s Guide.* [online]. [cit. 23.05.2017]. Dostupné z: http://www.ti.com/lit/ug/spruf30a/spruf30a.pdf

[24] GitHub. *stránka vývoje modulu gpio-pca953x.c* [online]. [cit. 22.05.2017]. Dostupné z: https://github.com/torvalds/linux/blob/master/drivers/gpio/gpio-pca953x.c

[25] Texas Instruments. *Sitara Linux LCDC fbdev User Space* [online]. [cit. 20.05.2017]. Dostupné z: http://processors.wiki.ti.com/index.php/Sitara_Linux_LCDC_fbdev_User_Space

[26] Texas Instruments Wiki. *Fbtest* [online]. [cit. 24.05.2017]. Dostupné z: http://processors.wiki.ti.com/index.php/Fbtest

[27] Texas Instruments. *How to get output of webcam through VGA port in OMAPL138 LCDK* [online]. [cit. 24.05.2017]. Dostupné z: https://e2e.ti.com/support/dsp/omap_applications_processors/f/42/t/328619#pi317270=1

[28] Texas Instruments Wiki. *Demonstration of VPIF raw capture using MT9T031 sensor on AM18X/DA850/OMAP-L138 running Linux* [online]. [cit. 15.05.2017]. Dostupné z: http://processors.wiki.ti.com/index.php/Demonstration_of_VPIF_raw_capture_using_MT9T031_sensor_on_AM18X/DA850/OMAP-L138_running_Linux

[29] *Zdrojové soubory aplikace uvccapture*. [online]. [cit. 22.05.2017]. Dostupné z: http://staticwave.ca/source/uvccapture/

[30] Texas Instruments Wiki. *Linux Host Support CCSv5* [online]. [cit. 17.05.2017]. Dostupné z: http://processors.wiki.ti.com/index.php/Linux_Host_Support_CCSv5

[31] Spectrum Digital. *Emulation Tech Note 11 Validating Spectrum Digital Emulation Drivers on Linux* [online]. [cit. 19.05.2017]. Dostupné z: http://support.spectrumdigital.com/ccs42/Docs/TechNote_11.pdf

[32] Texas Instruments. *66AK2H14 (ACTIVE) Multicore DSP+ARM KeyStone II System-on-Chip (SoC)* [online]. [cit. 17.05.2017]. Dostupné z: http://www.ti.com/product/66AK2H14

[33] Texas Instruments. *CODECS - Video and Speech- C64x+-based Devices (OMAP35x, C645x, C647x, DM646, DM644x, DM643x)* [online]. [cit. 24.05.2017]. Dostupné z: http://www.ti.com/tool/C64XPLUSCODECS?keyMatch=codec&tisearch=Search-EN-Everything

[34] Texas Instruments. *C64XPlus_Video_CODECS 1_00_003 Product Download Page* [online]. [cit. 25.05.2017]. Dostupné z: http://software-dl.ti.com/dsps/dsps_public_sw/codecs/C64XPlus_Video/index_FDS.html

[35] Texas Instruments, *DSPLink User Guide*, součást dokumentace v instalaci software [17]

[36] Texas Instruments, *DSPLink Programmers Guide*, součást dokumentace v instalaci software [17]

[37] Texas Instruments, *LNK_010_DES - PROCESSOR MANAGER*, součást dokumentace v instalaci software [17]

[38] Texas Instruments, *LNK_012_DES - LINK DRIVER*, součást dokumentace v instalaci software [17]

[39] Texas Instruments, *LNK_024_DES OS Adaptation Layer*, součást dokumentace v instalaci software [17]

[40] Texas Instruments, *LNK_031_DES - Messaging*, součást dokumentace v instalaci software [17]

[41] Texas Instruments, *LNK_040_DES - COFF Loader*, součást dokumentace v instalaci software [17]

[42] Texas Instruments, *LNK_041_DES - Zero Copy*, součást dokumentace v instalaci software [17]

[43] Texas Instruments, *LNK_082_DES - POOL*, součást dokumentace v instalaci software [17]

[44] 1016572B_AM1808_OMAP-L138_Baseboard_Schematic.pdf ve staženém OMAP-L138 eXperimenter Kit Hardware Design Files v [9]

[45] Souborová cesta *%DVSDK\docs\OMAPL138_Software_Developers_Guide.pdf* v instalaci [5]

[46] Souborová cesta *%DVSDK/psp/linux-2.6.37-psp03.21.00.04.sdk/drivers/media/video/uvc* v instalaci [5]

Přílohy

Demonstrování principů snímání přepínače

Připojení souborového systému debugfs je provedeno následujícím příkazem:

```
root@arago:~# mount -t debugfs none /sys/kernel/debug
```

Po něm je debugfs připojen do složky */sys/kernel/debug*. Následně je možné se do příslušné složky přesunout (příkazem *cd*) a zobrazit obsah složky (příkazem *ls*), která obsahuje podsoubory částí jádra podporujících výpisy v debugfs. Zobrazením (příkazem *cat*) obsahu souboru gpio v této složce je pak možné zjistit informace k jednotlivým GPIO pinům.

```
root@arago:~# cd /sys/kernel/debug/
root@arago:/sys/kernel/debug# ls
asoc            davinci_clocks  memblock        musb
bdi             gpio            mmc0            usb
bluetooth       hid             mmc1            wakeup_sources
root@arago:/sys/kernel/debug# cat gpio
GPIOs 0-31, DaVinci:
 gpio-15  (wl1271 BT Enable    ) out lo

GPIOs 32-63, DaVinci:
 gpio-36  (ON_BD_USB_DRV       ) out hi
 gpio-38  (mdio_clk_en         ) out lo
 gpio-40  (lcd pwr
           ) out hi
 gpio-47  (lcd bl
            ) out hi

GPIOs 64-95, DaVinci:
 gpio-64  (MMC CD
            ) in  lo
 gpio-65  (MMC WP
            ) in  lo

GPIOs 96-127, DaVinci:
 gpio-105 (wl12xx              ) out lo
 gpio-106 (wl12xx_irq          ) in  hi
 gpio-109 (ON_BD_USB_OVC       ) in  hi irq-210 edge-both

GPIOs 128-143, DaVinci:
```

Ve výpisu informace k I^2C expanderu ani k jednotlivým přepínačům desky zobrazeny nejsou. Důvodem je chybějící modul ovládající I^2C expander (modul pca953x) a modul řídící GPIO piny snímané pomocí I^2C expanderů (modul i2c-gpio). Po přidání uvedených modulů (příkazem *modprobe*) je již možné stav přepínačů zobrazit.

```
root@arago:/sys/kernel/debug# modprobe pca953x
root@arago:/sys/kernel/debug# modprobe i2c-gpio
at24 1-0050: 32768 byte 24c256 EEPROM (writable)
1-0018 supply IOVDD not found, using dummy regulator
1-0018 supply DVDD not found, using dummy regulator
1-0018 supply AVDD not found, using dummy regulator
1-0018 supply DRVDD not found, using dummy regulator
asoc: tlv320aic3x-hifi <-> davinci-mcasp.0 mapping ok
pca953x 1-0020: failed reading register
pca953x 1-0021: interrupt support not compiled in
regulator: VDCDC1: 3200 <--> 3300 mV at 3300 mV
regulator: VDCDC2: 1750 <--> 3300 mV at 3300 mV
regulator: VDCDC3: 950 <--> 1350 mV at 1200 mV
regulator: LDO1: 1800 mV
regulator: LDO2: 1150 <--> 1300 mV at 1200 mV
input: TPS6507x Touchscreen as /devices/platform/i2c-gpio.1/i2c-1/1-
0048/input/input0
i2c-gpio i2c-gpio.1: using pins 20 (SDA) and 21 (SCL)
input:     gpio-keys-polled     as     /devices/platform/gpio-keys-
polled.1/input/input1
```

```
root@arago:/sys/kernel/debug# cat gpio
GPIOs 0-31, DaVinci:
 gpio-15  (wl1271 BT Enable    ) out lo
 gpio-20  (sda                 ) in  hi
 gpio-21  (scl                 ) in  hi

GPIOs 32-63, DaVinci:
 gpio-36  (ON_BD_USB_DRV       ) out hi
 gpio-38  (mdio_clk_en         ) out lo
 gpio-40  (lcd pwr
           ) out hi
 gpio-47  (lcd bl
            ) out hi

GPIOs 64-95, DaVinci:
 gpio-64  (MMC CD
            ) in  lo
 gpio-65  (MMC WP
            ) in  lo

GPIOs 96-127, DaVinci:
 gpio-105 (wl12xx              ) out lo
 gpio-106 (wl12xx_irq          ) in  hi
 gpio-109 (ON_BD_USB_OVC       ) in  hi irq-210 edge-both

GPIOs 128-143, DaVinci:

GPIOs 160-175, i2c/1-0021, tca6416, can sleep:
 gpio-165 (user_pb1            ) in  hi
 gpio-168 (user_sw1            ) in  hi
 gpio-169 (user_sw2            ) in  hi
 gpio-170 (user_sw3            ) in  hi
 gpio-171 (user_sw4            ) in  hi
 gpio-172 (user_sw5            ) in  hi
 gpio-173 (user_sw6            ) in  hi
 gpio-174 (user_sw7            ) in  lo
 gpio-175 (user_sw8            ) in  lo
```

Nově jsou již ve výpisu zobrazeny i řídící piny expanderu (gpio-20, gpio-21) a jednotlivé přepínače (gpio-165 - gpio-175, resp. user_sw1 - user_sw8). Přepínače jsou

ve výpisu součástí modulu tca6416, který odpovídá typu expanderu uvedeného v HW dokumentaci vývojové desky. Pro každý z pinů je pak uvedena informace, jestli je nastaven jako vstup, či výstup (in/out), jaká je jeho aktuální hodnota (logická 0 - lo, logická 1 - hi) a také jaké má číselné označení.

Pomocí souborového systému debugfs tak bylo ověřeno, že jsou stavy přepínačů v rámci OS viditelné. Ve složce */sys/class/gpio* by pak mělo být možné zobrazovat jejich stavy. Po přesunu do dané složky a provedení příkazu, který umožní exportovat řízení a snímání GPIO pinu do uživatelského prostoru (příkaz zapisující číslo pinu do souboru export; pro sejmutí uživatelského přepínače 8 je to podle výstupu debugfs příkaz *echo 175 > export*) se však složka příslušného přepínače neobjeví.

```
root@arago:/sys/kernel/debug# cd /sys/class/gpio/
root@arago:/sys/class/gpio# ls
export       gpiochip128  gpiochip32   gpiochip96
gpiochip0    gpiochip160  gpiochip64   unexport
root@arago:/sys/class/gpio# echo 175 > export
root@arago:/sys/class/gpio# ls
export       gpiochip128  gpiochip32   gpiochip96
gpiochip0    gpiochip160  gpiochip64   unexport
```

Důvodem je, že přepínač je vlastněn jiným modulem. Není však přímá možnost zjistit jakým. Odebíráním jednotlivých modulů jádra souvisejících s řízením GPIO bylo zjištěno, že přepínač byl blokován modulem gpio-keys-polled. Po jeho odebrání je již řídící složka přepínače přístupná.

```
root@arago:/sys/class/gpio# rmmod gpio-keys-polled
root@arago:/sys/class/gpio# lsmod
Module                  Size  Used by
i2c_gpio                1835  0
i2c_algo_bit            4421  1 i2c_gpio
pca953x                 2784  0
dsplinkk              119140  0
dm365mmap               1716  0
cmemk                  20694  0
ipv6                  230498  12
minix                  25958  0
root@arago:/sys/class/gpio# ls
export       gpiochip128  gpiochip32   gpiochip96
gpiochip0    gpiochip160  gpiochip64   unexport
root@arago:/sys/class/gpio# echo 175 > export
root@arago:/sys/class/gpio# ls
export       gpiochip128  gpiochip32   gpiochip96   user_sw8
gpiochip0    gpiochip160  gpiochip64   unexport
```

Přesunem do řídící složky přepínače je pak možné číst jeho stav a nastavovat jeho jednotlivé řídící soubory. Pro aplikaci byly potřeba dva z těchto souborů - soubor value obsahující aktuální logickou hodnotu přepínače a soubor direction nastavující směr daného GPIO (vstup - in, výstup - out). Výpis obsahu souboru value po změně polohy přepínače ukazuje funkčnost řešení.

```
root@arago:/sys/class/gpio# cd user_sw8/
root@arago:/sys/devices/platform/i2c-gpio.1/i2c-1/1-
0021/gpio/user_sw8# ls
active_low  direction   subsystem   value
device      power       uevent
root@arago:/sys/devices/platform/i2c-gpio.1/i2c-1/1-
0021/gpio/user_sw8# cat direction
in
root@arago:/sys/devices/platform/i2c-gpio.1/i2c-1/1-
0021/gpio/user_sw8# cat value
0
root@arago:/sys/devices/platform/i2c-gpio.1/i2c-1/1-
0021/gpio/user_sw8# cat value
1
```

Problematika kamery - USB 1.1

V úvodu výpisu je hláška, která se objeví po připojení kamery k vývojovému kitu. Jak je vidět ve výpisu, USB ovladače oznámují, že zařízení (kamera) nepracuje v plné rychlosti kvůli rychlostnímu omezení daného USB portu. Po této informaci následuje příkaz (*lsmod*) pro vypsání aktuálně běžících dynamických modulů jádra, který ukazuje, že ovladače pro kameru byly úspěšně vloženy do jádra (především modul uvcvideo).

```
root@arago:~#
usb 2-1: new full speed USB device using ohci and address 2
usb 2-1: not running at top speed; connect to a high speed hub
Linux video capture interface: v2.00
uvcvideo: Found UVC 1.00 device HD Webcam C525 (046d:0826)
input: HD Webcam C525 as /devices/platform/ohci.0/usb2/2-1/2-
1:1.2/input/input0
usbcore: registered new interface driver uvcvideo
USB Video Class driver (v1.0.0)

root@arago:~# lsmod
Module                  Size  Used by
uvcvideo               55251  0
videodev               57723  1 uvcvideo
v4l1_compat            13040  2 uvcvideo,videodev
dm365mmap               1716  0
dsplinkk              119140  0
cmemk                  20694  0
ipv6                  230498  12
minix                  25958  0
```

Následuje zobrazení obsahu složky (příkaz *ls*), který ukazuje, že v aktuální složce se nenachází žádný obrázek. Poté je volána aplikace uvccapture, která slouží k jednoduchému snímání obrázků z kamer podporovaných UVC ovladači.

```
root@arago:~# ls
Final_App
root@arago:~# uvccapture -otest.jpg
root@arago:~# ls
Final_App  test.jpg
```

Dále je vypsán obsah logovacího souboru (příkaz dmesg), který zaznamenává důležité děje v jádře. Výpis je omezen na výskyt slova uvc (příkaz grep uvc za znakem |, který slouží k předávání výstupu příkazu před | do příkazu následujícího za |). Ve výpisu se nacházejí pouze údaje z registrace USB kamery při jejím připojení.

```
root@arago:~# dmesg | grep uvc
uvcvideo: Found UVC 1.00 device HD Webcam C525 (046d:0826)
usbcore: registered new interface driver uvcvideo
```

Logovací soubor však umožňuje zobrazovat i ladící informace k chování kamery. Jejich výpisy je však potřeba aktivovat. To je možné podle postupu uvedeného ve FAQ. Ten zahrnuje odebrání modulu uvcvideo (příkaz rmmod) a jeho následné vložení (modprobe) s parametrem trace. Ladících výpisů je více typů, pro omezení jejich počtu byl zvolen pouze výpis rychlostí komunikace, které po jádře požaduje připojená kamera (parametr trace=0x400 [14]). Poté byl opět sejmut obrázek kamerou, aby došlo k výpisu zadaného parametru do logovacího souboru.

```
root@arago:~# rmmod uvcvideo
usbcore: deregistering interface driver uvcvideo
root@arago:~# modprobe uvcvideo trace=0x400
uvcvideo: Found UVC 1.00 device HD Webcam C525 (046d:0826)
input: HD Webcam C525 as /devices/platform/ohci.0/usb2/2-1/2-
1:1.2/input/input1
usbcore: registered new interface driver uvcvideo
USB Video Class driver (v1.0.0)
root@arago:~# uvccapture -otest_report.jpg
root@arago:~# ls
Final_App         test.jpg         test_report.jpg
```

Následně byl opět zobrazen obsah logovacího souboru. Ten nyní obsahuje informaci o velikosti a počtu alokovaných bufferů a nastavené rychlosti komunikace.

```
root@arago:~# dmesg | grep uvc
uvcvideo: Found UVC 1.00 device HD Webcam C525 (046d:0826)
usbcore: registered new interface driver uvcvideo
usbcore: deregistering interface driver uvcvideo
uvcvideo: Found UVC 1.00 device HD Webcam C525 (046d:0826)
usbcore: registered new interface driver uvcvideo
uvcvideo: Device requested 944 B/frame bandwidth.
uvcvideo: Selecting alternate setting 6 (944 B/frame bandwidth).
uvcvideo: Allocated 5 URB buffers of 32x944 bytes each.
```

USB 2.0

Úvodní chyba USB 2.0.

```
uvcvideo: Failed to submit URB 0 (-90).
```

Tato chyba byla nalezena vždy pouze nevyřešená. Proto bylo zapnuto trasování modulu v jádře. Při jeho stejném parametru trasování se stejným parametrem jako u USB 1.1 byl výpis následující.

```
uvcvideo: Device requested 3060 B/frame bandwidth.
uvcvideo: Selecting alternate setting 11 (3060 B/frame
bandwidth).
uvcvideo: Allocated 5 URB buffers of 32x3060 bytes each.
```

Paměťová mapa procesoru

Start Address	End Address	Size	ARM Mem Map	DSP Mem Map	EDMA Mem Map	PRUSS Mem Map	Master Peripheral Mem Map	LCDC Mem Map
0x1170 0000	0x117F FFFF	1024K	DSP L2 ROM					
0x1180 0000	0x1183 FFFF	256K	DSP L2 RAM					
0x1184 0000	0x11DF FFFF							
0x11E0 0000	0x11E0 7FFF	32K	DSP L1P RAM					
0x11E0 8000	0x11EF FFFF							
0x11F0 0000	0x11F0 7FFF	32K	DSP L1D RAM					
0x11F0 8000	0x3FFF FFFF							
0x4000 0000	0x5FFF FFFF	512M	EMIFA SDRAM data (CS0)					
0x6000 0000	0x61FF FFFF	32M	EMIFA async data (CS2)					
0x6200 0000	0x63FF FFFF	32M	EMIFA async data (CS3)					
0x6400 0000	0x65FF FFFF	32M	EMIFA async data (CS4)					
0x6600 0000	0x67FF FFFF	32M	EMIFA async data (CS5)					
0x6800 0000	0x6800 7FFF	32K	EMIFA Control Regs					
0x6800 8000	0x7FFF FFFF							
0x8000 0000	0x8001 FFFF	128K	Shared RAM					
0x8002 0000	0xAFFF FFFF							
0xB000 0000	0xB000 7FFF	32K	DDR2/mDDR Control Regs					
0xB000 8000	0xBFFF FFFF							
0xC000 0000	0xCFFF FFFF	256M	DDR2/mDDR Data					
0xD000 0000	0xFFFC FFFF							

Obr. 15 Paměťová mapa procesoru s vyznačenými důležitými bloky